ACTIVE MESHES FOR MOTION TRACKING

BY

DEREK MOLLOY (B.ENG.)

(DEREK.MOLLOY@EENG.DCU.IE)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
(ELECTRONIC ENGINEERING)

SUPERVISED BY DR. PAUL F. WHELAN

SCHOOL OF ELECTRONIC ENGINEERING
DEPARTMENT OF ENGINEERING AND DESIGN
DUBLIN CITY UNIVERSITY

MARCH 2000

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:________________________ Date:________________

Acknowledgement

I wish to express my sincere gratitude to my supervisor, Dr. Paul F. Whelan for his constant support and advice. I wish to thank Charles McCorkell and the School of Electronic Engineering, DCU, for encouraging and supporting me during this research. I have benefited from many discussions with Ovidiu Ghita, Robert Saldleir and Alex Drimbareau. I value the help and advice that they have given me, and thank them for constructive comments on my work. I also wish to thank my parents for their support and Sally for her patience, love and proof reading skills.

I would also like to thank the external examiners Prof. Fionn Murtagh and Dr. Conor Hennigan for the time that they put into the examination of this work and for the suggestions that they made.

Abstract

Active Meshes for Motion Tracking

Derek Molloy

Under the supervision of Dr. Paul F. Whelan

at Dublin City University

Submitted in partial fulfilment of the requirements for the degree
of Doctor of Philosophy at Dublin City University, 2000

This thesis presents an integrated approach to modelling, extraction and tracking of deformable contour meshes through image sequences, with the aim of extracting motion information about the viewed scene. The thesis begins by reviewing the area of motion estimation in computer vision, leading to a review on the formulation and initialisation of active contour models. From this review the thesis develops and provides as its major contribution an active mesh structure that may be used for motion estimation. This active mesh structure approach is combined with feature matching to provide a stable, deformable motion tracking system for real-world scenes. This system is tested on various real-world scenes and varying conditions to provide extensive and rigorous experimental proof of the validity of the formulation. Further extensions to the system are implemented, including the use of multiple and region based active meshes. Future directions of research are also suggested.

Table of Contents

Table of Figures

Chapter 1 - Introduction

1.1 Motivation

In a scene that is being viewed by a digital camera, movement of objects in the scene, or the camera itself, will in general result in changes to the intensity values in the image being viewed. On the basis of these intensity changes, motion-tracking algorithms attempt to deduce the actual motion of the objects in the scene, or of the camera, that caused these changes. There are three possible motion situations: a stationary camera with moving objects, a moving camera (*termed egomotion*) with a stationary scene and a moving camera with moving objects.

To further avoid confusion over the use of the term 'motion', Nagel (1990) defined four categories of motion, to clarify the distinction between motion in the 2-D projected image plane and motion in the actual 3-D scene. These categories are:

- The real velocity of a 3-D point on a time-varying curve,
- The apparent velocity of the 3-D point on this curve,
- The real motion of the image of a point in the image plane,
- The apparent motion of a grey-level structure in the image plane.

Frequently motion approaches fail to make explicit which type of motion is being measured. In the case of Optical Flow, for example, it is the fourth type that is being calculated, but often under the mistaken assumption that it is the projection of the real motion of the 3-D velocity field onto the image plane (case three), that is being calculated.

Many methods are available for motion tracking, each with their own individual features and failings. Traditional methods such as optical flow and feature matching are well established, with numerous approaches available and currently in development. More recently, research in deformable templates has provided promising results in motion tracking, using weak models, such as 'snakes', that deform in response to salient image features.

The abilities of motion detection, tracking and estimation are essential for automation of the tasks of automatic vehicle guidance, traffic speed control (automatic ticketing) and traffic analysis (e.g. junctions, roundabouts). Other applications include characterisation of human motion, medical imaging, and data rate reduction for image compression and satellite imagery. The initial application area of this research was vision-based aids for the visually disabled[1] with primary research and ideas discussed in Molloy *et al* (1994).

If the camera is static in respect to the scene then the task of extracting motion information is substantially simplified. Techniques such as image frame differencing will in this case provide clear cues about the movement (or at least provide regions of interest) within the sequence, greatly aiding the determination of motion information. There is no such assumption being made in this work, where more dynamic (and indeed interesting) scenes are to be examined, such as scenes in which the observer may be moving, the objects in the scene may be moving or indeed the entire scene may be moving. For this form of motion analysis to be useful for real-world applications, it is necessary for the computation involved to be very efficient.

1.2 Contributions

This thesis presents an integrated approach to modelling, extracting and tracking deformable meshes directly in real-world image sequences. It begins by performing a review of current motion tracking techniques, developing these techniques for combination with deformable models to provide the basis of this work. The theories are developed for the approach and the validity is examined through extensive testing on synthetic and real-world image sequences.

The immediate contribution of this work is the overall method that provides an innovative approach to helping solve the motion-tracking problem, combining a self-initialisation technique, automatic modelling and unstructured meshes with suitable force handling to develop an active weak model approach. The technique was further extended to multiple meshes, region meshes and applied to depth estimation. Java technology was applied to computer vision applications, finding useful aspects of the language and possible difficulties for further use. The SUSAN[2] corner detection algorithm

[1] More recently other implementations in this area of research can be found in *Image and Vision Computing*, April 1998, a special issue on “Vision-Based Aids for the Disabled”.

[2] Smallest Univalue Segment Assimilating Nucleus – Discussed in detail in Section 2.6.2.

(Smith, 1992) was modified using pre-filters for improved computational performance and was also implemented within an active contour model framework.

1.3 Organisation

The organisation of this thesis is as follows: Chapter 2 presents a review of the 'classic' approaches to motion estimation, including methods such as optical flow that attempt to compute a measure of velocity at every point in the image. Feature matching is also discussed, that attempts to compute motion from matching distinct image features between frames. Feature extraction methods are also discussed in some detail. Chapter 3 discusses active contour models, including the formulation and initialisation aspects that are associated with them. The discussion further expands on the different forms of active contour models that are available and describes some current applications. Chapter 4 details the active mesh approach, providing a discussion on the structure, the energy formulation and the various 'flavours' of this approach. Chapter 5 gives an extensive set of results to support this method, including simulated and real-world image sequences. Finally, Chapter 6 summarises the work and suggests future research directions.

Chapter 2 - Methods for Extracting Motion Information

2.1 Introduction

There is no single technique that can solve the motion analysis problem for all scenarios and indeed each technique presented here works only if certain conditions are met. Prior knowledge of the scene, such as information about the movement of the camera and the frame rate can help reduce the complexity of the analysis, helping choose the best technique for a particular application. Traditionally, there have been two approaches to the motion estimation problem: *Optical Flow* and *Feature Matching*. This chapter will discuss these techniques and introduce some less traditional approaches.

2.2 Traditional Approach of Optical Flow

Optical flow techniques are concerned with the apparent motion of grey value structure in the image plane. Optical flow analysis consists of two distinct stages:

- The first stage is to assign a 2-D vector to the grey value structure at each pixel in the image by examining its neighbouring points. At this stage no attempt is made to relate this apparent local motion to the motion of real objects in the scene. The *Optical Flow Field* is calculated, and can be defined as: *the 2-D vector field of apparent velocities of grey-value structure in the image plane.*
- The second stage is to use this flow field to compute as much information as possible about the physical motion in the scene, i.e. to interpret the estimated flow field. For example, discontinuities in a flow field can help segment images into regions that correspond to different objects. This is a complex problem that is not aided by the difficulty in computing an accurate optical flow field.

The optical flow field cannot be computed at a point in the image independent of neighbouring points without introducing additional constraints. This is because the velocity field at each image point has two components while the change in image brightness at a point in the image due to motion gives only one constraint (Horn and Schunck, 1981). The relationship between the optical flow in the image plane and the velocities in the real world is not always obvious. For example, a uniformly coloured sphere when rotated around its centre axis gives a zero optical flow field at all points in the image, as the shading pattern does not rotate with the surface but is caused by the direction of the surface of the sphere (Nagel, 1987).

2.2.1 Assumptions

In the method of Horn and Schunck (1981), to avoid variations in brightness due to shading effects, it is initially assumed that the image surface is flat and uniformly lit. The brightness at a point in the image is taken to be proportional to the reflectance of the surface at the corresponding point on the object. It is assumed that the reflectance varies smoothly and has no spatial discontinuities, thus allowing the image brightness to be differentiable (discontinuities in reflectance will occur at object boundaries and are thus excluded). In this simple situation described, the motion of the brightness patterns in the image relate directly to the motion of the objects and computing the motion of the objects is a case of simple geometry.

Providing that these assumptions are made then the following theory results: If the intensity of a point (x,y) at time t is $I(x,y,t)$, optical flow theory assumes that the same point displaced by motion will have the same intensity at time $t+\Delta t$, when it is moved to the point $(x+\Delta x,y+\Delta y)$, therefore:

$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\Delta t) \qquad 2\text{-}1$$

Using a Taylor expansion, we can approximate this by[3]:

$$I(x,y,t) = I(x,y,t) + I_x\Delta x + I_y\Delta y + I_t\Delta t + \textit{higher order terms} \qquad 2\text{-}2$$

Neglecting higher order terms,

$$I_x\Delta x + I_y\Delta y + I_t\Delta t = 0 \qquad 2\text{-}3$$

Dividing by Δt and taking the limit as $\Delta t \to 0$ we obtain

$$I_x u + I_y v + I_t = 0 \qquad 2\text{-}4$$

This gives one equation with two unknowns, where I_x, I_y and I_t are the estimated partial derivatives at every point in the image and $u = dx/dt$, $v = dy/dt$ the unknown velocities in the x and y directions, as required. This implies that the optical flow field cannot be computed at a point in the image independently of neighbourhood points without introducing additional constraints.

[3] For an image $I(x,y)$, this work will use the notation $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_{xx} = \frac{\partial^2 I}{\partial x^2}$, $I_{xy} = \frac{\partial^2 I}{\partial x \partial y}$, $I_{yy} = \frac{\partial^2 I}{\partial y^2}$

2.2.2 Optical Flow with Added Constraints

Three of the possible assumptions that are commonly made are:

- Constant Velocity Constraint.
- Smoothness Constraint.
- Rigidity Constraint.

The next three sections explain these constraints.

2.2.3 The Constant Velocity Constraint

Much of the work on optical flow has assumed that velocity is constant over small patches of an image and so can be approximated by pure translation. If the projection of a scene onto the image plane is orthographic, the constant velocity constraint is strictly valid only when objects undergo pure translation (Hildreth, 1984). It is not strictly valid for rotations under orthographic projection or for any motion under perspective projection (Schunck, 1986). Let the image brightness at the point *(x,y)* in the image plane at time *t* be denoted by *E(x,y,t)*.

Now consider what happens when the pattern moves, the brightness of a particular point in the image in the pattern is constant (Schunck, 1986), so that:

$$\frac{dE}{dt} = 0$$

Using the chain rule:

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0 \qquad 2\text{-}5$$

Letting $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$,

results in a linear equation with two unknowns *u* and *v*, where *u* represents the *x* component of velocity and *v* represents the *y* component of velocity.

$$E_x u + E_y v + E_t = 0 \qquad 2\text{-}6$$

The constraint on the local flow velocity expressed by this equation is shown in Figure 2-1. This equation may be rewritten as:

$$\left(E_x, E_y\right) \bullet (u,v) = -E_t \qquad 2\text{-}7$$

Thus the component of the movement in the direction of the brightness gradient $\left(E_x, E_y\right)$ is:

$$-\frac{E_t}{\sqrt{E_x^2 + E_y^2}} \tag{2-8}$$

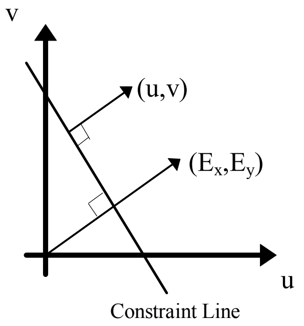


Figure 2-1. Optical flow constant velocity constraint.

2.2.4 The Smoothness Constraint

The smoothness constraint is the second type of constraint assumption. If every point in the image were to move independently then there would be little point in trying to calculate motion in the scene. The basis of the smoothness constraint is that in general neighbouring points on the same object have similar velocities, assuming that the world consists of solid objects whose surfaces are smooth compared to the distance to the observer. Assuming that a smooth surface in motion generates a smoothly varying velocity field then the smoothness constraint aims to find the velocity field that varies as little as possible. Discontinuities in flow can be expected, for example, in the case where one object occludes another.

Horn & Schunck (1981) use this additional constraint by minimising the square of the magnitude of the gradient of the optical flow velocity,

$$\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 \text{ and } \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \tag{2-9}$$

Another technique used to measure the smoothness constraint was the sum of the squares of the Laplacians of the *x* and *y* components of the flow. The Laplacians of *u* and *v* are defined as:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \text{ and } \nabla^2 v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}. \tag{2-10}$$

In the simple case, both Laplacians are zero. If the viewer translates parallel to a flat object, rotates about a line perpendicular to the surface or travels orthogonally to the surface, then the partial derivatives of *u* and *v* vanish (Horn and Schunck, 1981). This formulation breaks down when motion boundaries occur in the image, such as at the object boundary where the pixel belonging to the object will have non-zero velocity, while a pixel belonging to the stationary background will have zero velocity. This is in direct violation of the smoothness constraint and large errors occur at these motion boundaries. This is undesirable since motion boundaries often contain more information about the motion involved than the actual area of the object. Nagel (1987) tried to overcome this problem by supporting an 'oriented' smoothness constraint, designed so that in areas of strong grey-value gradient, the variation of the vector field is in the direction along the contour lines of constant grey value.

2.2.5 Rigid Motion in the Image Plane

Some motion measurement schemes allow objects to undergo rigid motion and translation in the image plane. Suppose a rigid curve undergoes a general motion in space. The motion may be described as the combination of:

- A rotation with angular velocity ω about a single axis in space which can be denoted by the unit vector $\mathbf{n} = [n_x,n_y,n_z]^T$ (i.e. transposed).
- A translation, denoted by $\mathbf{t} = [t_x,t_y,t_z]^T$.

Let the curve be given by $C = (x(s), y(s), z(s))$, where *s* denotes arc-length. The location of a point on the curve may be given by the position vector $\mathbf{r} = [x(s), y(s), z(s)]^T$ If it is assumed that the axis of rotation passes through the origin of the coordinate system, the velocity of a point given by the position vector **r** is equal to the cross-product of **r** with the vector $\omega\mathbf{n}$. Hildreth (1984) lets the optical axis lie in the *z*-axis, and claims that the 2-D velocity field is given by:

$$\mathbf{V}(s) = \mathbf{M}(\mathbf{r} \times \omega\mathbf{n} + \mathbf{t}) = -\omega z(s)\begin{bmatrix} n_y \\ -n_x \end{bmatrix} - \omega n_z \begin{bmatrix} -y(s) \\ x(s) \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad 2\text{-}11$$

where **M** denotes the matrix that performs the orthographic projection. The first term denotes the component of the velocity field due to rotation in depth about an axis parallel to the image plane (the axis $\mathbf{n} = [n_x,n_y,0]^T$). The second term denotes the component due to rotation in the image plane (the axis $\mathbf{n} = [0,0,n_z]^T$) and the third term denotes the translation component. Consider the restricted case of rigid motion in the image plane, giving a translation, and rotation about $\mathbf{n} = [0,0,1]^T$, thus **V**(s) becomes

$$\mathbf{V}(s) = -\omega \begin{bmatrix} -y(s) \\ x(s) \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad 2\text{-}12$$

V(s) is a simple translation, rotation and scaling of the image curve *(x(s),y(s))*, as shown in Figure 2-2.

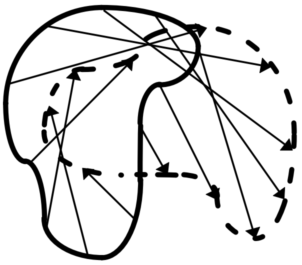


Figure 2-2. Velocity field for rotation in the image plane, under rotation and translation (Hildreth, 1984).

A simple geometric reconstruction can now be used on this velocity field, so that if the true direction of velocity is known at two points on the contour, then the direction of velocity can be computed everywhere by:

- At the two known points (P_1 and P_3), construct perpendicular lines to the direction of velocity (see Figure 2-3).
- Calculate the intersection of these two lines, *q*.
- From every other point (e.g. P_2) on the contour, construct the line to the intersection point *q*, and the true direction of velocity is perpendicular to this line.

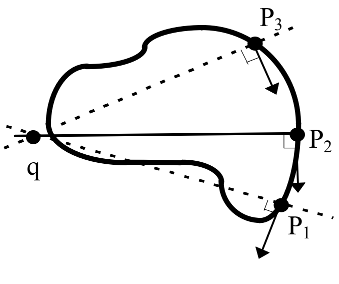


Figure 2-3. Geometric Reconstruction (Hildreth, 1984), where the construction of the velocity field can be performed when the direction of velocity is known at points P_1 and P_3 it can then be computed at P_2.

Hildreth (1984) uses the assumption that all significant motion information in an image is contained in the contours or edges, so the motion should only be calculated along these contours. This is a more general case of the ‘cornerness’ method used by Nagel (1987) as shown later. This approach has a very high computational load (of the order of $O(N^2)$, where N≈1000). It also has difficulty in dealing in junctions that result from occlusion, as they are represented as a single contour that crosses motion boundaries. Appendix A describes the estimation of the partial derivatives and the minimisation of errors in optical flow techniques.

2.2.6 Difficulties with Determining Optical Flow

In general the solution is most accurately determined in regions where the brightness gradient is not too small and varies in direction from point-to-point. On the other hand, if the gradient is too large then the estimation of the derivatives will be corrupted by under-sampling and aliasing, thus a good *tightness of constraint* is required. An area in which the brightness gradient is small leads to uncertain, noisy estimates that are estimated from the surrounding values.

One problem with optical flow in general, is that it suffers from the *aperture problem*. In Figure 2-4, the local movement as in the optical flow technique is to be examined. Suppose there is an extended line moving across this aperture from left to right, and the optical flow technique used determines a vector $\boldsymbol{V}^{\perp}$ to describe this motion. It is not sufficient to describe this motion as $\boldsymbol{V}^{\perp}$, since the true velocity could be anyone of the infinite velocity vectors $\boldsymbol{V}$. Most optical flow techniques compute only this orthogonal velocity $\boldsymbol{V}^{\perp}$. Nagel (1987) tries to overcome the problem by having a 'cornerness' measure in areas where the aperture problem cannot usually occur (e.g. corners). He then interpolates between these points in conjunction with the normal velocity measured between these points.

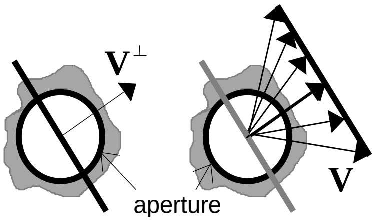


Figure 2-4. Illustration of the aperture problem.

2.2.7 Second Order Optical Flow Techniques

Differential techniques for optical flow compute velocity from spatiotemporal derivatives of image intensity. This problem is under-constrained and hence must be constrained using one of the previous three techniques described. Second order optical flow techniques use the second order derivatives to constrain the 2-D velocity. However, due to noise sensitivity and the aperture problem, results are in general less accurate than the first order methods, according to Barron *et al* (1992).

2.2.8 The Global Approach to determining Optical Flow.

Optical flow computation is based on two assumptions; (1) the observed brightness of any object point is constant over time, and (2) nearby points in the image move in a similar manner. This means that optical flow computation will be violated where there are dramatic changes in intensities, such as in highly textured areas or around moving boundaries. Unfortunately, real-world applications often have to deal with this problem. The global approach attempts to smooth the velocity field consistent with the entire image, reducing the problems with the above assumptions. Local constraints are propagated globally, although usually beneficial, can also cause local estimation errors to be propagated globally. A small number of problem regions will cause the errors to propagate and lead to bad optical flow estimates.

Torr and Murray (1992) suggest an interesting global approach where they examine the movement of the background as global flow. Areas of the image that do not obey the characteristics of the global flow of the background are marked as regions of interest. These areas are then examined using other optical flow techniques. Reasonably good results are recorded and their algorithm helps localise areas of interest within image sequences.

2.2.9 Local Approach to determining Optical Flow

The local approach to determining optical flow is based on the same principle as the global approach, where the optical flow assumptions are reinforced, in this case by breaking the image into small regions where the assumptions hold. Local based methods combine local estimates of the normal velocity through space and time. The method of local optimisation estimates optical flow by solving a group of gradient constraint lines obtained from a small region of the image as a system of linear equations. It has also been performed by fitting the local neighbourhoods to a 2-D velocity field (a low order polynomial model in (u,v) using a least squares fit).

This solves the error propagation problem of the global approach but there are problems where the spatial gradient changes slowly as the local approach becomes ill-conditioned due to a lack of motion information. The global approach would have little trouble with this region, as information from neighbouring locations would propagate into this region, giving a reasonably good estimation of the flow.

Kearney *et al* (1987) produced an approach with continuous adaptation to errors. By understanding how the errors occur, they attempt to define the inherent limitations of the local approach, obtaining estimates of the accuracy of the technique and enhancing the performance of the technique. Their basic local optimisation method performs a least squares minimisation on an over-determined set of gradient constraint equations to estimate optical flow at each point. Each image is Gaussian blurred to reduce noise in the image and to linearise the brightness function. They note that local optical flow techniques have problems with the following types of regions:

- Largely homogeneous regions.
- Highly textured regions that are moving.
- Areas with large discontinuities in the flow field.

Murray & Buxton (1987) attempt to merge the local and global approaches. The local approach suffers from the disadvantage that the 'resulting mosaic' of tiny pieces must be pieced together, using similar motion and structure, into larger surfaces. Taking a global view of this problem is beneficial to the combination of the 'resulting mosaic' and results are given for small regions.

2.3 Optical Flow - Recent Approaches

2.3.1 Velocity Tuned Filters

Velocity Tuned Filters are a more recent approach that is similar to optical flow in that it computes the velocity at a large number of points in the image. The velocity-tuned filters examine the motion phenomena in the frequency domain[4]. The method behind this may be explained by looking at the case of a 1-D sine wave grating that seems like horizontal bars, moving across the image. At any particular point in an image, the spatial intensity of the point will vary as the horizontal bars move across the point. The rate of change of intensity of the point f_t increases both with the velocity of the grating v and its spatial frequency f_x. This relationship may be given simply as $f_t = vf_x$ which may be written as:

$$v = {}^{f_t}\!\big/_{f_x}, \qquad \text{2-13}$$

[4] The frequency domain refers to either the temporal frequency domain or the spatial frequency domain.

In the spatiotemporal frequency domain, with f_x and f_t as the axes, this equation describes a slope defined by the velocity of the grating. Combining a *vertical* spatial frequency f_y that can move with a velocity u, results in a ‘checked’ pattern that can move in 2-D, giving the equation:

$$f_t = uf_x + vf_y, \qquad \text{2-14}$$

a plane in the spatiotemporal frequency domain, with orientation given by u and v. This means that the velocity of a translation in the image plane will be equivalent to a plane of high power in the frequency domain. The problem of estimating velocity in the image plane becomes; take small regions of the image plane and fit a plane to each of their power spectra in the spatiotemporal frequency domain (Simoncelli and Adelson, 1991).

2.3.2 Energy Outputs of Velocity Tuned Filters

One of the most common models is the Elaborated Reichardt Detector (ERD) model, in which responses from two spatial locations are multiplied together. The input stages include spatial-frequency-tuned receptive fields, such as Gabor filters[5] with pairs that differ in phase or position by about 90° to build a motion detection unit. It is also possible to develop direction sensitive filters, by carefully selecting these filters. The method of Heeger (1998) uses 12 Gabor filters at each spatial scale to formulate the problem as a least squares fit of the filter energies to a plane in frequency space (Barron *et al*, 1992). That is, it takes small spatiotemporal windows from the image sequence to find the plane in which the energy resides and thus the motion. A least squares estimate is used to minimise the difference between the predicted and estimated energies, thus solving for *u* and *v*. This method does not suffer from the aperture problem as calculations are performed at a number of scales, however smoothing occurs at motion boundaries.

2.3.3 Phase Outputs of Velocity Tuned Filters

An alternative approach to dealing with the output of velocity-tuned filters is to examine the phase of the outputs exclusively. In general, algorithms for determining the best value of *u* and *v* in the spatiotemporal domain will obtain only one value. It would however be useful to determine more than one velocity, such as in the case of motion

[5] Gabor filters - Gabor, D., Nobel Prize winner and inventor of holography. A 1-D odd-phase Gabor filter is a sine wave multiplied by a Gaussian window. Its power spectrum is a pair of Gaussians centered ±ω, thus a band pass filter. Heeger uses a 3-D version, with 2 spatial and 1 temporal dimension. Band pass filters are used to decompose the input signal according to scale, speed and orientation.

boundaries. The method is similar to the energy approach using 3-D Gabor filters although it examines the phase output of the filters. This method proposed by Fleet and Jepson, is described in Barron *et al* (1992), as looking at the phase of the outputs; they consider space-time surfaces of constant phase, which evolve due to the motion field. The phase output of the filters may be related to the normal velocity quite easily as shown in Byrne (1992).

2.3.4 Improving Results Using Kalman Filtering Over Image Sequences

One technique, which has been implemented successfully in a range of applications, is that of *Kalman Filtering*. It is used in areas where noise is present and repeated measurements are made for more accurate estimation. Kalman filtering allows the estimate to be incrementally updated with every measurement, while also providing a reduction in uncertainty for the measurement variable. This allows the velocity measurements of each frame to be integrated with previous ones, allowing a progressive reduction in uncertainty and a measure of uncertainty to be attached to the velocity of each point. This gives confidence in the attachment and accuracy of measurement in different areas of the image. Problems do exist however; as the initial measurement is relatively inaccurate, it takes time to build up confidence in the measurements. This means that scenes, in which objects are constantly entering and leaving, might not have enough time to build up an accurate estimate. Ancona (1992) applies the method of Kalman filtering to stabilise the estimates in the calculation of the focus of expansion[6] for obstacle detection. He defines an obstacle for his method as a plane lying on the ground, orthogonal to it and high enough to be perceived.

Another attempt at improved optical flow estimation is motion coherence theory. It is clear that regions moving together tend to belong to the same object, but this was not accounted for in prior theories of motion estimation. Yuille & Grzywacz (1989) present theories of motion correspondence and attempt to develop them mathematically. For example, if features are located spatially close then they tend to move together, so they define a Gaussian filter that falls off with distance from each feature for linking those features together. Secondly, local motion may be inaccurate, so they introduce a cost

[6] The focus of expansion (FOE) is often used to determine the direction of motion. See McQuirk *et al* (1998) for an up-to-date discussion on the FOE and time to collision (TTC) algorithms.

function for smoothness that integrates this local measurement over a large area, to improve performance. Their approach worked well on artificial images.

2.3.5 Discussion on Optical Flow Techniques

Each approach to optical flow analysis has been found to have its own individual strengths and weaknesses. According to a key paper on performance of optical flow techniques by Barron *et al* (1992) the differential methods of Horn & Schunck (1989) proved poorly in accuracy. This seemed due to the smoothness term, which produced attractive, but inaccurate flow fields. The local based approach was found to be 'encouraging', also providing a good measure of the accuracy of the estimated flow. Gradient-based methods that locally solve for optical flow suffer from three principle sources of errors:

- The brightness gradients will be poorly estimated in regions of high texture.
- Variations in optical flow will violate the constant local flow assumption.
- There must be sufficient local variation in the brightness gradient to avoid the poor error propagation characteristics that are associated with ill conditioned systems.

On the test sequences used in Barron *et al* (1992) the energy-based methods proved quite successful, and performed better than the general differential based flow methods, but not quite as good as the local based flow methods. The energy-based method can overcome the aperture problem (as previously described), as the image is processed at a number of scales and not just locally. However, due to Gaussian convolution and other factors, the velocity resolution is 'smoothed-over' in local regions, giving rise to errors at motion boundaries.

Phase based methods have some advantages over the energy-based methods in velocity tuned filters:

- Velocity resolution is preserved by also taking responses from neighbouring filters.
- There is sub-pixel accuracy.
- Phase information is more robust, especially with regard to changes in speed and contrast. Thus, this method is relatively insensitive to changes in viewing directions and lighting directions.

Unfortunately, in retaining velocity resolution in small areas this method suffers from the aperture problem. Results were found to be more accurate than the gradient based approach, but it is only the normal velocities that are computed, and not the full velocities as in Heger's method. It was quite sensitive to aliasing and corruption at high frequencies especially when a number of frames less than 15 were available (Barron *et al*, 1992).

Various techniques have tried to segment scenes by identifying *motion discontinuity boundaries* in the optical flow that measures the image motion in each individual pixel in the scene. There are several problems with this:

- First of all, optical flow is quite difficult to determine accurately and robustly.
- Along image edges only one component of the flow can be calculated due to the aperture problem.
- Smoothing is often used at these points, and this smoothes across the motion boundary, causing loss of information.
- Motion discontinuity boundaries do not necessarily correspond to the boundaries of independently moving objects (they could be rotating edges on an object).
- Image motion is often too small to show up the motion boundaries.

These problems suggest that an approach with the following characteristics should be taken (Wiles *et al*, 1995):

- Image motion should be measured only at strong image features. A feature is an image measurable, the image motion of which can be measured more accurately, such as corners, edges, regions and textures.
- An appropriate mathematical framework must be used to cluster observed features into rigid objects.
- The features must be tracked over an extended period of time.

In Figure 2-5, the vectors along the contour represent the local perpendicular components of motion that may be extracted. To compute the true motion of this region a second stage is required, that combines these local measurements. This combination stage faces huge theoretical difficulties, since the movement of elements in an image is not determined uniquely by the patterns of changing intensity. Thus, the true velocity is not determined uniquely from the initial local motion measurements. This can be attributed to two main factors; firstly the loss of information due to the projection of the 3-D world onto a 2-D image (different surfaces undergoing different motions, may translate to the same 2-D image), and secondly a loss of information due to the projection onto a pattern of changing intensity (e.g. a uniformly shaded sphere, rotating around its central axis).

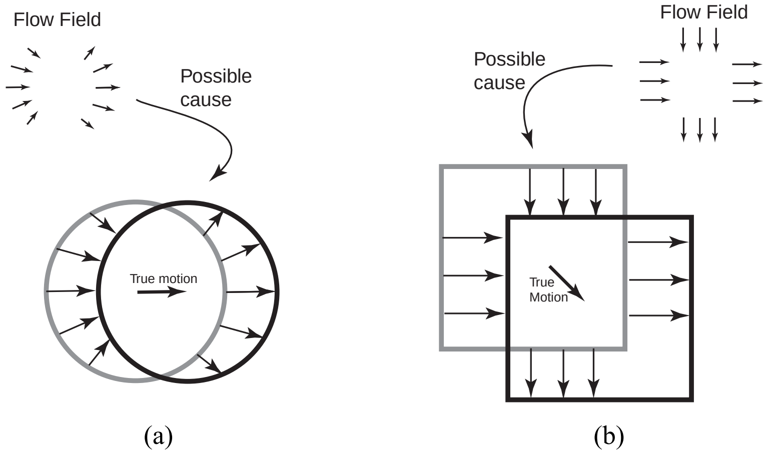


Figure 2-5. Components of motion (a) and (b) illustrating two examples of flow fields that are difficult to relate to true motion events.

Most optical flow algorithms rely on taking derivatives. This is a noise enhancing process; so many optical flow algorithms work well on artificial images but break down when exposed to noisy real world images.

2.4 Traditional Approach of Feature Matching

Optical flow computes a dense velocity field, but in direct contrast to this approach feature matching is based on the extraction of a relatively sparse, but highly discriminatory set of features in the image, such as corners, edges, or even grey-level regions. By processing image features, the volume of data is reduced by orders of magnitude compared to the original data volume, which is particularly important for an application executing in real-time.

