Examensarbete
LITH-ITN-MT-EX--05/054--SE

Model Flowing: Capturing and Tracking of Deformable Geometry

Kjell Reuterswärd

2005-10-14

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Science and Technology
Linköpings Universitet
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings Universitet
601 74 Norrköping

LITH-ITN-MT-EX--05/054--SE

Model Flowing: Capturing and Tracking of Deformable Geometry

Examensarbete utfört i medieteknik
vid Linköpings Tekniska Högskola, Campus Norrköping

Kjell Reuterswärd

Handledare Ken Museth
Handledare Doug Roble
Examinator Ken Museth

Norrköping 2005-10-14

Avdelning, Institution
Division, Department

Institutionen för teknik och naturvetenskap

Department of Science and Technology

Datum
Date

2005-10-14

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report category

Examensarbete
☐ B-uppsats
☐ C-uppsats
☒ D-uppsats

☐ _____

ISBN

ISRN **LITH-ITN-MT-EX--05/054--SE**

Serietitel och serienummer
Title of series, numbering

ISSN

URL för elektronisk version

Titel
Title

Model Flowing: Capturing and Tracking of Deformable Geometry

Författare
Author

Kjell Reuterswärd

Sammanfattning
Abstract

In this thesis a new markerless deformable model capture system is presented which is more accurate and controllable than similar systems. Computer vision methods are applied and modified to capture the animated geometry of deformable surfaces such as human faces and cloth. The markerless approach makes simultaneous recovery of animation, texture and lighting possible.

Nyckelord
Keyword

Computer Vision, Animation, Markerless motion capture

Model Flowing: Capturing and tracking of deformable geometry

Kjell Reuterswärd[1]

October 14, 2005

[1] Kjell.Reutersward@dice.se

ABSTRACT

In this thesis a new markerless deformable model capture system is presented which is more accurate and controllable than similar systems. Computer vision methods are applied and modified to capture the animated geometry of deformable surfaces such as human faces and cloth. The markerless approach makes simultaneous recovery of animation, texture and lighting possible.

The system assumes that the initial pose of the deforming object is known and that the deformation is captured by an array of carefully placed cameras. By modifying the basic 2D optical flow algorithm, which has been used in similar work, the movement of the 3D geometry is solved for directly and in all views simultaneously frame by frame. This technique incorporates the epipolar constraint into the solver, reducing the search space resulting in higher accuracy and less accumulated error.

The thesis outlines the basic mathematic tools that have been used such as projective math, 2D warps, Conjugate Gradient Optimization, image registration, and spring systems. It describes how these tools can be combined to build a Model Flow system and how such a system performs in comparison to an optical flow based approach that has been used in production.

CONTENTS

1. INTRODUCTION

1.1 Background

This master thesis is a mandatory part of a MSc degree in media technology and engineering at Linköping University (LiU). It contributes with 20 Swedish university credits towards the degree, which should correspond to twenty weeks of full time work. In practice this means six month of work which for this specific thesis was carried out at the digital studio and visual effect house Digital Domain (DD) in Venice, CA (USA).

I, the author of this thesis, applied for an internship at DD through LiU in the spring of 2004. I was approved by both the LiU and DD during the summer and left for Los Angeles in October of 2004. At this stage I was also assigned one academic supervisor, Professor Ken Museth, and one industrial supervisor at DD, Creative Director of Software Doug Roble.

The subject of my thesis was worked out in a process where my background and my requests were weighed against active research areas at DD and their requests. The project was to be an integrated part of the research and software development at DD and one of the requests was that the results should be submitted as a sketch to SIGGRAPH 2005, and if accepted, presented by me at the conference.

The topic which was initially agreed on was “Facial Capture and Animation”. The project involved extending the existing suite of computer vision tools at DD in a way they would allow capturing of markerless facial animation. DD has over the years noticed a growing need for detailed facial capturing and animation, a process which historically has been an expensive and labor extensive task. This project would hopefully automate this process somewhat and hopefully surpass previous work in the field.

As time passed the project evolved into a slightly wider field of application and the goal was no longer just to capture and animate faces but to develop a system that could capture the animation of any kind of deforming surface. Since this thesis describe a system especially designed for visual effects in motion pictures I will already in the introduction explain some of the central parts of the modern visual effect pipeline and the how my work relates to those parts.

1.2 Modeling and Animation in the entertainment industry

The process of modeling and animating faces can be broken down into a few basic steps. Even though these steps can be used for many types of objects that are animatable the face will be used as an example in this simple walk through.

The first step is to collect reference material of the face. This can be still images, short video clips of the face moving, and different kinds of physical measurement of the face. In motion pictures laser scans of either the face itself or a cast of the face, made from either plaster or latex, is used as a complete 3D-reference of the face. The purpose of the reference material is to capture all the characteristics of the face you want to model and animate.

The second step is to convert this reference material into useful models of the properties of the face. This means both models of skin characteristics, such as color, texture, and light scattering, and the geometric three dimensional (3D) appearance of the face. The process of creating a 3D model of the face is usually just called modeling and is an art form in itself, performed by trained professionals. If a laser scan, or any kind of physical measurements, are available that data is used as guidelines for the artist when the 3D model is sculptured out of mathematical curves and geometrical building blocks such as small triangles and polygons. A common way to represent 3D-geometry in computers are as triangle meshes which consists of hundreds, thousands, or millions of small connected triangles each consisting of three vertices, three edges and one face. This geometric representation will be used through out this thesis.

When the face is represented by a number of models describing all of its attributes the geometric model is given to a "rigger". In the third step of this process the rigger connects every part of the geometric model to handles, knobs, dials and sliders controlling the motion of the different parts of the face. This is usually done through a combination of sliders mixing between blend shapes and sliders controlling physical attributes, such as the dropping the jaw. Blend shapes are specific poses you can blend between. For example it is easy to imagine how a face can be modeled in various stages of happiness and a slider can be connected to those stages and that way act as a control for how happy the face should look. The purpose of rigging is to make the job easier for the animator whose job is to pose and animate the model according to a script. If the rigging is done correctly any desired pose of the face can be recreated by putting the sliders in the right positions.

The forth step is called animation and here an animator uses the animation rig created by the rigger to animate the face according to a script or the directions of a director. This might sound easy but if you consider that a human face has thousands of muscles that acts as controllers for the appearance of the face the task for the animator, who maybe only has twenty controllers at hand to recreate the exact same spectrum of expressions, suddenly seams a lot harder. On top of that humans are extremely well trained to interpret facial expressions and movements.

The fifth and final step in this process is to combine the skin appearance models, the geometric model and the animated movements of the face into synthetic photographs of the face in order to give the impression that this face actually exists and it has been captured performing with a camera. This step is called rendering and is still very much an active area of research and will not be covered at all in this thesis.

As can be seen above there are lots of manual work in this process and for decades research has been done to minimize the work load for the artists. Maybe the most important set of tools to speed up the process goes under the name "Motion Capture". As the name hints the technology makes capturing of motion possible. This way animators can get help from actual actors to "drive" the animation. The data captured with motion capture techniques is usually noisy and seldom rich in detail or even accurate, but the data can usually be cleaned up and modified by the animator. At least enough to spare them many hours of work creating the same data from scratch. Motion capture data can be captured in many different ways ranging from mechanical sensors to ultra sound sensors and computer vision systems. Many of these technologies are not well suited for capturing detailed motions of a deforming surface such as a face. In this thesis a computer vision based system that is specially designed to capture animations of deforming surfaces, such as faces and cloth, will be developed end evaluated. Such a system can both assist the animator and rigger with animated reference material and drive the animation itself which should speed up the production pipeline and hopefully result in higher quality of the final result!

1.3 Tracking and Registration

One of the core technologies of visual effects in motion pictures is the ability to seamlessly combine synthetic or "fake" footage with live action footage. Synthetic material can for example be computer generated material or hand painted matte paintings. "Fake" footage can be miniature footage or other kind of footage that are meant to fool the eye in some way. This technology is in general called compositing since many image sources are combined together in one composite image.

To make this composite believable as an actual photograph many problems have to be solved. One of the most complicated of these problems is camera tracking. Camera tracking means that the position, orientation and optical settings of the camera in the live action footage is captured and matched in the other image sources. An example scene could be a camera panning across a room in which there is a table. If a computer generated vase is to be placed on that table synthetic images needs to be generated, corresponding to the angle from which the camera sees the table. Camera tracking data can either be captured mechanically with robotic cameras and cranes or with computer vision and visual queues in the image material.

When the position, orientation, and the optical parameters of the camera are known, i.e. it is tracked; other objects in the scene can also be tracked and matched with synthetic or "fake" footage. In the example with the table and the vase the table can now be tracked and that way make the vase stick to the table even if someone in the scene nudges it over the floor or rotates it. Traditionally only rigid object such as boxes, walls, pillars, and tables can be tracked in a scene. In this thesis a computer vision based system that is specially designed to also track deforming objects such as faces and cloth, will be developed and evaluated. Such a system allows synthetic and "fake" footage to be matched with deforming surfaces.

1.4 Problem Statement

Both when the deformation of an object is captured for tracking reasons or for the sake of driving an animation rig the output should be an animated 3D model representing the geometry and the deformation of that geometry. In the visual effects industry it is reasonable to assume that a static 3D model of good quality is delivered by a modeling artist. Probably numerous tracked cameras can be placed around the object on the set, but it is not certain that neither the lighting conditions nor the surface properties of the object can be controlled since the lighting and the appearance of the object might also be captured for the final shot. This implies a scenario where the deformation of an animated object is wanted, given numerous camera feeds of the deforming object and a static geometric model of the object. In the worst case the lighting of the scene cannot be altered or markers be placed on the surface of the object.

The object of this thesis is to develop and evaluate a system that can track and capture deforming geometry under these conditions using computer vision. The system is customized for the creation of visual effects in feature films but the algorithm can be used in other situations as well. Both the algorithm and the implementation presented in this thesis are specifically developed for this thesis by the author and the co-authors mentioned under acknowledgements.

2. MATHEMATICAL OVERVIEW

In this section the mathematical core technologies of this thesis will be introduced and briefly explained. Since all of these methods and algorithms are well established in computer graphics and proven in numerous applications and scientific papers there will be no attempts to prove or explain them in detail.

2.1 Naming conventions

In this thesis the following naming convention will be used:

Notation	*Alt. notation*	*Description*
x, y, z	x, y, z	3D coordinates in space.
u, v	u, v	2D image coordinates, u = horizontal and v = vertical.
u_*, v_*, w	u_*, v_*, w	2D homogenous image coordinates.
A, **B**, **M**, **S**, …	$A, B, M, S, ...$	Capital letters usually represents matrices.
a, **b**, **m**, **s**,…	$\vec{a}, \vec{b}, \vec{m}, \vec{s}, ...$	Bold lower-case letters or letters with an arrow on top usually represents vectors.
a, b, m, s, …	$a, b, m, s, ...$	Regular lower-case letters usually represents scalar values.
$\mathbf{A}^T$	A^T	Matrix transpose
$\mathbf{A}^{-1}$	A^{-1}	Matrix inverse

2.2 Transformations and homogeneous coordinates

In computer graphics linear mappings from one particular space to another is called transformations. There are many different transformations and some examples are 2D translations (2D to 2D), 3D rotations (3D to 3D) and perspective projections (3D to 2D). Transformations are essential to computer graphics. Mathematically it is basically linear algebra where transformations are performed through matrix multiplications. Even translations, which also can be performed by additions, can be performed by matrix multiplications if homogeneous coordinates are used.

Homogenous coordinates are created by adding an extra "dimension". In 2D that means (u,v)→(u,v,1) and in 3D in means (x,y,z)→(x,y,z,1). Since the extra "dimension" is not guaranteed to equal one, after for example perspective projections, it is often referred to as w. To get a 2D point out of a 3D homogenous vector the vector is simply divided by w which will guarantee that w equals one. When projecting it is the divide that performs that actual flattening of dimensions.

2.2.1 Projective transformation

Since the world is in 3D and a computer screen or a digital image is in 2D the mapping between 3D and 2D is crucial in computer graphics and computer vision. Transformations that map a space of higher dimensionality to a space of lower dimensionality are called projective transformation and in this thesis perspective projections are in particular interesting since they mimic the behavior of both human eyes and photographic cameras fairly well. There are other kind of projective transformations such as orthographic projection but those will not be discussed here. In computer graphics text books the perspective projection matrix often looks like this:

$$\begin{bmatrix} x_* \\ y_* \\ z_* \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Eq. 2.1

Where "d" is the distance along the z-axis where the projective plane is located. In this thesis though, for the sake of simplicity and pedagogical reasons, the same equation will be written this way:

$$\begin{bmatrix} u_* \\ v_* \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Eq. 2.2

And especially when an arbitrary triangle in space is projected onto an image plane the following matrices are multiplied:

$$\begin{bmatrix} u_{*1} & u_{*2} & u_{*3} \\ v_{*1} & v_{*2} & v_{*3} \\ w_1 & w_2 & w_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \\ 1 & 1 & 1 \end{bmatrix}$$

Eq. 2.3

Or even simpler **S** = **P***__V__, where **S** is a 3x3 matrix with three homogenous screen coordinates describing a 2D triangle, **P** is 3x4 projection matrix, and **V** is a 4x3 matrix with 3 homogeneous vertex coordinates describing a 3D triangle in space. More sophisticated camera models are out of scope of this thesis and in this particular system the creation of the projection matrices was taken care of by propriety computer vision code at Digital Domain.

2.2.2 2D Warps

2D warps are typically defined as linear transformations that can map a 2D coordinate in an image to another 2D coordinate in another image using a 3x3 matrix. However, sometimes 2D warps also include mappings that cannot be described with a 3x3 matrix and this expression takes account of them as well:

$$\mathrm{I}(u,v) \rightarrow \hat{\mathrm{I}}(\hat{u},\hat{v})$$

Eq. 2.4

This thesis will concentrate on the warps that can easily be described with linear matrix algebra though:

$$\begin{bmatrix} \hat{u}_* \\ \hat{v}_* \\ w \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Eq. 2.5

And especially the case where an arbitrary 2D triangle in an image is warped into another triangle:

$$\begin{bmatrix} \hat{u}_1 & \hat{u}_2 & \hat{u}_3 \\ \hat{v}_1 & \hat{v}_2 & \hat{v}_3 \\ w_1 & w_2 & w_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \cdot \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ 1 & 1 & 1 \end{bmatrix}$$

Eq. 2.6

Which in compact notation is **T** = **W*****S** where **W** is a 3x3 warp matrix, **S** is a 3x3 source matrix describing the source triangle, and **T** is a 3x3 target matrix describing the target triangle. It will later in this thesis be clear that the 2D warp is a great instrument when implementing Model Flowing since much of the implementation is based on the ability to warp and compare pixels in two different images.

2.3 Conjugate gradient Optimization

There will be no attempt to fully explore the subject of conjugate gradient optimization in this thesis since there is plenty of good literature on the subject ([15] [2][20]) and since no new work on conjugate gradient optimization was done for this project. Conjugate gradient optimization will be used later in this thesis though, which is the reason for this brief introduction to the concept. Conjugate gradient optimization is one of the most effective and popular algorithms for finding the nearest local minimum of a function of n variables in situation where the gradient of the function can be computed. It is a highly effective method for symmetric positive definite systems, which is why it is used in this

thesis, and contrary to many similar algorithms it uses the conjugate directions instead the local gradient for going downhill towards the minima.

The method works by generating successive approximations to the solution (i.e. a vector sequence of iterates), residuals corresponding to the iterates, and search directions used in updating the iterates and residuals. Only a small number of vectors need to be kept in memory even when the length of the sequences becomes large. A typical system to be minimized by conjugate gradient optimization can be described by **Ax=b**, where **x** is a vector of variables for which you solve for and **A** is a positive definite matrix. In "Templates for the Solutions of Linear Systems: Building Blocks for Iterative Methods" [2] the conjugate gradient algorithm is described with the following pseudo code:

$$
\begin{aligned}
&\textit{Compute } \vec{r}^{(0)} = \vec{b} - A\vec{x}^{(0)} \textit{ for some initial guess } \vec{x}^{(0)} \\
&\textit{for } i = 1,\ 2,\ ... \\
&\quad \rho_{i-1} = \vec{r}^{(i-1)^T} \vec{r}^{(i-1)} \\
&\quad \textit{if } i = 1 \\
&\qquad \vec{p}^{(1)} = \vec{r}^{(0)} \\
&\quad \textit{else} \\
&\qquad \beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}} \\
&\qquad \vec{p}^{(i)} = \vec{r}^{(i-1)} + \beta_{i-1} \vec{p}^{(i-1)} \\
&\quad \textit{endif} \\
&\quad \vec{q}^{(i)} = A\vec{p}^{(i)} \\
&\quad \alpha_i = \frac{\rho_{i-1}}{\vec{p}^{(i)^T} \vec{q}^{(i)}} \\
&\quad \vec{x}^{(i)} = \vec{x}^{(i-1)} + \alpha_i \vec{p}^{(i)} \\
&\quad \vec{r}^{(i)} = \vec{r}^{(i-1)} - \alpha_i \vec{q}^{(i)} \\
&\quad \textit{Check convergence; continue if necessary} \\
&\textit{end}
\end{aligned}
$$

Do note that for scalars the iteration "i" is indicated with a subscript like "a_i" and for vectors it is indicated by $\mathbf{a}^{(i)}$ to avoid confusion regarding indexing of the vector elements. Luckily there is a free open source C++ library where the pseudo code above and many other matrix operations and algorithms are implemented in very efficient C++ code. The library is called SparseLib[21] and is especially designed to handle huge but sparse matrices in scientific computations. SparseLib is used in the implementation of the system described in this thesis.