The choice of the feature type may depend on the availability of that type of feature in the image and the reliability of its measurement. Feature matching, also known as motion correspondence may be described as: *given n frames taken at different time instants, and given m points in each, map a point in one frame to another point in the next frame such that no two points map onto the same point.*

There are two distinct problems to be addressed by feature matching techniques. The first is extracting and maintaining correspondence between the features extracted from one frame and the features extracted from the subsequent frame. The second is to compute the 3-D structure and/or motion of objects in the scene based on the calculated correspondences. The effects of occlusion and disocclusion, further add difficulty.

In solving the first problem there are two distinct steps. The first is to find the features in two or more consecutive frames, reducing the amount of information to be processed and hopefully providing a higher-level of understanding of the frame by removing the information that is less useful. The second step, in its simplest form, is to match the features in these two sets of features, to provide a single set of motion vectors.

2.4.1 Point Matching

Originally an orthographic or parallel imaging model was assumed whereby points in the scene were orthographically projected onto the image plane. Ullman (1979) showed that if three distinct images of four points in motion were taken, it was possible to develop a set of linear equations that could be solved to find the position and motion of the four points. However, this is not practical for real-world applications that require perspective (a pinhole camera model) and not just parallel projection, greatly increasing the complexity of the problem.

Perspective projection involves taking a 3-D scene and projecting it onto a 2-D surface, taking into account the effects of distance on its appearance. Generally with the perspective model the pinhole camera model is used, where the projection centre is taken as a point in space, with the image plane located between the optical centre and the scene being viewed. Roach and Aggarwal (1979) examined the situation of a static scene and a moving camera, to see if it would be possible to determine the position of the matched points in space, and the translational/rotational motion of the camera. They showed that it is possible if a minimum of five point matches are obtained between two views, but this however requires solving 20 non-linear equations (Faugeras & Maybank, 1990). Other researchers came forward with a similar formulation that resulted in a set of linear equations. The price for linearity was an increase in the number of matches from five to eight points for each set of frames. This method was unfortunately found to be quite sensitive to noise when the minimum number of points was used. For a complete review of early work, see Tsai & Huang (1981).

If the positions of feature point pairs are located without error, then each matched feature-point pair provides a single constraint on the camera parameters. In dealing with egomotion, using only image data, only five of the camera motion parameters are determinable, due to the speed-scale ambiguity. According to Harris (1987), using 5 feature points pairs, the motion parameters may be solved exactly in ideal cases. However, in real-world images the matches may not be determined as accurately, leading to large errors and an ill-conditioned problem. The more points that are available, the better conditioned the problem becomes. Some of the latest approaches that perform feature matching are discussed in the section on ASSET-2 (see Section 2.9).

Yao and Chellappa (1995) address the problem of tracking a set of features over a long sequence of image frames. They propose a localised feature-tracking algorithm that has the merits of two frame and long-sequence based approaches. The algorithm allows for the inclusion of new features to be tracked from subsequent frames. However, the algorithm is limited, in that it is explicitly designed to track feature points to determine the ego-motion of the camera. A correlation approach is used to identify the corresponding points in subsequent frames and so it is not a suitable algorithm for tracking feature points on the boundary of independently moving objects.

2.4.2 Line Matches

An alternative approach is to use lines or edges instead of points as features. Many factors such as lighting and surface reflection often cause a change in the position of points in an image. Lines are usually less sensitive to these effects and a line-fitting algorithm along a sequence of edge points can generally determine the location and orientation of a line. Therefore long lines are preferred and so the more edge points, the more accurate the measurement of the line position. When matching lines, it is necessary that at least three views be taken, rather than the two views needed for points. This is because lines possess an extra degree of freedom. This causes a large increase in difficulty when trying to use this method. Kalman filtering is often used to solve the non-linear equations involved in obtaining the best estimate of the motion parameters. In Figure 2-6*(a)*, matching *l* and *l'* simply defines a 3-D line *L*, but puts no constraint on the relative position of C_1 and C_2. However, in Figure 2-6*(b)*, three frames provide constraints about the motion. From C_3 it is apparent that the line is tilted in the *z*-direction, an observation that is not apparent from (a). See Medioni and Nevatia (1984).

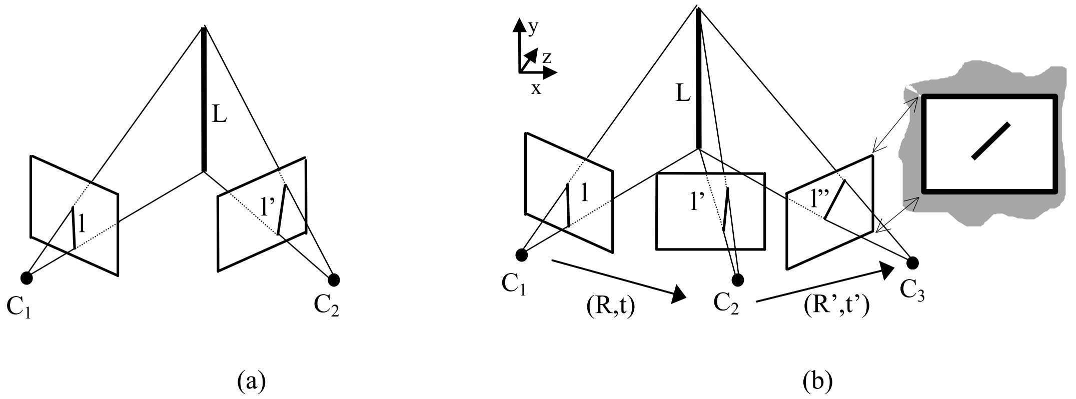


Figure 2-6. Lines as features (a) illustrates two views of a line segment whereas (b) shows an accurate estimation of the 3-D line using 3 views (modified from Faugeras (1993)).

2.4.3 Correspondence

Davis (1979) introduces shape matching using relaxation techniques as a useful approach that attempts to solve the problem of finding matches of shapes to other shapes. The shapes to be matched are usually represented by polygonal approximations (see Figure 2-7). During matching, *figures of merit* are assigned to the matches between pairs of angles on the two shapes. Relaxation methods are then used to find acceptable combinations of these matches.

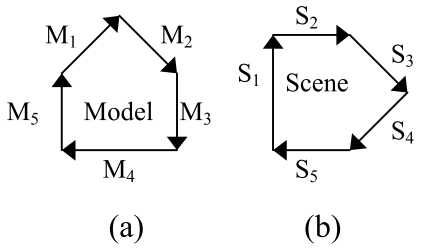


Figure 2-7. (a) The model and (b) the scene.

If structure is added to the object scene descriptions then they may be represented by graphs. The nodes of the graphs represent the primitives M_i and S_j, and there are labelled features attached to the primitives. These arcs represent relationships between the primitives, e.g. adjacency, nearness, parallelism, perpendicularity, etc. See Figure 2-8, where arcs marked *a* indicate adjacency, whereas arcs marked *p* indicate parallelism. These features attached to each segment include, for example, midpoint, length, orientation etc. Recognising the model *M* in the scene *S* is equivalent to finding that the graph representing *M* is isomorphic[7] to the graph representing *S* (i.e. ∃ a one-to-one corre-

[7] There is a one-to-one correspondence between the elements of two or more sets, and between the sums or products of the elements in one of these sets and those of the equivalent elements of the other set or sets.

spondence between nodes that preserves the graph structure). In practice, due to partial occlusion, only part of the model is present in the scene. The real problem then becomes finding the subgraph of *M* isomorphic to a subgraph of *S*. The vertices of the graphs to be manipulated are usually labelled with numerical values that are derived from the features of the corresponding geometric primitives. The values of these vertices should also be taken into account, in that the features attached to both sets of vertices should be 'close' in value. See Bhanu & Faugeras (1984) for more detail.

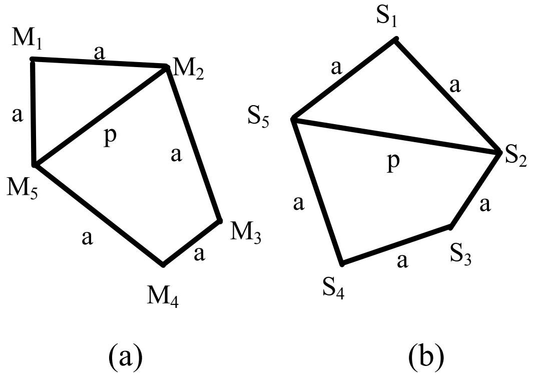


Figure 2-8. The graphs associated with the model (a) and scene (b) to be examined.

2.4.4 Constraints in Feature Matching

Due to the combinational increase in matching from a small increase in *n* (the number of frames) or *m* (the number of point matches) it is useful to introduce constraints to limit the search space. Some of these constraints include proximal uniformity, maximum velocity, small velocity change (smoothness of motion), common motion, consistent match, rigidity, etc. (Cédras and Shah, 1995). Some of the methods that have been employed to convert these constraints into formal cost-functions are as follows:

- Tracking the 3-D position of points from a stereo view of the scene based on the smoothness constraint that the position, velocity and direction of motion are virtually unchanged from one frame to the next.
- The proximal uniformity constraint states that most objects in the real world follow smooth paths and cover a relatively small distance in a small amount of time.
- Another method put forward was the use of a two-stage approach. The first stage is a forward search calculating trajectories as usual, but the second stage is rule based, moving backwards over the trajectories correcting bad correspondences.

Note that all these methods rely on the smoothness constraint, which although quite reasonable, is not however true in all cases, likely causing these algorithms to fail (Sethi & Jain (1987)). See Cox (1990) for a review of the statistical data association techniques required for feature correspondence. This information can also be successfully combined with a model based tracking approach to further increase the reliability of the matches.

2.4.5 Detection of Features of Interest

A fundamental stage in computer vision is the generation of descriptions of images, more useful than a large set of pixels. The main aim of this feature extraction is to reduce this large set of pixels into a list of features that are distinct from surrounding portions of the image so that the information available in the scene becomes more manageable. For example, in the case of feature matching, the distinctiveness of these points limits the potential matches in the following frames. Therefore, these points must more than likely correspond to significant features in the real-world scene, such as 'real' corners, 'real' boundaries or edges or significantly textured regions. Most image segmentation techniques are based on the search for local discontinuities or on the detection of regions in the image with homogeneous properties. The *trajectories* derived from locating particular feature points on an object, through time, are popular because they are relatively simple to extract.

The generation of motion trajectories from a sequence of images typically involves the detection of features in each frame, and the correspondence of such features from one frame to another. These features need to be distinct enough for detection and stable enough to be found in each frame. Features may include edges, corners, interest points, and regions (Nerriec *et al*, 1994). False features are also problematic, derived from noise, occlusion and lighting effects, the behaviour of which is usually impossible to predict. Some of these feature detection methods are now discussed.

2.5 Edge Extraction

Edges have many uses in image analysis as they characterise object boundaries, and so are useful for segmentation and identification of objects in scenes. Edge points are points in the image where there is a large change in grey level intensity within a small spatial distance, i.e. step discontinuities in the image signal. In the case of a binary image, edge points can simply be defined as black pixels with at least one surrounding white pixel. In greyscale images the method of localising these discontinuities becomes one of finding local maxima in the derivative of the image, or zero-crossings in the second derivative of the image. Edge detectors usually try to compute two properties for an edge; a *direction*, which is aligned in the direction of maximal grey-level change, and a *magnitude* describing the abruptness of the change. See Appendix A.5 for a further explanation and Jain (1989) and Ballard (1982) for a more detailed introduction.

Since edges are a high-spatial-frequency phenomenon, edge detectors are quite sensitive to high-frequency noise (such as 'snow' on a TV). After the extraction of the edge points, they should be represented by lines, as in feature matching, otherwise there will be a return to the case of too much information. The representation process is faced with difficulties in points where the edge 'breaks', or where noise will cause non-existent lines to be created. Curves are also difficult to represent with lines, as to get a perfect discrete curve the representation again returns to a pixel description. The line representation of edges also suffers from the aperture problem when dealing with local operations, such as optical flow. Common applications with relevance to motion applications are given in Kim (1990).

2.6 Corner, Junction and Feature Point Detection

The term 'corner' is quite vague, and can be defined no more accurately than: *a position in the 2-D array of brightness pixels which humans would associate with the word 'corner'* (Smith, 1992). Mathematically, corners are points where the gradient direction changes rapidly, and corresponds to the physical corners of objects, i.e. there is a strong 2-D intensity change and so is well distinguished from neighbouring points. Corners and junctions are 2-D image features, the identification of which provides important information for computer vision applications, such as stereo-vision, motion detection and scene analysis.

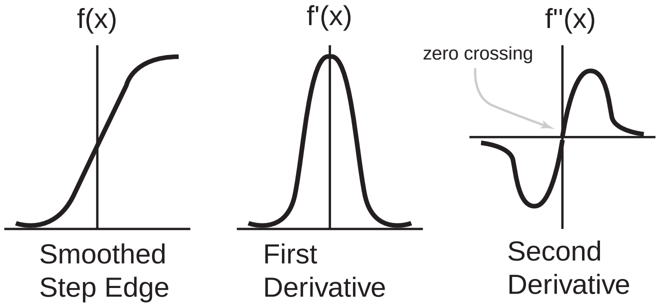


Figure 2-9. Edge Derivatives.

Historical approaches to determining image corners involved segmenting the image into regions, extracting the boundaries as a chain code and then identifying areas where directions changed rapidly. This approach is very complex and relies on region segmentation, which is in itself a complex process.

The first well-known algorithm for extracting ‘interest’ points was the Moravac algorithm (Moravec, 1977), which consists of three main stages (Noble, 1989). At a point in the image *I(i,j)*:

1. A window *W* of a specified size is displaced from each point in the four directions *{(1,0),(1,1),(0,1),(-1,1)}*.
2. In each direction a measure of change *C(dx,dy)* is computed. The window *W* is shifted by *(dx,dy)* and the following measure is computed:

   $$C(dx,dy) = \sum_{(x,y)\in W} w(x,y)\left|I(x+dx,y+dy) - I(x,y)\right|^2 \qquad \text{2-15}$$

   where the weight *w* has the value of 1 if *(x,y)* is inside the window *W* and a value of 0 if it is outside *W* and *x,y* represent a position within the window *W* with respect to the origin of that window.
3. The minimum value of *C* over these four directions is taken as the local measure of interest. A threshold is then performed over the entire image to define the points of interest.

Moravac defined points of interest as points where there is a large variation in intensity in every direction. As the variation is computed in only four directions, it is sensitive to strong edges only under certain directions and is very sensitive to noise.

2.6.1 Curvature Based Approaches

In simple scenes grey level corners of 2-D objects may be modelled as the junction point of two or more straight-line edges (step changes in intensity). For 3-D object grey level corners, this is generally more difficult. Nagel (1987) defined the grey level corner as: *the location of maximum planar curvature in the locus line of steepest grey level*, by the following reasoning:

If the intensity gradient is a maximum then,

$$\nabla\left(I_x^2 + I_y^2\right) = 0 \ , \qquad \text{2-16}$$

which can be written as (using the chain rule):

$$\begin{pmatrix} I_{xx} & I_{xy} \\ I_{yx} & I_{yy} \end{pmatrix}\begin{pmatrix} I_x \\ I_y \end{pmatrix} = 0 \qquad \text{2-17}$$

If the *x,y* axes are aligned with the principle axes then $I_{xy}=I_{yx}=0$. The second derivatives (I_{xx} and I_{yy}) are also directly proportional to the principle curvatures κ_1,κ_2, giving:

$$\begin{aligned} I_{xx}I_x &\propto \kappa_1 I_x = 0 \\ I_{yy}I_y &\propto \kappa_2 I_y = 0 \end{aligned} \qquad 2\text{-}18$$

There are four ways by which these conditions may be satisfied:

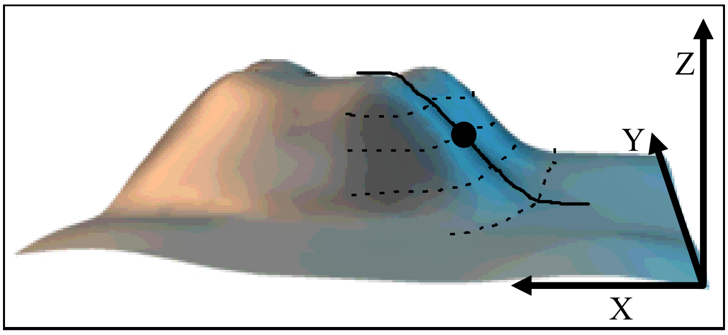


Figure 2-10. A grey level corner, defined by Nagel as the location of maximum planar curvature in the locus line of steepest grey level.

1. Both principle curvatures and thus the Gaussian curvature, are zero, $\kappa_1 = \kappa_2 = K = 0$
2. Gradient components are zero $|\nabla I| = 0$
3. One principle curvature is zero, the other crosses zero, or is non-zero.

$$\left.\begin{aligned} &\kappa_1 = 0 \\ &\kappa_2 \text{ crosses zero } \Rightarrow I_y = \max \\ &\kappa_2 \neq 0 \Rightarrow I_y = 0 \end{aligned}\right\}$$

4. The most interesting case, where one principle curvature crosses zero and the other is non-zero

$$\left.\begin{aligned} &\kappa_1 \text{ crosses zero } \Rightarrow I_x = \max \\ &\kappa_2 \neq 0 \Rightarrow I_y = 0 \end{aligned}\right\}$$

Thus the grey level corner is defined by:

$$\left.\begin{aligned} &|I_y| = \max \neq 0 \\ &I_{yy} = 0 \\ &I_x = 0 \\ &|I_{xx}| = \max \neq 0 \end{aligned}\right\}$$

This is shown as in Figure 2-10.

One problem with this model of corners is that the assumption that corners appear at a maximum of the gradient 2-16, implying that low gradient but highly curved corners will be overlooked. This is likely to be a common corner in real-world images. Nagel demonstrated that this definition of a corner could be used as a feature point in motion estimation, and claims that a grey level corner provides a link between the continuous approach and the feature based approach to motion analysis.

Corners may also be detected by locating the points of high curvature along the edge or boundary contours. There are two stages to this approach:

1. Edge detection and grouping.
2. Curve representation.

In this approach, directional and non-directional operators will effect the detection of curves and corners. Directional operators perform better at localising edges and sharp corners, as non-directional operators such as the Laplacian introduce a rounding effect.

Noble (1989) derives an equation for the total curvature for a Laplacian edge as,

$$\kappa_T = (\kappa_T)_{lap} = \frac{(\nabla I)^2}{g^3} \frac{\partial^2 I}{\partial n_\perp{}^2}, \qquad 2\text{-}19$$

where $g^2 = 1 + I_x^2 + I_y^2$. In general, for large gradients $g^2 \cong |\nabla I|^2$, from this she shows that many ‘supposed’ individual corner detectors, with individual mathematical derivations, all break down to have the general form of:

$$\kappa_T \cong \frac{1}{|\nabla I|} \frac{\partial^2 I}{\partial n_\perp^2} \qquad 2\text{-}20$$

Showing that most corner detectors are based on second order derivative schemes computing the strength of the corner based on the product between the rate of change of gradient direction along an edge and the gradient magnitude.

Another method for corner detection is based on a ‘cornerness’ approach. This method involves using non-linear second derivative operators, instead of the Laplacian, since the Laplacian method incorrectly often obtains corners at other features, such as edges, and is responsive to noise. The determinant of the Hessian matrix is a simple operator ($C = I_{xx} I_{yy} - I_{xy}^2$) and gives a high response in regions of image curvature. Other methods involve taking the ‘cornerness’ measure as the change of gradient direction along an edge contour multiplied by the local gradient magnitude, i.e. requiring an edge and a corner. The expression for corner strength is given as (Nobel, 1989):

$$C = \frac{I_{xx}I_y^2 - 2I_{xy}I_xI_y + I_{yy}I_x^2}{I_x^2 + I_y^2} \qquad 2\text{-}21$$

The use of second order derivatives has however the unfortunate property of amplifying image noise. This approach is also not very suitable for handling other types of junctions, such as T, X, or Y junctions.

The Harris algorithm (or Plessey algorithm) (Harris, 1987) is based on the Moravac interest operator. Importantly, it uses first order derivatives to approximate the second derivatives. The algorithm consists of three main stages:

- Compute the partial derivatives of the image I_x and I_y and compute the local Gaussian smoothed products $\langle I_x^2 \rangle, \langle I_y^2 \rangle$, and $\langle I_xI_y \rangle$.
- Compute the eigenvalues λ$_1$, λ$_2$ of the matrix a 2x2 matrix (called the A matrix) where $\mathbf{A} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_xI_y \rangle \\ \langle I_xI_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$
- Mark points as corners where the product λ$_1$λ$_2$ divided by (λ$_1$ + λ$_2$) is greater than a threshold (as below where *k* is usually 25), i.e. a corner exists when both eigenvalues are large.

A corner response function (CRF) is formulated where

$$R = \frac{k}{(k+1)^2}\left(\det \mathbf{A} - (Trace\mathbf{A})^2\right) = R = \frac{k}{(k+1)^2}\left(\lambda_1\lambda_2 - (\lambda_1 + \lambda_2)^2\right)$$

so, $$R = \frac{k}{(k+1)^2}\left(\left(\langle I_x^2 \rangle\langle I_y^2 \rangle - \langle I_xI_y \rangle^2\right) - \left(\langle I_x^2 \rangle + \langle I_y^2 \rangle\right)^2\right) \qquad 2\text{-}22$$

If both eigenvalues are large (i.e. *R* is large) the pixel is flagged as a corner. The algorithm is reasonably computationally expensive, as the Gaussian is applied three times. The Harris algorithm is used in the DROID system (to be discussed later in Section 2.9), however, Wang and Brady (1995) claim that the detector is not well localised and Smith (1992) finds dislocation at T-Junctions. Nobel (1989) claims that it is only useful for L-junctions.

Wang and Brady (1995) suggest a fast algorithm for corner detection that is based on the observation that curvature is proportional to the second order directional derivative in the direction tangential to the edge normal and inversely proportional to the edge strength. They state that for points with a strong gradient the total curvature may be approximated as:

$$\kappa = \frac{1}{|\nabla I|}\frac{\partial^2 I}{\partial \mathbf{t}^2}, \quad |\nabla I|^2 >> 0 \qquad \text{2-23}$$

Where $\partial^2 I/\partial t^2$ is a directional derivative along the unit tangent vector **t**, perpendicular to the edge normal **n**, computed using a linear approximation for the pixel values along the line ($\partial^2 I/\partial t^2 = \left(I_y^2 I_{xx} - 2I_x I_y I_{xy} + I_x^2 I_{yy}\right)/|\nabla I|^2$). Equation 2-23, has difficulty in dealing with image quantisation and noise effects. A Gaussian convolution could be added to reduce this effect but more importantly they show that Gaussian convolution causes linear signal displacement along the edge normal. Equation 2-23 assumes that $|\nabla I|^2 >> 0$, which means that the algorithm is only interested in corners that lie near pixels where the image gradient magnitude is high, causing a reduction in the amount of computation and reducing the effect of false corners. *F* denotes the grey-level image *I* after Gaussian Smoothing Convolution (σ = 0.5) to reduce the effect of noise and quantisation. Corners are points where κ is high and a local maximum. Wang and Brady (1995) define the corner operator as points where the following conditions are fulfilled (derived by assuming $\kappa^2 > S$):

$$\begin{cases} \Gamma = \left(\dfrac{\partial^2 F}{\partial \mathbf{t}^2}\right)^2 - S|\nabla F|^2 \ \text{ is a maximum} \\ \dfrac{\partial^2 F}{\partial \mathbf{n}^2} = 0 \\ |\nabla F|^2 > T_1, \Gamma > T_2 \end{cases} \qquad \text{2-24}$$

Γ is a Corner Response Function (CRF) where *S* is a user-defined threshold measure of image curvature and T_1 and T_2 are user-defined thresholds on edge and corner strengths that depend on the context of the image. The first condition of Equation 2-24 explains the main operation: The first term of the right hand side is the squared second order tangential derivative, at which corners as well as edges will respond well in the false corner case. The second term is the edge strength that responds well at edge pixels (including false corners). The difference of these two terms cancels the false corner response and leaves a clean corner response.

The Curvature Scale Space (CSS) corner detector is based on the Canny edge detector and was suggested by Mokhtarian *et al* (1998). From the resulting Canny edge map, gaps are filled and 'T-junctions' are determined and marked, using a local rule-based approach. The curvature is computed at the highest scale and corner candidates are determined by comparing the maximum of the curvature to a defined threshold and the neighbouring minima (i.e. the curvature of a corner should be twice that of one of the neighbouring local minima). Corners are then tracked to the lowest scale to improve the localisation of the corner using a simple curvature comparison and the knowledge that the corners will not move dramatically.

Some other techniques to determine the corners and junction points in an image include template matching, which involves a large set of templates where the search is performed using an adapted Hough transform. This is a very computationally intensive method on which not very much work has been developed. Fitting B-splines (to be discussed in Section 3.5.3) to corners has also been mentioned as a possible approach to estimate edge curvature after a suitable edge detector has been used, where corners are identified as the points of local maxima of the curvature of the spline. This approach is not suitable for junctions or fine detail and the computational overhead of an erroneous edge segmentation algorithm is also required. Nobel (1989) developed a detector based on morphological curvature. It has a high computational load and has problems with tails appearing at 'T' junctions. Other morphological techniques are discussed in Soille (1999).

Many techniques do not tackle the problem of corner orientation. A common solution is to calculate the mean gradient orientation over a small neighbourhood area, which is only accurate to 20° and not good at handling junctions, according to Chabat *et al* (1999). Chabat *et al* (1999) make the assumptions that: 1) like all edge points, corners and junctions have a strong intensity gradient and 2) unlike straight lines, corners and junctions do not have a single dominant orientation. They use a measure of anistropy (uni-directionality) at each point in the image, of which the orientation is defined by:

$$\theta(x,y) = \frac{1}{2}\tan^{-1}\frac{\int\int_{\Omega} 2I_x I_y dxdy}{\int\int_{\Omega}\left(I_x^2 - I_y^2\right)dxdy} + \frac{\pi}{2} \qquad 2\text{-}25$$

where Ω is a small neighbourhood of the point *(x,y)*, the strength of which is defined as:

$$g(x,y)=\frac{\left(\int\int_{\Omega}\left(I_x^2-I_y^2\right)dxdy\right)^2+\left(\int\int_{\Omega}2I_xI_ydxdy\right)^2}{\left(\int\int_{\Omega}\left(I_x^2-I_y^2\right)dxdy\right)^2} \qquad 2\text{-}26$$

Notice that only single derivatives are used and so this method is less sensitive to noise than many other methods. The value of $g(x,y)\in[0,1]$, where 1 is a strong orientation pattern (straight edges) and is 0 for isotropic regions (corners). Chabat *et al* (1999) define a function $s_{c(j)}(x,y)$, calculated at each point M_i in a small neighbourhood Γ around a labelled corner point C_j that signifies the likelihood of a pixel being part of the edges associated with that corner:

$$s_{c(j)}(x,y)=g(x,y)\left\|\nabla I(x,y)\right\|\cos^3(\alpha) \qquad 2\text{-}27$$

and α = (**u**,**v**) where **u** is the direction of uni-directionality at M_i, derived from θ in Equation 2-25 and **v** is defined as: $\mathbf{v}=\overline{C_jM_i}$ (the direction is compatible with the direction of the point M_i to the corner C_j). For each corner, the value of $s_{c(j)}(x,y)$ is calculated and a histogram is constructed, the analysis of which (the local maxima) gives the dominant direction of the edges associated with the corner, where 2 peaks define a corner and 3 peaks define a junction. Further information on the analysis of the histogram is available in Chabat *et al* (1999).

Matas & Kittler (1992) suggest a novel approach to junction and corner detection. First of all, they attempt to recover the projected 3-D lines. Using the property that these 3-D projected lines are, by definition, the projection of 3-D edges, and therefore must terminate at the intersection of two projected lines. They then use a Direche filter (Deriche, 1987) for edge detection and a line detector based on the Hough transform. Junction detection is then performed using probabilistic relaxation labelling, which in their case works quite successfully. They then stretch line segments to junctions, also performing gap bridging post-processing. They achieve excellent results, but their test results are based on images with purely quadrilateral models, e.g. blocks, the gable end of a house etc. (with the exception of *any* circles, spheres, and cylinders). Many of the methods examined here (with the exception of Wang and Brady (1995)) are not useful for motion analysis. They are inadequate, as they do not compute the curvature in the images reliably and are quite sensitive to noise.

2.6.2 The SUSAN Corner Detector

Noble (1989) showed that many different corner detection algorithms had unique definitions of a ‘corner’ and unique mathematical approaches, but derive very similar mathematical conclusions. The ‘Smallest Univalue Segment Assimilating Nucleus’ (SUSAN) by Smith (1992) is a form of corner detector that is quite unique.

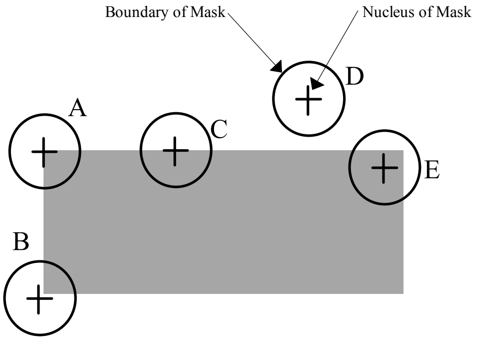


Figure 2-12. SUSAN sample data case (derived from Smith, 1992).

The principle behind SUSAN is as follows: It begins by taking a small mask, usually a small circular mask and sequentially moves its centre across every pixel in the image set and examines the local information at each step. The intensity of the pixel at the centre of the nucleus of the mask is determined and all pixels surrounding this point with similar intensity values (within a threshold *t*) are then grouped together and termed a ‘Univalue Segment Assimilating Nucleus’ (USAN). Figure 2-12, displays some sample cases for the SUSAN corner detector algorithm and in Table 2-1 the USANs are displayed (always represented in white). The Smallest-USAN is the point where the corner exists. One of the main advantages of this algorithm is that since there are no brightness spatial derivatives, the ‘snow’ noise in the image is not amplified. Many other corner detection methods have attempted to reduce the noise in the image by smoothing the image before performing the detection algorithm, but this leads to local distortion and a loss of local information (Smith, 1995).

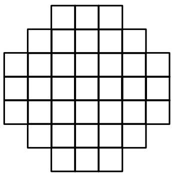


Figure 2-11. Approximate circular mask used by SUSAN.

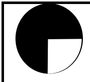
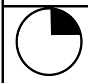
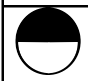
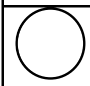
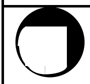
Mask	USAN diagrams	Detect	Explanation
A		☑	In this case the USAN (in white) has a small area, less than half the mask area, thus a corner is correctly detected here, at the pixel in the centre of the nucleus. This small area is usually taken as 2/5 of the mask area.
B		☒	In this case the USAN has a large area, and so does not meet the conditions. No corner detected.
C		☒	The USAN in this case has an area of exactly half that of the mask, but this does not detect a corner at the nucleus of the mask. No corner detected.
D		☒	Clearly the USAN area is too large. No corner detected.
E		☒	In this case again the USAN is too large, and so a corner is correctly not detected at the Nucleus of the mask. No corner detected

Table 2-1. The USAN (Univalue Segment Assimilating Nucleus) for the corners presented in Figure 2-12.

The SUSAN algorithm usually uses a 5 x 5 mask, with 3 pixels added to each edge, giving a reasonable approximation to a circle (see Figure 2-11). In the creation of the USAN a brightness threshold (*t*) is used to link pixels to the USAN area. For example, if all the pixels currently under the mask are within 15 grey levels of the centre pixel and the brightness threshold is 16 then all those pixels will be linked to the USAN. A rigorous mathematical evaluation of this algorithm is not really possible, as it is more 'intuitively picturesque' than analytic, according to Smith (1992). This entire process is quite computationally efficient, taking approximately one tenth of a second using a 256x256 pixel image[8]. This computation duration varies according to the number of likely corner features, as a significant part of the computation is involved in further processing the possible corners. In this calculation ~300 corners were eventually extracted.