2.4 Image registration

Image registration is one of the fundamental problems in the fields of computer vision and image processing. The basic idea is to transform or deform the pixels in an input image in a way they line up as close as possible with the pixels in a template image. This problem can have a number of different sets of variables depending on the kinds of transformations or deformations one allows. Over the years different algorithms and refinements of those have been presented. A classic, and powerful, algorithm is the iterative image registration technique presented by Lucas and Kanade in 1981 [11]. Since the image registration element of the capturing system presented in this thesis is very similar to the Lucas-Kanade algorithm, the algorithm will now be described in short. For more details on the algorithm the original paper by Lucas and Kanade is highly recommended.

As with all image registration the goal of the Lucas-Kanade algorithm is to align a 2D template image T(u,v) to an input image I(u,v). Algebraically that amounts to minimize the sum of the squared error between the template T and a warped version of the input I. If a column vector containing the pixel coordinates is defined as $\mathbf{u} = (u,v)^T$ and a vector of parameters as $\mathbf{p} = (p_1, \ldots p_n)^T$, $W(\mathbf{u};\mathbf{p})$ denotes the parameterized set of allowed warps. $W(\mathbf{u};\mathbf{p})$ is the mapping between pixel $\mathbf{u}$ in template T and the subpixel coordinates $W(\mathbf{u};\mathbf{p})$ in the input image I. $W(\mathbf{u};\mathbf{p})$ can behave like a simple 2D translation or describe a much more complex deformation with an arbitrary number of parameters n. Mathematically this is a minimization problem where the following equation is to be minimized:

$$\sum_{all\ pixels\ \vec{u}\ in\ T} \left[I(W(\vec{u};\vec{p})) - T(\vec{u}) \right]^2$$

Eq. 2.7

The minimization is typically a non-linear optimization with respect to $\mathbf{p}$. Lucas and Kanade assumes that there is a current estimate of $\mathbf{p}$ from which the equation can be approximately minimized by iterating the following two steps until the parameters $\mathbf{p}$ converges or a lower threshold for the error is reached:

$$Calculating\ Error: \quad \sum_{all\ pixels\ \vec{u}\ in\ T} \left[I(W(\vec{u};\vec{p}+\Delta\vec{p})) - T(\vec{u}) \right]^2$$

$$Updating\ Parameters: \quad \vec{p} \leftarrow \vec{p} + \Delta\vec{p}$$

Eq. 2.8

The Lucas-Kanade algorithm is a Gauss-Newton gradient descent non-linear optimization algorithm. To make the error calculating step linear, a first order Taylor expansion of I(W) is performed which gives us:

$$\sum_{all\ pixels\ \vec{u}\ in\ T}\left[I(W(\vec{u};\vec{p}))+\nabla I\frac{\partial W}{\partial \vec{p}}\Delta\vec{p}-T(\vec{u})\right]^2$$

Eq. 2.9

Where

$$\nabla I=\left(\frac{\partial I}{\partial u},\frac{\partial I}{\partial v}\right)$$

Eq. 2.10

is the image gradient of I evaluated at W(**u**;**p**) and the term

$$\frac{\partial W}{\partial \vec{p}}$$

Eq. 2.11

is called the Jacobian of the warp. The resulting equation has a closed form solution which can be derived by taking the partial derivative with respect to delta **p,** setting that expression to equal zero and solve for delta **p**. Like this:

$$Partial\ derivative: \quad \sum_{all\ pixels\ \vec{u}\ in\ T}\left[\nabla I\frac{\partial W}{\partial \vec{p}}\right]^T\left[I(W(\vec{u};\vec{p}))+\nabla I\frac{\partial W}{\partial \vec{p}}\Delta\vec{p}-T(\vec{u})\right]$$

$$Solve\ for\ \Delta\vec{p}: \qquad \Delta\vec{p}=-H^{-1}\cdot\sum_{all\ pixels\ \vec{u}\ in\ T}\left[\nabla I\frac{\partial W}{\partial \vec{p}}\right]^T\left[I(W(\vec{u};\vec{p}))-T(\vec{u})\right]$$

Eq. 2.12

Where H is the nxn Gauss-Newton approximation to the Hessian matrix:

$$H=\sum_{all\ pixels\ \vec{u}\ in\ T}\left[\nabla I\frac{\partial W}{\partial \vec{p}}\right]^T\left[\nabla I\frac{\partial W}{\partial \vec{p}}\right]$$

Eq. 2.13

All the key steps in the Lucas-Kanade algorithm have now been presented. The complete iterative loop looks like this:

- Calculate new delta **p**
- Calculate error
- If error is getting lower: update parameters and start over, else: Done!

2.4.1 Optical Flow

Given two consecutive frames in a continuous image sequence the optical flow is the 2D vector field which, to the extent possible, describes the motion of each pixel from frame A to frame B (or from frame B to frame A). Another loose definition of optical flow is the apparent motion of the image color pattern. In the perfect of worlds the optical flow would also give information of how objects in the scene have moved, between the two instances in time when the images were sampled. For the idea of a correct optical flow to be valid a long list of assumptions need to be made. These are some of the more important ones:

- Objects do not change color between two consecutive frames.
- There is no change of occlusion in the scene.
- No object leaves or enters the scene.
- All objects has texture that can be tracked.
- All moving surfaces are Lambertian (no specular highlights).
- The lighting condition is constant.
- There is no photometric distortion.

These assumptions are in practice never all true but in most cases true to some extent if the frames are captured close enough in time. This makes optical flow, in all its simplicity, a surprisingly useful computer vision tool. But even if the idea of optical flow is fairly simple it is not trivial to calculate in most cases. Ideally the flow of every single pixel is calculated but in reality that is too expensive and the result would be very noisy. Since neighboring pixels usually move in similar paths it is fair to assume that the flows of only a subset of pixels need to be calculated. The flow for the pixels in between can be interpolated.

A sequential image sequence, such as most normal video clips, usually fulfills the requirements of image registration fairly well. That is, almost any given sequential frame pair is almost identical in color and contents and they, almost without exception, picture the same objects. With that said it is very understandable that image registration algorithms are used to approximate optical flow.

When an image registration algorithm, such as the one presented by Lucas and Kanade, is used to calculate optical flow a set of key points in the image are chosen. These key points can either be evenly spaced on a grid or placed according to some image processing scheme. If the Lucas-Kanade algorithm described above is used each key point is given at least two parameters, one for horizontal translation and one for vertical translation. Other parameters, that for example enforce global smoothness of the vector field, can be added to the solver before the minimization begins. The areas between the key points are warped according to the movement of the closest key points when the error is

calculated for every iteration. When the optimal movements of the key points have been found sub pixel optical flow for each and every pixel can be interpolated.

2.5 Spring Systems

Many physical simulations of membranes, solids and surfaces are simulated using systems of particles interconnected with springs. Sometimes masses are assigned to the particles (mass-spring systems) and a range of different kinds of linear and non-linear springs can be used. In all these systems the spring forces are calculated and summed up for each particle. Other kind of forces such as friction, gravity, drag, and inertia can easily be added for each particle and the complete system can be simulated over time with either explicit or implicit time integration. Systems of this kind has proven to simulate the inner forces of materials relatively well and especially materials like paper and cloth seam to be particularly well suited for this kind of simulation model. Surface characteristics such as bending, stretching, shearing and even ripping can be mimicked in at least a visually correct way (i.e. the solution looks plausible).

Cloth simulations in computer graphics are often simulated with mass-spring systems [1][13]. For cloth three different springs are typically used: stretching springs, bending springs, and shearing springs. Stretching springs simulates the elasticity in cloth in the vertical and horizontal direction by connect quads of neighboring particles in squares (see figure 2.1). Bending springs interconnects every other row and column of particles with longer springs which gives the material stiffness and resistance to bending (see figure 2.2). Shearing springs locks the stretching quads over the diagonal to limit shearing motions of the material (see figure 2.3). In the following three figures the layouts of the three different kinds of springs in a cloth simulation are illustrated.

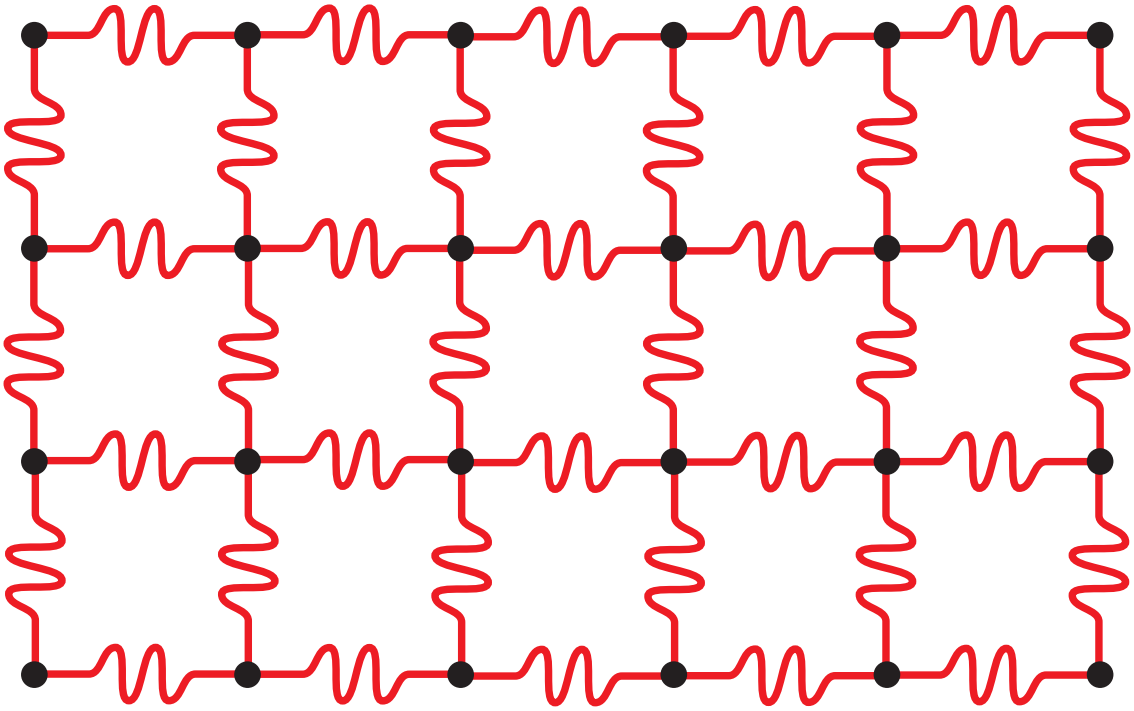


Figure2.1 *Stretching springs*

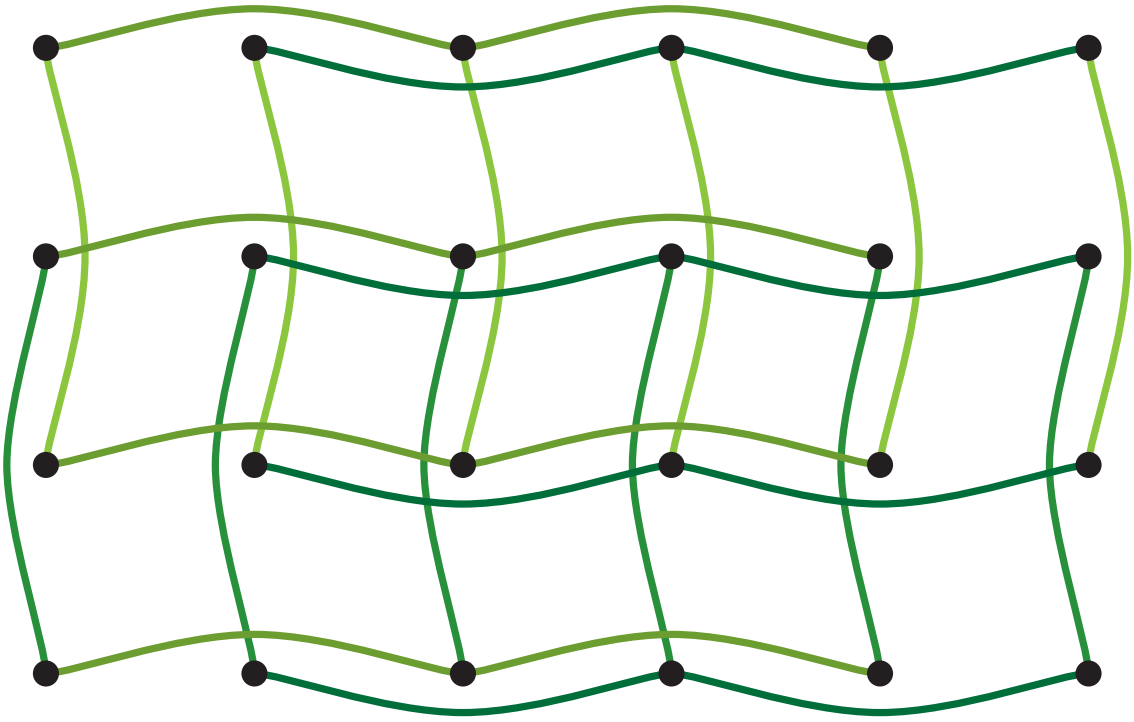


Figure 2.2 *Bending springs*

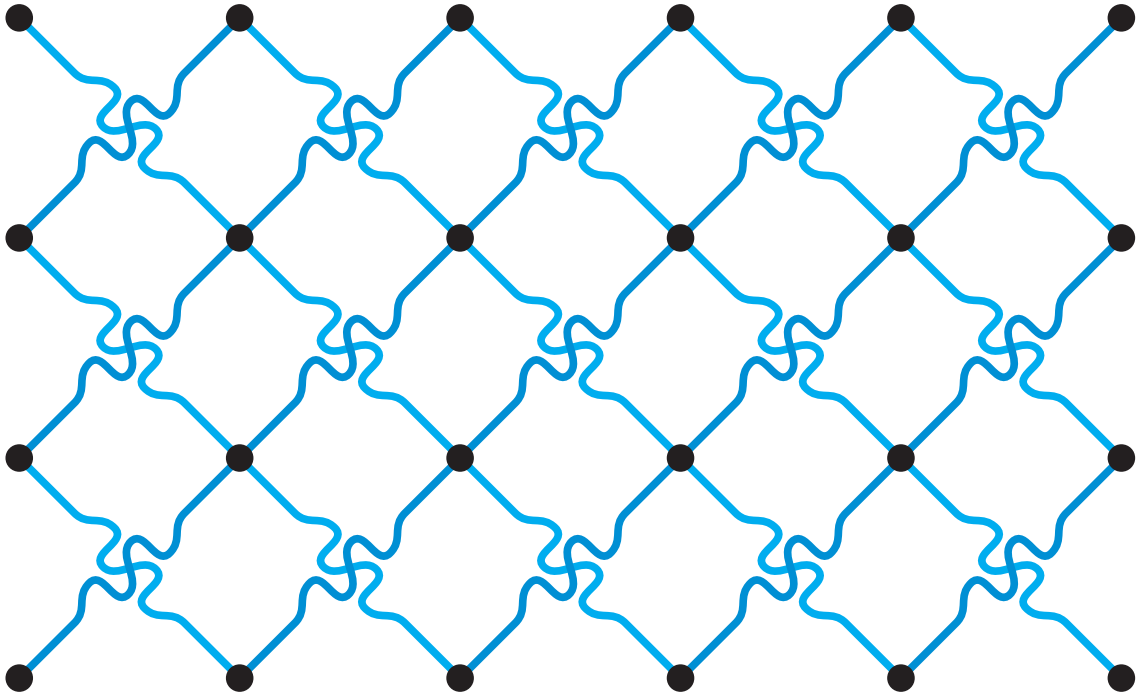


Figure 2.3 *Shearing springs*

The math and physics in spring models is quite simple. The characteristic element of springs is that they always strive to get back to its equilibrium state. This manifests itself through forces acting on its ends toward the center of the spring if the spring is extended or away from the center if the spring is compressed. The simplest model is the linear spring which is described by Hooks law:

$$F = -k \cdot \Delta x$$

Eq. 2.14

Where F is the force along the spring, k is the spring constant, and delta x is the amount the spring has been compressed or extended ($x\text{-}x_0$). The minus sign can be included in the spring constant but emphasizes that the force has opposite sign than the change of length. The springs that will be used in the capturing system described in this thesis will be slightly more advanced and the forces will be calculated according to:

$$F = -\frac{k \cdot (\Delta x)^3}{\mathrm{T}^2}$$

Eq. 2.15

Where T is a threshold value for the ratio between delta x and the original length of the spring. I.e. if T is 0.10 the threshold is 10% of the original length of the spring. Below the threshold the spring will be softer than the linear case. Above the threshold the spring gets stiffer fast since the function is a cubic function. Such a spring will allow small fluctuations but will punish large

changes. Exactly the kind of behavior the Model Flowing algorithm will benefit from, where expected deformations are allowed but extreme distortions should be panelized and unrealistic solutions prohibited.

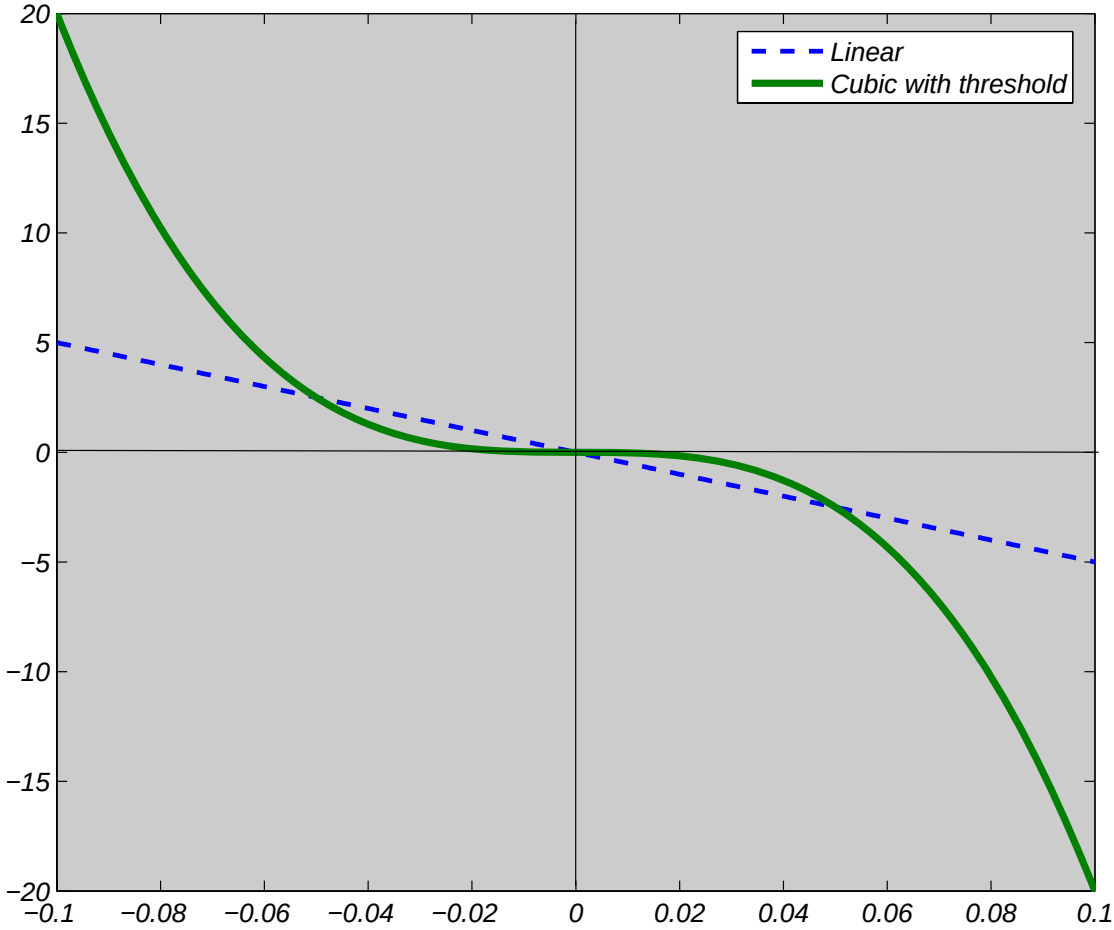


Figure 2.4 *This plot illustrates how the spring force for the cubic spring is small below the threshold, which will let the spring flex quite freely around its equilibrium state, but quickly grows large when the spring is contracted below, or extended beyond, the threshold. The blue dotted line represents a linear spring with a spring constant of 50 while the green solid line represents a cubic spring with threshold of 0.05 (5%) and a spring constant of 50.*

3. RELATED WORK

In this section previous work relating to capturing of deforming geometry in general, and capturing of faces in particular, will be discussed. As in many other fields of computer graphics there are two distinctive categories of earlier work: research done by graphics labs at universities, schools, and dedicated research facilities, and work done on productions in the entertainment industry. The first category is very well documented and recognized by a high degree of automation and in most cases dubious results from a artistic standpoint (mostly because of lack of funding and/or artistic skills). The latter category is very seldom documented in detail and usually has a large degree of user supervision from talented artists and the visual results are generally stunning. The work presented in this thesis is very much in between industry and academia and therefore references from both camps have been used extensively.

3.1 Facial Capturing

Faces are still somewhat of a holy grail in computer graphics. For tens of years massive research has been done on everything ranging from facial recognition and facial scanning to facial modeling, animation and rendering, and still loads of new research is presented every year on graphics subjects relating to faces. One face-related area of research that has been around for a long time and still is going strong is automated capturing of facial animation or facial capturing in short. There are many reasons for capturing the movements of faces and some of the more common objectives of research done on the subject are:

- Capturing and isolating the animation, which in turn can be applied to other faces or geometric models.
- Capturing and isolating the geometric properties of a face, with the intention of animating by hand or with the motions captured from another face. [19]
- Capturing and isolation the reflectance properties of the face in order to be able to relight the animated face afterwards. [10]
- Capturing and isolating the animation in terms of poses, i. e. the face is parsed to a set of already defined poses such as happy, sad or grumpy, or a mix between those poses. [4][12]
- Capturing and isolating the geometric properties of a face in terms of poses, with the intention of defining a mapping between the geometry of the face and already defined poses such as happy, sad or grumpy. [10]

In the two following sections some of the major influences in this project will be described and similarities and differences will be discussed.

3.1.1 Using markers

Traditionally tracking markers, or at least well defined features, have been used for all kinds of motion capture and tracking in general. A tracking marker can be anything from a piece of green tape on the wall in the background to hi-tec super reflective silver spheres on a motion capture suit to a simple dot on a human nose made with a pen. Williams [17] pioneered the technique of facial capturing with markers, tracking 2D points on a single image. Guenter et al. [9] extended this approach to tracking points in 3D using multiple images. Terzopoulos and Waters [16] estimated muscle contractions from the displacement of a set of face markers. Simulating muscles to some extent is a fairly common approach to limit the search space and thus making the capturing process more robust. Muscle simulations also fit well into the concept of blend shapes and model rigging in the animation pipeline.

In the visual effects industry tracking markers are well established in the pipeline and most successful facial capturing has been done with markers. Unfortunately a lot of the capturing done in production is not well documented but the research facility ICT of University of Southern California has both done substantial academic work on facial capturing and consulting work for the industry. In [10] Hawkins et al describes the system ICT has developed to capture facial animations and animatable facial reflectance fields. The system has been used for facial capturing in the academy award winning motion picture Spiderman II among other movies. The ICT capturing rig is a good example of the state of the art in facial capturing with markers.

The system is built around a light stage which can very rapidly light an object more or less arbitrary which enables them to capture the facial reflectance field while capturing the facial animation. Apart from the light stage, the system is quite traditional in the sense that it uses 6 cameras for capturing and the deforming model is basically a triangle mesh connecting the markers on the face. The face is initially captured in approximately 60 different poses, covering the most common expressions, visemes, head poses, and eye positions. Since they use multiple geometrically calibrated cameras they can triangulated the 3D-positions of the fiducial dots, i.e. the markers, and thus form the basis of their 3D face model. They use approximately 300 facial markers which is a fairly large number of dots in a human face but still a rather coarse representation of a human face.

When all poses are triangulated in 3D they can define them as blend shapes. The animation can now be driven by the actor, deforming the geometry through combining/blending the captured blend shapes. The strength of this approach is the robustness due to the fiducial dots and the limited search space due to the fixed number of blend shapes. The biggest drawbacks are the substantial work involved in creating the bland shapes and the coarseness of the final animation both because of the limited number of markers and blend shapes. The fiducial dots has to be removed before the data can be used for rendering, a process which both degrades quality and takes time and effort.

3.1.2 Without markers

Even if tracking markers is an established tool in motion capture they are not suitable in all situations. As computers have become more powerful and digital video cameras have gotten faster and sharper (higher resolution and less noise) more advanced algorithms have emerged which opens up new possibilities and problems. Most markerless tracking solutions are very similar to their counterparts using markers. Instead of tracking the markers, image registration or optical flow algorithms are usually used to drive the deforming geometry.

An interesting exception is the work done by Zhang et al. [Zhang et al. 2004] which uses structured projected light to eliminate the need for traditional markers or even textures in the scene. Since they project light patterns onto the face the image data is quite useless for pretty much anything else and the annoyance factor for the actor/actress is not negligible. When the person is captured they can drive the animation without the light patterns though, and the results are very good.

Others have calculated the optical flow from the image sequence and decomposed the flow into muscle activations. Essa and Pentland designed and implemented a physically-based face model and developed a control-theoretical technique to fit it to a sequence of images [6][7][8]. Douglas DeCarlo and Dimitris Metaxas have done some excellent work on combing optical flow, edge information and Kalman filtering [5] to track faces with great results. Unfortunately the face models they use are not very detailed and they solve for a very limited number of parameters, but their method seams to be very robust.

At SIGGRAPH of 2003 Borshukov et al. [3] presented their "Universal Capture" approach in a sketch. Their capturing approach is somewhat similar to the one used for Model Flowing, but when they calculate 2D optical flow on the camera feeds independently the Model Flow algorithm solve for a flow in 3D right away. The strength of the Model Flow approach is that it incorporates the epipolar constraint in the solver and not in a separate triangulation step.

4. SYSTEM OVERVIEW

The most common approach for capturing deforming geometry such as faces is to use multiple cameras and stereo triangulation. The capturing system described in this thesis is taking this approach although it is more targeted towards feature film productions and the tracking pipeline at Digital Domain. The cameras used in this setup are assumed to capture color with three channels and have fairly high resolution even though the algorithm is very flexible when it comes to resolution and image format. The cameras do not have to be of the same type, be color calibrated or have the same resolution, but they need to be tracked in space, optically calibrated, reasonably synchronized, and carefully placed. Potentially a moving "hero" camera could be used on set even though such a setup has not yet been tested and will not be evaluated in this thesis.

4.1 Physical Setup

For facial capturing in feature film productions it is common to use between 5 and 8 cameras positioned in a way such that all interesting parts of the face is captured by at least 2 cameras at all time. When capturing cloth or any other deforming object the number of cameras may have to go up or down depending on the situation but still all geometry that is to be captured need to be captured by at least 2 cameras in every frame. Ideally the camera views should both overlap to a large extent and be spread wide apart, which is an inherited dilemma of stereo triangulation. The process of putting the capturing system together and capturing deforming geometry includes the following key steps:

- Positioning of the cameras around the object to be captured.
- Calibrating of the cameras. That is finding the intrinsic parameters.
- Tracking of the cameras. That is finding the extrinsic parameters. If one or more of the cameras will be moving during the shot that movement needs to be tracked as well.

4.2 Camera Calibration

Calibrating a camera is basically to estimate the values of the intrinsic and/or extrinsic parameters of the camera. The intrinsic parameters of a camera are the parameters necessary to link pixel coordinates of an image point to a corresponding coordinate of the theoretical image plane of a camera. These parameters are the focal length of the camera, the location of the image center in pixel coordinates, the effective pixel size in horizontal and vertical direction, and, if required, the radial distortion coefficient. Estimations of the intrinsic camera parameters are usually calculated by linking the known coordinates of a set of 3D points and their projections in an image. In practice this means that a test pattern of some sort is waved around in front of a camera to generate test

images that can be processed in special calibration software. The test pattern is usually some sort of checkerboard that can easily be tilted and moved within the camera view. A common set of images for camera calibration consists of roughly 20 images of the checkerboard rotated around all its axles and moved in all direction covering all corners of the image. The capturing system described in this thesis requires that all cameras used for capturing are calibrated and that the resulting image sequences are rectified accordingly.

Calibrating cameras is somewhat of a necessary evil for almost all kinds of computer vision applications. It is far from a solved problem and still papers are published on the subject. This thesis will not attempt to fully cover the subject of camera calibration.

4.3 Camera Tracking

Extrinsic parameters are the parameters that define the location and orientation of a camera with respect to a known world reference frame. In the movie industry the process of finding the extrinsic parameters of a camera is usually known as "tracking the camera". At Digital Domain a proprietary software package named ***TRACK*** is used by "trackers" to capture the necessary extrinsic parameters. ***TRACK*** can both handle automated tracking and more traditional tracking where the images are lined up with simple 3D-geometry more or less by hand. The concrete results of tracking a camera in a shot are the animated (if the camera, or the world, is moving) translation vector and the animated (if the camera, or the world, is rotating) rotation matrix describing the placement and movement of the camera throughout the shot. This information is usually exported and passed on to animators, modelers and compositors. In the case of capturing deforming geometry in 3D this information is vital for triangulation.

The easiest way to track cameras in a shot is to incorporate some kind of known geometry in the shot. When multiple and/or static cameras are used custom made "tracking objects", such as tracking cubes, can be used. A tracking cube is a carefully constructed wire frame cube with well defined tracking markers placed in its corners and sometimes along its edges. With a tracking cube in the scene it is straight forward to calculate the exact position and orientation of the cameras in ***TRACK***. Camera tracking can also be achieved with advanced mechanical robotic arms that can export, and repeat camera positions and movements with astonishing accuracy. Equipment for mechanical camera tracking is very expensive and rarely used other than when a shot has to be completed with multiple and identical passes.

4.4 Camera positioning

To get optimal results from triangulation the base line (the distance between the lines of sight of the cameras) should be large. At the same time there should be a substantial overlap in image information. To triangulate the position of a piece of geometry that piece needs to be captured by at least two cameras at all time.

5. MODEL FLOWING

In this section the heart and soul of Model Flowing will be described. First the algorithm will be summarized fairly quickly followed by a simple walk-through using the simplest example possible. Last in this chapter the different parts of the algorithm will be discussed in detail. For even more details chapter 6 is especially devoted to implementation.

At this point of the capturing process a suitable data set has been captured with a setup described in chapter 4. In other words: for the Model Flow algorithm to work the following requirements has bee fulfilled:

- The deformation was captured with multiple calibrated, synchronized, and tracked cameras.
- There is a polygon model describing the geometry in its initial state.
- The polygon model can be represented with triangles and it is perfectly lined up with the captured image sequences in frame 0.
- Every part of the geometry was captured by at least two cameras at all time.

A fundamental concept for this algorithm is that it finds the best possible deformation one frame at the time in sequence starting with the very first frame. This means that the algorithm will treat the state of the system in frame 0 as absolute truth when calculating the deformation in frame 1. When the best possible deformation in frame 1 is found it will go to the next frame and solve for the best possible deformation in frame 2 treating the state in frame 1 as absolute truth. From now on the frame which is already lined up will be called the current frame or frame 0. The frame which is solved for will be called the next frame or frame 1. Since the deformation has been captured with multiple cameras a "frame in time" corresponds to multiple images.

As the name of the algorithm suggests the algorithm is "flowing" the 3D geometric model from one frame to the next. It is important to understand that 2D pixels are not flowing, as in 2D optical flow, but the actual 3D geometry. In order to optimize this 3D flow the captured image data is used in a cost function and in the estimated derivative of this cost.

The core idea is that by deforming the geometry in the next frame, and by projecting the image data from the next frame onto that deformed geometry, the current frame can be synthesized. The difference between the pixels in the actual current frame and the synthesized current frame is the cost function that will be minimized. This cost is a function of the vertex positions in the next frame so reducing it will deform the geometry. Do note that the cost always is calculated as the difference between two frames from the same camera but the

cost is summed up for all camera views in the solver. Thus the Epipolar constraint is enforced in the solver and all image data contribute to one single deformation solution.

By deriving the cost function with respect to the x-, y-, and z-coordinates of the geometry in the next frame the gradient of the error can be estimated as a function of image derivatives. Similar to the Lucas and Kanade [11] image registration algorithm presented in chapter 2. Analytic derivatives to each parameter are not available and numeric differencing is prohibitively expensive.