In simulated scenes the above description would suffice and also work quite well, however when dealing with real-world images the algorithm invokes a noise filter and the

[8] Calculated experimentally during this research using the GNU 'C' implementation of the regular SUSAN algorithm on a SUN Sparc II (with the dual processor not used) 266MHz. Result will vary depending on the number of corners determined.

algorithm evolves to: Compare the brightness of each pixel in the mask to the central pixel using the comparison:

$$c(\vec{r},\vec{r}_0) = 100e^{-\left(\frac{I(\vec{r})-I(\vec{r}_0)}{t}\right)^6}, \qquad \text{2-28}$$

where: $\vec{r}_0$ is the position of the nucleus in the 2-D image,

$\vec{r}$ is the position of any other point within the mask,

$I(\vec{r})$ is the brightness of any other point within the mask,

t is the brightness difference threshold and

c is the output of the comparison.

According to Smith (1992) this equation allows a pixel's brightness to vary slightly without having a large effect on *c*. For speed purposes it was possible to implement a lookup table on this comparison, i.e. $I(\vec{r}) - I(\vec{r}_0)$ can only produce a value of –255 to +255 in a greyscale image of 256 levels (since *t* has only one value (usually 20-25), determined at run-time by the user). This lookup table adds greatly to the computational speed of the algorithm, as 512 values need only be computed instead of 65535 operations in the case of a 256x256 image of 256 greyscales. For each pixel in the mask (37 pixels) as described in Figure 2-11 a running total is computed of the form:

$$n = \sum_{\vec{r}} c(\vec{r},\vec{r}_0) \qquad \text{2-29}$$

n is then compared with a fixed geometric threshold *g*. As the maximum value of this computation is (37 pixels in mask x 100 (i.e. 100 is the max value of c)) = 3700. Smith chooses

3700 / 2 = 1850 as the threshold value.

The brightness threshold *t* does not have the same effect as this threshold value. The brightness threshold effects only the contrast allowed in the detected corners and so affects the number of corners detected in the image. The threshold *g* affects the shape of the corners detected; by lowering this value it in effect forces sharper corners. See Figure 2-13.

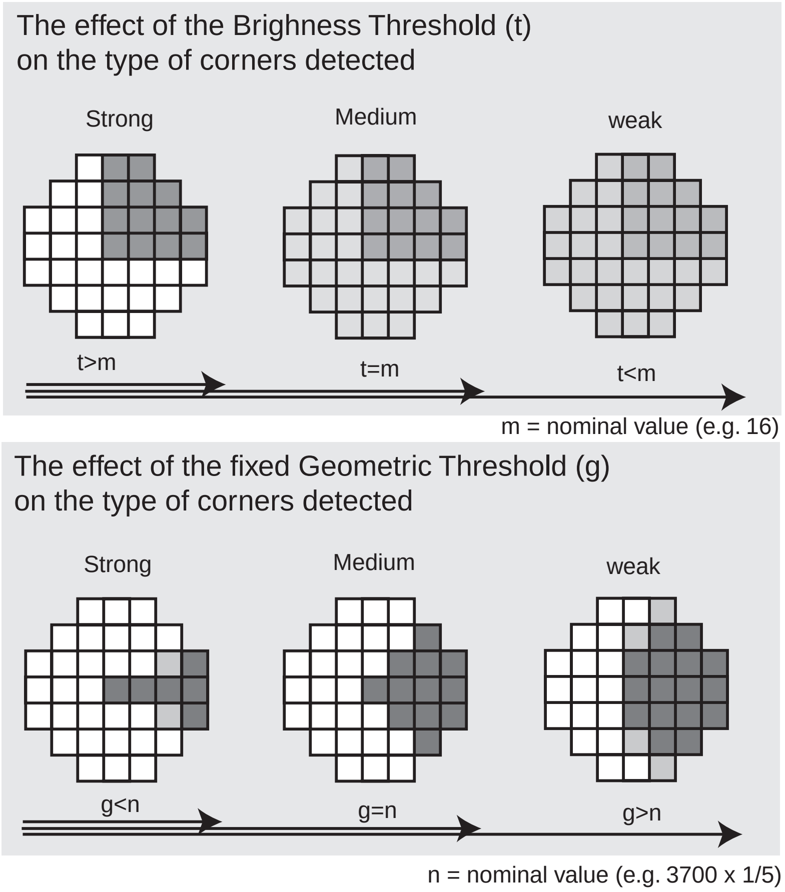

The effect of the Brighness Threshold (t)
on the type of corners detected
Strong
Medium
weak
t>m
t=m
t<m
m = nominal value (e.g. 16)
The effect of the fixed Geometric Threshold (g)
on the type of corners detected
Strong
Medium
weak
g<n
g=n
g>n
n = nominal value (e.g. 3700 x 1/5)

Figure 2-13. Effects the type of corner detected by changing the brightness threshold (*t*) and the geometric threshold (*g*)

The next step is to remove any false positives that might occur. For example, Figure 2-14, shows an example mask, in which the area of the USAN is indeed less than 2/5 of the area of the mask, however, it is not appropriate to classify it as a corner. To combat this problem a relatively simple solution is to calculate the centre of gravity of the region within the mask. In Figure 2-14, the centre of gravity is very close to the centre of the mask. This however is not the case when a corner is detected. This simple addition to the algorithm need only be calculated when the threshold is reached and so does not add significant computation to the overall algorithm. There is one further step to reduce the case of false positives. All the points between the centre of the mask and the centre of gravity must belong to the USAN for a corner to be detected. This insures a degree of uniformity in the USAN and thus in the corner being detected.

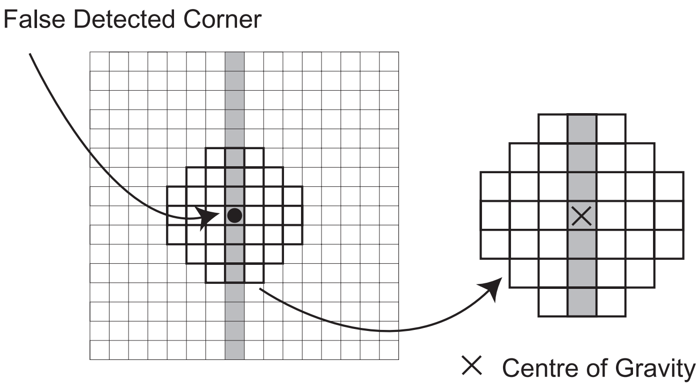


Figure 2-14. A possible falsely detected corner using the SUSAN algorithm and mask, prior to the calculation of the Centre of Gravity.

An independent review of the SUSAN algorithm by Trajković *et al* (1998) has estimated the complexity of the algorithm at a total cost of 48.3 additions and 0.8 multiplications per pixel (based on a 25% transfer from stage one to stage two processing). Trajković *et al* suggest a new approach that uses a similar algorithm structure to SUSAN. Instead of calculating the areas of the USAN masks (involving a large number of operations), this approach calculates the intensity differences between points on the circumference of the mask and the nucleus of the mask, to provide information about the presence or absence of a corner.

The aim of this approach is to be computationally efficient and so uses a multi-stage approach. At the first stage a computation of the Curvature Response Function (CRF) is performed of the form:

$$R_N = \min\left(\left(f_P - f_C\right)^2 + \left(f_{P'} - f_C\right)^2\right), \qquad 2\text{-}30$$

where f_C refers to the image intensity at the centre point, f_P refers to the image intensity at the point P, and this is calculated for different lines. Referring to the three cases in Figure 2-16, in Case (a) there is at least one line l where the points P and P' both belong to the USAN so the min value of R_N is low. In case (b) there is a single case where the line l is parallel to the edge of the USAN where the value of R_N is very low. In case (c) the value of the CRF R_N is high as there are no points P and P' along the line l where they can both belong to the USAN.

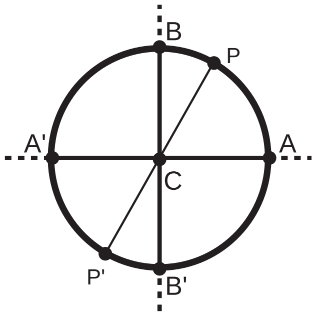


Figure 2-15. The neighbourhood of the Nucleus and the linear positions used. The points *P* and *P'* are on an arbitrary line *l*, that may be rotated around the mask. Note that the mask used is of the same form as the SUSAN algorithm 5 x 5 with 3 added to the edges in the largest case.

In practice however this approach has to deal with problems due to strong edges, with directions different than in Figure 2-16. The horizontal and vertical intensity variations are first calculated using:

$$r_A = (f_A - f_C)^2 + (f_{A'} - f_C)^2 \qquad \text{2-31}$$

$$r_B = (f_B - f_C)^2 + (f_{B'} - f_C)^2 \qquad \text{2-32}$$

Then the CRF is calculated as:

$$R = \min(r_A, r_B) \qquad \text{2-33}$$

If the value of *R* is less than a certain threshold then the current point is not a corner and no further computation is performed. If *R* is greater than that threshold then an inter-pixel approximation is applied to check for diagonal edges, and the CRF is derived as:

$$R = r_A - \frac{B^2}{A} \qquad \text{2-34}$$

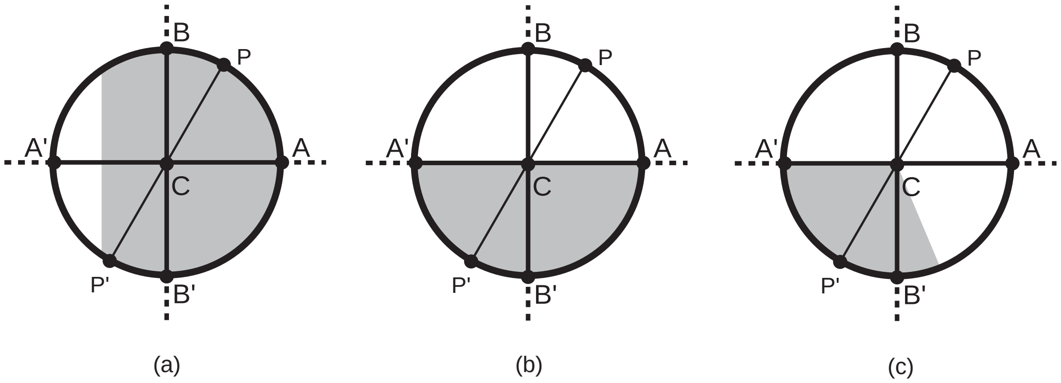


Figure 2-16. Some example cases, representative of the USAN of the mask.

The value of $A = r_B - r_A - 2B$ and *B* is defined as *B* = min (B_1, B_2) where,

$$B_1 = (f_B - f_A)(f_A - f_C) + (f_{B'} - f_{A'})(f_{A'} - f_C),$$

$$B_2 = (f_B - f_{A'})(f_{A'} - f_C) + (f_{B'} - f_A)(f_A - f_C),$$

provided that $B<0$ and $A+B > 0$. A circular inter-pixel approximation is also available. This value R from Equation 2-34 is calculated for every pixel that conforms to the initial comparisons. The last stage is to find the pixels with the locally maximal CRF and mark them as corners, noting that several pixels around the vicinity of a corner will have a high value of CRF. This process is called non-maximum suppression and is also carried out at the end of the SUSAN algorithm.

2.6.3 Discussion on Feature Detectors

To be useful for feature matching in motion tracking applications a feature detector should satisfy the conditions of:

- Accuracy: Features should be detected as close as possible to the true image plane position
- Stability: The features should be detected consistently and should not move when multiple image frames of the same scene are captured (insensitive to variation of noise)
- Speed: For applications where real-time implementation is required, the detector must be very efficient.

An examination of published literature on corner detectors has allowed the determination of the results of Table 2-2.

Algorithm	**Accuracy**	**Stability**	**Speed**[9]
Trajković (MIC)	Good	Good	Very Fast
Harris (Plessey)	Good for L junctions, not for other types *Poor localisation according to Wang.*	Very Good	Slow
Smith (SUSAN)	Good (more accurate with pre-filtering)	Good	Fast
Wang and Brady	Good	Good	Fast
Mokhtarian (CSS)	Good	V.Good	Reasonable

Table 2-2. A general comparison between different corner detectors.

The SUSAN algorithm was chosen as feature detector for the primary implementation in this research as it performs well, localising consistently features in image sequence

[9] The Smith (1992) SUSAN and Wang and Brady (1995) corner detectors have been used in real-time applications (see Section 2.9). The Trajković (1998) MIC algorithm is theoretically more computationally efficient, however has not been used in any real-time applications to date.

frames. The computational speed aspect of the algorithm is also examined and improved later by the author in Section 4.10.

2.7 Model Based Tracking

Model-based tracking uses prior knowledge of the structure and appearance of particular objects before the interpretation of a scene. The aim is to try to correspond these predefined models to actual objects in a real-world scene. In general, these models consist of precise 3-D geometrical representations of known objects (commonly vehicles) that may be rotated and examined in any position or direction. Often very carefully calibrated camera models are used. Using the known camera and scene geometry, it is possible to project this 3-D model onto a 2-D image plane, and a confidence of match may be determined by comparing the features of the translated model to the physical image to be examined.

In the case of Worrell *et al* (1991), they apply a 'goodness-of-fit' value to the score obtained in matching the physical image to the translated model image. At each position and orientation, they evaluate this goodness-of-fit value, using a search in position space and orientation-space to speed up the evaluation of this score. Once a maximum value is found and claimed to be successful, it is used to estimate the position and orientation for the same object in the next frame. Thus, if they obtain the initial position of a specific object to track in the first frame, then it may be possible to track this object easily through subsequent frames, calculating the position and orientation of this object.

They have applied this technique to tracking a vehicle through a sequence of images, breaking the process into stages:

- *'Goodness-of-fit':* in this stage the model is translated and rotated to the appropriate position in the real world and each line of the model is projected onto the image. These visible line features are then evaluated in the Gaussian smoothed image using a process called iconic evaluation (as introduced by Brisdon *et al* (1988)), and the scores from all the lines are aggregated to give an overall score for the model in the given position.
- *Pose recovery:* Here they determine the pose of the vehicle, by locating the peaks that indicate likely matches in the six-degree of freedom function that they use.

They minimise the computation by assuming that the object is on the ground plane, with two dimensions of translation and one degree of rotation.

- *Separated gradient ascent:* this iterative algorithm is applied to find the peaks in the evaluation function, where each iteration consists of three 1-D searches taking place a number of samples either side of the initial position.

Large problems result due to occlusion, where models can drift and be applied to objects other than the vehicle in question. In their case, they showed an unrecoverable error while tracking where the model of the vehicle became trapped on a roadside advertisement.

2.8 Discussion

In Section 2.3.5 the various optical flow techniques were examined and it was found that for many problems gradient-based methods offer greater performance than matching techniques. Matching techniques are sensitive to ambiguity between the features to be matched, but optical flow may only be calculated accurately at distinct feature points in the image, and so can provide a very sparse number of points in the image. Although it is reasonable not to desire a large number of points for computational reasons, insufficient points can lead to inaccurate results. Gradient-based approaches avoid the sometimes-difficult task of finding distinguishable points or regions. The differential approaches (all except local), have been found to be quite inaccurate in results for extracting motion information. As discussed, according to the literature the local approach and the energy and phase approaches have been found to be the most accurate flow techniques. This is important especially at this stage, as errors will be further multiplied when segmentation is attempted.

The use of feature matching techniques using line segments as features has two main advantages over other matching techniques:

- The number of primitives to be matched is usually much smaller, than for example the number of edge pixels.
- In matching symbolic primitives attached properties may be used, such as geometric properties that are robust and reliable.

And two main disadvantages:

- It assumes that objects have a polyhedral structure. If not, then the curves introduce extra lines, which introduce errors, such as lines flickering (appearing and disappearing).
- The density of correspondences is less than with other methods, but however these correspondences are generally more accurate.

The number of views required for the line-based method is undesirable over that of the point-based methods.

Unfortunately all matching methods are computationally expensive. While some efficiency is possible, much effort is placed in the avoidance of false matches and the matching of points densely enough to be useful. The *'coarse to fine'* strategy is quite often used, where initial accurate matches are made, then finer resolution matching are performed by other matches, using the information gained by the accurate matches. In the case of the unconstrained motion sequences to be examined in this work, it is not feasible to use a traditional model based tracking method. Objects to be examined include humans, and unknown obstacles that must be avoided. In the case of human 'objects', the models required would have to be deformable and a set of models to allow all possible shapes of the human body would be very difficult, using standard model based techniques. This is examined to some extent in Section 3.5.1 on the Leeds people tracker, using active shape models (Heap and Hogg, 1996).

A basic assumption that is often made in computing optical flow is that the image changes enough from frame to frame so as the motion can be tracked, but not so much that the correspondence cannot be realised. Edge and corner feature points can be computed rapidly and reliably in images and they are in general stable over image sequences. As discussed earlier, corner points also provide more constraint than edge points.

2.8.1 Correspondence of Points of Interest

The task of maintaining a correspondence of points of interest between multiple frames has proven difficult. This task is made more difficult by features being occluded, false features appearing and current features disappearing.

There are two different ways to match these features. One way would be to extract a set of features from the first frame and try to find the same features in subsequent 2nd, 3rd, 4th frames etc. The second way is to extract features with certain attributes from the first frame and then match these in the second frame, similarly for the 2nd and 3rd.

There are several image motion assumptions that are commonly made:

- Maximum Velocity – Assume that a point in one frame of an image sequence must lie within a certain distance of that point in the next frame. This assumes that the point has a maximum velocity.
- Common Movement – Points on the same object move in a common way.
- Rigid Motion – Points on a rigid object are related to each other from one frame to the next provided there is no occlusion or disocclusion.

Tomasi and Kamade (1992) track corner features over many frames using a nearest neighbour strategy that assumes the frame-to-frame motion is very small. This method worked well; however, if significant frame-to-frame motion occurs then ambiguities quickly arise. Partial occlusion and significant changes in the background of the frame can be problematic for these methods (Cox and Hingorani, 1996). These methods are suitable for tracking existing features and are not suitable for determining new trackable features on a frame-by-frame basis. Cox and Hingorani (1996) develop an efficient approximation implementation of Reid's multiple hypothesis-tracing (MHT) algorithms for tracking features over long image sequences. The MHT is computationally exponential ($O(N^4)$) in both time and memory, so they implement an approximation to the MHT. They achieve impressive results, however their algorithm still involves a high level of computational complexity.

2.9 ASSET-2 and Other Navigation Implementations

There have been many attempts in recent years at implementing an autonomous vehicle navigation system. Many of these implementations require structured environments, such as near 'perfect' road edges, leading to problems in the navigation of narrow, cluttered country roads.

NavLab, developed by Thorpe *et al* (1988), is a self-contained mobile platform for use in outdoor environments. It was used primarily for integrating perception and navigation capabilities, with colour vision and 3-D vision capabilities. The main algorithm involved the use of edge and region detection algorithms, where the edges were extracted and fitted to a road shape model. The region information was extracted and grouped according to colour and texture information, with road pixels being grouped together and off-road pixels being grouped together. Unfortunately, the road-extraction problem is a difficult one, with the shadow edges often being the strongest edges, and the shadowed area of the road having different colour intensity values than the road itself. The initial resolution of the captured image is 480 x 512 pixels, but they reduce this to 30 x 32 subregions by averaging, to improve computation and to smooth the regions. The developed algorithm searches for the vanishing point of the road using the edge information (assuming that the road edges are parallel) and the colour and texture information using adaptive colour classification to classify the image points as either ‘road’ or ‘non-road’ values. The vanishing point of the road is calculated as the point where the calculated road edge lines intersect in the ground plane, providing the road’s direction relative to the observer. In the calculation of the texture image the gradient is normalised by dividing a high-resolution gradient of the image by the mean pixel value of the image subregion and a low-resolution gradient of the image. The division by the mean pixel value is performed to handle shadow interiors and the low-resolution gradient removes shadow boundaries. The calculated sub-region colour and texture values are compared to pre-defined classes determined from a training set of 50 images, to provide a classification as ‘road’ or ‘off-road’ and also to provide a confidence value of this label.

The algorithm functions well, using a laser range finder to calculate the 3-D information required. By using this range information and grouping pixels that are expected to belong to the same surface the algorithm attempts obstacle detection. The speed of the vehicle was limited to 10 cm/sec, due mainly to the computational requirements of the algorithm. No motion information cues were calculated, such as the optical flow information and the algorithm functioned, however, it is unclear as to how it would deal with other vehicles, occlusions due to leaves, snow, wet and dry, sun flashes (dramatic changes in illumination on a frame-by-frame basis) and other environmental factors that change the perceived texture and colour of the road and road edges.

Dickmanns and Graefe (1991) also develop an algorithm for their mobile test vehicle, VaMoRs and manage to drive the vehicle at speeds of up to 96 km/hr for lengths of more than 20km. Again however, the algorithm is limited to well structured motorways, with small radii of curvature, determining navigation using the edges and usually lane markings.

With regard to the more complex tasks such as obstacle avoidance and badly defined country roads the guidance technique must be capable of calculating 3-D structure and motion information. The DROID Image Processing System[10] provides real-time 3-D scene measurements at localised areas within the image. It uses 2-D features over time, using a Kalman filter for the 3-D tracking of each point (Charnley and Blissett, 1989). The search regions around each pixel in this application were of the order of 3 or 4 pixels and this greatly simplified the matching process. However, the large number of features used (typically 300 - 500 for a 256 x 256 image) allowed an accurate reconstruction of the scene using their structure from motion technique.

The ASSET-2[11] (A Scene Segmenter Establishing Tracking, Version 2) tracker implements a more complex algorithm that avoids the need for advanced road texture and colour models. Motion information is calculated using feature-based image motion estimation. The features that are used for the implementation are 2-D features (corners), with edges often used to refine the results. Feature matching is performed on the image plane, with the corners calculated using either the SUSAN or Harris corner detectors (see Section 2.6.1). The Harris detector was used along with the SUSAN detector, as a hardware implementation of the detector was available to the research group at the time.

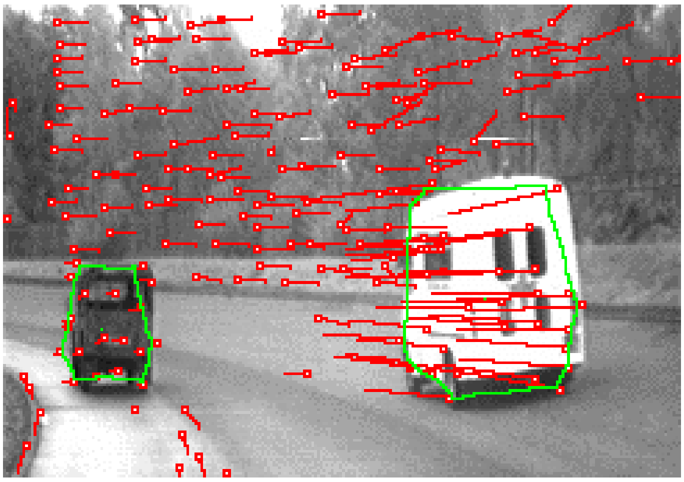


Figure 2-17. The ASSET-2 Tracking algorithm in an unconstrained image sequence. (Image is from the Smith ASSET-2 report (Smith, 1995)).

[10] Developed at Roke Manor Research Ltd.

[11] Developed largely at Oxford University and Defence Research Agency, Farnborough, Hampshire.

Smith (1995) uses a list of cluster groups of flow vectors, which have been segmented according to the similarity of the flow vectors. The boundaries of these clusters are further refined to use image edges if they are available (i.e. edges could be derived when tracking a mobile vehicle, but maybe not when examining general background information). The segmentation algorithm is based on first-order flow fields, where the distance function is defined as:

$$D = \frac{|\vec{u} - \vec{u}_m|}{\frac{|\vec{u}| + |\vec{u}_m|}{2} + \sigma} \qquad \text{2-35}$$

This compares the candidate flow vector $\vec{u}$ with the estimate vector $\vec{u}_m$ from the constant flow model. This distance function allows small vectors to be matched to each other without bias by including a noise term into the fractional vector error. σ is of the order of the error estimate of the flow vector derived from the flow model (as described in Smith (1995)). The feature tracking is performed using simple 2-D motion models, where either constant velocity or constant acceleration models are assumed. Smith claims from implementation experience that there are very few benefits in using the more complex constant acceleration model. A threshold D_{max} is used to decide if the new flow vector should be added to the current cluster. Each flow vector is initially placed in a list, once it is added to a cluster it is removed from that list, so that it does not get added to more than one cluster (Smith, 1995). Once the list of independent clusters is established, properties are calculated for the cluster, such as bounding box, centroid and motion model. The ASSET-2 system then tracks these clusters over time using a radial map representation of the cluster shape to update information about the interframe differences and to attempt to detect occlusion.

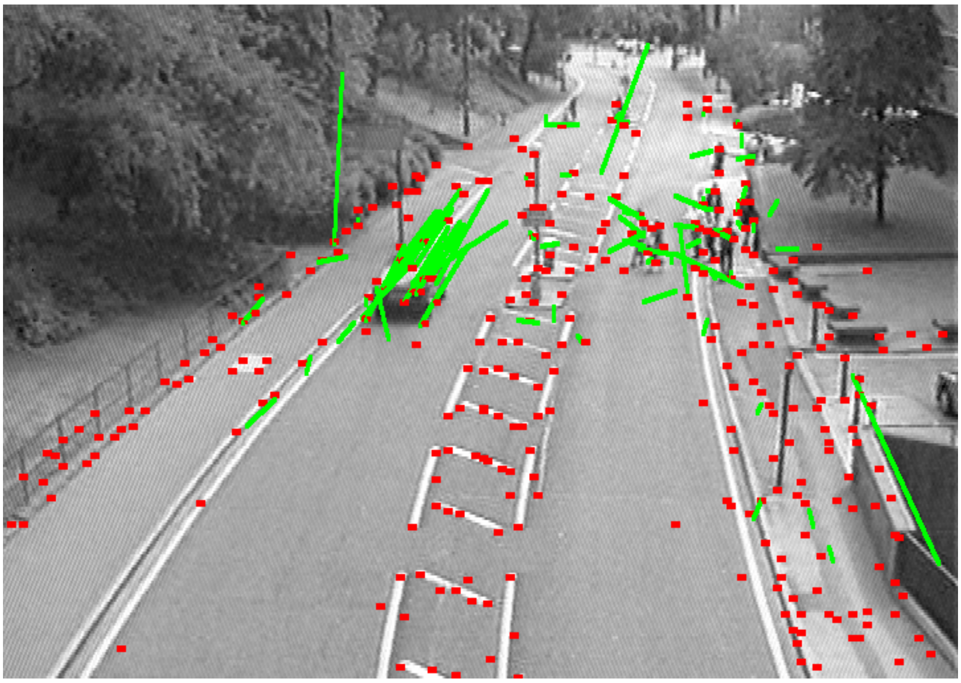


Figure 2-18. The ASSET-2 tracking algorithm in a fixed camera scene, where the ground plane has been identified by hand and masks only the roads.

The application implementation is of a vehicle travelling along a road taking video pictures of what is in front of it. In the case of ASSET-2 implementation moving vehicles were required to be extracted (or segmented) from the background, so that navigation could be performed safely on busy roads. The list of spatially and temporally significant clusters is used to provide information about the motion of the objects being viewed by the camera. The 3-D positions and motions of the objects are estimated by making assumptions about the real world (see Sinclair *et al* (1992)). The complete system runs in real-time, requiring no calibration and no knowledge of the camera's motion.

The algorithm used in the ASSET-2 method was extended to the ALTRUISM (Automatic Local Three-Dimensional Representation Using Image-based Scene Measurements) (Smith, 1995b). The ALTRUISM approach attempts to use the information from the ASSET-2 implementation and the 3-D structure measurements to provide a useful level of understanding about the scene, in particular the location of areas where it is safe to drive an autonomous vehicle. The implementation included a large array of hardware image processing devices connected to the Charnley and Blissett (1989) DROID system (as discussed at the beginning of this section).

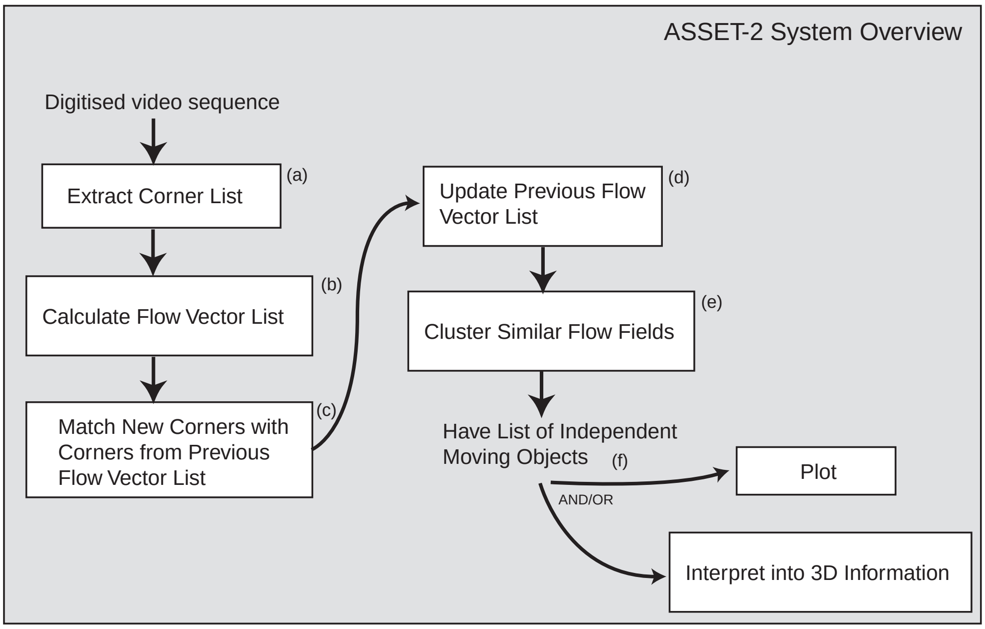

ASSET-2 System Overview
Digitised video sequence
Extract Corner List
(a)
Calculate Flow Vector List
(b)
Match New Corners with Corners from Previous Flow Vector List
(c)
Update Previous Flow Vector List
(d)
Cluster Similar Flow Fields
(e)
Have List of Independent Moving Objects
(f)
AND/OR
Plot
Interpret into 3D Information

Figure 2-19. The ASSET-2 system overview.

A similar approach to robotic navigation (as a part of the DROID team) involved using localised feature points for computing the scene structure over image sequences and was suggested by Brady and Wang (1992). In particular, they use corner points, with the correspondence being computed using the intensity values and gradients of the corners. The system computes structure from motion by tracking corners over a sequence of images and uses corners as 'seed' points for computing optical flow. Another version of the DROID system is discussed as in Shapiro *et al* (1992) that uses a basic form of the corner detector discussed in Wang and Brady (1995) along with a feature-matching algorithm based on a two-frame approach. The algorithm constructs a search window around each corner in frame 1 and lists each corner in frame 2 that exists within that search space window. A local patch correlation is performed on the candidate corner matches and the match corner is the corner with the best match correlation, provided that a certain threshold is reached. The correlation measure that is used is of the form:

$$c = \frac{\sum_{i=1}^{n}(t_i - \bar{t})(p_i - \bar{p})}{\sqrt{\sum_{i=1}^{n}(t_i - \bar{t})^2}\sqrt{\sum_{i=1}^{n}(p_i - \bar{p})^2}} \quad -1 \le c \le 1 \qquad \text{2-36}$$

where t_i and p_i are the intensity values of the template and of the patch and $\bar{t}$ and $\bar{p}$ are their means. This measure has the advantage of being invariant to a linear change between the data sets, so c compares the structure of the patches rather than individual intensities of the patch. The main problem with the approach of Shapiro *et al* (1992) is that features moving in the real world do not always project to simple trajectories in the image plane and so the linear predictors in the algorithm will fail.