5.1 The Algorithm – A walk through

In the most basic case there is one triangle captured by two cameras according to the figure on the right. Of course in a real world scenario there would be substantially more triangles and probably more cameras as well.

If the system is correctly initialized there is a perfect match between the geometry and the images in the current frame.

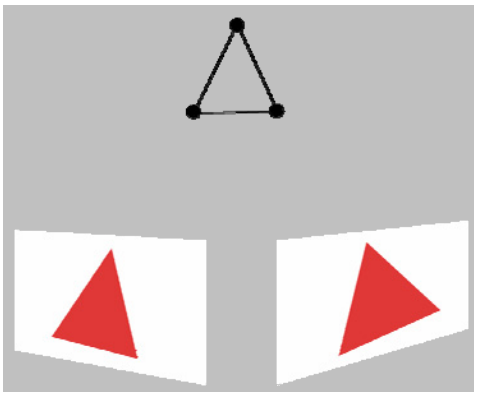


Figure 5.1

The geometry can now be projected out onto the image planes as in this figure. Using a simple scan line rasterizer each and every pixel corresponding to this triangle can be identified. These are the pixels that will be synthesized and compared. The number of pixels defines the number of residuals in the solver.

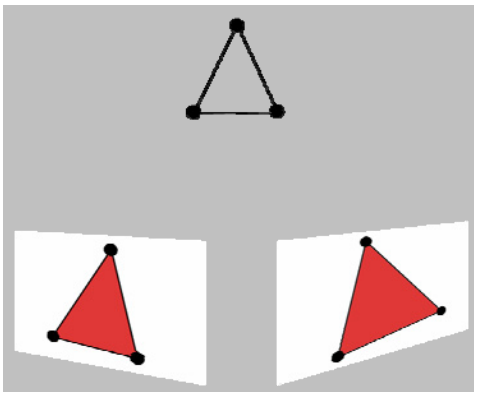


Figure 5.2

In the next frame the geometry and the image data no longer line up. By comparing the pixels corresponding to the triangle in the current and the next frame an error, which indicates how far off a correct deformation is, can be calculated. The pixel error is the cost function of the solver. The gradients and the costs can be plugged into a conjugate gradients solver which will minimize the cost according to the conjugate gradient method.

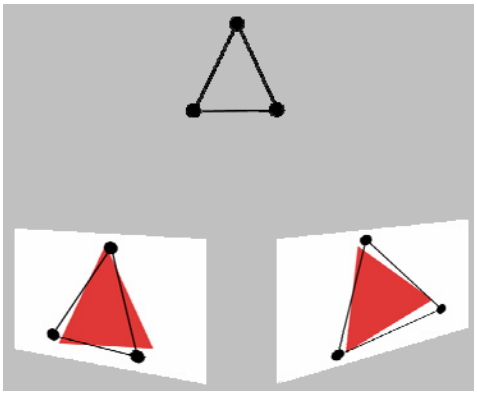


Figure 5.3

The conjugate gradient method is an iterative algorithm which will iterate until the total cost reaches a lower threshold or the algorithm finds a local minimum from which it can't continue. The resulting deformation hopefully results in a, once again, perfect match between the geometry and the image data.

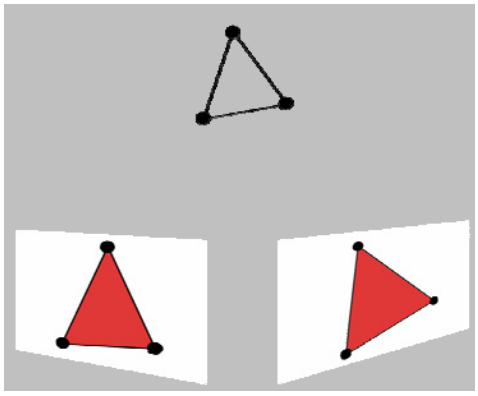


Figure 5.4

5.1.1 The spring extension

In situations where there is very little image information to track or the texture in the image is invariant to scale there is nothing in the Model Flow algorithm that conserves size or shape of the triangles. To make the algorithm more robust in those situations spring forces has been added along the edges of every triangle. While this is no attempt to accurately model any physical system, these spring forces are a simple approximation to the elasticity of many deformable surfaces such as cloth and human skin. The behavior of the springs is controlled by the capturing artist when running the Model Flow command in *TRACK*.

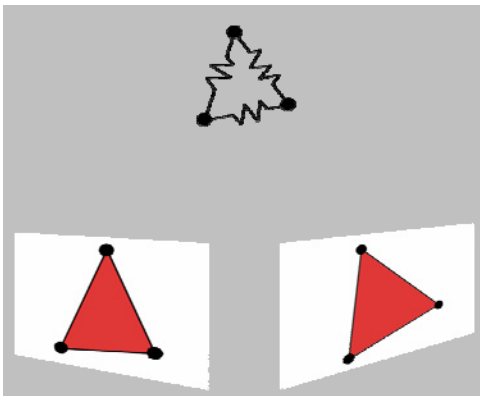


Figure 5.5

5.2 Conjugate Gradient Solver

The conjugate gradient (cg) solver is the work horse of the capturing system described in this thesis. A typical system of equations that is to be fed into a cg solver is usually written as Ax=B but in this thesis it will be written as **Aq**=**b** for consistency and to avoid confusion between the column vector **x** and the x-coordinates. Figure 5.7 will hopefully explain this seemingly childish system of equations even further:

The conjugate gradient method is a iterative algorithm which requires several iterations for optimal results. While model flowing no deformation is assumed for the first iteration and the vertex coordinates from the current frame, $\mathbf{q}_0$, is used. After each iteration, i.e. solution of the **Aq**=**b** system, the deformation is updated according to $\mathbf{q}_{n+1}=\mathbf{q}_n+\mathbf{q}$, where **q** is the latest solution. When the method no longer finds better solutions the iterative process stops and the optimal deformation found so far is used.

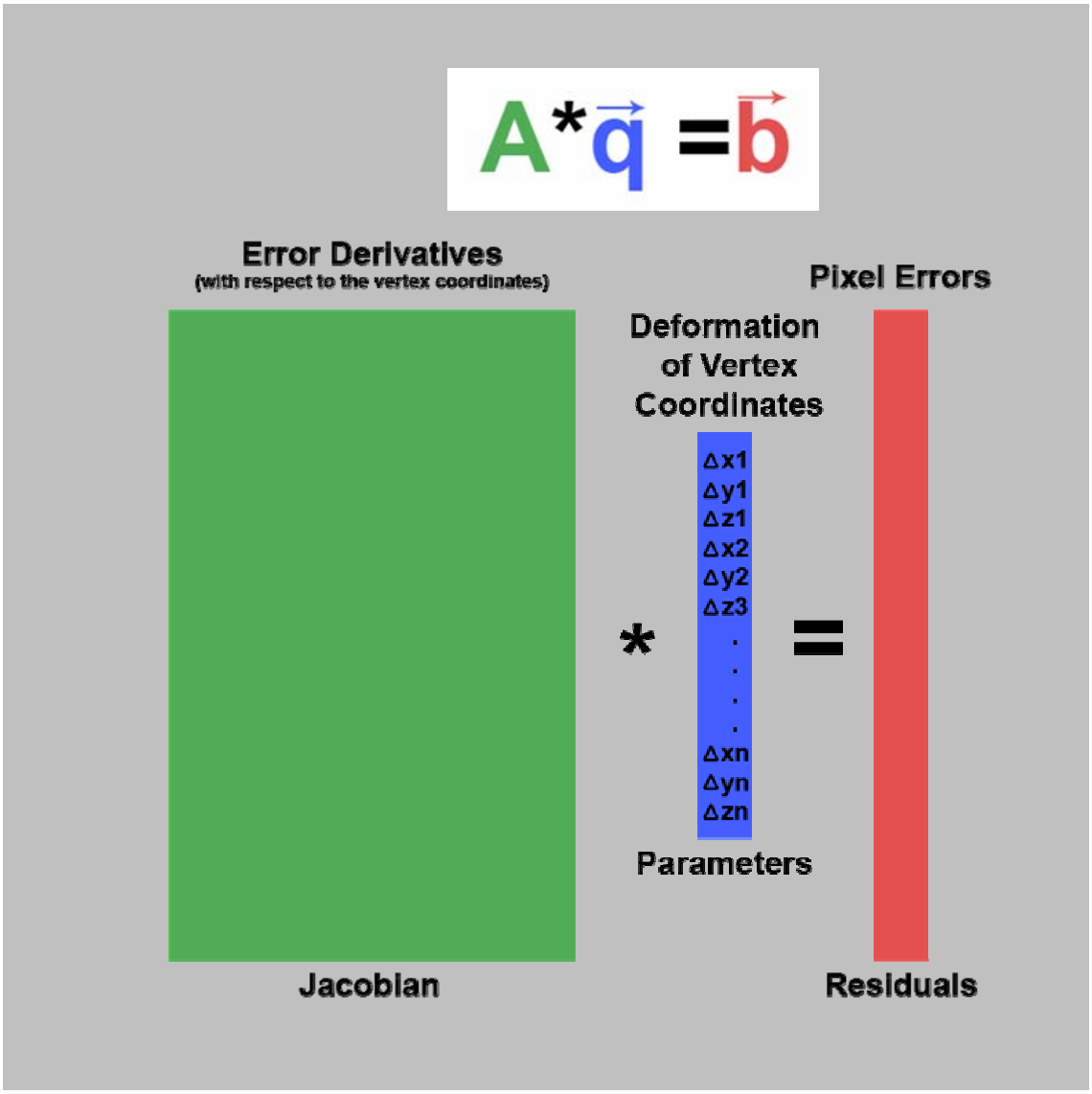


Figure 5.7 *Illustration of the system of equations that is to be minimized.*

Figure 5.7 clearly illustrates that the matrix **A** will be quite huge and far from square. By taking the pseudo inverse of **A** the equation turns into $\mathbf{A}^T\mathbf{A}\mathbf{q}=\mathbf{A}^T\mathbf{b}$ which can rewritten as $\mathbf{A}_2\mathbf{q}=\mathbf{b}_2$ but this time the Jacobian $\mathbf{A}_2$ is guaranteed to be square, diagonal, positive definite and in most cases extremely sparse. Exactly how most cg solvers like it.

The residuals in **b** are easily calculated since they are just the costs from the cost function, i.e. pixel differences. The Jacobian is a different story though, since it consists of gradients of the cost function with respect to the variables which are solved for. There is no way of calculating the analytic derivative of such a complex cost function, but the gradients can be approximated as functions of the image gradients which can easily be calculated. Note that the solution vector **q** is in practice a suggested step towards the correct solution rather than the final solution. This means that **q** is added onto the total deformation iteration after iteration until the cost no longer is getting smaller.

5.2.1 The Cost Function

The complete cost function for a full Model Flow system looks like this:

$$Cost = \sum_{Vertices} \sum_{Triangles} \sum_{Cameras} \sum_{Pixels} \sum_{RGB} CostFunction(u,v)$$

Eq. 5.1

Where u and v are horizontal and vertical pixel coordinates. Luckily the solver will take care of all the summations and in a moment it will be clear that each pixel only depend on nine variables at the most. This means that the important part is the actual cost function inside all the summations:

$$CostFunction(u,v) = \sqrt{\left(I_{next}(\hat{u},\hat{v}) - I_{current}(u,v)\right)^2}$$

Eq. 5.2

This is nothing but a pixel difference. Since only the minimization of this function is interesting the square root can also be ignored:

$$CostFunction(u,v) = \left(I_{next}(\hat{u},\hat{v}) - I_{current}(u,v)\right)^2$$

Eq. 5.3

5.2.2 The Warp

If perfect conditions are assumed, that is no occlusion and that the deforming surface in 3D can be accurately modeled with a triangle mesh, all pixels will belong to a triangle which is fully visible in both the current and the next frame. This means that the deformation of the triangle in 3D equals a 2D warp of the triangle in 2D. That is for every triangle there exists a 3x3 matrix which takes a homogenous screen coordinate in the current frame, $\mathbf{s}_{current}$, and warp it to a homogenous pixel coordinate in the next frame, $\mathbf{s}_{next}$:

$$\vec{s}_{next} = W \cdot \vec{s}_{current}$$

Eq. 5.4

This warp enables a mapping between the pixels in the current and the next frame:

$$\hat{u}(u,v) = \frac{\hat{u}_*(u,v)}{w(u,v)} = \frac{w_{11} \cdot u + w_{12} \cdot v + w_{13}}{w_{31} \cdot u + w_{32} \cdot v + w_{33}}$$
$$\hat{v}(u,v) = \frac{\hat{v}_*(u,v)}{w(u,v)} = \frac{w_{21} \cdot u + w_{22} \cdot v + w_{23}}{w_{31} \cdot u + w_{32} \cdot v + w_{33}}$$

Eq. 5.5

Since the warp is the same for all pixels belonging to the same triangle only one warp matrix per triangle is needed and a complete triangle can be warped with one matrix multiplication:

$$S_{next} = W \cdot S_{current}$$

Eq. 5.6

Using this warp it is easy to compare pixels in two different triangles regardless of the deformation in 3D. Further exploration of this equation gives:

$$\begin{aligned} S_{next} \cdot [S_{current}]^{-1} &= W \cdot S_{current} \cdot [S_{current}]^{-1} \\ S_{next} \cdot [S_{current}]^{-1} &= W \\ W &= S_{next} \cdot [S_{current}]^{-1} \\ W &= [P_{next} \cdot V_{next}] \cdot [S_{current}]^{-1} \end{aligned}$$

Eq. 5.7

The projection transformations presented in section 2.2.1, and especially equation 2.3, enables the introduction of some new and very important matrices. $\mathbf{P}_{next}$ is the known 3x4 perspective projection matrix from the camera and $\mathbf{V}_{next}$ is the 4x3 matrix describing the coordinates of the vertices in space. This last equation is very important! This warp describes the mapping between the current and the next frame as a function of the vertex coordinates of the triangle in the next frame. This means that the pixel mapping equation can be rewritten for the case where the pixels in the current frame are fixed and the vertex positions in the next frames are the variables:

$$\begin{aligned} \hat{u}(V_{next}) &= \frac{\hat{u}_*(V_{next})}{w(V_{next})} \\ \hat{v}(V_{next}) &= \frac{\hat{v}_*(V_{next})}{w(V_{next})} \end{aligned}$$

Eq. 5.8

5.2.3 The Jacobian

The Jacobian is a carefully built matrix containing the gradients of the cost function with respect to the parameters of the equation system. In this case the derivatives with respect to the x-, y-, and z-coordinate of the vertices need to be approximated. Deriving the cost function in equation 5.3 with respect to the parameters **q** leads to:

$$\frac{\partial}{\partial \vec{q}}\left(CostFunction(u,v)\right) = \frac{\partial}{\partial \vec{q}}\left(I_{next}(\hat{u},\hat{v}) - I_{current}(u,v)\right)^2 =$$

$$= 2 \cdot \left(I_{next}(\hat{u},\hat{v}) - I_{current}(u,v)\right) \cdot \frac{\partial}{\partial \vec{q}} \cdot \left(I_{next}(\hat{u},\hat{v}) - I_{current}(u,v)\right) \approx$$

$$\approx \left(I_{next}(\hat{u},\hat{v}) - I_{current}(u,v)\right) \cdot \frac{\partial}{\partial \vec{q}} \cdot \left(I_{next}(\hat{u},\hat{v})\right)$$

Eq. 5.9

The constant "2" can be dropped for simplicity in the last step since it is just a constant and since $I_{current}(u,v)$ is not a function of the parameters describing the geometry in the next frame that part equals zero. Now apply the chain-rule:

$$\left(I_{next}(\hat{u},\hat{v}) - I_{current}(u,v)\right) \cdot \left[\frac{\partial I_{next}}{\partial \hat{u}} \cdot \frac{\partial \hat{u}}{\partial \vec{q}} + \frac{\partial I_{next}}{\partial \hat{v}} \cdot \frac{\partial \hat{v}}{\partial \vec{q}}\right] =$$

$$= diff \cdot \begin{bmatrix} \frac{\partial \hat{u}}{\partial \vec{q}} & \frac{\partial \hat{v}}{\partial \vec{q}} \end{bmatrix} \cdot \begin{bmatrix} \dot{I}^{u}_{next}(\hat{u},\hat{v}) \\ \\ \dot{I}^{v}_{next}(\hat{u},\hat{v}) \end{bmatrix}$$