Wiles and Brady (1995) suggest an algorithm for feature matching as follows:

- Extract features.
- Predict feature's image locations based on previous frames information only.
- For each trajectory search for strong matches locally about the predicted image location.
- While new matches have been found:
    - For each cluster containing new matches, validate the cluster (are matches consistent with clusters structures) and transfer all unmatched and mismatched features into the current image to give new predicted feature locations.
    - For each trajectory with a new predicted feature location, search for forced matches about the predicted image location, if forced matching fails record trajectory as a *zombie*.
- Destroy old clusters and trajectories.
- Update trajectory descriptions and strengths.
- Initialise new trajectories.

This technique is motivated by the need to track independently moving objects in an image sequence, again using clustering. Other autonomous vehicle implementations include Nashashibi *et al* (1994), who suggest an approach for outdoor navigation in unstructured terrain.

The approach taken in this thesis is largely performed by matching 2-D image features, which has the strong advantage of not suffering from the aperture effect. 2-D image features are well localised and have been shown to be consistently detectable between image frames. The motion vectors calculated from these feature points also represent the true motion of the points in the image plane. For this implementation, computational complexity should be minimised and one way to reduce this is to perform operations on a subset of important image points. It therefore follows that this strong set of image features should provide the highest level of motion information possible, and for this task corner points are ideal.

The use of corners as feature points provides low-level information for a feature matching technique, with no facilities other than matching based on feature properties. This can be erroneous when similar features appear. Therefore, it is logical that a higher-level approach be developed, and for this ‘active contour models’ were examined.

Chapter 3 Active Contour Models

3.1 Introduction

Active Contour Models (ACMs) are a popular method for tracking ‘regions’ as features through image sequences, using a similar method to the way in which humans observe the scene of a moving vehicle. Humans often focus on a particular feature point of the vehicle and then rapidly refocus (*optokinetic nystagamus*) to follow a different point on the same vehicle. Developed largely by Kass *et al* (1987), snakes[12] are ACMs that use an energy-minimising spline to help solve numerous computer vision problems, such as the analysis of dynamic image data, image segmentation and image understanding. The model is active as it is always attempting to minimise its energy function, showing dynamic behaviour. They are an example of the generalised technique of matching a deformable model to an image using energy minimisation techniques, where the shape of the contour determines its internal energy and the external energy is determined by the spatial location of the contour within the image plane. External forces are used to attract these contours towards image features, such as edges, lines, corners and image regions (Figure 3-1).

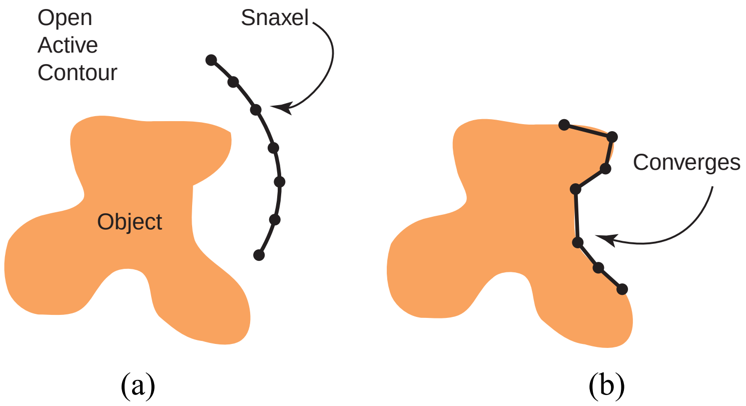


Figure 3-1. An example open ended active contour model. Figure (a) shows the snake being placed in proximity to the object to be tracked and figure (b) shows the snake having converged to the object.

There are several different types of deformable contours that may be used, such as the snake model suggested by Kass *et al* (1987) that ‘wraps around’ image features, or the

[12] Named as ‘snakes’ by Kass *et al* (1987), in part due to the way that the models ‘slither’ while minimising their energy values.

balloon model introduced by Cohen (1991) that expands to locate desired image features. Staib and Duncan (1992) deal with another method, elliptic Fourier decomposition for objects with shape irregularities, where a Fourier shape model is used that represents a closed boundary as a sum of trigonometric functions of various frequencies. They then use an iterative energy minimisation technique to fit the model within the image. This technique is limited to closed boundaries and does not always provide an appropriate basis for capturing shape variability.

Some of these methods deal with different types of problem; for example, Kass *et al* (1987) deal with local deformations only, while Staib and Duncan (1992) deal with global deformations. Global deformations might be described by scaling, rotation, stretching or dilation of a contour (rigid motion) in space, whereas local deformations might be caused by some higher level complex motion, such as the movement of human lips, living cell deformation and other deformations related to the shape of the feature and not just its 3-D spatial location. There are indeed two separate problems to be examined. Global deformations of rigid objects are too varied to be described adequately by single shape attributes such as bending energy or elasticity, while these descriptions may be adequate for local deformations in deformable objects.

3.2 Formulation and Initialisation

3.2.1 Formulation

Marr's Paradigm (Marr *et al*, 1978) treats boundary extraction as two independent problems, low-level edge extraction followed by higher-level edge linking. The edge detector (as discussed in Section 2.5) acts directly on the image intensities to produce an edge map that contains information such as the location of the edges, direction of edges and the strength of the edges. This edge map is then passed to an edge-linking algorithm that has numerous predefined rules to improve the continuity of detected edges. While much research has been carried out in both areas, many problems exist with this approach, such as: (Marr, 1978)

- There is no single solution to the edge extraction problem, with many different edge detectors producing quite varying results (the problem is ill-posed).
- Errors from the edge detection stage are passed directly to the edge linking stage without the opportunity for correction.

- To model a contour, the contour must first be extracted from the image. However, without prior knowledge of the contour of interest the extraction is an ill-posed problem (Lai, 1994).

Active contour models were proposed as an approach to solving the ill-posed edge detection problem. Kass *et al* (1987) claim that the low-level process of edge detection should provide sets of possible alternate solutions for the higher-level process of edge linking, rather than forcing forward a single unique solution. They propose an energy minimisation framework as a solution, designing energy functions with local minima that provide alternate solutions for the higher-level edge linking process, resulting in an active model that minimises to the desired solution when placed spatially close to that solution.

The structure of a snake is an ordered set of control points (or snaxels[13]) of the form $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]$ where each snaxel $\mathbf{v}_i \in \mathbf{I} = \{(x, y) : x, y = 1,2,\ldots, M\}$, allowing every snaxel to have a 2-D coordinate position on the image plane, *I* (see Figure 3-2). Alterations to the location or shape of the snake are possible by moving the position of the individual snaxels. The number of snaxels is chosen by selecting an appropriate internal distance *h*. This value is chosen on an application specific basis, where the coarseness of fit of the snake to the object is the defining factor. The smaller the value of *h*, the greater the number of snaxels that are required and the more tightly defined the minimised snake will be to the image object.

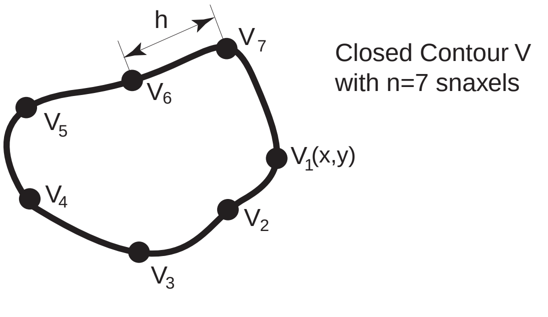


Figure 3-2. An example structure of a closed active contour

A snake can either be structured as open or closed, with a closed snake having the end points connected, so snaxel $\mathbf{v}_n$ is connected to $\mathbf{v}_1$ (i.e. $\mathbf{v}_{n+1} = \mathbf{v}_1$). Allowing the snake to

[13] For the rest of this chapter ‘snaxel’ will be used to refer to such points or elements of the snake. The term is derived from a contraction of the term “snake elements”.

be open in structure leads to difficulties in determining the desired energy at the first and last snaxels. Sometimes however, for certain applications, it may be necessary to require the end points to remain at specific pre-defined image locations.

3.2.2 Initialisation

Many methods for developing active contour models have emerged in recent years, since the introduction of snakes by Kass *et al* (1987), with many applications including Staib and Duncan (1992) and Leymarie and Levine (1993). However, in most of these applications it is assumed that the initial position of the snake is relatively close to the desired solution, in fact often initialised by a human operator (such as the method of Etoh *et al* (1993), Cohen (1991)). While this might be a suitable assumption in some cases, automated initialisation is required for many vision applications, including most motion based applications.

The initialisation of the snake is a difficult problem that has a significant impact on the outcome of the snake minimisation. If the snake's initial position is spatially distant from the desired solution, it is quite common for the snake to become trapped in local energy minima, due to irrelevant edge information or noise. The snake is also limited in spatial movement since the snake's own potential energy prevents the snake from moving far from its current position (Neuenschwander *et al*, 1994). Many methods require the user or other mechanisms to place the initial snaxels near the desired boundaries of the object to be tracked. If the snake is placed close to an intended image object, its energy minimisation behaviour will force the correct solution. Snakes do not attempt to solve the problem of detecting prominent image contours, but rely on other methods to place the snake near the desired contour. Some of these methods include:

- **The Hough transform** is commonly used for the extraction of the initial estimates of the contour position for *rigid* objects. These rigid templates cannot account for deformations that may occur, thus a rigid template chosen *a priori* cannot produce satisfactory results in all cases. Lai (1994) shows that performance actually degrades with deformation, so they use the generalised Hough transform to provide the initial contours when substantial prior knowledge is available.

- **Short snakes** may be initialised at strong edges and allowed to expand and even overlap until the entire boundary is covered.
- If enough computational power is available thousands of **randomly initialised snakes** may be placed on the image, until a suitable solution is found, however, this is rarely practical.

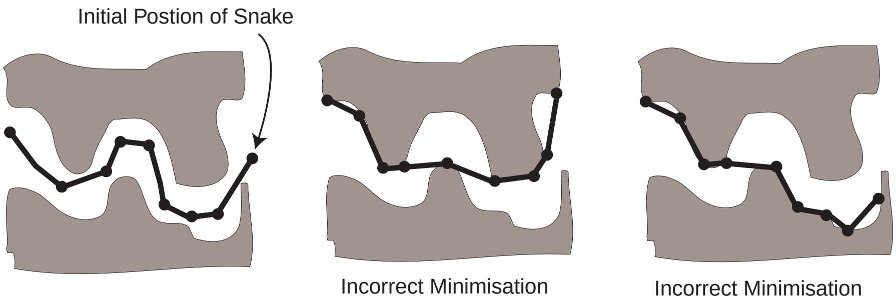


Figure 3-3. This figure shows an example of how a user might initialise a snake when objects are very close together. Unfortunately, because the objects are close, the user must outline the contour very carefully, otherwise if the snake comes close to undesirable objects then incorrect solutions may occur.

Neuenschwander *et al* (1994) have an interesting approach to the initialisation problem by first allowing a user to pick the beginning and end point of the snake. They then 'turn on' the image component at the end points individually, followed by 'turning-on' the other snaxels progressively from both ends towards the center, causing the snake to find the smoothest path. They term these snakes 'static snakes'. After the snake has been correctly initialised the dynamic component can then be activated. See Figure 3-3 and Figure 3-4.

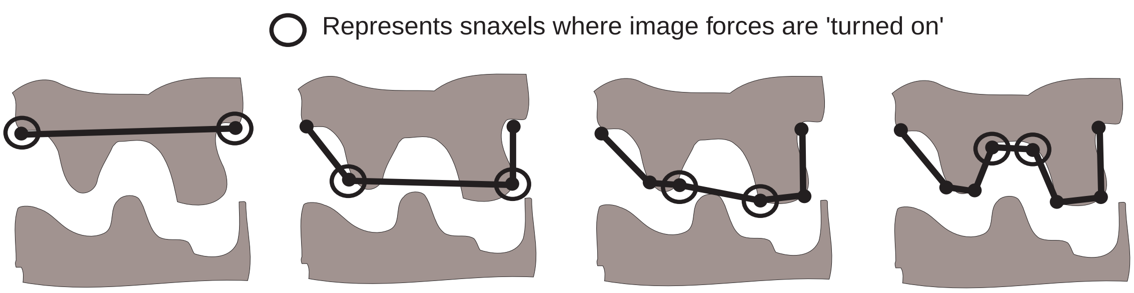


Figure 3-4. the method of Neuenschwander *et al* (1994) begins by placing the snaxels at each end of the snake, as close to the desired object as possible, then allowing each successive pair of snaxels to converge in an ordered fashion towards the centre snaxel of the snake. When the two active snaxels meet at the same snaxel location the optimisation method is complete and all snaxels are frozen in place except for the two active snaxels.

This method is implemented successfully for both synthetic and real-world images, showing their results closer to the desired result than a standard initialisation in many cases. They use an optimisation method to smooth over the snake's potential energy to remove small gaps, without removing valleys. Their method still requires a user to place the starting and ending snaxels very close to the object to be examined, however it is more forgiving that the method of Kass *et al* (1987) where an 'expert user' is required to place the entire snake, snaxel by snaxel.

Conditions might be defined for the use of the snake, such as: the feature to track might form a closed contour and be smooth. A snake might be placed around the outside of this feature by *a priori* knowledge or the initialisation of the contour might receive information from the Hough transform, if its shape is known. The snake must then be allowed converge around the object, or evolved to have certain properties such as smoothness.

The energy function for a snake consists of three distinct parts, internal, external and constraint energies:

$$E_{Snake} = E_{Internal} + E_{External} + E_{Constraint} \tag{3-1}$$

$E_{Internal}$ depends on the properties of the snake such as length or curvature, $E_{External}$ depends on factors such as image structure and $E_{Constraint}$ determines the constraint energy that the user can impose.

As $v(s) = (x(s), y(s))$ the snake energy may be written:

$$E_{Snake} = \int_0^1 E_{Snake}(v(s))\,ds \tag{3-2}$$

so,

$$E_{Snake} = \int_0^1 [E_{Internal}(v(s)) + E_{External}(v(s)) + E_{Constraint}(v(s))]\,ds \tag{3-3}$$

over the length of the snake, i.e. the linear parameter *s* is usually defined in the closed interval [0,1] and increases as the snake is traversed.

3.2.3 Internal Energy

The internal energy of the snake is determined by the intrinsic properties of the snake, such as its length or curvature. A snake will in general be better behaved if the control points are equally spaced around the object to be enclosed (smoothness), with the closer the points the more tightly the object can be enclosed. In other words, the internal spline energy imposes a smoothness constraint on the spline.

The internal energy of the contour may be written as:

$$E_{Internal}(v(s)) = \frac{1}{2}\left(\alpha(s)|v_s|^2 + \beta(s)|v_{ss}|^2\right) \qquad 3\text{-}4$$

where $v_s \equiv \frac{\partial v}{\partial s}$ and $v_{ss} \equiv \frac{\partial^2 v}{\partial s^2}$. The first order term $\alpha(s)|v_s|^2$ causes the snake to behave like a string (or membrane) so that it resists stretching, while the second order term $\beta(s)|v_{ss}|^2$ causes it to behave like a rod (or thin plate) that resists bending. Adjusting the weights $\alpha(s)$ and $\beta(s)$ controls the importance of the string and rod terms. Large values of $\alpha(s)$ will cause an increase in the internal energy as the snake is stretched, while small values of $\alpha(s)$ will make the internal energy function insensitive to the amount of stretch. Large values of $\beta(s)$ will cause an increase in the internal energy as the snake curves or bends, whereas small values of $\beta(s)$ will make the external energy function insensitive to the bending of the snake. Small values of both $\alpha(s)$ and $\beta(s)$ will place little constraint on the size or shape of the snake.

Setting the $\beta(s)$ term to be zero at a single point causes the snake to become second order discontinuous at that point and possibly develop a corner (Kass *et al*, 1987). The values of $\alpha(s)$and $\beta(s)$ can alter the minimisation solution quite dramatically, for example:

If $\alpha(s)$= 1 and $\beta(s)$= 0 are chosen,

$$E_{Internal} = \frac{1}{2}\left(|v_s|^2 + 0\right) = 0 \text{ when } v(s) = v(0) \qquad 3\text{-}5$$

This means that the contour will collapse down to a single point, to minimise the internal energy.

If $\alpha(s)$= 1 and $\beta(s)$= 0 are chosen and the end points are fixed at different spatial locations $E_{Internal}$ = 0 cannot be achieved, since the contour cannot collapse down to a single point. However, the energy would be minimised by a straight line between the two points:

$$v(s) = v_0 + s(v_1 - v_0),$$

as in Figure 3-5.

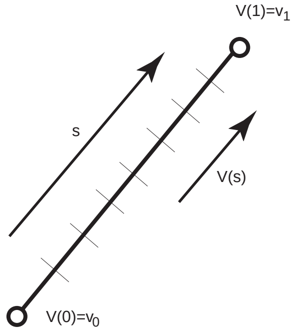


Figure 3-5. Minimised energy of the open contour with fixed end points and the second order term removed.

If $\alpha(s)$= 0 and $\beta(s)$= 1 are chosen,

$$E_{Internal} = \frac{1}{2}\left(0 + |v_{ss}|^2\right) \qquad 3\text{-}6$$

is required to minimise v_{ss}(s), but v_{ss}(s) = 0 for a straight line of the form $v(s) = as + b$, providing the optimal solution.

To define the discrete form of the internal energy function (Blake *et al*, 1992):

$$E_{Internal} = \frac{1}{2}\left(\alpha(s)|v_s|^2 + \beta(s)|v_{ss}|^2\right) \qquad 3\text{-}7$$

may be written in the form:

$$E_{Internal} = \frac{1}{2}\sum_{i=1}^{N}\left\{\alpha_i|v_i - v_{i-1}|^2 + \beta_i|v_{i-1} - 2v_i + v_{i+1}|^2\right\} \qquad 3\text{-}8$$

Approximating by the Finite Difference Method (to be discussed in Section 3.3.1) the first derivative v_s may be represented by $v_i - v_{i-1}$ and the second derivative v_{ss} may be represented by $v_{i-1} - 2v_i + v_{i+1}$, where $v_i = (x_i, y_i)$ and $|v|^2 = x^2 + y^2$. Rewriting this equation in terms of x and y gives:

$$\begin{aligned} E_{Internal} = &\frac{1}{2}\sum_{i=1}^{N}\alpha_i(x_i - x_{i-1})^2 + \beta_i(x_{i-1} - 2x_i + x_{i+1})^2 + \\ &\frac{1}{2}\sum_{i=1}^{N}\alpha_i(y_i - y_{i-1})^2 + \beta_i(y_{i-1} - 2y_i + y_{i+1})^2 \end{aligned} \qquad 3\text{-}9$$

The primary purpose of $E_{Internal}$ is to impose a smoothness constraint on the snake. In the absence of any external forces, $E_{Internal}$ should cause the snake to minimise its form to a line in the case of an open snake and minimise its form to a circle in the case of a closed snake.

3.2.4 External Energy

External forces determine the relationship of the snake to the rest of the image. The way in which the image *I(x,y)* might be considered is as a potential surface, where the snake is constrained to lie on *I(x,y)* under the action of a gravitational force *g* that has a constant magnitude. This means that the snake has a weight that causes it to fall down the slopes of the surface *I(x,y)*, where the surface might correspond to image intensities or to contrast values.

The external energy may be expressed as a combination of the functionals of the form:

$$E_{External} = k_{Line}E_{Line} + k_{Edge}E_{Edge} + k_{Corner}E_{Corner} + \ldots \qquad 3\text{-}10$$

Adjusting the user definable constants allows the exact attraction behaviour of the snake to be controlled.

Two very different forms of force, pushing and pulling may affect the external energy $E_{External}$. Kass *et al* (1987) suggested springs and volcanoes to model these two types of force. A spring is typically used to force the snake to attach to desirable features, while a volcano is used to push them away from undesirable features or noise. After determining the image location to apply these external forces, the strength and type of the forces must also be calculated.

The spring force is best determined from the points where the force is to be placed, rather than examining the energy term that will result. There are two points that must be known for the spring force; the fixation point and the snaxel to which it will be attached. The force can then be evaluated easily, based on the distance between the points, as:

$$E_{Spring} = {}^{k}\!/_{2}(x_i - x_f)^2 + {}^{k}\!/_{2}(y_i - y_f)^2, \qquad 3\text{-}11$$

where (x_f, y_f) is the fixation point, (x_i, y_i) represents the snaxel to which it is attached and *k* is a constant for a particulart spring. In the case of the volcano this is much different. A volcano constraint involves a number of snaxels on the snake and so it is not practical to determine the force between each individually effected snaxel and the volcano. Instead a local cone of energy $E_{Volcano}$ may be added to the constraint energy term $E_{Constraint}$. The slope of the edge of this local cone will then determine the repulsive effect of the volcano.

There are several different functionals that may be used for determining external energies, such as:

1. The Line Functional

One of the simplest forms of external forces is the line functional. It may be useful in certain circumstances to have the snake attracted to lines of a particular intensity.

$$E_{Line} = \pm\sum_{i=1}^{N} I(v_i) \tag{3-12}$$

where the sign that is chosen attracts the snake to either light intensity (-ve) or dark intensity lines (+ve). The snake will attempt to align itself with the nearest lightest or darkest line.

2. The Volcano Functional

Volcanoes are a way of adding high-level information to the image. The volcano could be assigned to an area of the image that is known *a priori* to be problematic. In medical imaging applications volcanoes can be placed in identified areas to prevent the snake from leaving the organ region of interest. In this case springs could be added to contain the snake within an area of interest and the volcanoes placed in regions to avoid.

$$E_{Volcano} = \sum_{i=1}^{N} \frac{1}{|v_i - v|}, \tag{3-13}$$

where *v* is the point where the volcano is placed. A volcano is usually formed by locally deforming the potential surface, often by adding a conic surface; see Figure 3-6(a).

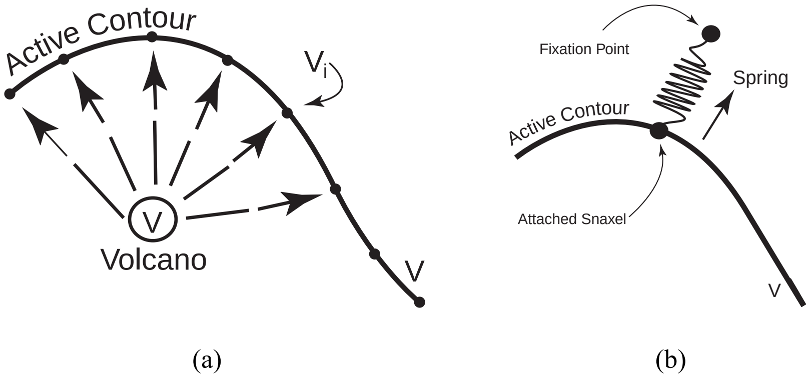


Figure 3-6. (a) The volcano pushing the contour away and in (b) the spring force attached to a snaxel on the contour.

3. Edge Functional (Edge loving)

For this functional the snake is to be attracted to contours with large image gradients. This can be defined as:

$$E_{Edge} = -\sum_{i=1}^{N} \left| \nabla I(v_i) \right| \qquad \text{3-14}$$

See Figure 3-7 for an edge-loving contour minimising to a desired object.

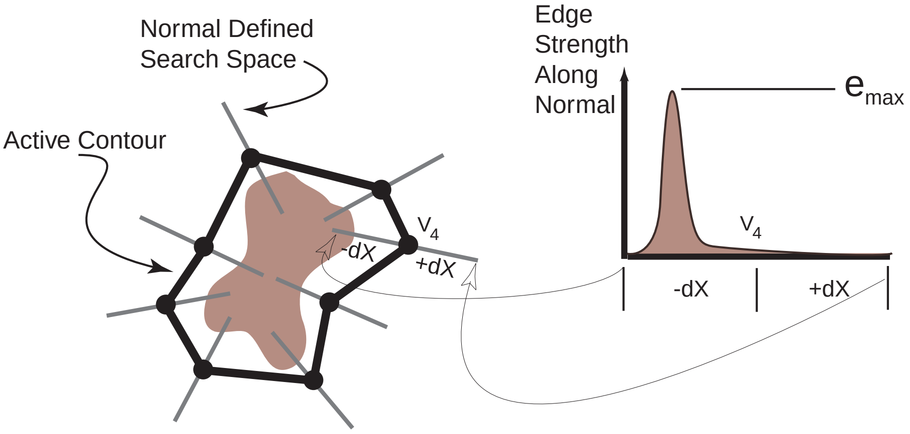


Figure 3-7. An example of an active contour with a linear search space for edge strength.

4. Corner Functional (Corner Loving)

$$E_{Corner} = -\sum_{i=1}^{N} C(v_i) \qquad \text{3-15}$$

This functional may be used to find the terminations of line segments and corners. Kass *et al* (1987) use a slightly smoothed version of the image to remove high frequency noise and they denote the gradient directions along the snake in the smoothed image C as $\psi(x, y)$. Let

$$n(x, y) = (\cos\psi(x, y), \sin\psi(x, y)) \text{ and} \qquad \text{3-16}$$

$$n_{\perp}(x, y) = (-\sin\psi(x, y), \cos\psi(x, y)) \qquad \text{3-17}$$

be unit vectors along and perpendicular to the gradient directions $\psi(x, y)$. The curvature of the level contours in the smoothed image C can be written as (Kass *et al*, 1987):

$$E_{Corner} = \frac{\partial\psi}{\partial n_{\perp}} = \frac{\partial^2 C / \partial n_{\perp}^{\ 2}}{\partial C / \partial n} \qquad \text{3-18}$$

so

$$E_{Corner} = \frac{C_{yy} C_x^{\ 2} - 2C_{xy} C_x C_y + C_{xx} C_y^{\ 2}}{\left(C_x^{\ 2} + C_y^{\ 2} \right)^{3/2}} \qquad \text{3-19}$$

5. Image Regions

Etoh *et al* (1993) have developed a method to precisely determine the contour that should be extracted from a roughly drawn initial contour (see Figure 3-8). The method uses region extractors (using colour information and position) to determine the regions of the initial contour. An active contour model is then developed based on a log-likelihood strategy. In general for pixels in the object region the log-likelihood description: θ^{in}

$$l\left(y\middle|\mathbf{R}^{in},\theta^{in}\right) > l\left(y\middle|\mathbf{R}^{out},\theta^{out}\right) \tag{3-20}$$

where θ^{in} is the mixture density description of the object region $\mathbf{R}^{in}$ and θ^{out} is the mixture density description of the background region $\mathbf{R}^{out}$. The Maximum Likelihood Estimate (MLE) position is where the sum of the log-likelihood of $\mathbf{R}^{in}$ and $\mathbf{R}^{out}$ is maximised. This should identify the boundaries between image regions, however in real-world images another measure is used to insure accuracy, the maximum gradient position of the log-likelihood. The closer these two measures, the more accurately this algorithm performs.

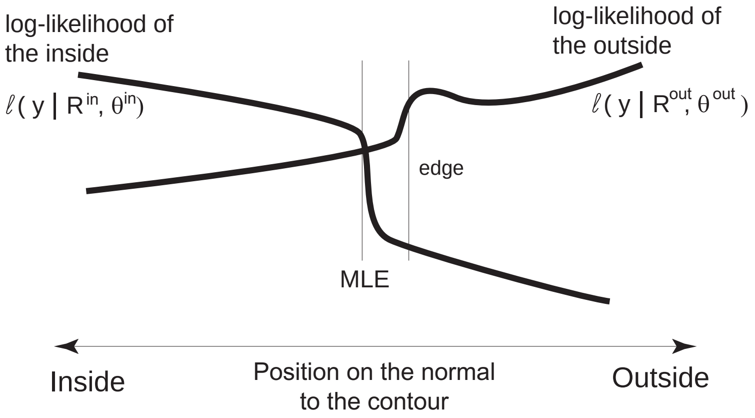


Figure 3-8. The Log-likelihood Search of Etoh *et al* (1993)

3.2.5 Regularisation

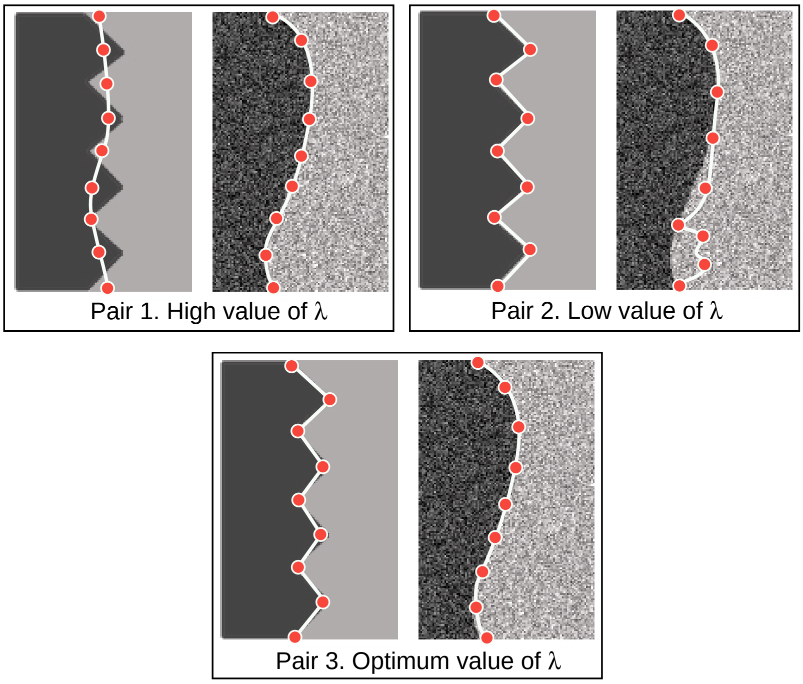
To encourage the active contour model to be more robust and stable it may be necessary to regularise the terms of the dynamic equations. This can however cause restrictions on the action of the contour. For example, the snake is usually required to be short and smooth. Figure 3-9 describes the effects of regularisation on an open snake. If regularisation is strongly encouraged (in Figure 3-9 (pair 1)) then the snake will fail to capture the corners but will be very robust to noise. If the regularisation parmeter is set too low (in Figure 3-9 (pair 2)) then the snake will capture the corners, but will be very sensitive to noise. A suitable compromise is given in Figure 3-9 (pair 3). It is however difficult to select this parameter exactly. Kass *et al* (1987) originally proposed snakes as a regularisation approach to the ill-posed edge detection problem. As described previously, the Internal Energy ($E_{Internal}$) imposes continuity and smoothness constraints, while the External Energy ($E_{External}$) attracts the snake to image features. In this case the snake **V** is required where:

$$\mathbf{V} = \min_{\mathbf{V}}\left(\sum_{i=1}^{n}\lambda_i E_{Internal}(v_i)+(1-\lambda_i)E_{External}(v_i)\right) \tag{3-21}$$

and $\lambda_i \in [0,1]$ are the regularisation parameters. Setting $\lambda_i > (1-\lambda_i)$ strengthens the regularisation at each snaxel, providing noise-resistant models, and inversly, small values of λ_i allow the models to capture features but to be noise sensitive.