Eq. 5.10

In the last expression "diff" is the scalar difference between two pixels and the column vector on the far right consist of the image gradient in the next frame in u and v direction. Both the pixel difference and the image gradients are easy to calculate when the interesting pixels are identified. The row vector in the middle is still in need of more work. Expanding it with equation 5.8 gives:

$$\begin{bmatrix} \frac{\partial \hat{u}}{\partial \vec{q}} & \frac{\partial \hat{v}}{\partial \vec{q}} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial \vec{q}}\hat{u}(V_{next}) & \frac{\partial}{\partial \vec{q}}\hat{v}(V_{next}) \end{bmatrix} =$$

$$= \begin{bmatrix} \dfrac{\frac{\partial \hat{u}_*(V_{next})}{\partial \vec{q}} \cdot w(V_{next}) - \hat{u}_*(V_{next}) \cdot \frac{\partial w(V_{next})}{\partial \vec{q}}}{\left[w(V_{next})\right]^2} & \dfrac{\frac{\partial \hat{v}_*(V_{next})}{\partial \vec{q}} \cdot w(V_{next}) - \hat{v}_*(V_{next}) \cdot \frac{\partial w(V_{next})}{\partial \vec{q}}}{\left[w(V_{next})\right]^2} \end{bmatrix}$$

Eq. 5.11

Where (from eq. 2.5)

$$\begin{bmatrix} \hat{u}_* \\ \hat{v}_* \\ w \end{bmatrix} = \begin{bmatrix} Warp \\ Matrix \\ W(V_{next}) \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Eq. 5.12

Which means

$$\begin{bmatrix} \frac{\partial \hat{u}_*(V_{next})}{\partial \vec{q}} \\ \frac{\partial \hat{v}_*(V_{next})}{\partial \vec{q}} \\ \frac{\partial w(V_{next})}{\partial \vec{q}} \end{bmatrix} = \begin{bmatrix} \frac{\partial W(V_{next})}{\partial \vec{q}} \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Eq. 5.13

As declared in equation 5.7 the warp **W** is created by multiplying $\mathbf{P}_{next}$, $\mathbf{V}_{next}$ and $\mathbf{S}_{current}$. $\mathbf{P}_{next}$ is projection matrix at the next frame, which is known and will stay constant. $\mathbf{S}_{current}$ is the screen coordinates of the triangle in the current frame, which are also known and constant. At this point it is clear that building the Jacobian is not trivial but let's discuss what we have so far and what these letters really mean. The cost function is derived with respect to the column vector of parameters **q**. In most cases there will be thousands or at least hundreds of parameters in **q** but luckily the cost is calculated per pixel and every pixel will be affected by (at the most) 9 parameters. This is because every pixel belongs to a triangle which in turn is defined by three vertices. The parameters belonging to those three vertices are the only nine parameters that can affect the triangle and in turn the pixels it corresponds to. 3 vertices times 3 dimensions (x, y, and z) equal 9 parameters. Not surprisingly it is the same nine x-, y-, and z-coordinates in the matrix that define the triangle in space, $\mathbf{V}_{next}$.

In practice this means that there will be nine 3x3 matrices with warp gradients for every triangle and camera. Those can be pre-computed and the values looked up and multiplied with pixel gradients and pixel differences when building the Jacobian. The nine gradient-matrices for each triangle are computed thru deriving equation 5.7 with respect to the nine parameters of the triangle.

$$\frac{\partial W}{\partial \vec{q}} = \frac{\partial}{\partial \vec{q}} \left[\left[P_{next} \cdot V_{next} \right] \cdot \left[S_{current} \right]^{-1} \right] =$$
$$= \frac{\partial}{\partial \vec{q}} \left[\left[P_{next} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \\ 1 & 1 & 1 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} \right]$$

Eq. 5.14

This equation (5.14) equals the following nine matrices of gradients:

$$\frac{\partial W}{\partial x_1} = \left[P_{next} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} p_{11} & 0 & 0 \\ p_{21} & 0 & 0 \\ p_{31} & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{11}s_{11} & p_{11}s_{12} & p_{11}s_{13} \\ p_{21}s_{11} & p_{21}s_{12} & p_{21}s_{13} \\ p_{31}s_{11} & p_{31}s_{12} & p_{31}s_{13} \end{bmatrix}$$

$$\frac{\partial W}{\partial y_1} = \left[P_{next} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} p_{12} & 0 & 0 \\ p_{22} & 0 & 0 \\ p_{32} & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{12}s_{11} & p_{12}s_{12} & p_{12}s_{13} \\ p_{22}s_{11} & p_{22}s_{12} & p_{22}s_{13} \\ p_{32}s_{11} & p_{32}s_{12} & p_{32}s_{13} \end{bmatrix}$$

$$\frac{\partial W}{\partial z_1} = \left[P_{next} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} p_{13} & 0 & 0 \\ p_{23} & 0 & 0 \\ p_{33} & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{13}s_{11} & p_{13}s_{12} & p_{13}s_{13} \\ p_{23}s_{11} & p_{23}s_{12} & p_{23}s_{13} \\ p_{33}s_{11} & p_{33}s_{12} & p_{33}s_{13} \end{bmatrix}$$

$$\frac{\partial W}{\partial x_2} = \left[P_{next} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} 0 & p_{11} & 0 \\ 0 & p_{21} & 0 \\ 0 & p_{31} & 0 \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{11}s_{21} & p_{11}s_{22} & p_{11}s_{23} \\ p_{21}s_{21} & p_{21}s_{22} & p_{21}s_{23} \\ p_{31}s_{21} & p_{31}s_{22} & p_{31}s_{23} \end{bmatrix}$$

$$\frac{\partial W}{\partial y_2} = \left[P_{next} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} 0 & p_{12} & 0 \\ 0 & p_{22} & 0 \\ 0 & p_{32} & 0 \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{12}s_{21} & p_{12}s_{22} & p_{12}s_{23} \\ p_{22}s_{21} & p_{22}s_{22} & p_{22}s_{23} \\ p_{32}s_{21} & p_{32}s_{22} & p_{32}s_{23} \end{bmatrix}$$

$$\frac{\partial W}{\partial z_2} = \left[P_{next} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} 0 & p_{13} & 0 \\ 0 & p_{23} & 0 \\ 0 & p_{33} & 0 \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{13}s_{21} & p_{13}s_{22} & p_{13}s_{23} \\ p_{23}s_{21} & p_{23}s_{22} & p_{23}s_{23} \\ p_{33}s_{21} & p_{33}s_{22} & p_{33}s_{23} \end{bmatrix}$$

$$\frac{\partial W}{\partial x_3} = \left[P_{next} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} 0 & 0 & p_{11} \\ 0 & 0 & p_{21} \\ 0 & 0 & p_{31} \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{11}s_{31} & p_{11}s_{32} & p_{11}s_{33} \\ p_{21}s_{31} & p_{21}s_{32} & p_{21}s_{33} \\ p_{31}s_{31} & p_{31}s_{32} & p_{31}s_{33} \end{bmatrix}$$

$$\frac{\partial W}{\partial y_3} = \left[P_{next} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} 0 & 0 & p_{12} \\ 0 & 0 & p_{22} \\ 0 & 0 & p_{32} \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{12}s_{31} & p_{12}s_{32} & p_{12}s_{33} \\ p_{22}s_{31} & p_{22}s_{32} & p_{22}s_{33} \\ p_{32}s_{31} & p_{32}s_{32} & p_{32}s_{33} \end{bmatrix}$$

$$\frac{\partial W}{\partial z_3} = \left[P_{next} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \right] \cdot \left[S_{current} \right]^{-1} = \begin{bmatrix} 0 & 0 & p_{13} \\ 0 & 0 & p_{23} \\ 0 & 0 & p_{33} \end{bmatrix} \cdot \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix} = \begin{bmatrix} p_{13}s_{31} & p_{13}s_{32} & p_{13}s_{33} \\ p_{23}s_{31} & p_{23}s_{32} & p_{23}s_{33} \\ p_{33}s_{31} & p_{33}s_{32} & p_{33}s_{33} \end{bmatrix}$$

Eq. 5.15

It is important to remember that both the projection matrices and the current screen coordinates of the vertices will stay fixed and are well known during the iterative minimization process. This means that the gradient matrices can be calculated once for every triangles and camera and used over and over again for all the pixels in that triangle iteration after iteration.

For every residual there will be nine (or less) entries in the Jacobian. The Jacobian must have one column per parameter, which means that every group of three columns represents one vertex. It also must have one row for every pixel comparison. If all three color channels are used this means three rows per pixel. In every row all values but nine must be zero! When building the Jacobian the following pseudo code is used:

```
For every triangle:
 For every camera:
  Get the pre-computed gradient matrices (Eq. 5.15)
  For every pixel:
   Get the pre-computed image gradients
   For every vertex that belong to the triangle (3)
    For every dimension (3/xyz)
     For every color channel (typically 3/rgb)
      Calculate the pixel difference (the cost)
      Calculate the nine entries in the Jacobian
      according to eq. 5.10
```

5.2.4 The spring extension

When a cg solver is set up correctly it is quite easy to extend it with other cost functions that also affects the solution in one way or the other. A cost function does not necessary capture all kinds of behaviors in a complex system like this one, or it might have weaknesses that can be corrected by other costs.

The cost function described above is not perfect and it has a couple of apparent weak spots. One of them is that it assumes that there is pixel information to compare at all time and that the pixel information is so rich it will always give correct costs when minimizing. In situations where large areas have hardly any color variation or the color patterns are fairly scale invariant the pixel comparing cost function will run into trouble. Since it doesn't penalize unrealistic size or shape changes of triangles weird things can happen in regions where the pixel tracking assumptions fail.

The system described in this thesis incorporates an extension that will mimic the elasticity and shape conservation properties that most deformable surface show. By adding costs to the solver corresponding to theoretical spring forces along the edges of the triangles in the polygon mesh area and shape preservation is encouraged. While this extension is no attempt to model any physical system the spring forces greatly improves the robustness of the system.

The springs are non-linear cubic springs and their exact behavior can be controlled by the capturing artist while capturing. The spring forces which are used as costs are described by this equation presented in chapter 2:

$$F = -\frac{k \cdot (\Delta d)^3}{\mathrm{T}^2}$$

Eq. 5.16

Where T is a threshold value for the ratio between delta d (change of length) and the original length of the spring. I.e. if T is 0.10 the threshold is 10% of the original length of the spring. Below the threshold the spring will be softer than the linear case. Above the threshold the spring gets stiffer fast since the function is a cubic function. Such a spring will allow small fluctuations but will punish large changes. Exactly the kind of behavior which is sought after in a system which does not hinder expected deformations, penalizes extreme distortions, and prohibits unrealistic solutions. For the cg solver the spring extension make the following changes to the system of equations:

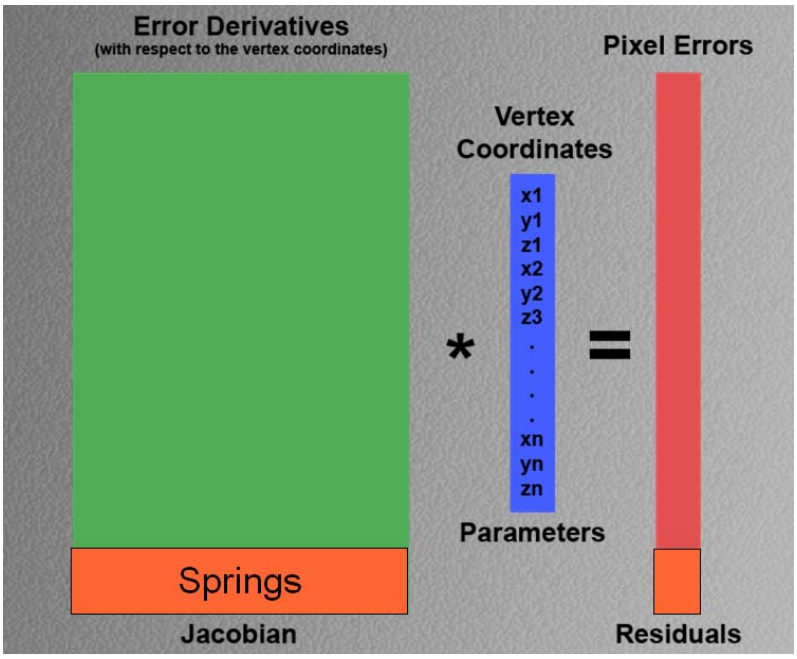


Figure 5.8 *Illustration of the system of equations, including spring extension.*

In the Jacobian the spring costs are derived with respect to the parameters representing the vertices in each end of the spring. This means that every extra row in the Jacobian represent one spring and in one of those rows all but six entries must equal zero. The spring contributions to the system of equations are much easier and faster to calculate since they are calculated for every edge and not for every pixel. A simple example for the spring forces is a spring stretching from (0,0,0) to (5,5,0) in it's original position. If the spring is extended by the square root of two it go from (0,0,0) to (6,6,0) delta d will be:

$$\Delta d = \sqrt{(6-0)^2 + (6-0)^2 + (0-0)^2} - \sqrt{(5-0)^2 + (5-0)^2 + (0-0)^2} =$$
$$= \sqrt{72} - \sqrt{50} = \sqrt{2}$$

Eq. 5.17

And the spring cost will be:

$$F = -\frac{k \cdot (\Delta d)^3}{\mathrm{T}^2} = -\frac{k \cdot (\sqrt{2})^3}{\mathrm{T}^2} = -\frac{k}{\mathrm{T}^2} 2\sqrt{2}$$

Eq. 5.18

Which is a scalar value that will be added to the bottom of the **b** vector. The derivative of this cost, with respect to for example the x-coordinate, is:

$$\frac{\partial F}{\partial x} = -\frac{k}{\mathrm{T}^2} \cdot \frac{\partial}{\partial x}\left((\Delta d)^3\right) = -\frac{k}{\mathrm{T}^2} \cdot \left(3 \cdot (\Delta d)^2 \cdot \frac{\partial}{\partial x} \Delta d\right) =$$
$$= -\frac{k}{\mathrm{T}^2} \cdot \left(3 \cdot (\Delta d)^2 \cdot \frac{\partial}{\partial x}\left(\sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} - \sqrt{(\Delta x_0)^2 + (\Delta y_0)^2 + (\Delta z_0)^2}\right)\right) =$$
$$= -\frac{3k}{2\mathrm{T}^2} \cdot \left((\Delta d)^2 \cdot \left(\frac{\frac{\partial}{\partial x}\left((x-0)^2\right)}{\sqrt{(x-0)^2 + (y-0)^2 + (z-0)^2}}\right)\right) = -\frac{3k}{2\mathrm{T}^2} \cdot \left((\Delta d)^2 \cdot \left(\frac{2x}{\sqrt{x^2 + y^2 + z^2}}\right)\right)$$

Eq. 5.19

Which, if evaluated at (6,6,0), equals:

$$\left.\frac{\partial F}{\partial x}\right|_{x=6, y=6, z=0} = -\frac{3k}{2\mathrm{T}^2} \cdot \left(\left(\sqrt{2}\right)^2 \cdot \left(\frac{12}{\sqrt{36+36}}\right)\right) = -\frac{k}{\mathrm{T}^2} \cdot \left(\cdot \left(\frac{36}{6\sqrt{2}}\right)\right) = -\frac{k}{\mathrm{T}^2} 3\sqrt{2}$$

Eq. 5.20

If k and T are positive the cost is negative and so is its derivative. This makes perfect sense since an increase of x would indeed make the spring force stronger in a negative direction. Since the solver wants its costs to equal zero this information will affect the solver to decrease the value of x. Thus bringing the spring closer to its equilibrium state.

6. IMPLEMENTATION

The implementation of this system was all done in C++. Most of the code written specifically for this project was written in Microsoft Visual Studio 2003 but the system runs in both Windows and Linux.

6.1 TRACK

TRACK is the name of the academy award winning propriety computer vision suite used at Digital Domain. It is a top of the line 3D tracker which has support for both automated and manual tracking as well as extra features such as optical flow based tracking and processing of laser scan data. In ***TRACK*** there is excellent support for data input/output and it has a flexible and powerful user interface which was used and extended to suit this project. The system described in this thesis was implemented as an integrated part of ***TRACK*** 5.