Lai (1994) (and Lai and Chin, 1998), develop a method called *g*-snakes[14], using a local minimax criterion of Equation 3-21 (shown as in Figure 3-9) to develop a tradeoff at each location along the boundary, of the form:

$$e\left(\mathbf{V}^*,\Lambda^*\right)=\min_{\mathbf{V}}\left(\sum_{i=1}^{n}\max\left(\mathbf{E}\left(\mathbf{v}_i\right)\right)\right) \tag{3-22}$$



Pair 1. High value of λ

Pair 2. Low value of λ

Pair 3. Optimum value of λ

Figure 3-9. Effects of regularisation. In pair 1, the snake fails to capture the corner information, but is robust to noise. In pair 2, the snake captures the corner information, but is highly noise sensitive. Pair 3, shows an illustration of the choice of the optimum values of λ_i..

where the solution snake $\mathbf{V}^*$ and the solution regularisation parameters Λ^* (where $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$) are calculated by finding the snake $\mathbf{V}$ with the minimum energy determined by the sum of the maximum of $\mathbf{E}(\mathbf{v}_i) = \lambda_i E_{Internal}(\mathbf{v}_i) + (1-\lambda_i) E_{External}(\mathbf{v}_i)$ for each snaxel $\mathbf{v}_\mathrm{i}$ by choosing the appropriate value of λ_i local to that snaxel. Their solution allows automation in the selection of suitable regularisation parameters and they obtain results that were not possible without some form of local automation. Figure 3-9 shows the effect caused by the choice of optimum values of λ_i.

3.2.6 Scale Space

One advantage of active contours is the way in which a group of snaxels pulled towards a low energy image feature will allow the internal energy of the snake to pull neighbouring snaxels towards a possible continuation of the feature. The same effect may be produced in a scale-based approach where the snake may be allowed to achieve equilibrium on a blurred external energy functional, and then by reducing the blurring, the result is minimisation by scale-continuation (Kass *et al*, 1987).

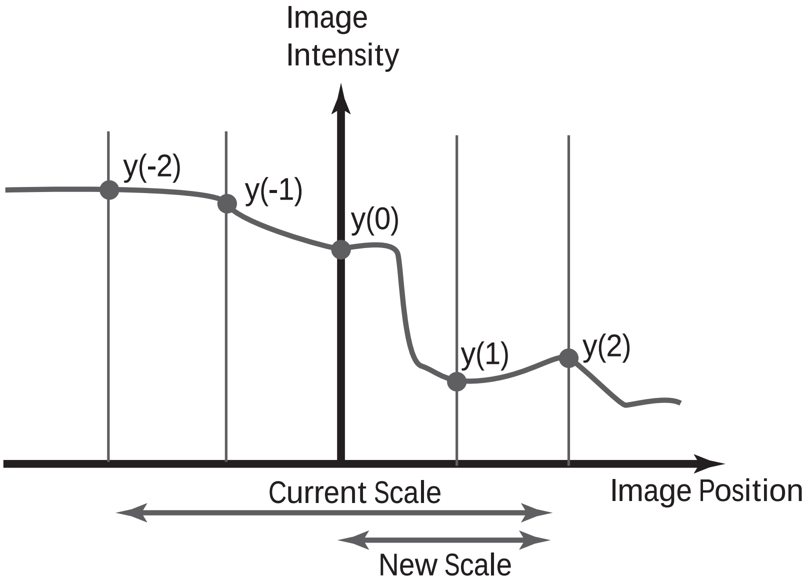


Figure 3-10. Curwan and Blake (1992) coarse-to-fine scale-based search for an image feature (Modified from Curwan and Blake (1992)).

This approach will limit the snake from becoming locally 'trapped' on certain pixel locations (possibly due to noise). Instead the snake will be attracted to the local minima from a greater spatial distance, initially providing a poor localisation of the edge, but improving as the blurring is reduced. Gaussian blurring is often used for this task, how-

[14] *g*-snakes – generalised-snakes, using a generalised method, based on the minimax criterion.

ever Curwen and Blake (1992) suggest that large blur implies considerable computational power for real-time tracking and is unnecessary. Instead, they use a controlled-scale search along linear paths originating from the contour. Their feature search algorithm is recursive and operates with a coarse-to-fine strategy. At each scale, finite difference approximations to the image gradient are calculated at the search point and surrounding points. The new search point is the point with the greatest calculated gradient. The scale is then halved, as in Figure 3-10 with the search bracket moving towards $\left|y(2)-y(0)\right|$.

3.2.7 Constraint Functional

The external energy component of the snake can have contributions other than directly from image energy functionals. The indirect constraint energy is often included and determined by the user or some form of pre-analysis. A particular image point might pull the snake as a form of power assisted operation (Young, 1995). This might also be necessary if the snake settles (becomes trapped) in a local energy minimum, due to noise or a local feature, that a higher-level process (such as a human operator) determines to be incorrect. In this case, an area of high $E_{Constraint}$ may be placed at this location to force the snake to converge to a different local minimum. To evaluate the constraint forces caused by this constraint energy, the directional derivatives of the constraint energy are often used, i.e. $-E_{Constraint_{\bar{x}}}$ and $-E_{Constraint_{\bar{y}}}$. This is equivalent to calculating the slopes in the *x* and *y* directions at each snaxel for the last known position in the image, so as to force the snaxel to move by following the steepest slope in the constraint energy field in the neighbourhood of each snaxel.

3.3 Energy Reduction

3.3.1 The Finite Differences Method (FDM)

The original method to reduce the energy in Equation 3-10 was developed using the Euler method with finite differences. The contour is represented as in Figure 3-2, as a set of snaxels, connected together by an elastic, stiff link. The image forces act on the snaxels to alter the snake from its natural shape. However, its two immediate neighbours, representing elasticity and its nearest four neighbours representing stiffness, are also acting upon each snaxel. The solution involves solving *N* equations of *N* unknowns. The system of equations is repeated iteratively, with each iteration moving closer to a stable solution. The algorithm requires *O(n)* time, i.e. an iterative technique using sparse matrix methods, with each iteration taking Euler steps with respect to the internal energy and implicit steps with respect to the external and constraint energies (Kass *et al* (1987)).

Difficulties with this approach include:

- The image is only sampled at the snaxels, so to make the contour sensitive to fine detail the number of snaxels and so number the of equations to be solved increases.
- If the number of equations *N* is increased, then the gap between the snaxels must also be reduced, causing the effect of stiffness to occur only on a local scale. This could cause the snake to become trapped on undesirable features or noise. This could be overcome in part by changing the weight of the stiffness term.

In the original method of Kass *et al* (1987), variational calculus was used to minimise the energy functionals, which in some cases can become unstable, causing many snaxels to bunch up on strong edge features.

3.3.2 The Finite Element Method (FEM)

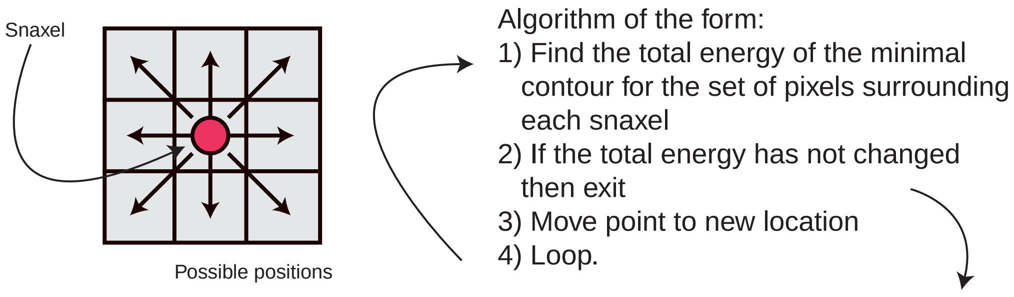
Another approach to solving the minimisation problem involves the Finite Element Method (FEM), suggested by Sullivan and Baker (1992), where all the forces are allowed to find their own local minimum. The position of the contour ($x(s),y(s)$) at each snaxel is described by a low order polynomial in *s*. They allow successive elements to be connected together by constraints, which try to enforce low-order continuity in *x* and *y* at their links. The problem then becomes solving *2N* equations with *2N* unknowns, where *N* is the number of elements. The image forces act along the entire element in this method, not just at the discrete snaxels, so the Gauss-Legendre n-point rule is used for choosing the sample points for numerical integration, and their relative weights, as it is an optimal solution for approximating integrals (Press *et al*, 1995). They use the FEM with an adaptable order of integration to give a method that allows better use of the local image information, without being over-sensitive to noise. It was found that integration along the elements is much less computationally expensive than increasing the number of elements in the finite difference method, to achieve a more localised solution.

Leymarie and Levine (1993) claim that the FDM is appropriate for use in tracking with active contours, since its convergence and numerical stability are adequate. However, for more difficult problems, such as active surface models, they recommend the use of the FEM. Cohen and Cohen (1993) argue for the use of the FEM, even in dealing with active contours. This was essential for reducing computational cost in their application, which involved constructing 3-D models from multiple 2-D contours. They discovered that when using the FEM, the solution became less sensitive to deformations in the model than the FDM, and that the FEM has better level of complexity, as a larger step size is possible, resulting in a linear system of a smaller size.

Amini *et al* (1990) point out some of the problems involved in the method of solution of Kass *et al* (1987). They propose an algorithm using dynamic programming that allows the addition of 'hard' constraints that cannot be violated, allowing the algorithm to be numerically stable. The optimisation problem is set-up as a discrete multistage decision process that is solved using a discrete 'time-delayed' dynamic programming algorithm, aiming to bypass local minima. They discretise the integral term to the form:

$$E_{Total} = \sum_{i=0}^{n-1} \left(E_{Internal}(v_i) + E_{External}(v_i) \right), \qquad \text{3-23}$$

to take advantage of the discrete nature of the problem, treating the minimisation as one of a finite set of stages with each stage having a finite set of possible solutions (see Chandran and Potty (1998)). The algorithm takes the form of Figure 3-11.



Algorithm of the form:
1) Find the total energy of the minimal contour for the set of pixels surrounding each snaxel
2) If the total energy has not changed then exit
3) Move point to new location
4) Loop.

Figure 3-11. The Dynamic Programming Technique of Amini *et al* (1990).

The variational methods for energy minimisation have several restrictions:

- High order derivatives of discrete data are required. This can cause instabilities unless the image is smoothed (leading however to poor localisation).
- If no image forces are available the contour will minimise to a point or line.

- If the contour is not placed near to the desired feature, it will not be extracted.
- It is important to note that the external field changes with each frame iteration. Amini *et al* (1990) state that unless this field remains constant that this can also cause instability in the variational approach.

Computation requirements of the dynamic programming approach are however linear, which increases to $O(nm^3)$ in the algorithm of Amini, where *n* is the number of snaxels on the snake, and *m* is the number of possible locations that a snaxel may move in a single iteration. The memory requirement also increases from *n* x *m* to *n* x m^2 when considering 2nd order terms. This method is claimed to give a more controlled contour where convergence is guaranteed and so the optimality of the solution. It is however not suitable for a scale based approach and will become more complex as the image resolution increases.

3.4 Snakes and Motion (Dynamic Contours)

Snakes can be applied statically to single images, or dynamically to a sequence of images. Once a snake has 'locked on' to a salient image feature, the snake will follow this feature if it moves slowly from its location. If the object accelerates too quickly for the snake to track, it can cause the snake to localise to a different local minimum, losing track of the object. Snakes are particularly useful for tracking deformable objects as the structure of the snake can deform with the object over time. A simple way to add interframe constraints would be to give the snake a *mass* property (another level of modelling), which would improve the motion tracking, by allowing a prediction of the next location of the snake based on its current velocity. As shown in Figure 3-12, the dynamic contour becomes a two-step process. In the first step, the dynamic model is used for prediction, to calculate motion from one frame to the next and predict the next location. This predicted position is refined in the next step using the external image forces and subsequently used in the next iteration.

It is possible to attach a dynamic component to a snake to allow it to track objects in an image sequence. Momentum might be added to allow it to move in the same direction from one frame to the next. A dynamic snake may be represented by introducing a time-varying mapping *v(s,t)* and a kinetic energy,

$$\int_0^1 \mu(s)\left|v_t\right|^2 ds\,, \qquad 3\text{-}24$$

where $\mu(s)$ is the mass density, and $v_t \equiv \frac{\partial v}{\partial t}$, which results in an equation for the snake of the form:

$$E_{Total} = \frac{1}{2}\int_0^1 \underbrace{\mu(s)\left|v_t\right|^2}_{Kinetic} - \underbrace{\alpha(s)\left|v_s\right|^2 - \beta(s)\left|v_{ss}\right|^2}_{Internal} - \underbrace{\rho(v)}_{External}\, ds\,, \tag{3-25}$$

where $\rho(v)$ includes all external potentials that attract the snake to edges, corners and also including any constraint energy that might be added (Terzopoulus and Szeliski, 1992). Using this formulation it is possible that the snake may oscillate around a solution unless some form of damping is used. One such method is using the Rayleigh dissipation functional

$$D(vt) = \frac{1}{2}\int_0^1 \gamma\left|v_t\right|^2 ds\,,$$

where $\gamma(s)$ is the damping density.

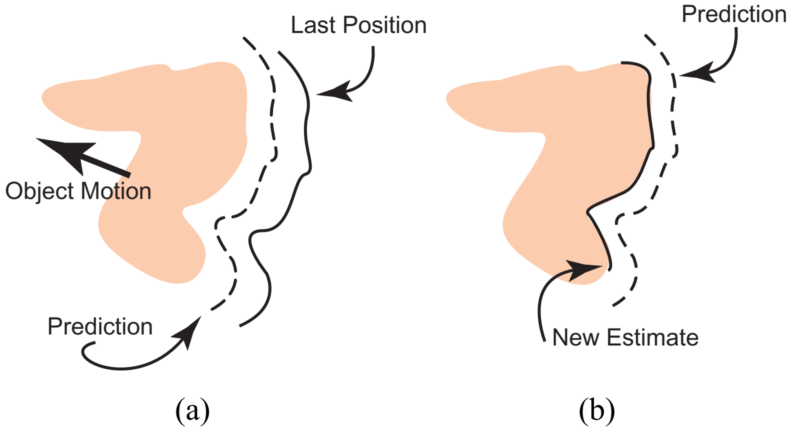


Figure 3-12. Dynamic contours with prediction.

Terzopoulus and Szeliski (1992) develop a technique called Kalman snakes for motion tracking to allow the tracking of non-stationary objects. They found that the probabilistic approach of using Kalman filtering proved successful in dealing with noisy measurements and allowed them to combine measurements from multiple sensors and over time sequences. Predictive models require probabilistic treatment in order to avoid forcing forward an incorrect solution and they still include a level of uncertainty to prevent the prediction from dominating the calculation. The Kalman snake that they develop allows the prediction of possible motions and optimally extracts measurement in uncertain noisy cases. Peterfreund (1999) proposes a new Kalman filtering approach that uses the image gradient and optical flow measurements along the contour as system measurements for calculating the velocity of the snake. Velocity measurements that are not consistent with the previous estimation of motion and the corresponding edge information are rejected during the update step. This method shows the combination of optical flow measurement and active contour models may be performed.

Curwen and Blake (1992) point out a problem with the assumption of uniform velocity in defining a contour with inertia. In Figure 3-13, no assumptions are made about the contour, and it tends to flow around the shape (reluctant to move). However, when a shape assumption is added the contour moves along correctly with the shape. This assumption is created by allowing a standard flexible contour to relax on the object. It is then frozen and becomes the template shape (Curwen and Blake, 1992).
A second B-spline curve, identical to the first, is then initialised and coupled to the first. The second B-spline is allowed to deform, but is constrained by the template spline so that it can only deform slowly.

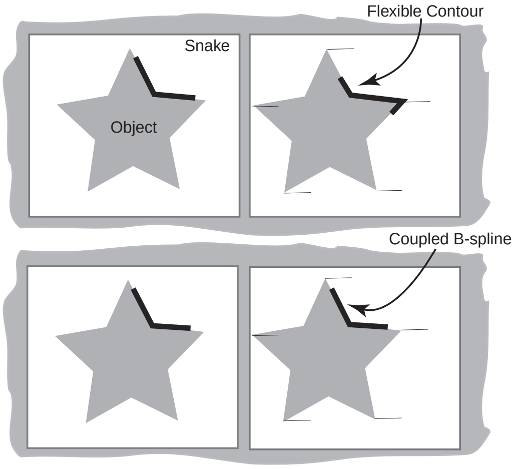


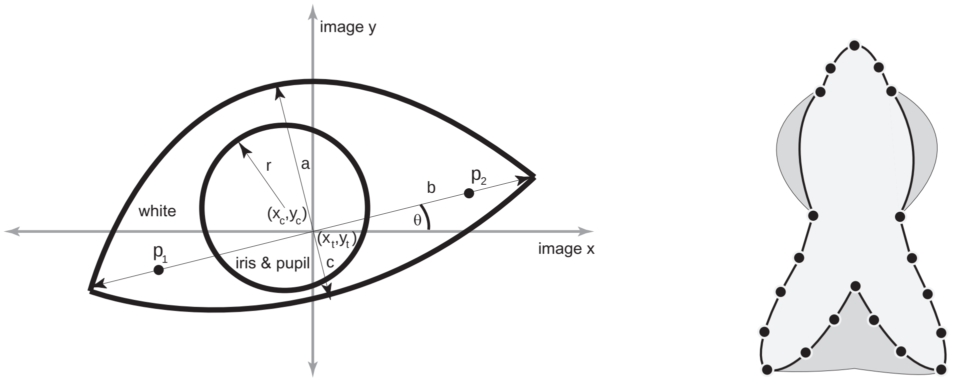
Figure 3-13. Showing a dynamic contour tracking an object. In the top case the flexible contour slides around the object as it moves to the left. In the bottom case, the coupled B-spline is attached and follows the real motion of the object.

3.5 Other Forms of Active Contour Models

3.5.1 Deformable Templates and Active Shape Models (ASMs)

Classical template matching attempts to match a template mask to an image. Deformable templates attempt to solve the problems associated with template matching, such as failing if the object in the image is slightly deformed or if the lighting conditions vary. The deformable template approach of Yuille *et al* (1989) involves three main stages. The generation of 1) a ‘geometrical model’, 2) an ‘imaging model’ that specifies the image intensity properties of the template and 3) an algorithm that determines the goodness-of-fit between the template and the image. Figure 3-14(a), shows the deformable template model of Yuille *et al* (1989), where a geometric model and an image model are created. The geometric model is parameterised by all the geometric measures from the figure $\boldsymbol{g}=(\boldsymbol{x_c},\boldsymbol{x_t},r,a,b,c,\theta,p_1,p_2)$. The image model assumes that the iris corresponds to a valley in the image intensity and the whites to a peak and the boundaries to image edges. This model is then successfully applied to tracking the human eye through image sequences. However, these models are 'hand-built' and the models and energy minimisation techniques have to be developed for each individual application.

Cootes and Taylor (1992) suggest 'Active Shape Models'(ASMs) that allow initial estimates of pose, scale and shape of an object in an image to be refined, using a Point Distribution Model (PDM)[15] that is derived from sets of training examples. At each point on the ASM a motion value is calculated that would improve the position of that point. The overall position, orientation and scale of the model that will satisfy all the displacements is then calculated, while always taking care that the global shape constraints are enforced by ensuring that the shape parameters remain within appropriate limits. The search for new positions is usually performed by a search for close edge points. They apply this method to finding resistors on printed circuit boards and the location of human hands. The method fails if the strong edges are outside the search space area.



(a)

(b)

Figure 3-14. (a) The deformable template of the eye, Yuille *et al* (1989). (b) The Leeds people tracker model.

[15] A Point Distribution Model is a way of representing an object (or indeed a class of an object) using a flexible model of carefully located labelled points.

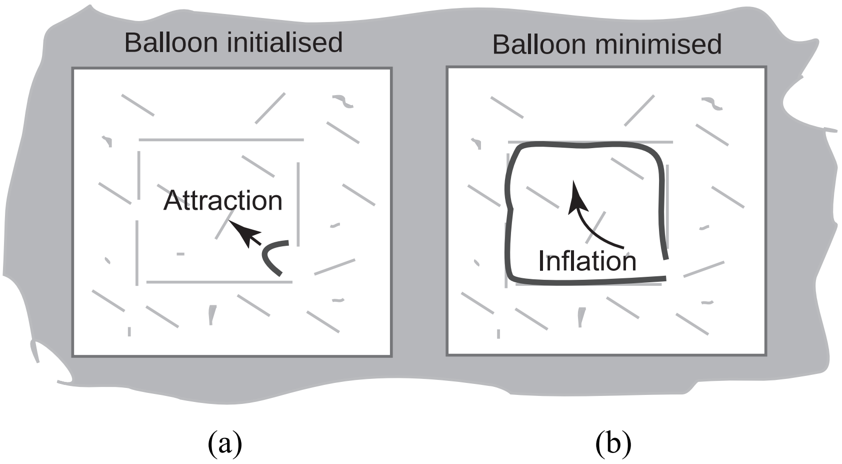
The Leeds People Tracker[16] uses flexible 2-D *a priori* models to recognise and track pedestrians in image sequences. The shape model is automatically created from test shapes from training images created using image subtraction. Each shape is traced by a cubic B-spline with equally spaced control points. A set of 'modes of variation' is determined from the variability observed in the training data (see Figure 3-14(b)). This variability is especially large at the arms and legs of the pedestrian model. Each object in the scene is represented by a set of parameters (the object centre, the scale, the orientation and a small set of shape parameters that are the weights for the eigenshape basis vectors) and Kalman filtering is used for model fitting and tracking. The contour has regularly spaced sample points along the B-spline contour, where suitable features are searched for along the line normal to the contour direction. Using the Kalman filter equations the measurements are combined to update the parameter estimates and the uncertainties (Baumberg, 1996). A motion-based event detector is used that is based on image frame subtraction with a dynamically updating background image. The motion regions of the image are then extracted with the top of the bounding box corresponding to the person's head and the bottom of the bounding box representing the person's feet. A fitness measure is determined for the model and if it is high for several frames then it is accepted and tracked. The system can deal with partial occlusion, such as when a person walks behind a car and results are very encouraging[17]. Heap and Hogg (1996) use similar techniques to construct 3-D PDMs from 3-D MRI volumetric training images where key features are located by hand. Tracking is performed on a human hand and the results displayed are noise-free. One point to mention is that the modes of variation used are linear, but they had begun the development of a non-linear PDM.

[16] The Leeds People Tracker – developed at the University of Leeds – http://www.scs.leeds.ac.uk/vislib/imv/

[17] For more work on deformable templates and occlusion *see Mardia et al (1997)*, a computationally intensive method based on a Bayesian recognition system (applied to automated mushroom picking).

3.5.2 The Balloon Model

Another approach to the energy minimisation process was suggested by Cohen (1991) based on the Galerkin solution of the FEM. The Galerkin solution of the FEM has the advantage of greater numerical stability and better efficiency (see Press *et al* (1992)). This approach is applied to the closed contour case and finds exceptionally good stability. He attempts to overcome the problems associated with the original snake, including the behaviour of minimising to a straight line, when not placed in close proximity to a desired object. The balloon model uses an additional inflation force, so that the balloon constantly inflates, passing through edge fragments that are too weak to contain the inflation, but resting on stronger edge features. The applications are clear for the balloon model, such as medical imaging, tracking closed internal organs (such as the heart) in noisy image scans (using noisy ultrasound and magnetic resonance images). It is very important in dealing with the balloon model, that it be prevented from overstepping the edges of the feature of interest. Cohen uses the FDM and prevents a step size of more than 2 pixels. The algorithm works very accurately in the example application of tracking the contractions of the left ventricle of the human heart.



(a) (b)

Figure 3-15. The balloon active contour model (Cohen, 1991). The balloon is inflated from the inside and expands, overcoming isolated valleys and noise giving better results than snakes could in particular cases. In (a) the algorithm begins with a simple curve (open or closed) placed close to the intended contour. External forces caused by the image energies are applied to the model, and in (b) the final position of the model corresponds to the minimum of the models energy.

3.5.3 B-splines

B-splines allow a compact parametric representation of active contours. A spline of order *n* is a piecewise polynomial function, consisting of concatenated segments, each of some polynomial order *n*, joined together at breakpoints. Polynomials that make up these splines can be of any degree, but higher-order polynomials can have undesirable non-local properties. A spline function *x(s)* is constructed from a weighted sum of N_B basis functions $B_n(s)$, $n=0,1,\ldots,N_B-1$. In the most straightforward case, each basis function has *d* polynomials, each defined over a unit length of the *s* axis. The unit lengths of the polynomials are joined together at knots. See Figure 3-16.

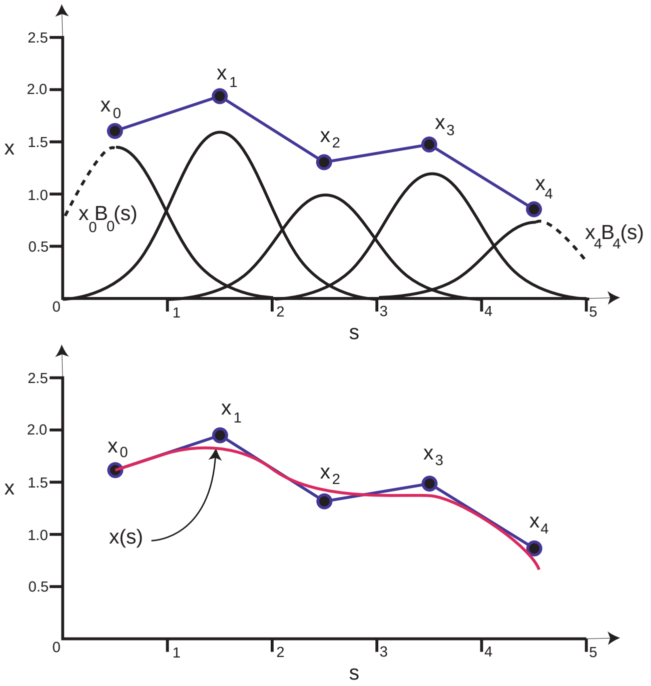


Figure 3-16. Construction of B-splines, the spline functions are weighted by weights x_0 to x_4 and combined to obtain spline function *x(s)*.

Blake *et al* (1998) set out the framework for using B-splines for motion tracking. If a snake were required to represent a desired object very closely then using the original snake form would require a snake with hundreds of elements. At every iteration of the snake's position it would be necessary to solve a matrix equation with several hundred variables, being quite computationally intensive. The spline function is:

$$x(s) = \sum_{n=0}^{N_B-1} x_n B_n(s) \qquad \text{3-26}$$

where x_n are weights applied to each basis function. See Figure 3-17.

Using B-splines, real-time tracking has been performed at video rate. There are some problems with B-splines, such as the difficulty involved in matching splines over image sequences, especially when re-parameterisation occurs.

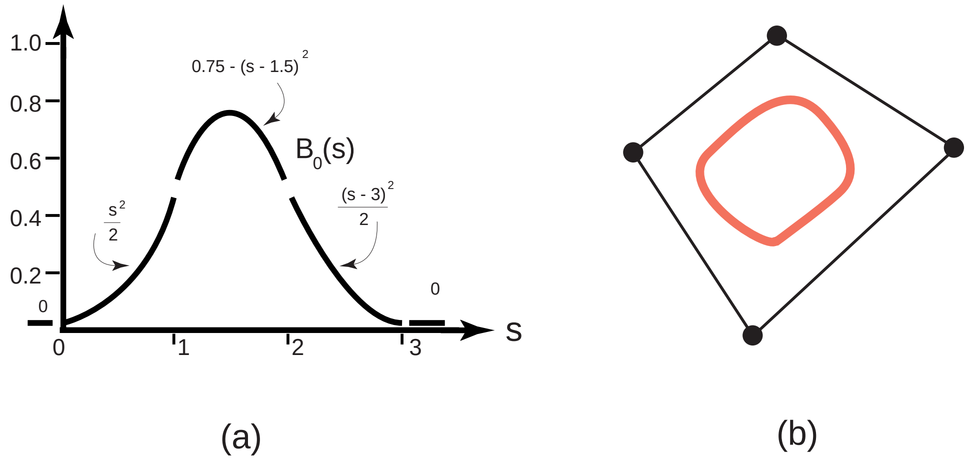


Figure 3-17. B-spline example. (a) An example B-spline basis function *$B_0(s)$* with knots at *$s=0,1,2,3$* and other basis functions are translated copies of *$B_0(s)$*. (b) An example of a closed curve contour with four control vector points.

3.6 Terminating Condition

It is necessary to obtain stable snake behaviour and this can involve choosing a suitable terminating criterion for obtaining the optimum solution in the shortest possible time. Often the terminating condition for a snake is when none of the snaxels move, i.e. the snake has reached it lowest possible potential energy.

Leymarie *et al* (1993) propose a steady-state criterion, requiring a minimal stable height of a snake based on the potential surfaces topography and not on the internal or external energies of the snake. At each snaxel and snaxel interval, they calculate the sum of the local heights of the snake along its length and divide this by the snake length - they wish to minimise:

$$E_{Length}(v) = \frac{\int_0^1 E_{Field}(v)ds}{\int_0^1 |v_s| ds} \qquad \text{3-27}$$

which is designed to cause the length of the snake to have little effect on the determined stable state, providing only a terminating condition where the variation in $E_{Length}(v)$ is tracked, until:

$$\left|\Delta E_{Length}(v)\right| < \varepsilon \qquad 3\text{-}28$$

When the magnitude of the difference is below a certain threshold ε then the algorithm terminates to prevent oscillatory behaviour in the snake.

3.7 Applications and Variations

Kass *et al* (1987) applied snakes to track the local deformations of a speaker's lips through image sequences. Leymarie and Levine (1993) applied snakes to the problem of tracking deformable cell movement, as they believe that living cells provide an ideal testing situation for active contours since their boundary can deform simultaneously or adopt any form in the image plane. They find however that snakes provide solutions to the segmentation and tracking problems only in constrained cases. They experienced problems in selecting bounds for the forces (necessary to obtain a stable snake) and selecting a suitable terminating criterion.

Gunn and Nixon (1997) suggest a dual active contour, where one active contour expands from the inside of the desired feature and another active contour contracts from the outside, where it is assumed that the two contours are placed inside and outside of the desired feature respectively. Kass *et al* (1987) require the initial contour to lie outside of the feature of interest and contraction forces to cause the snake to converge to the desired solution. Cohen (1991) proposed the balloon technique that requires the initial contour to lie within the feature of interest, with an inflating contour that reduced the sensitivity to noise. The technique of Gunn and Nixon (1997) technique helps reduce the sensitivity to initialisation of a desirable solution, by allowing a comparison between the two contours energy, allowing a selective rejection of weak minima solutions. They apply this technique to very noisy scenes and the dual active contour converges to a desired solution where other methods do not. This confronts some of the problems associated with initialisation, as the inner and outer contours do not have to be placed very accurately in the image.

Etoh *et al* (1993), do not expect the user to outline the contour exactly, rather they use a snake that determines a precise boundary by using sub-regions, created from clustering of colours and positions. They use region description techniques to determine if contours are stably obtained as boundaries or background regions. They use the dynamic programming technique of Amini *et al* (1990) to minimise the contour energy that is based on these region descriptions.

Cohen and Cohen (1993) define another form of approach to the use of active contours. Unlike the method of Kass *et al* (1987) they extract the global edge information using a good local edge detector and a *global* active contour model. They use the balloon model, with an added internal pressure force pushing out the boundary. They use the FEM to take advantage of the subdivisions between the snaxels without the added computationally complexity associated with a large number of elements in the FDM (See Section 3.3).