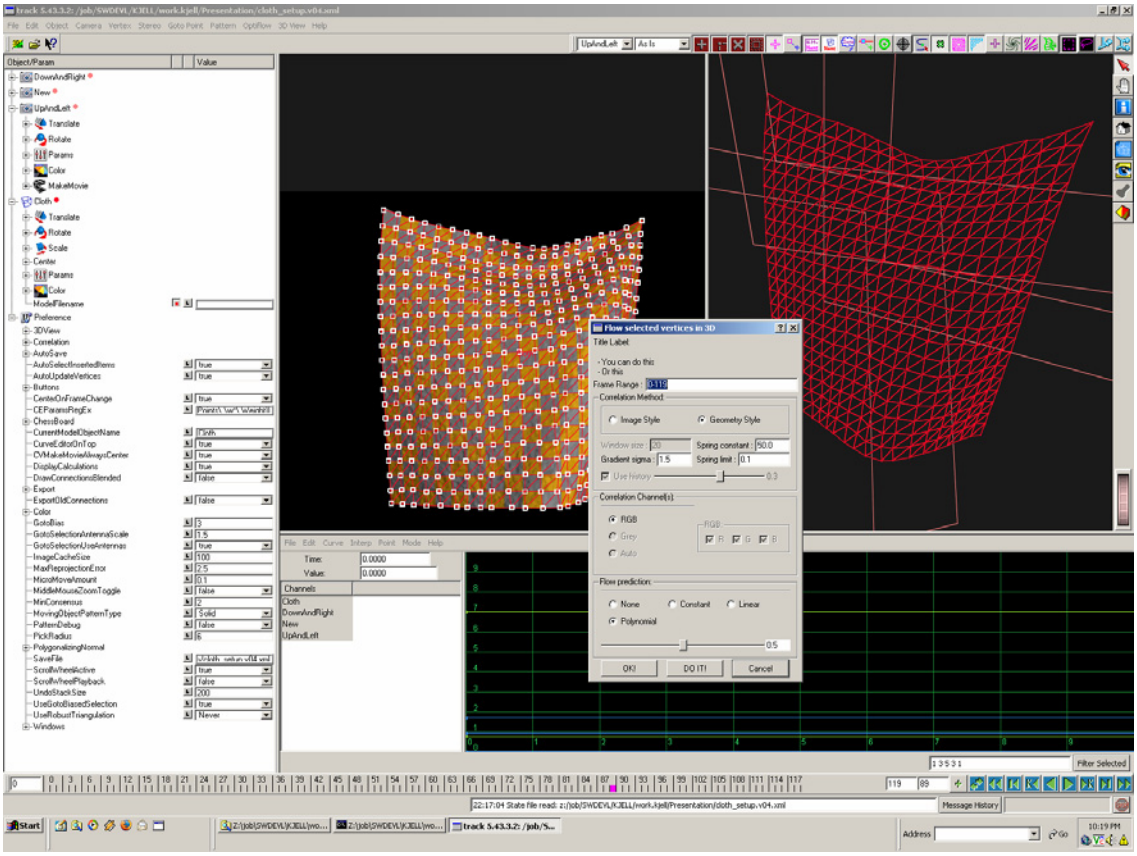


Figure 6.1 *The flexible and powerful user interface of track. On the very left the parameter panels shows an editable tree-structure of all parameters of the current project. Along the bottom there is a time line with controllers. On top of the time line you can see the curve editor in which any animated parameter can be plotted and edited. In the top right corner the 3D-panel is displayed and on its left the input image sequence is shown with tracking data overlaid. The dialog box is the user interface for tracking deforming geometry with Model Flowing in* ***TRACK****.*

One of the key ideas behind 3D tracking software is the connection between 2D image data and 3D geometry, position, and orientation. The link between 2D and 3D is the camera model; therefore the camera is fundamental in ***TRACK***. All image data in ***TRACK*** belong to a camera. Another key component in ***TRACK*** is the model object which holds 3D geometry data. For the purpose of Model Flowing a new model object was implemented, a deforming model object, which has independently animatable vertices. Not only is the geometry globally animatable but every vertex can be animated individually.

For normal tracking assignments the tracker usually has numerous pieces of geometry which is to be lined up with one image sequence associated with one camera. When performing Model Flowing the situation is the opposite, only one piece of geometry is deforming but it is captured with multiple cameras. This requires a slightly different link between cameras and geometry. This link is created when the model-flowing-command is executed.

6.2 Data Structure

Naturally a multi-camera capturing system generates lots of data and during the Model Flowing large amounts of data needs to be compared and modified. In ***TRACK*** all image data is owned by a camera and all 3D geometry data is owned by a model object. The 3D geometry is basically a list of coordinates in space, the vertices, and a list of triangles defined by three indices in the list of vertices. For every vertex that is "flowing" some new data need to created and managed. First of all a list of new coordinates needs to be created. These are the variables that will be solved for. Secondly all the data that will be used to calculate those new coordinates needs to be gathered. Since the algorithm is iterative much of the data will be stored in two versions, one for the current iteration and one for the last iteration.

6.2.1 Vertex

The vertex both keeps records of the error associated with the triangles surrounding it and the estimated derivative of that error with respect to its x-, y-, and z-coordinates. A deforming vertex also needs to keep track of its 3D coordinates in the last frame and two sets of coordinates in the present frame, one for the current iteration and one for the last iteration.

6.2.2 Triangle

Every vertex keeps record of which triangles it belongs to and every triangle knows its vertices. The triangles also keep track of its projections out into the surrounding cameras. This means that a triangle stores a series of index patches and pixel patches associated with the cameras. For every camera a triangle stores one reference index patch from the last frame, one pixel patch from the last frame, one warped pixel patch according to the vertex positions in the last iteration, and pixel patches with image derivatives in u- and v-direction. Since

the warps used for pixel comparison is defined per triangle the triangle also keep track of the warp and gradient matrices associated with the cameras.

6.2.3 Index Patch

The index patch stores the pixel coordinates for a patch of pixels corresponding to a triangle projected onto an image plane and then scan-line rasterized. Note that the patch is not storing the RGB-values but stores which pixels in an image belong to a triangle.

6.2.4 Pixel Patch

The pixel patch is the lowest level of data structure storing actual pixel values. It is basically a 2D float container storing RGB-values corresponding to an index patch.

6.3 Warping and Unwarping

The Model Flow algorithm assumes that there is a perfect match between the 3D geometry and the image data from the cameras in the frame prior to the one it is solving for. That is it knows it has a perfect match to start with and it wants to solve for the deformation that will give a perfect match also in the following frame. By projecting the geometry out onto the camera planes and scan line rasterize every triangle into index patches each triangle knows what pixels belong to it. With the projection matrices from the cameras the associated 2D warps that will correspond to the pixel deformation when the geometry is deforming can be calculated. Of this data most of it is not depending on the new vertex coordinates which means it will be constant during the iteration process and can therefore be precomputed and stored.

Before the iterations start the actual pixel values for a triangle is gathered according to the index patch and written to a pixel patch. In every iteration 2D warp matrices are built that maps pixels from the deformed triangle back to the already tracked triangle. A per pixel comparison can now be made between the pixels in the pixel patches and the corresponding pixel values gathered from the next frame using the 2D warp matrices. The difference is stored as the cost at vertex level. Using the same warp and pixel derivatives the gradient of the error can be estimated and stored as well.

6.4 CG Solver

Systems that is to be solved by the conjugate gradient method is often written as $\mathbf{Ax}=\mathbf{b}$ where $\mathbf{x}$ is a column vector with the variables that are solved for and $\mathbf{b}$ is a column vector with the cost for every residual. $\mathbf{A}$ is a big matrix as wide as the number of variables to solve for and as tall as the number of residuals in the system. Since A is not square but otherwise very well behaved the pseudo inverse is calculated which will give us $\mathbf{A}^T\mathbf{Ax}=\mathbf{A}^T\mathbf{b}$. $\mathbf{A}^T\mathbf{A}$ is square, symmetric, positive definite, and very sparse. Implementationwise $\mathbf{A}$ or $\mathbf{b}$ will never be

stored. $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$ is calculated right away and given to a conjugate gradient solver. In the figure below is a visualization of what $\mathbf{A}^T\mathbf{A}$ looks like for the cloth data set that will be introduced in chapter 7.

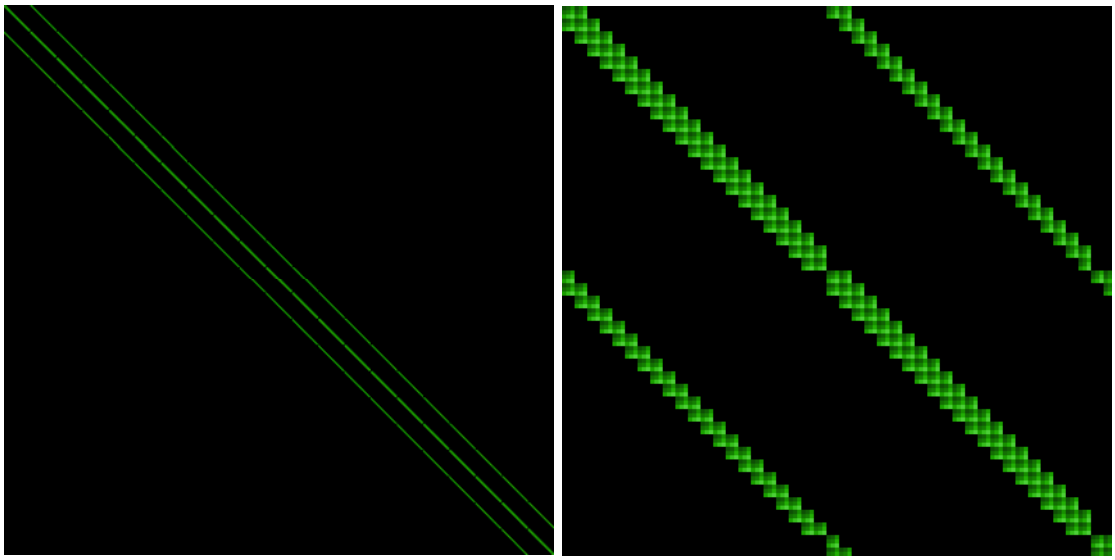


Figure 6.2 *Visualization of an $\boldsymbol{A}^T\boldsymbol{A}$ matrix. On the left the complete matrix where black pixels represent zeros and green pixels represent real data is illustrated. The matrix is clearly both diagonal and extremely sparse. On the right is a close up of the very left top corner of the matrix. In $\boldsymbol{A}^T\boldsymbol{A}$ every group of three columns represents one vertex. Every group of three rows represents vertices that are connected to each other. The "gaps" in the diagonal is caused by the fact that the vertices along the edges are not connected to as many vertices as the vertices in the middle of the triangle mesh.*

6.5 SparseLib

"SparseLib++ is a C++ class library for efficient sparse matrix computations across various computational platforms"[21]. SparseLib is an excellent library which supports a range of different storage formats and basic matrix operators. It also incorporates some very fast iterative solvers and fitting preconditioners for linear systems.

When all per triangle setup is done for an iteration the costs and the gradients are collected and written to compact SparseLib-matrices. SparseLib then solves the equation system in matter of milliseconds and a new estimated deformation is given as the result. If the new estimate renders a better result the iterative process goes on, if not the deformation is rolled back to best known deformation and the system takes on the next frame.

6.6 Pseudo Code

At the core of the Model Flow algorithm there is a main loop that takes two sets of images and a piece of geometry which lines up with the first set, and returns a deformation for the geometry which makes it line up with the second set as well. This main loop looks like this in pseudo code:

```
// Assume no deformation is needed
Do warping;
Calculate error;
If (error < threshold)   done = true;

While (done!=true)
{
 Old error = error;
Remember vertex positions;
 Build matrices for solver;
 Solve equation system;
 Update vertex positions (p = p + delta_p);
 Do warping;
 Calculate error;
 If (error < threshold)   done = true;
 Else if (error > old error)
{
      restore old vertex position;
      done = true;
 }
}
```

7. RESULTS

In this section some early test results will be presented and discussed. As Model Flowing has not yet been used in production and the implementation described in this thesis still has some basic functionality missing. It is reasonable to believe that Model Flowing can perform better than this. Since both inputs and results are animated it is extremely hard to illustrate their characteristics in print. It is to some degree up to the reader to "fill in he blanks" and "envision" the animations.

7.1 CG Cloth Data Set

In this data set an animated piece of synthetic cloth was rendered from 2 camera views in a piece of software called ***Maya*** from ***Alias|Wavefront***. The reason for using computer generated imagery in this data set was to avoid problems related to image noise, camera calibration and other factors which are not part of this research project. Computer generated imagery is also cheaper to produce in terms of man power and technical resources. The data set is 120 frames long and the resolution of each image is 480 by 640 pixels. The physical setup (in the virtual world of ***Maya***) looks like this:

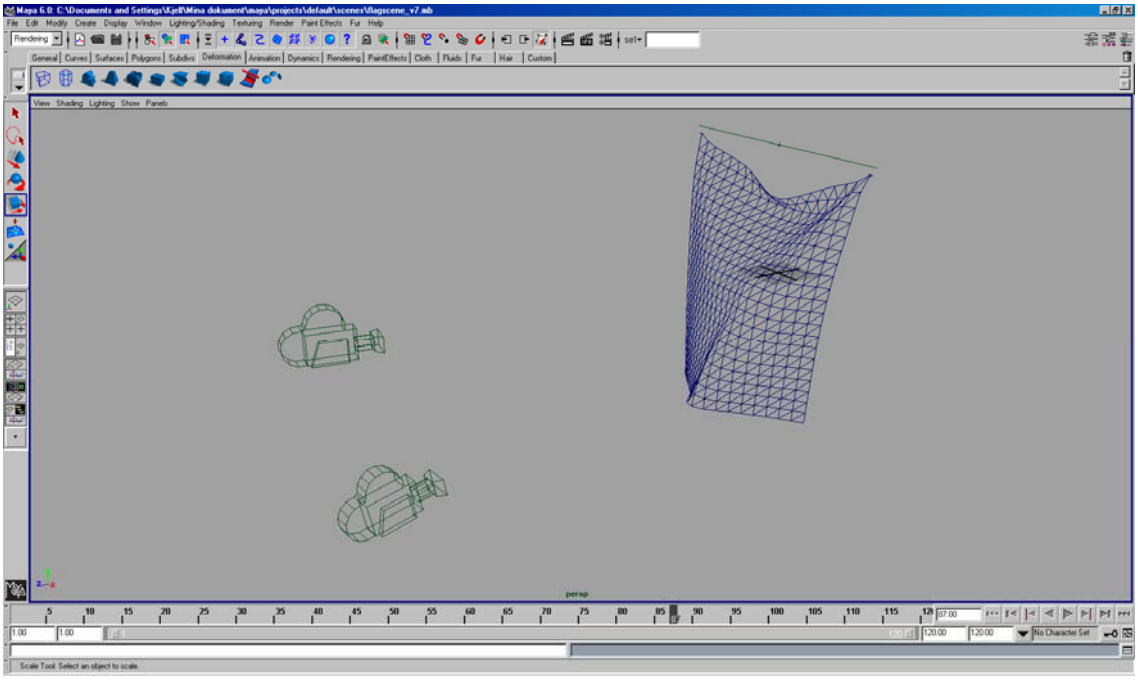


Figure 7.1 *Illustration of the physical setup in* ***Maya****. The geometry describing the cloth can be seen on the right and the two cameras can be seen on the left.*

When rendered the triangle mesh was textured and given material properties to make it resemble some kind of cloth-like material. This is what a pair of captured images look like:

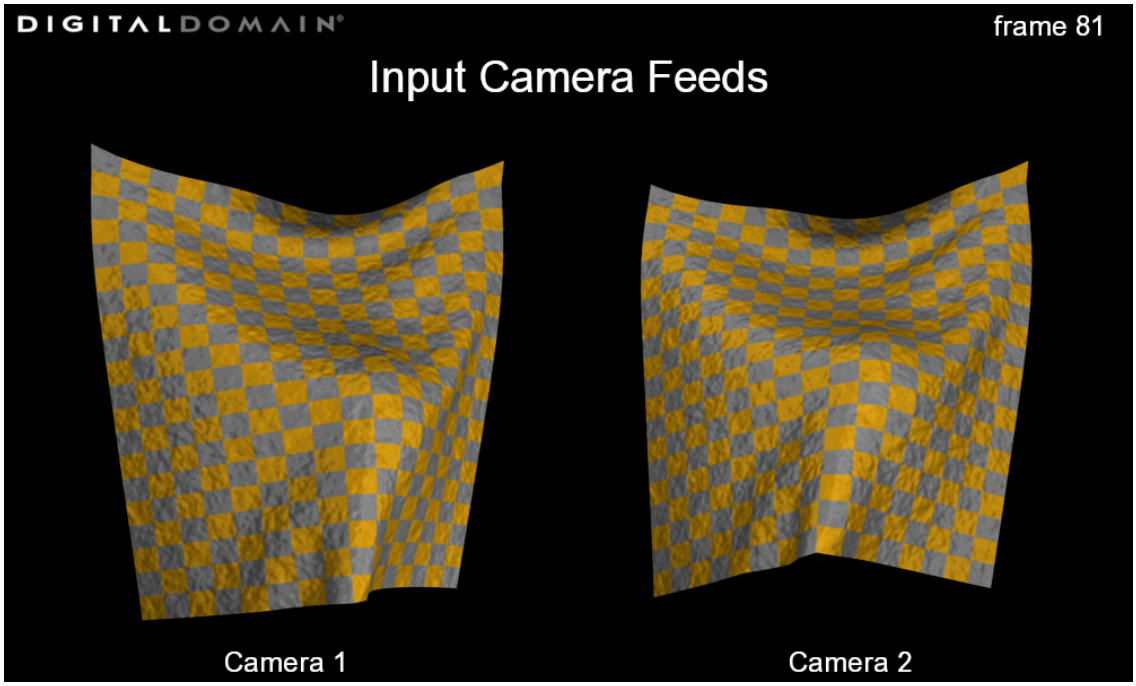


Figure 7.2 *Example of a pair of input images from the cloth data set. Clearly the base line is not very wide.*

On the cloth data set both the universal capturing approach [3] and Model Flowing was used for comparison. Figure 7.3 picture a frame from a camera view very close to the ones used for capturing. The geometry produced by the capturing algorithms are overlaid the original image. If the deformation has been captured perfectly the cloth and the polygon mesh should line up perfectly. The checker board pattern on the cloth has nothing to do with the polygon model though.

Figure 7.4 to 7.6 show some renderings of the final captured deforming geometry from a more aggressive angle than figure 7.3. This view point is far from where the cameras captured the input data and will show more of the accumulated noise and error in depth estimation. Once again the Universal Capturing has been used on the left and Model Flowing has been used on the right.

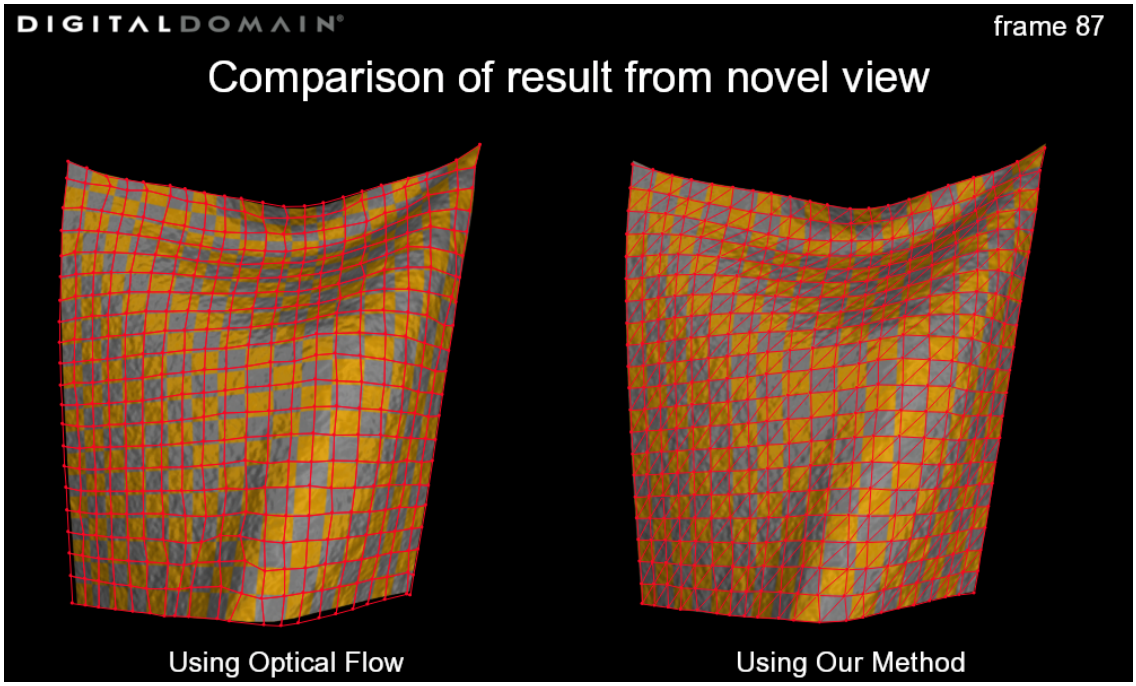


Figure 7.3 *Two illustrations of how the resulting mesh animation fits the original cloth movements. Universal capturing on the left and Model Flowing on the right. Model Flowing renders better results than the optical flow based capturing method. This is particularly noticeable along the edges of the mesh.*

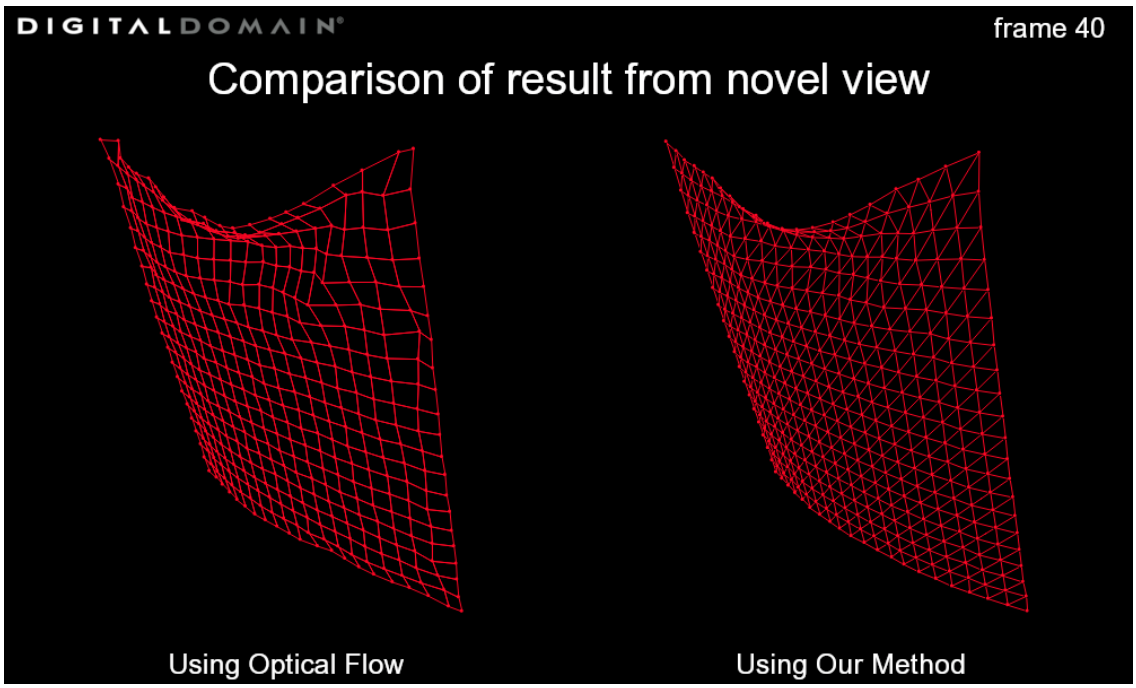


Figure 7.4

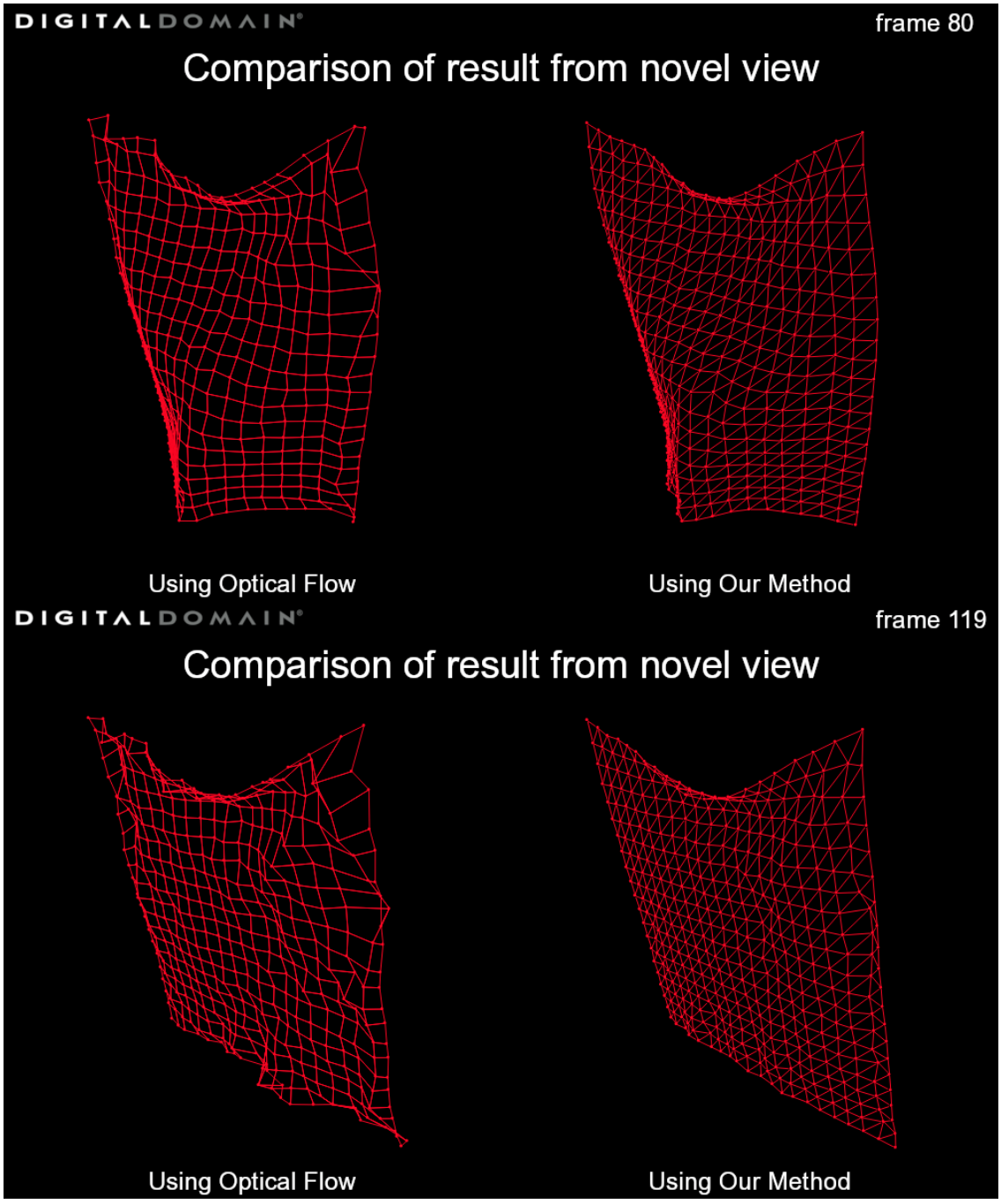


Figure 7.4(previous page), 7.5(top), and 7.6(bottom) *Wire frame renderings of the final captured animation. A perfect result should show a smooth and cloth-like polygon mesh. These three frames clearly show how error accumulates over time and that Model Flowing generates significantly less errors than the optical flow based algorithm.*

7.2 Regina Face Data Set

Since capturing of deforming geometry mainly is used for facial capturing a crude facial capturing rig was built and a short test sequence was shot at the Digital Domain stages. The sequence is 120 frames long and was captured with three off the shelf standard definition DV-camcorders. This is what the physical setup looked like in the real world and after it had been recreated in ***TRACK***:

Figure 7.7 *Physical setup for the Regina data set. Notice the tracking cubed placed where the head of the actress will be located. The tracking cube was used to track the cameras with respect to each other and the scene.*

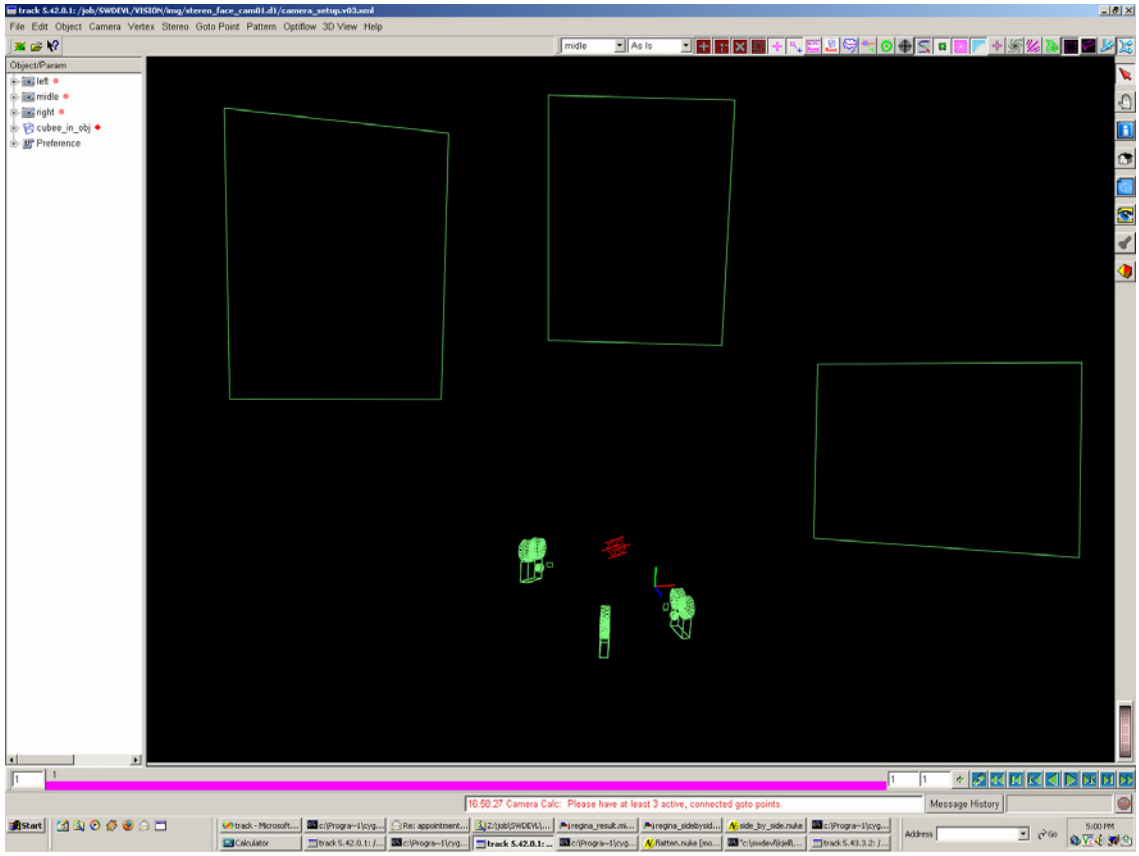


Figure 7.8 *Resulting initial setup in* ***TRACK*** *with virtual cameras and tracking cube placed according to the real world setup.*

Here are some frames from the captured material:

Figure 7.9(top) and 7.10(bottom) *Examples of input data in the Regina data set.*

Unfortunately there was neither time nor money to create a high quality geometric model of Regina's face. Luckily ***TRACK*** has built in features for triangulating simple polygon meshes by hand. This method is both slow and inaccurate but it resulted in some kind of geometry that could be used for a simple test. Figure 7.11 illustrates how the geometry was produced.

The course geometric model was lined up with the first frame in the data set and used as the initial geometry when the Model Flow process was kicked off. For this data set only Model Flowing was evaluated and figure 7.12 to 7.14 show the resulting deforming geometry rendered on top of the input data. Even though the course geometric model obviously breaks the assumption that an accurate static 3D representation of the geometry exists the triangle mesh deforms fairly well. Surprisingly the algorithm manage alright even in areas of heavy deformation around Regina's mouth and eyes.

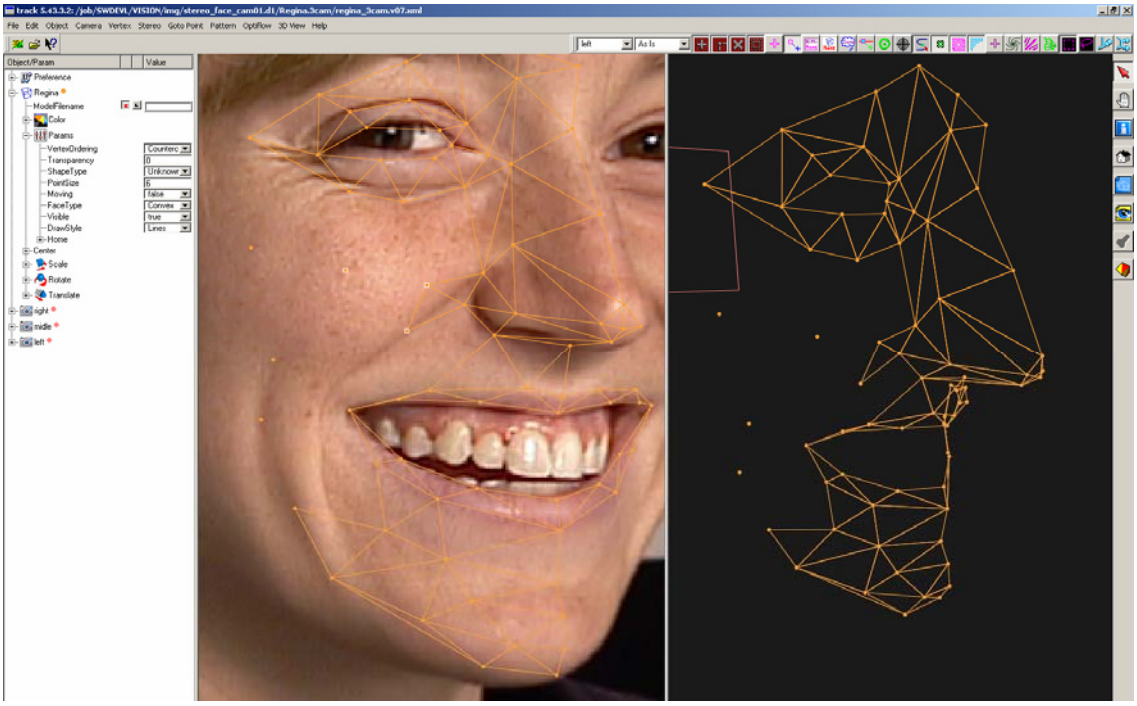


Figure 7.11 *Triangulating geometry in* ***TRACK*** *is possible but it is far from an accurate way of modeling 3D geometry. Unfortunately this was the only affordable way of producing geometry data for this data set.*

7.3 Performance

This system is targeted specifically towards the visual effects industry with no-compromise-photo-realistic-results in mind. That means that there is no real need for real time capturing as long as the results are good enough to save time and money in the creation of an effects shot. Since no production class test sequence has been shot or evaluated it is hard to say how well the system will scale but the test sequences seamed alright.