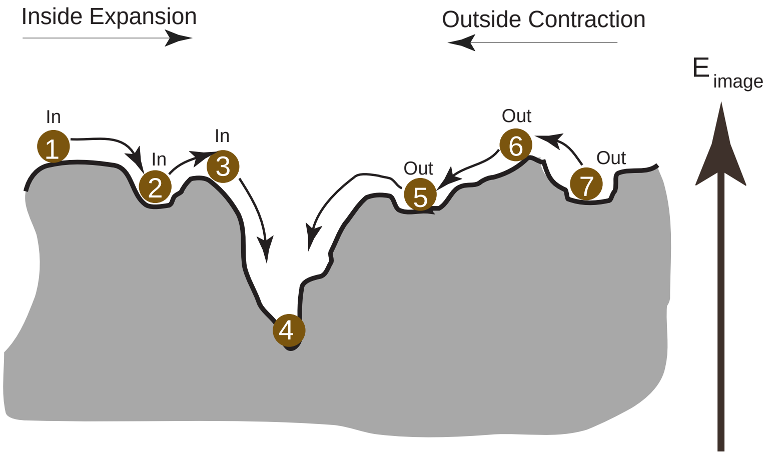


Figure 3-18. As can be seen in this figure, the internally initialised snake is expanding and the externally initialised snake is contracting. From this image energy diagram a number of local energy minima are visible, however the obvious (to a human viewer) optimum minimum is at point 4. When both contours become stationary (in this case the internal snake begins at 1 and the external snake at 7), the contour with the highest energy is minimised with an additional force pushing it towards the other contour. This local snaxel force is always in the direction of the equivalent snaxel on the other contour and derived from the distance between them. This allows the contours to 'climb out' of weak local minima. In this example the inside and outside snakes pull each other until they eventually reach their termination criteria at point 4.

3.7.1 Snakes and Stereo

Snakes can be applied to the problem of stereo matching. If a pair of stereo views with two corresponding contours is available then the disparity should vary smoothly along the length of the contours. A constraint could be added of the form:

$$E_{Stereo} = \left(v_s{}^{L}(s) - v_S{}^{R}(s)\right)^2 \tag{3-29}$$

Where this constraint would cause information obtained by the contour in one view to be passed to that corresponding constrained contour in the other stereo view. Kass *et al*

(1987) used this method to extract a true 3-D representation of a scene containing a piece of paper. Cohen (1991) also applied the balloon model to the extraction of stereo views in medical images with impressive results.

Terzopoulos *et al* (1987) propose a 3-D reconstruction algorithm based on snakes using a monocular sequence of a scene, with the snakes providing silhouettes of the objects of interest. They propose deformable models that are matched to the 2-D images using the external forces derived from those images.

Cohen and Cohen (1993) use MRI images of sequential slices across the left and right ventricles to construct a true 3-D model. On each image slice the 2-D model is applied and assuming that the variation from one slice to the next is very small they use the information from frame to frame. The inflation force that is associated with the balloon model is only used in the first frame, as in subsequent frames, it is assumed that the derived contour is very closely related to the previous contour.

3.7.2 Recent Applications

Schnabel and Arridge (1998), use active contours for active shape focusing, where multi-scale techniques are used. A high scale (very blurred) version of the shape is where a cubic B-spline is initialised. Slowly the scale is reduced and the B-spline is allowed to evolve to the shape, even more accurately using an increased number of snaxels. At each scale the shape and number of snaxels is recorded on a ‘shape stack’, allowing further investigation of shape changes across scale for future use. The internal energy curvature mechanism is relaxed using a non-parametric soft constraint to allow the contours to accurately deform to the desired shape.

3.8 Discussion

Active contour models have provided a uniform treatment to a collection of problems, historically treated differently under Marr’s paradigm (as discussed in Section 3.2.1). They provide an integration of image data, an initial estimate, desired contour properties and knowledge-based constraints, in a single extraction process (Gunn and Nixon, 1997). They also provide a uniform framework for localisation of objects, motion tracking and the stereo approach.

Active contour models can change their role within the scene by providing additional external information in the form of the constraint energy. This allows a higher-level process to decide if the task is being performed correctly or actually add information to find a more localised response.

The snake model has several difficulties according to Leymarie and Levine (1993):

- The snake tends to shrink due to its intrinsic bias towards solutions that reduce its length.
- There are no bounds imposed on the snake, possible leading to chaotic oscillation of the snake.
- The steady-state criterion for the termination of the optimisation is quite strong, possibly leading to oscillations and invalid solutions. An appropriate terminating criterion would allow the snake to find a suitable solution in the shortest possible period.

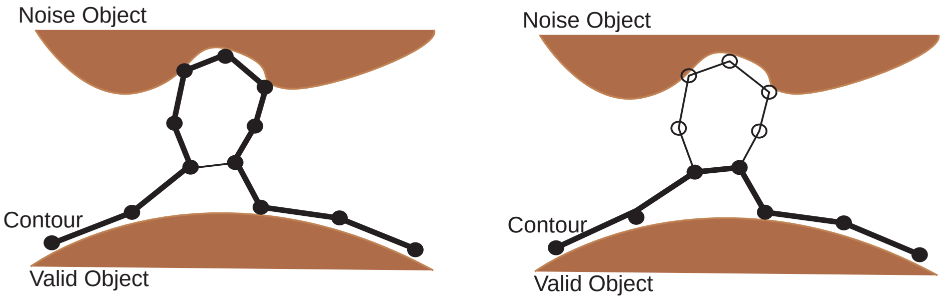


Figure 3-19. Noise reduction in the snake using the technique of Hashimoto *et al* (1994).

Hashimoto *et al* (1994), suggest a useful technique for reduction of noise on the snake, by 'cutting off' sections of the snake that are not useful. In Figure 3-19, it is visible that when non-neighbour snaxels are very close such a condition exists. By removing the snaxels that are between these two neighbouring snaxels the noise is removed from the snake, hopefully allowing the snake to converge more closely.

It should also be remembered that snakes are relatively computationally cheap, allowing them to be combined with other techniques to improve their performance. Active contour models have other advantages over classical feature extraction methods:

- It is a higher-level approach that can help ignore missing boundary information, which is not possible in a local approach.

- Can be made sensitive to scale by adding Gaussian smoothing in the image energy functionals.
- Snakes are self-adaptive, even changing role.
- They are easy to manipulate, using constraint forces, such as springs or volcanoes.
- They can be used for spatial location tasks or temporal tracking with little modification.

However:

- Snakes require a suitable *a priori* knowledge to achieve good results.
- Snakes can become trapped in local minima.
- The accuracy is governed by convergence criteria, which involves more computation, but provides more accuracy.
- Snakes can smooth over minute features.

To solve some of these problems, some methods have tried to combine multiple methods such as Zhu *et al* (1995) that attempt to combine active contours models, region growing and Bayesian (Geman and Geman, 1984) or MDL (Minimum Description Length) for image segmentation using energy function criteria. Each individual method has disadvantages and advantages. Local edge detection cannot guarantee continuous or closed edges. Active contour models only give information along the border and require good initial estimates. Region growing gives image information within the region but often generates irregular boundaries and small holes. Global optimisation techniques such as the Bayes or MDL have global criteria but it is difficult to find the minima.

An approach is suggested in the next chapter that attempts to deal with some of these problems. An active mesh is suggested that:

- Requires little *a priori* information for initialisation.
- Has representation within image boundaries.
- Does not become trapped on local minima.
- Retains the advantages of active contour models, such as multi-scale implementation, self-adaptive ability, force based manipulation, and can be used for spatial location or temporal tracking.

Chapter 4 - Methodology - The Active Mesh Implementation

4.1 Introduction

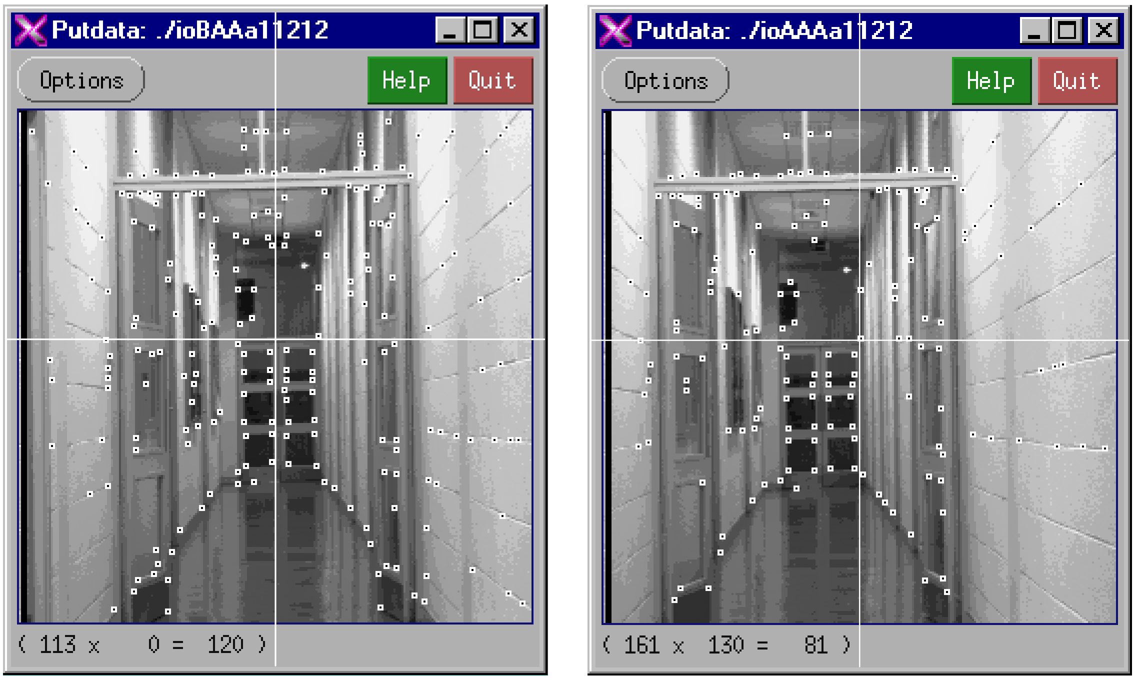
Most of the methods examined in the previous chapter exhibit advantages for use in different situations and applications. It became apparent from an early stage that a combination of techniques would be best suited to developing a solution to the moving camera, moving scene-tracking problem.

4.2 Primary Approach Investigated

The initial approach that was examined was the use of corners as features for feature matching. For motion analysis using feature-matching techniques, the detection of features must be:

- **Consistent**, in that features to be used as features must be detected consistently through image frames, if they are to be used as the basis for subsequent higher level processing.
- **Accurate**, in that these features must be located precisely from frame to frame. This is especially important in structure from motion techniques where an error will be dramatically magnified into 3-D space.
- **Non-complex**, in that computational complexity is a very important issue for real-time applications, in which this primary stage of corner detection must be performed at each iteration of the algorithm.

Edges have the advantage of being more stable than feature points in real-world scenes as they are less affected by lighting conditions. The use of edges was however dismissed, due to the difficulty in curve fitting and the local based aperture problems that can occur. These problems suggested point features as more suitable features for feature matching. As discussed in Section 2.6.2 the SUSAN corner detector was chosen as the primary feature detector based on the work of Smith (1992) and practical implementation has shown it to be a very suitable corner detector for examining motion in image sequences, as it is fast, stable, producing well-localised corners, and also performing well in the presence of image noise (again, partly due to the fact that there are no second order derivatives used). After testing the SUSAN corner detector on sequences of images (such as those shown in Figure 4-1) it was found by hand that corners were consistently detected from one frame to the next. Smith (1995) used this corner detector in a motion tracking application, calculating the optical flow vectors at the corner points. This is discussed in the ASSET-2 approach in Section 2.9.



(a) (b)

Figure 4-1. Real-world example operation of SUSAN – a white grid is superimposed on the image frames (a) and (b) to help demonstrate the scene motion between the two image frames.

After examining the results obtained by using the SUSAN corner detector on the sequence as shown in the two frames of Figure 4-1, it could be seen that many of the corners determined in the first frame had a possible match corner in the subsequent frame, even thought the frame had undergone a substantial translation. A primary algorithm was developed and discussed in Molloy and Whelan (1996) to determine possible matches using the 3x3 pixel area surrounding each corner, template matching and maximum velocity assumptions. The basis of the algorithm was the assumption that when a corner was found in the first frame of an image sequence the surrounding pixel intensities would be similar to that same corner determined in the subsequent frame (this assumption is discussed further in Section 4.4.2). This algorithm was implemented in Khoros[18]. During the implementation stage of this research, the SUSAN algorithm was modified and re-written for Khoros, to take advantage of the visual environment and to allow its integration into a full modular system.

[18] Khoros™, A product of Khoral Research Inc. (http://www.khoral.com)

Resultant vectors calculated from the corner-matching algorithm are shown in Figure 4-2, with the results superimposed on the scene image. The vectors are plotted in the form of a square head positioned on the feature in the initial frame with the tail pointing to the exact location of the matched feature in the subsequent frame. This technique was quite successful, showing all the vectors that were determined between the two frames. The stray vectors represent the inability of a corner at a particular point to determine a possible match corner in the subsequent frame. Approximately 20% of the vectors are ‘stray’ in Figure 4-2 and can be easily removed using a tolerance of match threshold, based on a 3 x 3 intensity patch difference.

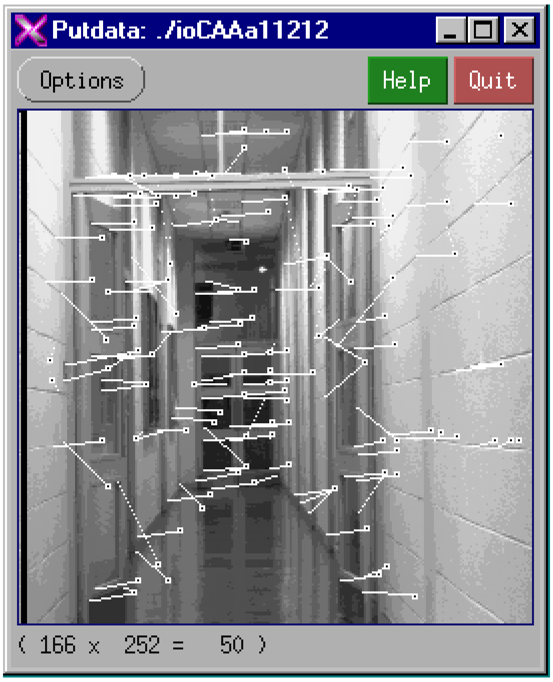
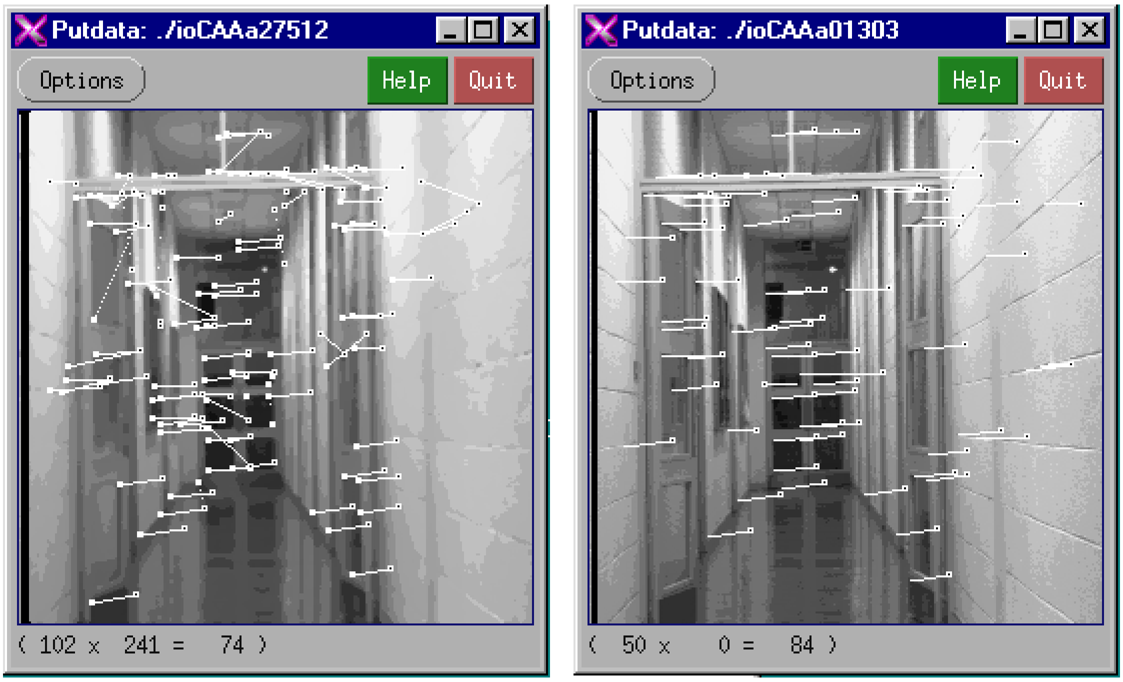


Figure 4-2. The extracted motion vectors using the developed algorithm.

Numerous steps were taken to optimise the corner-matching algorithm, such as:

- A **distance threshold** was implemented that defined the maximum distance that a corner may move between two frames, to prevent the case in which a corner is matched to a corner that is visible in one frame, but subsequently in the next frame becomes occluded, or fails to be detected. This basic measure can be calculated dynamically based on the general movement of the corners in the sequence. When feedback is used, it allows the use of a constraint such as velocity smoothness.
- The addition of a **factor of importance of distance** also improved results in certain cases, where the distance between a detected corner in one frame and in the subsequent frame allowed the discrimination between similar match cases.

- The addition of **a pre-median filter** was also found to improve results. A median filter implemented before the SUSAN algorithm removes some problem texture information, without smoothing over the corners in the image. This was found to remove weak detected corners, and improve the overall results of the matching block. It does however reduce the density of vectors, but if the threshold is lowered, this is not problematic. See Figure 4-3(a).



(a)

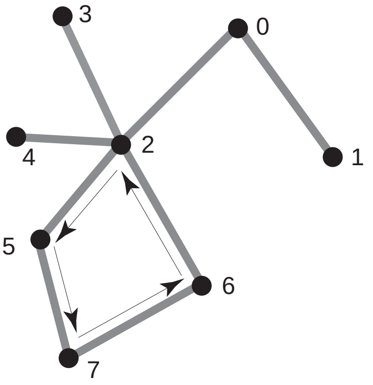
(b)

Figure 4-3. The output after pre-median filtering in (a) and the use of post-median filtering in (b), without pre-median filtering.

- The addition of **a post-median global filter** dramatically improved results, but this was found especially applicable to the motion sequence used (i.e. uniform translation). It was expected to implement a local approach to perform the same filter, without the error propagation of the global approach. The vector field is sparser than the previous results, but using a 'coarse-to-fine' strategy could improve the overall results of the algorithm. See Figure 4-3(b).

More information on these filter algorithms is available in Molloy and Whelan (1996). The global filtering results were promising but especially applicable to this particular case. Under rotation or scaling this form of filtering is difficult to compute and so will fail. It was decided that some form or relationship needed to be determined between the features in the image if a suitable form of global filtering was to be performed. A few options were examined for this:

- A form of relational database based on relative position in the scene. However, due to the presence of moving objects it was not clear how such a database would handle the relationship between multiple moving segments of the database.
- A model approach based on relative position was also considered but due to the non-rigid behaviour of the scene, also proved difficult.



(a)

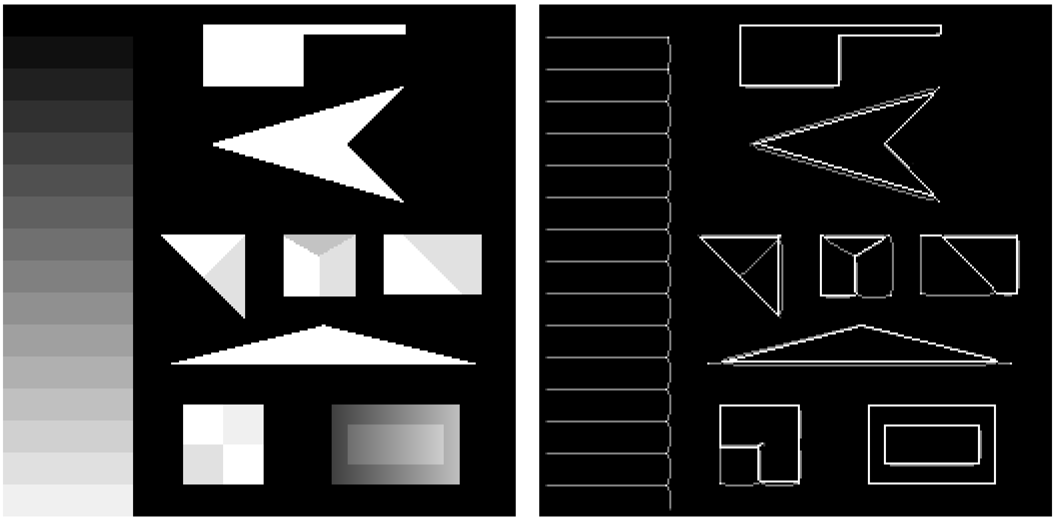
Feature Point	Connected to Feature				
⓪	①	②			
①	⓪				
②	⓪	③	④	⑤	⑥
③	②				
④	②				
⑤	②	⑦			
⑥	②	⑦			
⑦	⑤	⑥			

(b)

Figure 4-4. Developed edge linking algorithm example and extracted table.

Edge linking was examined to determine this relationship. The edge detection algorithm that was used was also based on the SUSAN algorithm of Smith (1992), as it was found to be fast and accurate, with the corner points localised on the edges. It also applies filters to the edge map to attempt to close broken edges in line segments. An edge-linking algorithm was developed to attempt to extract the relative position of the connected corners within the image. This algorithm had the following form (examining Figure 4-4(a)): For each feature point detected (identified in any order, but placed in a sorted list) – follow the edges until each possible direction is found and the linked feature points are identified. A list is created of the form of Figure 4-4(b) and this list is then passed to the next algorithm.

From this table, a routine is then called to follow all the possible paths that may exist. Examining feature ① there is only one path to ⓪, which in turn breaks into two directions towards ① and ②, but since it came from ① it ignores ① and goes to feature ②. At feature ②, it then examines all the directions except in the direction of feature ⓪, so there are four more paths that are possible ③,④,⑤ and ⑥. Each path is examined in turn in the same way. This routine attempts to find closed loops within the path list with greater than two steps (as this would be a line). It also attempts to find the longest path if a closed loop does not exist. The rational for this is so that these contours can be converted into the initialisation of closed and open snakes. This algorithm was quite complex to develop and involved significant computation to calculate the contours. The calculation only has to be performed at the beginning of a sequence, or after significant changes in the scene. It was written in ‘C’ and implemented using Khoros as the GUI front-end.



(a) Standard test template for SUSAN (b) Output of results after detection

Figure 4-5. Detection of closed loops for a test template (loops in brightest intensity).

In Figure 4-5(a), the sample image used by Smith to demonstrate the SUSAN algorithms is shown and the edge detection and following algorithm that was developed is then used to generate the feature list and connectivity list as previously described. In Figure 4-5(b), the output of results from the algorithm is displayed where the closed loops are detected within the image. These closed loops are indicated in the figure by being double the intensity of the edge-linked detection.

On examining the figure, it is clear that more than one loop exists in some of the shapes in the image, however the algorithm only identifies one loop for a given set of feature points. It could easily be altered to determine multiple loops or the loop with the largest number of corner points if required, but was thought unnecessary in this particular case, since more than one contour may not be needed to track a single object in the scenes. Figure 4-6(a) shows the algorithm being applied to a real-world corridor scene, where the edges that are detected are outlined with a white pixel on either side of the point where the edge exists. Figure 4-6(b) shows the closed contours displayed over the edge-linked corner image. Clearly, the windows in the centre of the figure show the clearest closed loops. In other areas of the image closed loops are detected, but are not clearly visible. This is because the number of corner points in the loop is quite low.

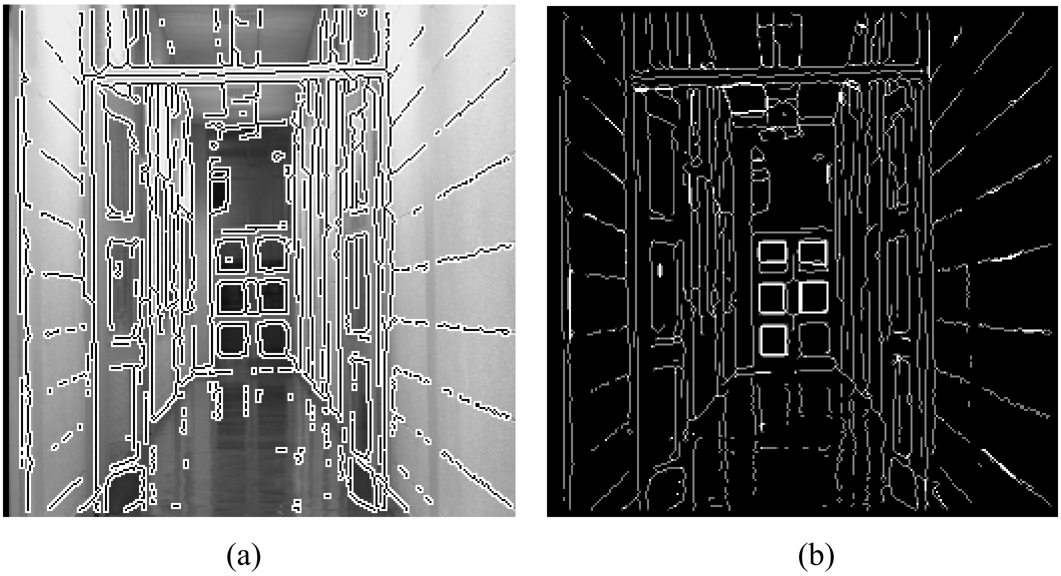


Figure 4-6. Application of this algorithm to detect closed contours in a real-world image. (a) Real-world image with edges superimposed and (b) closed contours displayed brightest on edge image.

Figure 4-7, displays an enlarged simulated view of the initialisation of the closed active contour models from Figure 4-6. The primary snaxels are placed on the detected corner points in the image frame. When the distance between the corner-initialised points is greater than a certain threshold, more snaxels are placed equidistant on the edge-followed path between the points. The snake is allowed to form itself around the edges of the object, by minimising the external energies constrained by image edges.

Once the initialisation is performed, the process of tracking the features is assigned to the multiple initialised snakes. When the number of open and closed active contours in the sequence falls below a certain threshold the algorithm halts, re-initialises a new set of contours for the current frame and then continues. This may occur when multiple contours are lost due to severe occlusion or features that leave the view of the camera. After new contours are initialised, an attempt is made to match the contours extracted in the new initialisation with the contours that still existed at the end of the previous image frame. This method attempted to track multiple features with automated initialisation, however it suffered from several difficulties (Molloy and Whelan, 1997a, 1997b).

- One problem is that the multiple snakes drift into each other and begin to track the same image features. This can be explained by examining Figure 4-7, where a sudden motion jerk to the left or right, can cause the contours surrounding the windows to drift onto each other as they are very similar.

- There is no direct relationship between each individual snake and so the solution is less global than would be expected.
- The process of edge following is erroneous due to gaps in edges, which can be filled to some extent, but not perfectly. One of the main advantages of using active contour models is they are a higher level approach to edge detection. This approach removes that advantage to some extent.

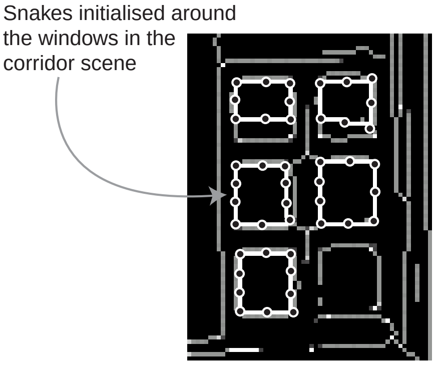


Figure 4-7. Snakes as they are initialised on the real-world image.

Subsequent to the difficulties in this approach, Park and Han (1998) managed to provide a solution for the problem of multiple contours in image sequences. By first comparing the curvature distributions of the contours and matching extreme curvature points, then using these points as a basis along the contour, they match the changes in curvature, with the criteria of minimum curvature differences. The assumption that is made is that curvature distribution changes smoothly, no matter what the contour motion. The contours in Figure 4-7 would still provide difficulty due to the similar curvature of each snake.

4.3 The Active Mesh

4.3.1 Mesh Generation

Mesh generation may be described as the process of breaking up a physical domain into smaller sub-domains in order to facilitate the numerical solution of a partial differential equation. Surface domains can be subdivided into triangle or quadrilateral shapes, while volumes may be subdivided into tetrahedral or hexahedra shapes. Meshing is used in a wide variety of applications, from mathematical simulation to 3-D graphics applications. There are two main forms of mesh, structured and unstructured. Structured meshes (see Figure 4-8) have regular connectivity (i.e. the same number of neighbours), such as simple, boundary fitted and multi-block, whereas unstructured meshes have irregular connectivity such as advancing front and Delaunay meshes.

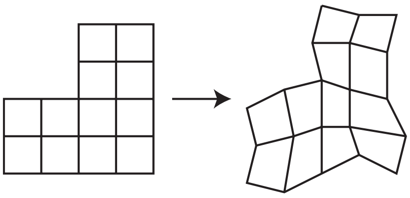


Figure 4-8. Multi-block structured mesh.

4.3.2 Voronoi Diagrams

The Voronoi diagram, named after mathematician M.G. Voronoi, is widely used in geometrical construction. These diagrams, sometimes called the Dirichlet or Thiessen tessellation, represent the regions of a plane, which are closer to a particular point in the plane than any other point. If $P = \{p_1, p_2, \ldots, p_n\}$ is a set of points in the 2-D Euclidean plane, where partition in the plane is developed by assigning each point to its nearest site then those points assigned to the site p_i form the Voronoi region $V(p_i)$, where $V(p_i)$ represents all the points at least as close to p_i as any other site.

$$V(p_i) = \{x : |p_i - x| \leq |p_j - x|, \forall i \neq j\} \qquad \textbf{4-1}$$

Some of these points do not have a unique nearest site and are called Voronoi edges or vertices. In Figure 4-9 the Voronoi polygon is displayed, defined by the lines that bisect (perpendicularly) the centre of the lines joining a site p_i and its surrounding sites. After applying this rule to every site in the area, an area remains that is completely covered by adjacent polygons. The polygons at the boundary of the area are ‘open’ because they have no neighbouring sites in that direction.

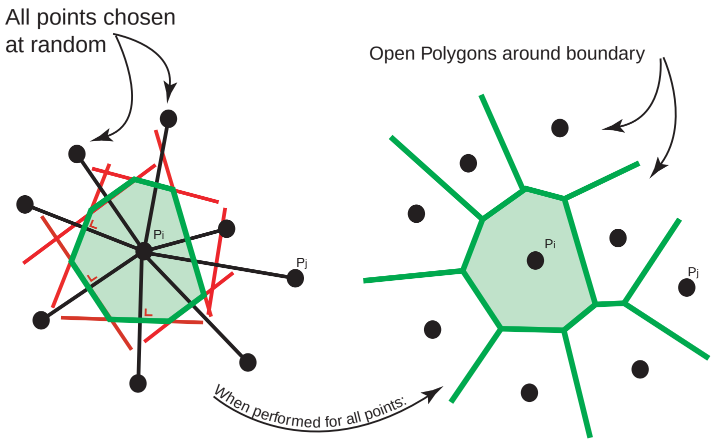


Figure 4-9. The construction of a Voronoi polygon and the Voronoi diagram for some random scattered points.

4.3.3 Delaunay Triangulation

Given a set of data points, the Delaunay triangulation produces a set of lines connecting each point to its natural neighbours. Delaunay triangulation is very closely related to the Voronoi diagram. The Voronoi diagram can be used as a basis for constructing the Delaunay triangulation by drawing lines between the points in adjacent polygons. Delaunay proved that when the dual graph of a Voronoi diagram is drawn with straight lines, it produces a planar triangulation of the Voronoi sites. This can be seen in Figure 4-10.

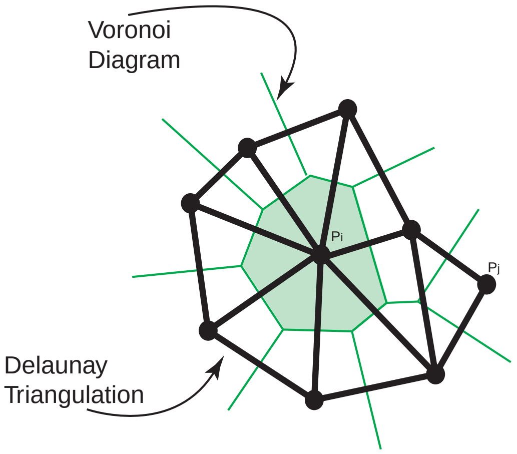


Figure 4-10. The relationship between the Voronoi diagram and the Delaunay triangulation.

It may not be suitable in many cases to determine the Voronoi diagram before determining the Delaunay triangulation. This is especially true in the case of implementation using computer programming, as the Voronoi diagram is often more complicated to construct than directly determining the Delaunay triangulation. A Delaunay triangulation is desirable for approximation applications because of its general property that most of the triangles are almost equiangular and also because there is a unique triangulation for a given set of points. The main rule for the Delaunay triangulation may be formulated as: *A Delaunay network in two dimensions consists of non-overlapping triangles where no points in the network are enclosed by the circumscribing circles of any triangle*. The basic properties of a Delaunay triangulation *D(P)* are:

- *D(P)* is the straight line dual of the Voronoi diagram *V(P)* (by definition).
- *D(P)* is a triangulation if no more than three points of *P* are co-circular.
- Each face of *D(P)* corresponds to a vertex of *V(P).*
- Each edge of *D(P)* corresponds to an edge of *V(P).*
- Each vertex of *D(P)* corresponds to a region of *V(P).*
- The boundary of *D(P)* is the convex hull of the sites.
- The interior of each face of *D(P)* contains no sites.

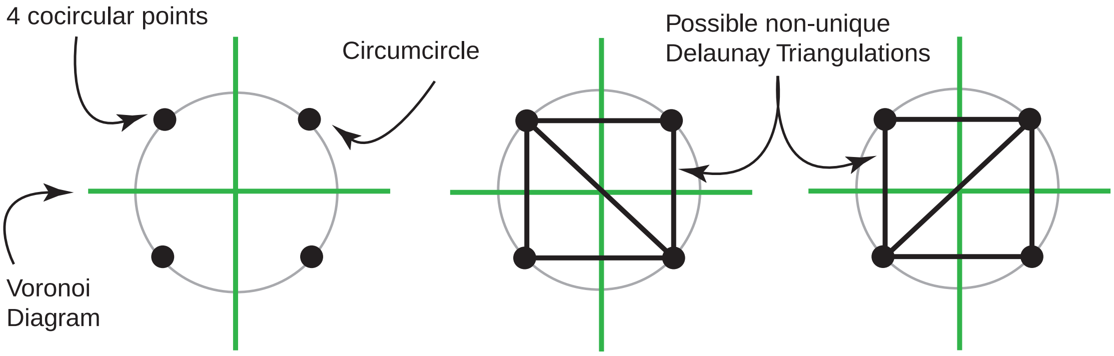


Figure 4-11. Degeneracy in the Voronoi diagram with four co-circular points.

If only three points exist on each circle in the mesh, the Delaunay triangulation is unique, however, if more than three points exist on each circle the mesh is still a Delaunay triangulation, but not unique (see Figure 4-11).

Classical algorithms for constructing the Delaunay triangulation may be classified as either incremental or divide-and-conquer algorithms. The incremental algorithms construct the triangulation by starting with a single point and then adding the remaining points step-by-step. The divide-and-conquer algorithm recursively splits the data set into equally sized subsets until triangles are obtained. Incremental algorithms have a maximum $O(n^2)$ time complexity, while the divide-and-conquer algorithms have an optimal *O(n log(n))* time complexity (Brown, 1995). The incremental algorithm was chosen, so that nodes could be added to the active mesh dynamically.

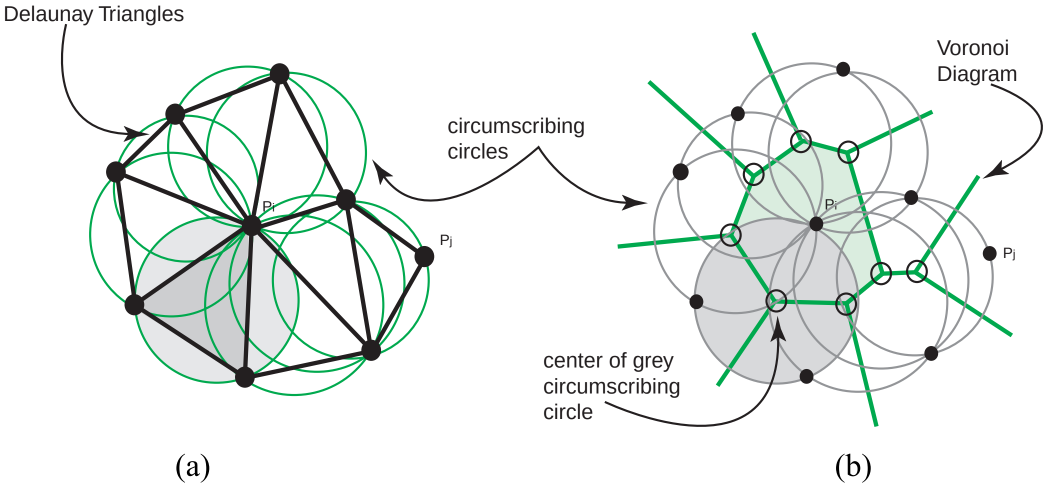


Figure 4-12. (a) The relationship between the Delaunay triangles and their circumscribing circles, and (b) the relationship between the Voronoi diagram and its circumscribing circles.

4.3.4 The Incremental Algorithm

Unlike many other algorithms for determining the Delaunay triangulation, the incremental algorithm has the main advantage of keeping the triangular network as a Delaunay triangular network during the actual triangulation process. The initial starting point of the algorithm is a single triangle (and so valid) and each point is included in the network at each iteration, re-organising the network until the circle criterion is met for all triangles in the network.

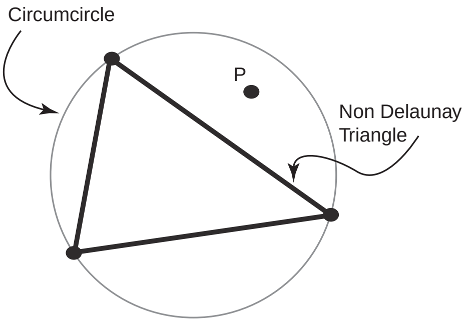


Figure 4-13. The triangle is not a Delaunay triangle since a point *P* lies within its circumcircle.

- The initial triangular network is created, so the convex hull is used for this purpose (see Figure 4-15). The triangular network must meet the max-min angle criterion as given in Figure 4-14 and the circle criterion as given in Figure 4-13.

- The first point in the interior of the network is included. This point is then connected to its enclosing triangle by three new edges, as in Figure 4-15(b).
- The quadrilaterals that have the previous edges as their diagonals are tested using the max-min angle criterion – where if they do not meet the criterion then their diagonals are swapped (as in both quadrilaterals in Figure 4-15(c)). If they are swapped then the new opposite edges to the inserted point will be recursively examined as diagonals in their quadrilaterals (see Figure 4-15).
- The Delaunay network now contains one more point and is a true Delaunay triangulation. Any further points are then added in the same way.

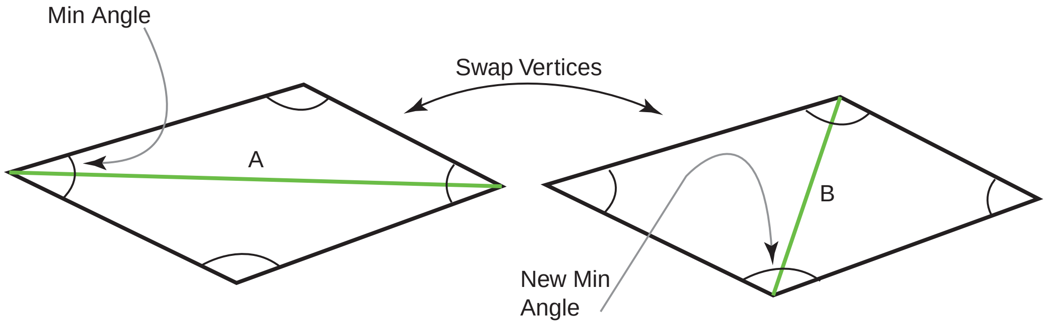


Figure 4-14. The max-min angle criterion for a Delaunay triangulation, states that; if the diagonal of any strictly convex quadrilateral formed by any two adjacent triangles of a Delaunay triangulation is replaced by the opposite vertex, the minimum angle of the six internal angles does not increase. In this figure the vertex *A* is replaced by the vertex *B* to give the maximum value for the minimum angle.

Several different forms of the incremental algorithm were researched according to the work of Shewchuk (1997):

- The Bower-Watson algorithm - which starts with a large triangle enclosing all the points to be triangulated. Each point is added, one-by-one to the triangulation. The algorithm finds and deletes all existing triangles, where the circumcircle is intersected by the new point and joins this point to all the vertices on the boundary of the cavity created. This algorithm has a worst-case complexity of $O(N^2)$, but also has the problem that all point positions must be known in advance.
- Green and Sibson's algorithm - an efficient approach that uses Voronoi regions, where each region has a list of its boundary neighbours. Each point is added point-by-point to the region and the new point ‘wins territory’ from the list of neighbouring points. This updated region adds all the neighbours to its list and

the neighbours add the new point to their lists. The overall level of complexity can be as low as *O(N log N)*.

- De Floriani's algorithm - does not make the same assumptions about a convex hull being present as many other algorithms and is mainly applied to terrain modelling. The algorithm is based on a software stack approach and takes advantage of the *push* and *pop* type operations where the triangle that encloses the new (iteratively added) point is identified. This triangle is split into sub-triangles around this point. The new edges are then examined for validity under the Delaunay criteria, being modified (swapping vertices etc.) until all points are added.
- Guibas and Stolfi's algorithm - give another Delaunay triangluation algorithm that has divide-and-conquer and incremental versions. This approach is conceptually quite difficult but quite easy to implement.

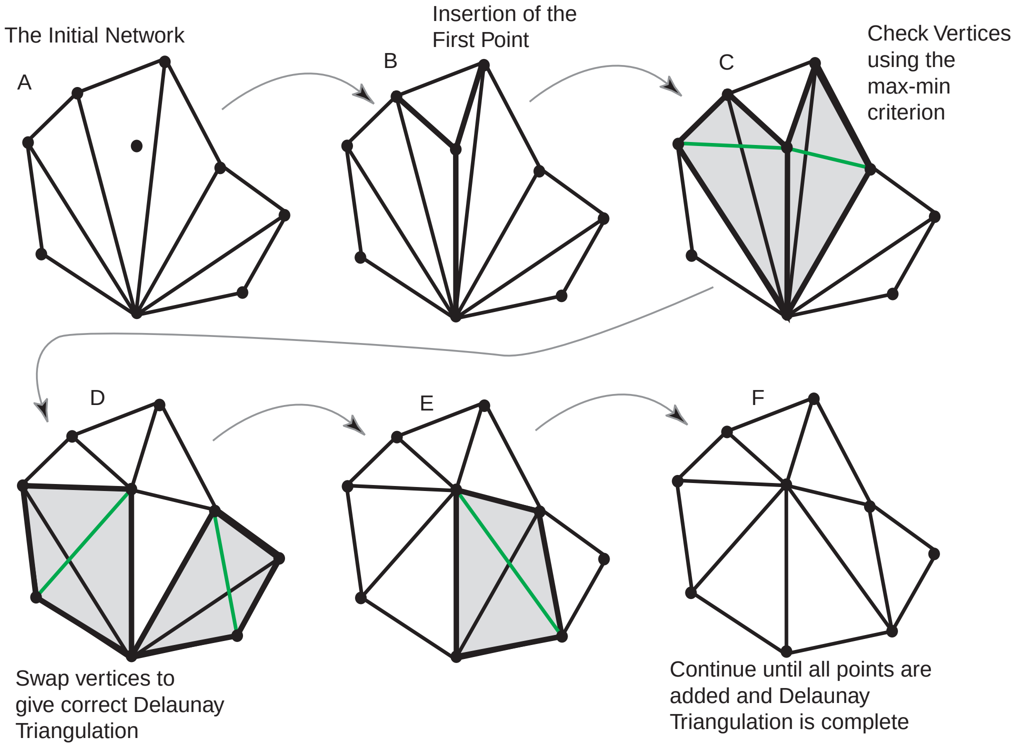

The Initial Network
A
Insertion of the First Point
B
C
Check Vertices using the max-min criterion
D
E
F
Swap vertices to give correct Delaunay Triangulation
Continue until all points are added and Delaunay Triangulation is complete

Figure 4-15. The creation of the Delaunay triangulation using an incremental algorithm.

The Delaunay triangulation that was developed for this implementation was based around a contraction of the *De Floriani* and the *Bower-Watson Algorithm*, taking advantage of the Vector class structure of Java. The algorithm has tested to be very stable but was not developed to be as computationally efficient as possible. The Bower-Watson algorithm was not used entirely as the assumption is made that all the points to be added are surrounded by a single large triangle. This assumption could not be made for this application, as the location of the points was not known in advance. The Green and Sibson algorithm was not used, as it requires an intensive use of the Voronoi regions, which are difficult and computationally intensive to calculate. The De Floriani algorithm and Guibas and Stolfi algorithm were not used entirely as the convex hull was to be used as part of the mesh and because of their difficulty of implementation. It is however very likely that some of these algorithms are more computationally efficient than the algorithm used and could be implemented at a later stage as an implementation improvement. It is worth mentioning that the mesh generation stage is only carried out once at the start of the sequence and points are only added and removed as the sequence progresses. The current algorithm developed in Java executes in 180ms for a mesh of 300 nodes[19]. The transition to Java as the development environment is discussed in Section 4.12 .

4.3.5 The use of Delaunay and Voronoi in Computer Vision

There have been several different applications that use the Delaunay and Voronoi methods in computer vision applications to-date. Guan *et al* (1992) use the Voronoi technique as a measure in place of the more commonly used Euclidean distance (Euclidean skeleton) using segments, rather than points as the basic units of operation. A distance map can be easily created from the segment list representation of the Voronoi diagram. Bowden *et al* (1997) use the Delaunay triangulation along with the Balloon model of Cohen (1991) as the basis of a volumetric reconstruction algorithm. An initial triangular mesh is placed within the boundary of the object and allowed to expand in all directions until it is constrained by the boundary of the object. Importantly, it uses no explicit attraction forces to image features, but is constrained to stay within the boundary of the object.

Bowden *et al* (1997) suggest a novel curvature based subdivision approach (see Figure 4-16) that aims to maximise the number of snaxels in the areas of high curvature on the snake. This approach allows the segmentation approach to perform better than possible with a uniformly distributed mesh.

An example operation of this approach is given in Figure 4-17, where the initial contour is allowed to expand under the balloon energy model, giving the approximation to the

[19] On a Pentium Pro II 233MHz using JIT compilation (Sun Java JRE1.2 using native threads)

object as shown. At the edges of the object border the number of snaxels increases as described to allow a closer approximation to the shape. A 2-D application is shown here, but the algorithm works with 3-D objects also.

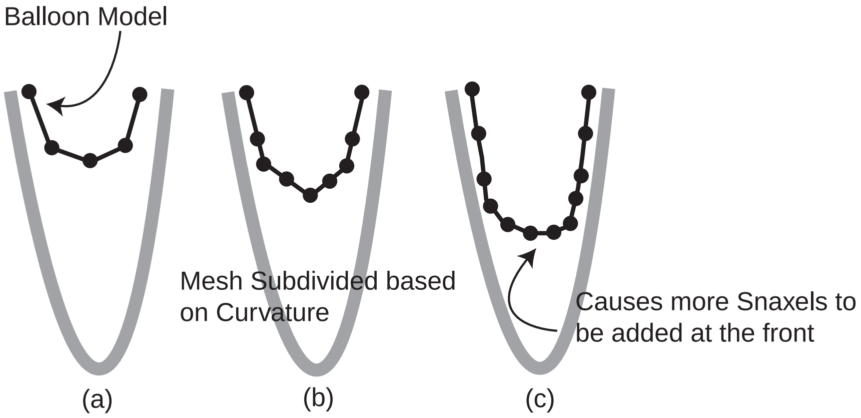

Balloon Model
Mesh Subdivided based on Curvature
Causes more Snaxels to be added at the front
(a)
(b)
(c)

Figure 4-16. Curvature based subdivision, as suggested by Bowden *et al* (1997).

The Delaunay triangulation has also been used in numerous video coding applications where an approximation to image regions with similar intensities is performed using a Delaunay triangulation, with the approximation being improved by increasing the mesh complexity.

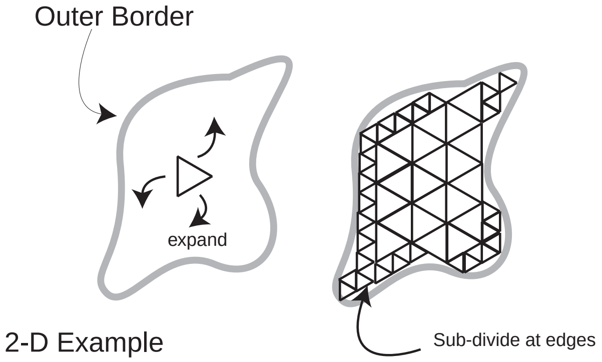


Figure 4-17. Volumetric Segmentation approach of Bowden *et al* (1997).

Another such active mesh implementation is that of Wang and Lee (1994). Their approach uses a quadrilateral structured mesh that is applied to video coding. The energy minimisation approach that is used allows non-uniform area samples of image sequences to predictively describe that image sequence. They do not apply this mesh to motion tracking or estimation, but do however suggest that an active mesh representation would be a suitable technique. Sclaroff and Isidoro (1998) suggest ‘Active Blobs’ as a region-based approach to nonrigid motion tracking. They use a mesh model, from which they capture the object shape and the colour texture map that describes the objects image properties. They then apply an energy based minimisation technique, based on the reverse projection of the texture information onto the image, taking advantage of OpenGL hardware architecture. The bounding area of the 'active blob' is drawn by hand and has been shown tracking and deforming to confectionary item wrappers.

4.3.6 Discussion on the Mesh Structure Used

The Mesh structure that is used in this algorithm is based around an iterative Delaunay triangulation algorithm, adding each point, point-by-point, and structuring the mesh, until all points are added. Once the mesh is constructed using the triangulation algorithm, it is then initialised as an Active Mesh. This was initially attempted using an edge following algorithm, but proved difficult due to problems with incomplete edges and image noise.

In this implementation the mesh lines have no image feature relationship, i.e. the mesh lines do not correspond to feature edges or lines within the image. However, physical representation is not required for motion tracking using this approach. In this algorithm the energy calculations take place only at the mesh nodes (vertices) and the mesh lines represent the spatial relationship of these nodes. As can be seen in Figure 4-19(a) at the bottom centre of the image, there are numerous mesh lines with no possible connected image features. However, in the bottom left of the image the mesh lines follow closely the edges of the object. In this mesh structure:

- The spatial relationships of multiple objects are apparent and can be described by the relationship and connections of the mesh lines.
- The mesh can provide reasonable region estimation since the points used for creating the mesh are on the boundaries of objects. If a more defined mesh is required then the number of feature points can be increased.
- The mesh is bounded by a convex hull that may be defined in advance of the algorithm to have a certain size or position (i.e. the mesh can be initialised within a certain region of the image frame, see Section 4.8).

If the mesh had physical representation within the image, it is possible that it could aid in helping to deal with occlusion. The mesh lines could represent physical image edges, and it would be likely that the various objects in the scene could be completely bounded by a subset of the mesh.

The mesh is constructed as a set of triangles, each triangle *referencing* 3 different mesh nodes and 3 different mesh lines (see Figure 4-18). The referencing is performed to allow a node's properties to be altered without having to locate that node on each triangle or alter a triangle. Instead the reference is modified and so updated on each triangle without having any prior knowledge about each triangle. All the nodes exist in a Java *vector list*[20] of nodes, where a force may be applied to each node without having to apply the force to the individual triangles. The lines also exist in a separate vector list, where each line again references two nodes.

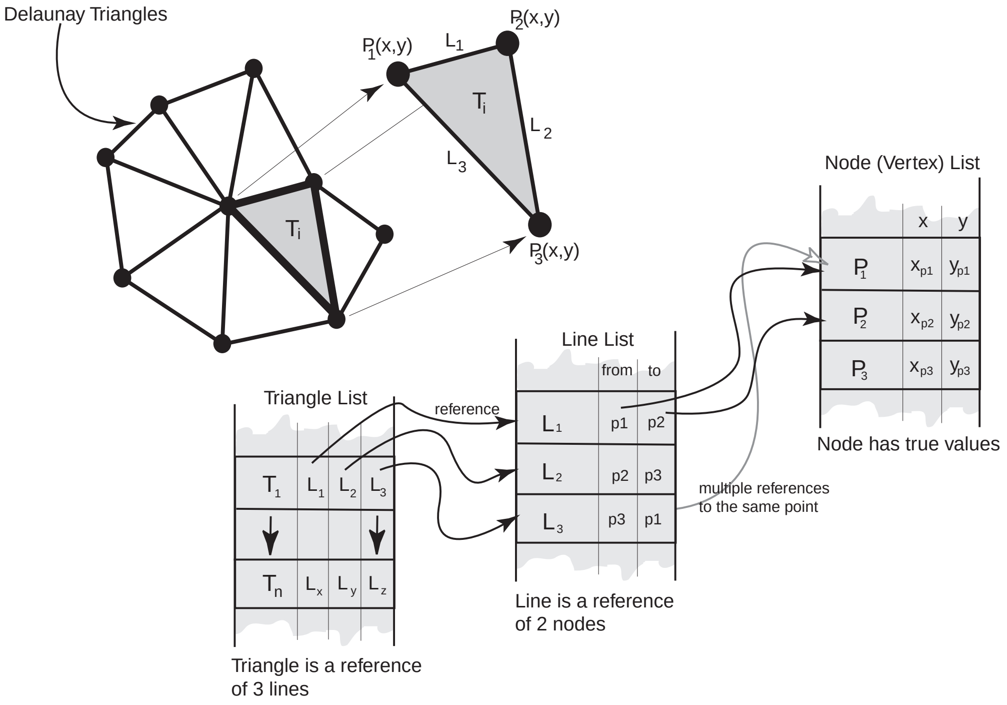
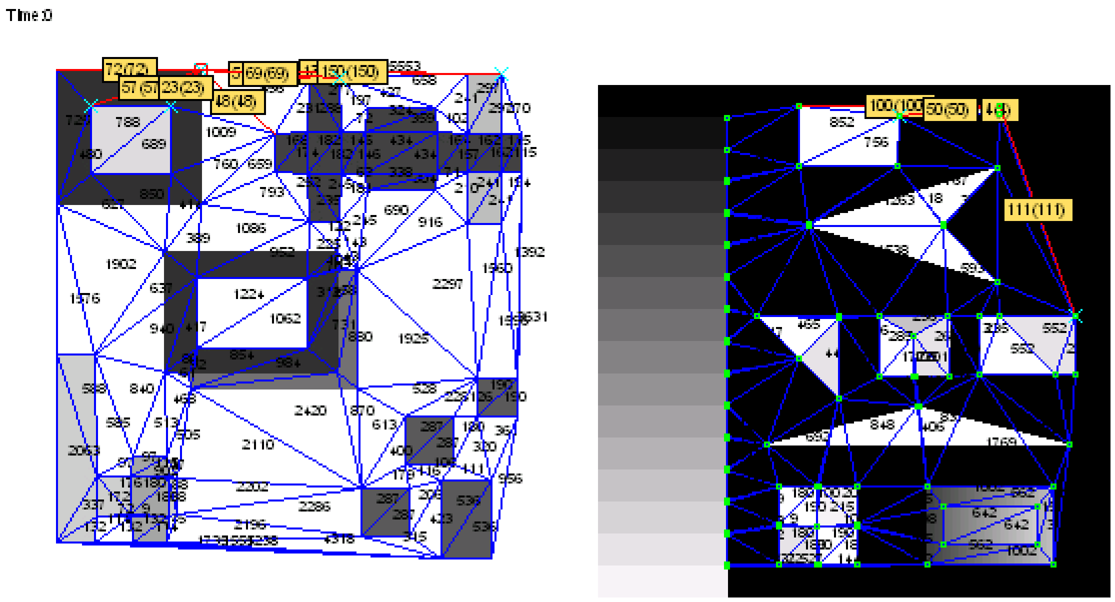


Figure 4-18. The structure of the mesh used, showing the mesh triangles broken down into lines and nodes at the top of the illustration. The equivalent software structures are shown below the physical mesh structure.

This independence of data objects allows forces to be applied to nodes, mesh lines to be updated or modified and triangles to be altered independently. This gives efficiency in the overall algorithm, as there are no search processes to run to find particular nodes or vertices. For example, to update all the node forces the code is simply outlined as *for all nodes 1 to N, update forces*, rather than *for all triangles, find the triangle mesh lines, for all of these lines, find the two mesh nodes, for each mesh node update forces*.

[20] A vector list in Java is a way of storing unformatted objects, in a link-list like structure, only more carefully and in a more structured manner. The Vector class is used to replace stack operations that would likely be required in other language implementations of the algorithm.

One of the main advantages of the incremental mesh algorithm is the way that mesh vertices can be added or removed from the mesh as required. This may be necessary when points in the mesh leave the viewable area of the image frame or when particular features disappear for a number of frames due to occlusion or image noise. The way in which the mesh is currently structured allows this to be performed without a large effect on the structure of the mesh. Figure 4-19 displays the Delaunay triangulation algorithm being performed on two test templates, using the SUSAN corner detector to locate mesh nodes.



(a) (b)

Figure 4-19. The template images after the mesh generation has been performed, using the Delaunay triangulation adding the corner points iteratively to the mesh. (a) Illustrates the test template used for testing and (b) illustrates the generation of a mesh from the SUSAN test Image (from Smith (1992)). The numbers in the triangles represent the pixel areas of the triangles. The currently selected node is highlighted in red and the lengths of the lines given in yellow boxes.

4.4 The Force Equations

The method presented here initialises the mesh using feature points extracted from the initial image frame to provide the initial node locations of the mesh. Not only is it necessary to place the snake in its initial position, the force equations must be initialised such that the mesh is created in a state of equilibrium.

Once the mesh structure is available the internal and external forces within the mesh are established at each node and mesh line, so that the mesh is initially in equilibrium with no internal structure or external image forces being applied. The mesh will then remain in equilibrium until a change in the underlying image structure occurs as the ‘active-mesh’ is designed so that it deforms in response to salient image features.

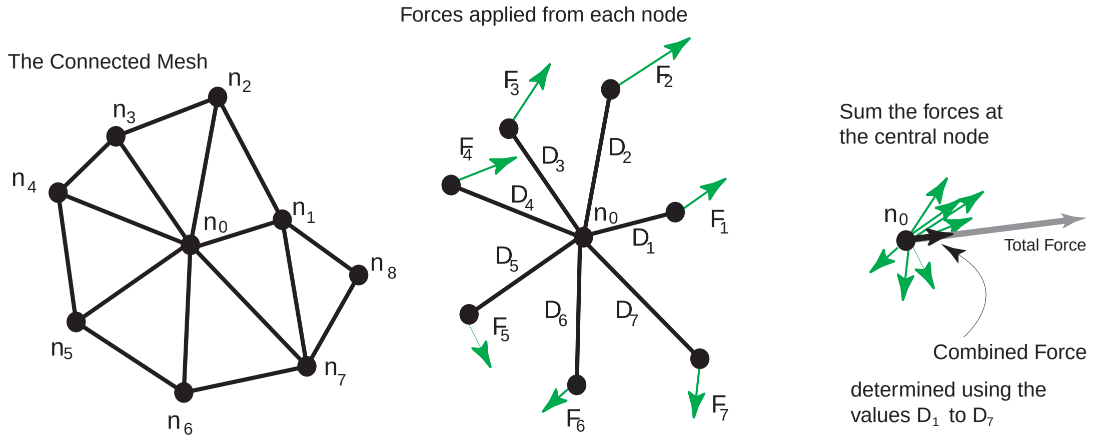


Figure 4-20. Multiple forces being applied to a single node, the combination of which is calculated using the force strength and the node distance of the forcing nodes from the centre node.

To allow for the complexities involved in dealing with ‘active-meshes’ as opposed to regular contours, the mesh algorithm must allow for varying numbers of line connections to each node and provide a method for dealing with the numerous forces that will be applied at each node. Originally it was decided for each connected node to have an equal effect on the central node, however, it was discovered experimentally that this was not suitable. It was found that connected nodes at a greater distance should have a smaller structural impact on the central node, since distant nodes could in fact exist on different objects. The closer connected nodes are more likely to exist on the same object or image region. For this reason a formulation was developed to allow greater weighting on the forces caused by the closer connected nodes (see Figure 4-20).

This algorithm was developed to deal with the varying number of connected nodes, resulting from the initialisation algorithm, with a method of combining the effects of the forces from the connected nodes on a particular node. Examining the centre node $n_0(x,y)$ with connected nodes $n_1, n_2, \ldots n_N$ the combination is of the form:

$$\beta_i = 1 - \frac{D_i}{\sum_{i=1}^{N} D_i} \text{ where, } D_i = \sqrt{(n_0(x) - n_i(x))^2 + (n_0(y) - n_i(y))^2} \qquad 4\text{-}2$$

$$F_0(x) = \sum_{i=1}^{N} \beta_i F_i(x) \qquad 4\text{-}3$$

$$F_0(y) = \sum_{i=1}^{N} \beta_i F_i(y) \tag{4-4}$$

where F_0 is the force on the centre node. The larger the distance of the node applying the force from the current node, the smaller the effect it has on the movement of the current node. These forces being applied from the surrounding nodes can be caused due to those nodes being pulled away from or towards salient image features, or indeed from a structural correction of the mesh. The algorithm to compute the forces on all the nodes is given as in Algorithm 4-1.

```
For each Node
    For each MeshForce Stored at this Node
        ForceLengthSum = ForceLengthSum + ForceLength
    For each MeshForce Stored at this Node
        $ForceFactor = \left(1 - \left(\frac{ForceLength}{ForceLengthSum}\right)\right)$
        SumForceDx = SumForceDx + ( ForceFactor x ForceDx)
        SumForceDy = SumForceDy + ( ForceFactor x ForceDy)
```

Algorithm 4-1. The node force combination algorithm

For each node in the vector node list the total sum of all the lengths of the lines separating the connected nodes causing the forces is calculated. This is the basis of the *ForceFactor* value calculation, allowing forces applied from nodes that are closer to the current node to have a greater impact on the current node. Figure 4-21 shows an example of the way in which multiple nodes might have an effect on the single node at the centre. Each force represents an instance of the *MeshForce* class with *dx* and *dy* representing the *x* and *y* components of the force to be applied. The *length* state of the class represents the distance that the force is being applied from. In the same figure the numerous forces are shown that have to be applied. At this stage the forces are stored, without being applied or combined in a list at the node. Algorithm 4-1 is used to sum these forces together after calculating the *ForceFactor* for each force and then summing the forces based on this force factor.