Both the cloth sequence and the Regina sequence took around 3 to 4 seconds per iteration to process. The cloth has much more vertices whereas Regina was captured with one more camera. The cloth sequence would allow 10 to 20 iterations whereas the Regina sequence would take 5 to 10 iterations. This is probably because of the noisier nature of photographed material which probably introduces more local minima where the minimizer can get stuck. Both sequences could easily be worked on with a fairly standard workstation with two 1 GHz processors (only one processor was used for capturing) and 2 GB of memory.

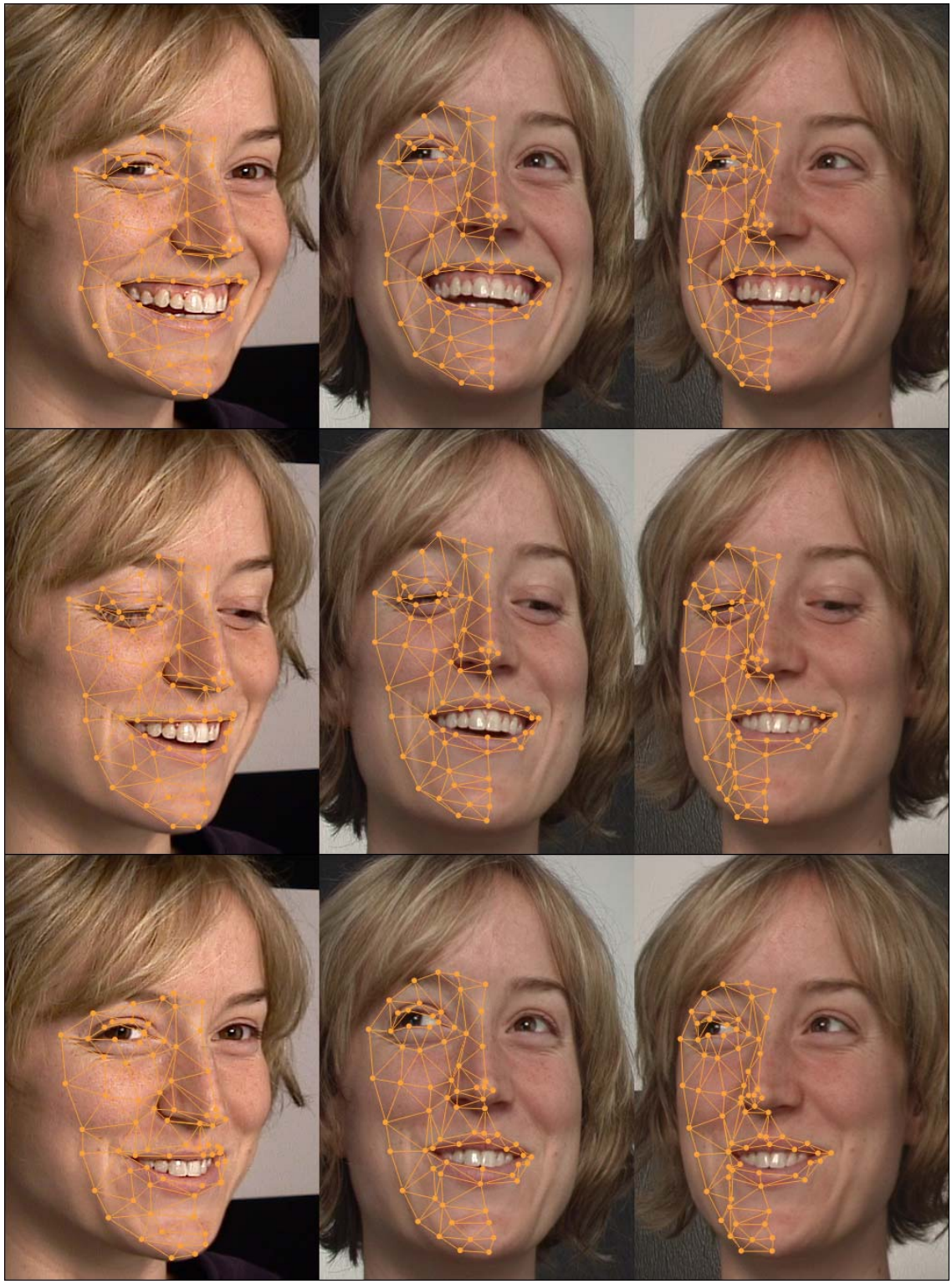


Figure 7.12(top), 7.13(middle), and 7.14(bottom) *Resulting deforming mesh rendered on top of the input data. Even this course mesh captures the animation fairly well and sticks to the deforming face.*

8. DISCUSSION

Overall this project went well and according to plans. The method presented here has yet to be used in production but the potential was good enough for a publication at SIGGRAPH 2005. In most cases neither markerless tracking nor per vertex animation are necessary and this approach may seam like overkill. These kinds of features may very well never be required but in research all new ideas and approaches are interesting ideas and approaches and this system obviously has its upsides.

8.1 Summary

In this thesis a new markerless deformable model capture system is presented which is more accurate and controllable than previous methods. Computer vision methods are applied and developed to capture the geometry of deformable objects such as human faces and cloth. The markerless approach allows recovery of animation, texture and lighting at the same time.

The system assumes that the initial pose of the deforming object is known and that the deformation is captured by an array of carefully placed cameras. By modifying the basic 2D optical flow algorithm, which has been used in similar work, the movement of the 3D geometry is solved for directly and in all views simultaneously frame by frame. This technique incorporates the epipolar constraint across the cameras, reducing the search space and resulting in higher accuracy and less accumulated error.

Image information from all camera-feeds and virtual spring forces are incorporated into a single cost function. This cost function is a function of the geometry deformation and its gradients can be approximated as functions of the image derivatives. Minimizing this cost function deforms the geometry towards a better match between the image streams and the final animated geometry.

The thesis covers all basic mathematic tools that has been used such as projective math, 2D warps, Conjugate Gradient Optimization, image registration, and spring systems. It describes how these tools can be combined to build a Model Flow system and how such a system performs in comparison to an optical flow based system.

8.2 Contributions

Contributions in research are best recognized and evaluated by others than the originator of the research and hopefully by people with far more experience and knowledge than the author of this thesis. In spite of that some comparisons can be made with very similar recent work. In 2003 Borshukov et al. for example presented Universal Capture [3] which was a complete system for markerless capturing of deforming geometry using 2D optical flow and an array of cameras. Where as optical flow driven methods solve for solutions in 2D rather than solving for 3D, Model Flowing has a huge advantage. By incor-

porating the epipolar constraint into the solver and gathering all image data into one system of equations the ambiguity of optical flow methods is eliminated and the search space is radically reduced.

Other interesting components of the Model Flowing approach is the way the Jacobian incorporates gradients as functions of both image gradients in 2D and vertex positions in 3D and the extension with spring forces. 2.5D image registration has been proposed before but not incorporated in a markerless capturing system like this. Systems of springs are also well known in simulation applications but have not previously been used to support capturing of faces.

8.3 Conclusions

Model Flowing is an algorithm which has shown very promising results in initial tests on capturing deforming geometry such as human faces and cloth. The method is mathematically and practically superior to similar optical flow based methods but is still fairly simple and straight forward to implement. Some functionality need to be added and more production like testing need to be conducted before Model Flowing can be used on a large scale project but the potential is obvious.

Over all the project resulting in this thesis must be considered a success since both the supervisors at University of Linköping and at Digital Domain are happy with the results and the project resulted in a publication of a sketch at SIGGRAPH 2005, which was one of the initial goals. The time frame of this project has well overshot the intended six months but the important deadline to finish all implementation work while in LA was met.

8.4 Future Work

The code base and all rights to the implementation of Model Flowing described in this thesis is fully owned and controlled by Digital Domain. More testing is being conducted on the work that has been done so far and there are plans to use the system in production within a year. Even though a lot of work has been put into the system already it is far from perfect and there are many ways in which it can be improved. In this section some of the possible refinements and extensions are presented.

8.4.1 Occlusion

As Model Flowing is presented in this thesis it has one major flaw which makes it close to useless for most applications. It has no support for occlusion which enforces unnecessary constraints on the capturing process. Instead of just have to capture every part of the geometry with at least two cameras at all time the system now has to capture all geometry with all cameras at all time. This obviously puts huge limitations on what kind of geometry that can be captured.

Support for at least self occlusion would not be especially difficult to implement. The problem of occlusion has been worked on for decades in com-

puter graphics and is in most cases a solved problem. Information about how much of and from what angle a triangle is seen could also be used as weights for the solver. Some cameras have better views of some triangles and should therefore have greater impact on their deformations.

8.4.2 Weighting

Both for user supervision and for automation reason there should be more weighing between the different local systems in Model Flowing. An obvious extension would be controlled and automatic weighting of the spring forces. One could imaging that areas of very little color variations would automatically get less weight in the solver and the neighboring areas with good patterns to track and the springs in the area would get higher weights. This way the system as a whole would be more robust and sort of tone down its own weaknesses. These weights should also be controllable by the user who then could manually turn up the spring weights in areas where appropriate.

8.4.3 Sticky Edges

Since exact knowledge of the initial geometry at every frame is known it should not be hard to calculate areas where contours and edges in the images are likely to appear. With statistical models or edge detecting algorithms the corresponding edges and contours in the input images should be quite easy to track if the interesting areas are already identified. This in turn would open up the possibility of encourage coherence of the contours in the image and in the deforming geometry and counteract the sliding, slipping and drifting associated with flowing algorithms where error accumulate over time. This approach has been implemented by DeCarlo and Metaxas [5] with astonishing results in terms of robustness.

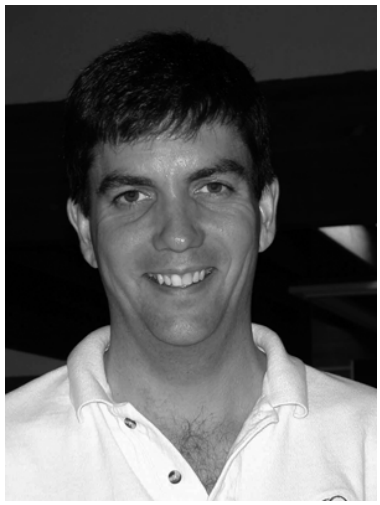
8.4.4 Optimizations and speed-ups

Even though the current implementation is not painfully slow to work with there is always room for speed-ups. Within the next generation/generations of graphics hardware, or maybe even with the current generation, a lot of the Model Flow algorithm should be possible to implement in hard ware. Projecting geometry and interpolating and comparing pixels are what graphics hardware do best. On top of that most of the slow parts of the current implementation are very parallel and should therefore be ideal for hardware implementations.

9. ACKNOWLEDGEMENTS

This project would not have been possible without the great connections between Digital Domain (DD) and the University of Linköping (LiU), and especially not without my great supervisors at DD and LiU: Dr. Douglas Roble and Professor Ken Museth. At DD the help and tutoring given by John Flynn was priceless and the main reason for Model Flowing being implemented in time. These three persons are also my co-authors to the SIGGRAPH sketch [14] that I presented on Model Flowing at SIGGRAPH 2005.

John P. Flynn
jflynn@imageworks.com
Sony Imageworks



Dr. Doug Roble
doug@d2.com
Digital Domain

Professor Ken Museth
museth@acm.org
Linköping University

I would also like to thank the rest of the software department at DD for being a great group of people to work with and especially the Swedes Magnus Wrenninge, Henrik Fält, and Mårten Larsson for making Swedes look good in the effects industry and helping new Swedes out. Thanks to Nicole Schwulera for being a great flat mate during most of my stay in LA and thanks Johan Törne, Jonas Unger, and Daniel Bernardsson for letting me stay with you at Oakwood the first couple of weeks!

Finally I would like to thank my parents of backing me up financially for this project and for visiting me on site. Other visitors I would like to thank are Hanna Jonsson, Johan Åkesson and Josefin Leander who joined me on my only vacation during my six months in LA.

10. REFERENCES

[1] D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In Proceedings of SIGGRAPH 1998, ACM SIGGRAPH, pages 43-54.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM 1994, Philadelphia, PA.
(2005-09-14) http://netlib2.cs.utk.edu/linalg/html_templates/report.html

[3] G. Borshukov, D. PIPONI, O. LARSEN, J. P. LEWIS, and C. Tempelaar-Lietz. Universal capture: image-based facial animation for "the matrix reloaded". In Proceedings of the SIGGRAPH 2003 Conference on Sketches & Applications, 1–1, 2003.

[4] J. Chai, J. Xiao, and J. Hodgins. Vision-based Control of 3D Facial Animation. Eurographics Symposium on Computer Animation. 2003

[5] D. DeCarlo and D. Metaxas. Optical Flow Constraints on Deformable Models with Applications to Face Tracking. In International Journal of Computer Vision, 38(2), pages 99-127, July 2000.

[6] I. Essa and A. Pentland. A Vision System for Observing and Extracting Facial Action Parameters. IEEE CVPR 1994 Conference, pages 76-83, Seattle, Washington, June 1994.

[7] I. Essa and A. Pentland. Coding, analysis, interpretation, and recognition of facial expressions. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7):757-763, July 1997.

[8] I. Essa, S. Basu, T. Darrel, and A. Pentland. Modelling, Tracking and Interactive Animation of Faces and Heads using Input from Video. In Proceedings of Computer Animation 96 Conference, Geneva, Switzerland, June 1996.

[9] B. Guenter, C. Grimm, D. Grimm, D. Wood, H. Malvar, and F. Pighin. Making Faces. In SIGGRAPH 98 Conference Proceedings, pages 55-66. ACM SIGGRAPH, July 1998.

[10] T. Hawkins, A. Wenger, C. Tchou, A. Gardner, F, Göransson, and P. Debevec. Animatable Facial Reflectance Fields. Eurographics Symposium on Rendering, 2004.

[11] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 674-679, 1981.

[12] F. Pighin, R. Szeliski, D. H. Salesin. Resynthesizing Facial Animation through 3D Model-Based Tracking. In the Proceedings of the Seventh IEEE International Conference on Computer Vision, pages 143-150 vol.1. September 1999.

[13] X. Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior, In Proc. Graphics Interface '95, pp 147-154. 1995.

[14] K. Reuterswärd, J. Flynn, D. Roble, K. Museth. Model Flowing: Capturing and Tracking of Deformable Geometry. In ACM SIGGRAPH 2005 Conference on Sketches & Applications, 2005.

[15] J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, School of Computer Science, Carnegie Mellon University. August 1994.

[16] D. Terzopoulos, K. Waters. Analysis and synthesis of facial image sequences using physical and anatomical models. IEEE Pattern Analysis and Machine Intelligence. 1993

[17] L. Williams. Performance-driven facial animation. In SIGGRAPH 90 Conference Proceedings, volume 24, pages 235-242. August 1990.

[18] L. Zhang, N. Snavely, B. Curless, and S. M. Seitz. Spacetime Faces: High-resolution capture for modeling and animation. In ACM SIGGRAPH Proceedings, Aug., 2004.

[19] Q. Zhang, Z. Liu, B. Guo, and H. Shum. Geometry-Driven Photorealistic Facial Expression Synthesis. Eurographics Symposium on Computer Animation. 2003

[20] Conjugate Gradient Method on mathworld (2005-09-14):
http://mathworld.wolfram.com/ConjugateGradientMethod.html

[21] SparseLib (2005-09-14):
http://math.nist.gov/sparselib++/