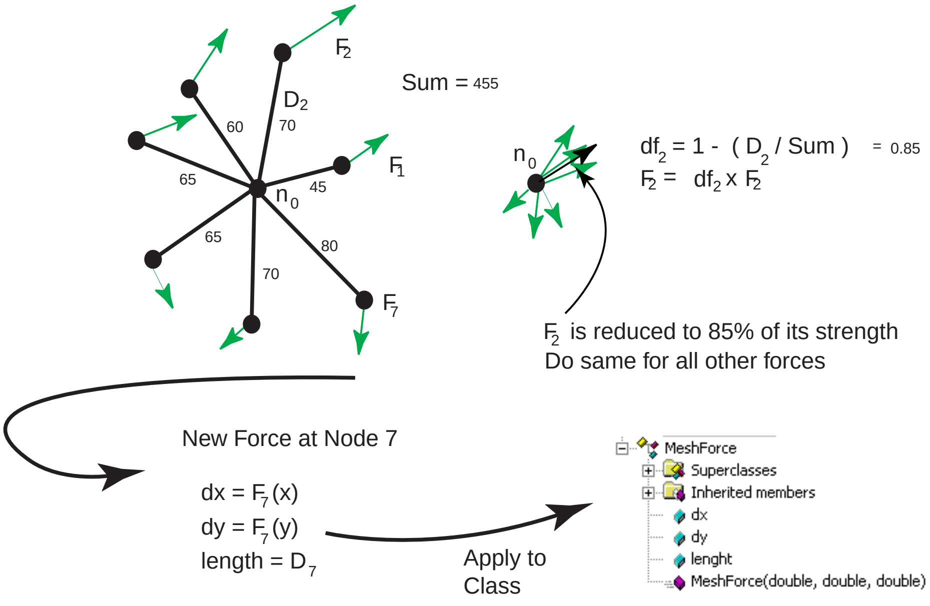


Figure 4-21. An example combination of forces, with relation to the Force class.

4.4.1 The Internal Forces

For the internal forces a rigid/elastic formulation is used, where every node is pulled or pushed by the connected nodes. The mesh-lines have an elastic property so that the mesh can deform when required over a number of time iterations, to track deformations in the scene, the scene objects or deformations due to scaling. There is no requirement for the higher order ‘rod’ term associated with snakes, as they would have no relation to the structure of the active mesh. The elastic properties give the mesh its flexibility while the rigid properties give the mesh structure. The rigid properties of the mesh lines cause the lines to attempt to return to their determined length. This determined length is permitted to expand or contract slowly over a time period, and is influenced by the elastic properties of lines. In other words, if the mesh is stretched by a number of consistent external forces for a significant number of iterations then the mesh will slowly assume a new default shape. This default shape is now the rigid shape of the mesh and will remain so, until similar forcing conditions arise. For each line in the mesh:

$$F_x = L(x)_{cur}\left(\frac{L_{set} - L_{cur}}{\alpha_{Line} L_{cur}}\right) \tag{4-5}$$

$$F_y = L(y)_{cur}\left(\frac{L_{set} - L_{cur}}{\alpha_{Line} L_{cur}}\right) \tag{4-6}$$

where *L(x)*, *L(y)* represent the *x* and *y* components of the mesh line lengths. The internal forces are determined by the current-length of each individual mesh line in comparison to the set-length of that mesh line. The constant α_{Line} is a factor that is user definable in the algorithm and represents a factor on the size of the forces to be applied. The mesh line has two node references n_1 and n_2 where,

$$F_{n_1} = \left(-F_x, -F_y\right) \text{ and } F_{n_2} = \left(F_x, F_y\right) \qquad 4\text{-}7$$

At each iteration:

$$L_{set} = L_{set} + \alpha_I \left(L_{cur} - L_{set}\right) \qquad 4\text{-}8$$

where α_I is the user-defined factor for mesh line deformity as displayed in Figure 4-22.

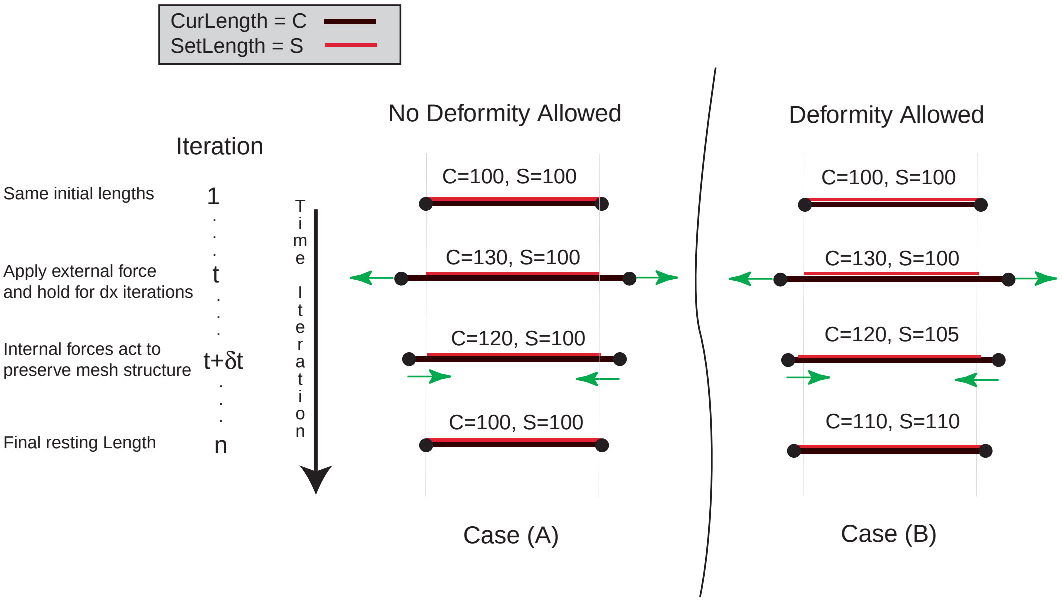


Figure 4-22. An example of a mesh line deformity in (a) how a rigid mesh (i.e. $\alpha_I \cong 0$) might react and in (b) how a deformable mesh (i.e. $\alpha_I > 0$) might react.

The algorithm for the internal energy calculation is given as in Algorithm 4-2. Each line in the mesh is examined individually. The *setLength* of the line is the rigid component of the structure of the mesh and the *curLength* is the current length of that line. When the *curLength* is equal to the *setLength* then there is no need to apply forces to the nodes to reduce the difference between these two values. If there is a difference between these lengths then there is a need to apply forces to the two connected nodes to cause these two lengths to equalise.

If the mesh structure is rigid then the *setLength* would be ‘set’ at the initialisation of the mesh and would remain constant at each time iteration. In Figure 4-22(a) an example is given of these two lengths on the line at (*iter.1*) the initialisation point of the mesh, where the *setLength* is equal to the *curLength* and the mesh component line is stable. However in (*iter.t*) if two opposite forces are applied for a short number of iterations δt (where $\delta t \geq 1$ iteration) to the nodes of the line causing the *curLength* to stretch to a larger value then in (*iter.t*+δt) the line will shrink until the two values are equal as in (*iter.n*).

```
In each iteration

For each Line
        Get curLength, setLength, toNode, fromNode
                $Force = \left( \frac{setLength - curLength}{\alpha \times curLength} \right)$
        tempForceDx = Force x (toNode.X – fromNode.X)
        tempForceDy = Force x (toNode.Y – fromNode.Y)
        toNode.addForce(tempForceDx, tempForceDy, curLength)
        fromNode.addForce(-tempForceDx, -tempForceDy, curLength)

For each Node
        Compute Forces on curNode using Algorithm 4-1
        tempForceDx  = forceFactor x curNode.Dx
        tempForceDy  = forceFactor x curNode.Dy
        curNode.Push(tempForceDx, tempForceDy)

For each Line
        lengthDifference = curLength - setLength
        setLength = setLength + (lengthFactor x lengthDifference)
```

Algorithm 4-2. Internal Energy Algorithm.

The method of shrinking is shown as at the start of Algorithm 4-2. If on the other hand the mesh is allowed to be deformable to some extent as in Figure 4-22(b) then applying the same force to the same length segment (*iter.t*) may cause the *setLength* of the mesh to expand as in (*iter.t*+δt), so that while the *curLength* will eventually equal the *setLength* the line will have expanded slightly at (*iter.n*). This formulation allows the structure of the mesh to be deformable, but still defined by a time factor, so that the mesh cannot deform too quickly. This is similar to the method of the dual active B-spline method, discussed previously in Section 3.7. This allows the prevention of a noisy or inaccurate measurement from altering the structure of the mesh, while still allowing deformation of the mesh in the case of stable, consistent forces.

4.4.2 The External Forces

The external forces are applied to the mesh nodes independent of the mesh lines and are derived from the image data. These image forces pull the mesh nodes towards suitable feature match points that are found within the circular image search space of the mesh nodes. If a suitable match feature appears within the circular search space then the node is pulled towards that feature point by a force magnitude determined by the suitability of the match feature. This in turn pulls the connected mesh nodes (due to the internal forces of the interconnecting mesh lines) in the direction of the new feature. There is a choice of several matching techniques that have been implemented:

Correlation Matching

One of the implemented matching algorithms, the best match corner, is found by comparing the 5x5 area surrounding the current node *n₀* with the 5x5 area surrounding the possible match corners *cₙ* detected within the circular search space of radius α_S. So, $\forall c_n(x, y)$ where the distance from n_0 the current mesh node,

$$d = \sqrt{(c_n(x) - n_0(x))^2 + (c_n(y) - n_0(y))^2} \qquad 4\text{-}9$$

is less than the search space of radius α_S, the total intensity difference is:

$$I_T = \min_{n_0} \sum_{i=-1}^{1} \sum_{j=-1}^{1} \left| I\left(x_{n_0} + i, y_{n_0} + j\right) - I\left(x_{c_n} + i, y_{c_n} + j\right) \right| \qquad 4\text{-}10$$

For a 5x5 area the corner point c_n is chosen that minimises the value of I_T in the range 0 to 6375 (i.e. 255 x 25 pixels). Based on this intensity difference, match strength is established. The larger this value the weaker the match strength and a factor is established:

$$S_M = 1 - \left(\frac{I_T}{25 \times 255} \right) \qquad 4\text{-}11$$

A larger value of I_T implies a weaker intensity patch match, averaged over 25 pixels and over the maximum intensity value 255, so $S_M \in [0,1] = 1$ for the best match and 0 for the worst match.

The method of correlation of small intensity patches is a popular method used in tracking applications. However, the theoretical justification of using the correlation-based method in this method is weak as the features are often formed at the physical corners of objects, where the background is moving and will occupy a large proportion of the patch. This may lead to a poor correlation between the background area components of the intensity difference patches, and indeed for a corner with high curvature on the border of an object the background may occupy as much as ¾ of the total area.

Feature Structure Method

Another implemented method uses the information about each feature derived from the feature extraction process. If it is possible to have a three or four parameter description of the feature rather than needing 25 differencing operations, the computation aspect may be reduced. This is especially important at the matching stage, as this is the aspect of the algorithm that is performed frequently, at each feature in the surrounding search-space of each mesh node, at each iteration of every frame.

Some of the feature parameters that were examined (particularly for corner features) were:

- The **centre of gravity** (COG) of the feature. In the case of corner features, this will not occur at the corner point and gives information about the direction of the corner and the shape of the corner. In the case of the algorithm used, the COG is a useful measure as its calculation is required in the later stages of the SUSAN algorithm. The COG is calculated with respect to the origin at the centre of the corner. The COG measure is rotation variant, but the distance between the COG and the centre of the mask is rotation invariant.
- The **area of the corner** as it is used in this algorithm refers to the sum of the intensities of the pixels within the mask connected (i.e. within a threshold) to the located corner point. This measure is almost invariant to rotation, however due to the discrete nature of the mask it is slightly affected.
- **Direction of corner**, as discussed previously defines the gradient direction of the corner and is seriously affected by rotation.
- **Intensity difference** (or contrast) of the corner defines the gradient strength of the corner, calculated by averaging the intensity over the 'connected-corner' and 'non-connected-corner' regions of the mask. The intensity difference is reasonably invariant to rotation and because it is a difference it is moderately unaffected by global changes in scene intensity.

A number of different combination formulations of the measures described where used as the feature comparison techniques.

- The first was a straightforward differencing method between the various parameter descriptions using (i) the distance to the centre of gravity from the centre of the corner (*C*), (ii) the area of the corner (*A*) and (iii) the contrast of the corner (*I*), to try to make the matching process as unaffected as possible by rotation and global intensity changes. These values were normalised over the range *0-1* to allow an equal contribution from each value.

  C, distance to the COG from the corner centre; $0 < C < 3.5$
  A, area of corner; $3 < A < 18$ (normalised over the 37 pixels)
  I, contrast of corner; $t < I < 255$ (where *t* is the SUSAN brightness threshold)

  Normalise these metrics over the range *0-1*:

  $$C' = \left(\frac{C}{3.5}\right),\ A' = \left(\frac{A-3}{18-3}\right),\ I' = \left(\frac{I-t}{255-t}\right)$$

  $$S_M = 1 - \sqrt{\frac{1}{3}\left(C_1^{'} - C_2^{'}\right)^2 + \frac{1}{3}\left(A_1^{'} - A_2^{'}\right)^2 + \frac{1}{3}\left(I_1^{'} - I_2^{'}\right)^2}\,, \qquad 4\text{-}12$$

  so S_M is normalised in the range $S_M \in [0,1]$, where *0* is the weakest and *1* is the strongest match.

- The second was a constrained version of the first using user-defined factors of importance of the particular feature properties:

  $$S_M = 1 - \sqrt{\lambda_C\left(C_1^{'} - C_2^{'}\right)^2 + \lambda_A\left(A_1^{'} - A_2^{'}\right)^2 + \lambda_I\left(I_1^{'} - I_2^{'}\right)^2}\ , \qquad \mathbf{4\text{-}13}$$

  where λ_C, λ_A and $\lambda_I \in [0,1]$ and $\lambda_C + \lambda_A + \lambda_I = 1$, tailored by the user to specific needs. Again S_M is normalised in the range $S_M \in [0,1]$, where *0* is the weakest and *1* is the strongest match.

The following matching percentage measure was used to test the performance of the feature matching techniques and can be computed simply as:

$$\text{matching percentage} = 100 \times \frac{\text{number of correct matches - number of incorrect matches}}{\text{maximum possible number of correct matches}}$$

4-12

which will be dependent on the pair of images chosen but as a measure it gives the estimated performance of the matching technique. The *maximum possible number of correct matches* defines the number of matches that is possible, assuming that certain features cannot be matched as they disappear and appear between image frames. This value was calculated by hand, with a small Java application used for matching between the two frames by the user choosing correct pairs of features.

It is worth noting that the measure above is defined for regular matching techniques and in the case of the active mesh algorithm the measure is better defined as:

$$\text{matching percentage} = 100 \times \frac{\sum \text{correct match strengths - } \sum \text{incorrect match strengths}}{\sum \text{ correct + incorrect match strengths}}$$

4-13

as the effect of the matching on the overall mesh will be determined by the calculation of the strengths of the matches rather than just the number of matches as this is the true effect of the match performance on the movement of the mesh.

From the relationship developed for matching the external forces may be calculated by the following equations:

$$F_{ext(x)} = \alpha_E S_M d(x)\left(\frac{r-d}{r}\right) \qquad \text{4-14}$$

$$F_{ext(y)} = \alpha_E S_M d(y)\left(\frac{r-d}{r}\right) \qquad \text{4-15}$$

where α_E is a user-defined factor to allow the external forces to have a larger or smaller effect on the mesh (i.e. similar to the regularisation parameter as discussed in Section 3.2.5) and *r* is the search space radius. The last term weights the distance of the force as weaker the further the distance from the examined node.

```
Set the searchSpaceArea
For each Node (curNode)
        Search CornerList for best match corner to curNode
                bestCorner = the best match corner returned
                pullFactor = factor based on the strength of the match
        distance = the distance between curNode and BestCorner
        distanceFactor = (searchSpaceArea – distance) / searchSpaceArea
        matchStrength = pullFactor x distanceFactor
        diffX = bestCorner.X – curNode.X
        diffY = bestCorner.Y – curNode.Y
        if (diffX not equal 0.0)
                unitX = diffX / (absolute value of diffX)
        if (diffY not equal 0.0)
                unitY = diffY / (absolute value of diffY)
        distanceXFactor = ExternalForceFactor x (absolute value of diffX)
        distanceYFactor = ExternalForceFactor x (absolute value of diffY)
        theForceX = distanceXFactor x unitX x distanceFactor x pullFactor
        theForceY = distanceYFactor x unitY x distanceFactor x pullFactor
        push curNode by (theForceX, theForceY)
```

Algorithm 4-3. External energy algorithm.

The active mesh allows constrained feature matching to take place on a frame-by-frame basis in an image sequence. Once the features have been correctly matched the real-motion of the image of selected points in the image plane are available as vectors.

The external forces are caused by the spatial position and match strength of the mesh corners detected in the new frame relative to the current mesh nodes. For each detected corner in the area around the node (determined by the search area), a comparison is computed between the similarity of the corner and the stored node information (from the initial computation of the mesh). This comparison of similarity between corners and nodes is performed using an intensity patch comparison or the other comparisons outlined that return a match strength value between 0 and 1, where 1 implies an ideal match. The force is then calculated for the node as shown in Algorithm 4-3, based on the distance of the best match corner from the current node. This algorithm allows forces that are further away from the node to apply a smaller force on the current node. It is likely that matches closer to the current node are more correct. However it is important not to discount the forces that are a large distance from the node. For example, if the scene undergoes a large translation then a large number of small forces could pull the mesh towards a possible correct solution.

4.5 The Final Algorithm

The completed general algorithm may be described as follows, referring to Figure 4-23.

- Load the initial frame.
- Perform the feature detector on the image (usually using the SUSAN corner detector with a modified 5x5 mask). The threshold value is chosen as appropriate for the image sequence, decided by a desired number of features, but it was found that this value does not vary significantly for the real-world image sequences used.
- If this is the primary frame, a mesh is built using the Delaunay triangulation from the features that are detected. Each feature point is added to the mesh, point-by-point, becoming a node independent of the image feature. The energy formulation that is used causes the mesh to be initially in equilibrium.

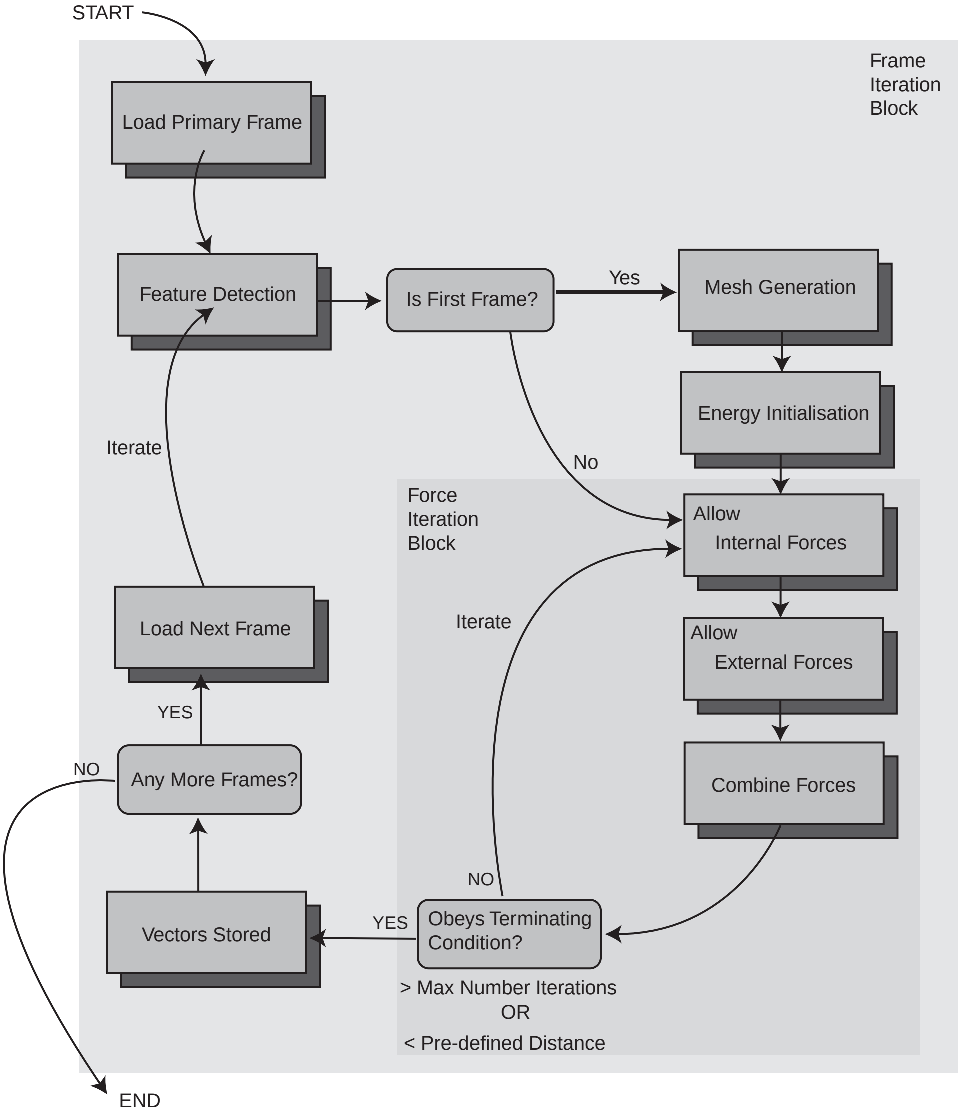


Figure 4-23. The outline of the overall active mesh algorithm.

- The mesh is applied to the image and is set active. Initially there should be no change in the mesh as the mesh is minimised, with the external forces being zero and the mesh line lengths at their desired values. A parallel thread is then started that:
    - Calculates the internal forces as in Algorithm 4-2, attempting to minimise the internal energy of the overall mesh.
    - The external forces that are caused by the change of the image are calculated and the external energies are minimised as in Algorithm 4-3.
    - All the forces are combined.
    - Is the mesh terminated? If not, then repeat last step otherwise go to next step.
    - The node positions of the mesh are recorded at each iteration and stored in a time-tagged array.
    - If there are more frames, repeat.

4.6 The Adaptive Active Mesh

The adaptive mesh algorithm is very similar to the general active mesh algorithm but with a few extensions. The concept behind the adaptive mesh is to allow features to be added or removed from the mesh as they become available or unavailable in the image sequence. This property can be added to the mesh algorithm as shown in Figure 4-24.

- Drop features: If the mesh is tracking a set of features for several frames, it should drop the features that are weakening the matching by being inconsistently detected over image frames. So the following cases should cause features to be dropped:
    - Weak features: features that are inconsistently detected.
    - Features that go out of scope: features that are tracked may leave the image view and should be removed from the mesh. This is application dependent, as the mesh can maintain information outside the image bounds quite successfully, but often these features should be removed.
    - Features that become occluded may also be removed. Again, it is application dependent. In fact using this algorithm the relative position of these features can be maintained until they re-appear, but these features must be tagged as occluded.

- Add features: If the mesh is tracking a set of features for several frames, it can also add features to the mesh. The following cases might cause features to be added:
  - Dis-occlusion of image features. Features might appear in the image frame due to an object moving and revealing previously occluded features.
  - New frame edge features will likely be detected at the edges of the image frame when the observer view moves to the left or right of the original view.
  - New moving object. The entrance of a new independently moving object may cause new features to appear.

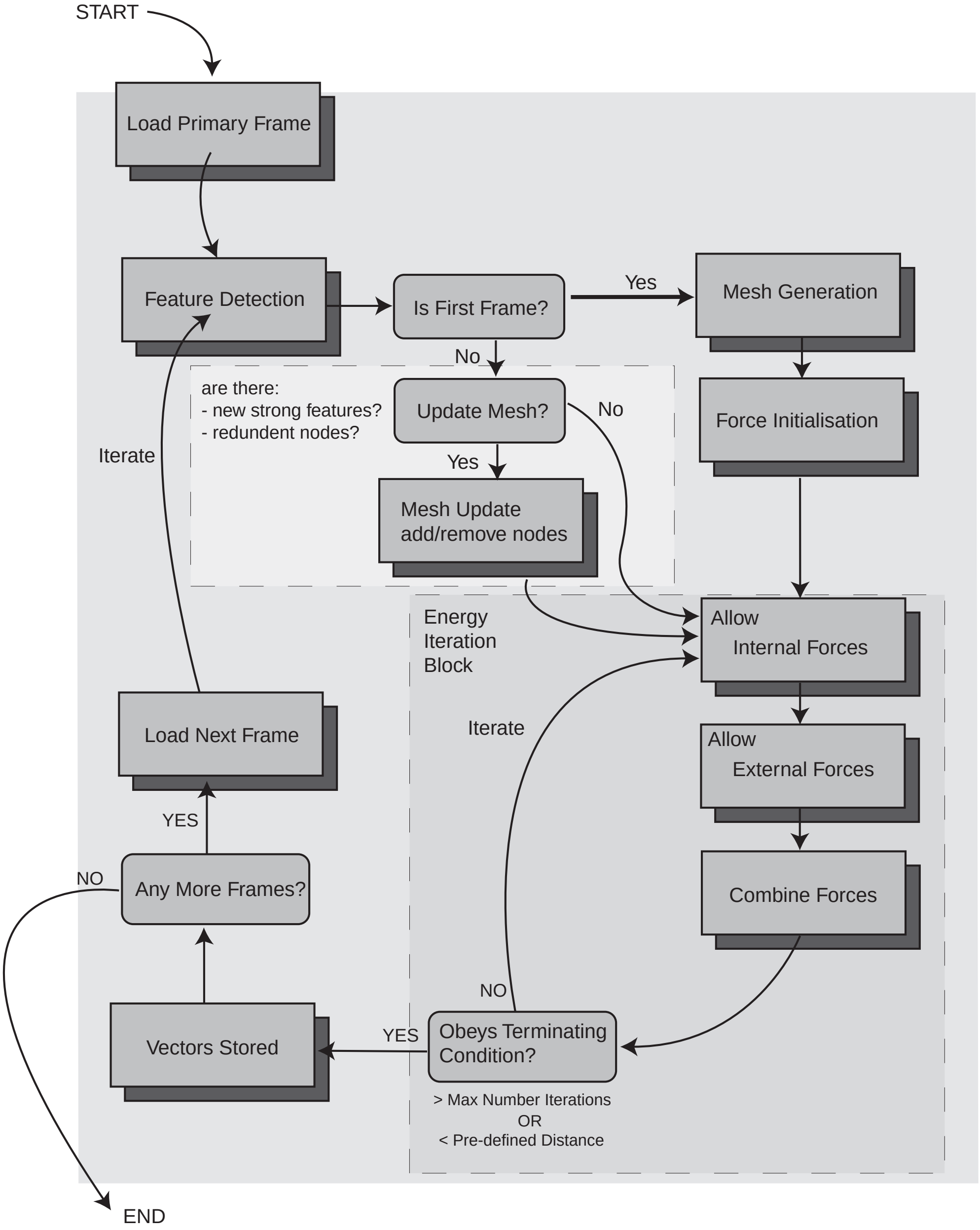


Figure 4-24. The adaptive mesh overall algorithm.

It was important to modify the active mesh to be capable of changing in structure as the image sequence progressed. The reason for using the iterative Delaunay triangulation was for this very reason, only it was found to be more difficult than initially expected. Problems occur with the adaptive mesh when it is modified and no longer obeys the conditions of a true Delaunay triangulation, i.e. when the structure of the mesh has been altered and no longer conforms to the Delaunay structure due to the effect of external forces. This is described in detail in Appendix B.

The solution to this difficulty involved updating the mesh at each frame, rather than updating the mesh at each iteration. The mesh structure is verified and modified to force it to conform to the structure of a Delaunay mesh, with vertices being swapped to conform to the min-max angle criterion and connection references being updated. This mesh does not however have to conform to the Delaunay triangulation rules while the iterations are taking place. The referencing structure of the software implementation of the mesh allowed this to be performed seamlessly, making it easy to remove mesh nodes from the centre of the Java vector list. The remaining vector list elements are recombined into a mesh, still retaining all the mesh node information. This methodology kept the mesh clean and prevented crossover cases as discussed in Appendix B.

4.7 The Multiple Active Mesh Approach

This active mesh algorithm structure allows for multiple meshes to exist in the same sequence. The determination of the locations of the individual meshes can be performed in several ways:

- The primary and most obvious method of determining multiple meshes is by examining the motion vectors of a large mesh and sub-dividing the mesh into a smaller mesh that is a subset of the larger mesh. This smaller mesh can then exist with no ties to the larger parent mesh. In the parent mesh the mesh nodes can be removed that correspond to the child mesh and both meshes can exist independently. The identification of an independent similarly moving cluster of mesh nodes was implemented.

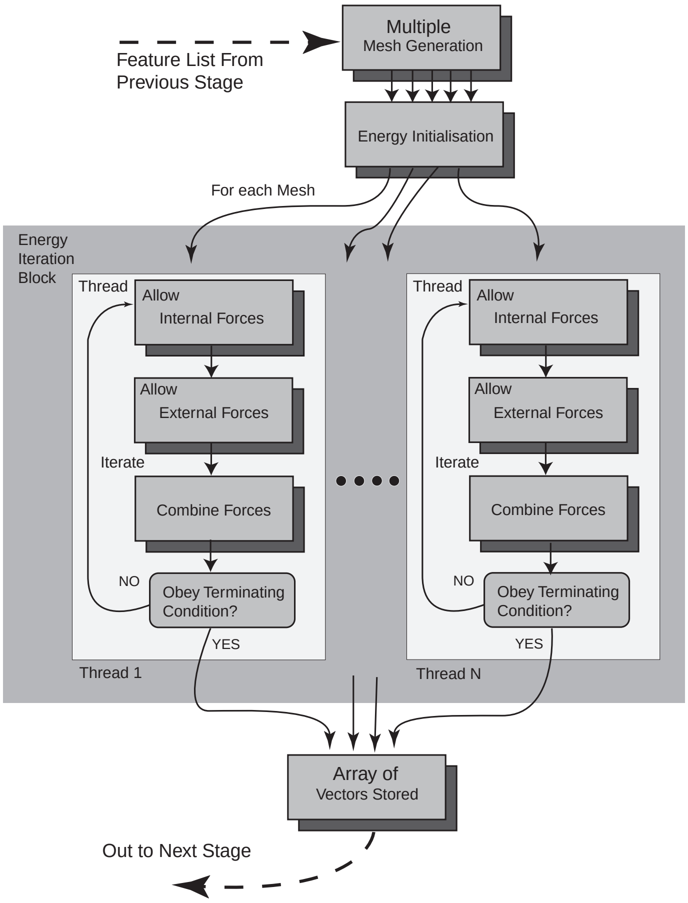


Figure 4-25. The multiple mesh algorithm segment.

- Another method would be to use region information in the scene. A region segmentation technique could divide the image into areas of similar intensity or texture patterns. From this information and the location of features within this region, a mesh could be constructed that is a small part of the entire image. Each mesh node is tagged with the region information and mesh regions that display similar motion patterns could be combined to form a super region that is the sum of these smaller regions.
- 3-D information may be available that would also allow the segmentation of sub meshes, where the depth map could be region segmented as information for the creation of multiple meshes. The mesh nodes of these small meshes would also be tagged with the estimation of depth of that node, leading to more informed segmentation in later stages. This would be a form of the 3-D ACMs discussed previously, in Section 3.7.1.
- *A priori* information is always useful and if available for a particular scene could lead to the creation of sub-meshes. For example, if the algorithm was being applied to purely corridor scenes and models were available from some other tech-

nique (various authors have developed models of corridor scenes based on the vanishing point and image edges (see Grosso *et al* (1992)) then multiple sub-meshes could be applied to different regions of the model (walls, floor, roof etc.). These meshes could then be tracked individually.

In the case of multiple sub-meshes image features could be shared between the meshes. This could have advantages in most cases as the image features could be corners, often existing on the physical boundaries of objects and should be present in the inner object mesh and the outer 'background' meshes. This phenomenon could also cause the problem of the sub meshes drifting into each other and getting 'lost'. This was a characteristic of the initial multiple snake implementation of this research. The addition of constraining region spring forces to the mesh might confine the sub-meshes to the correct regions, but this assumes that the region determination is accurate to a large extent.

The facility to have multiple meshes was added to the algorithm by adding an array of meshes, individually determined by some process (as outlined above). Once initialisation has been performed the individual meshes are threads that operate in parallel, with no transfer of information between individual threads. Currently the initialisation of the multiple active meshes is based on a region mask.

4.8 The Region Based Initialised Active Mesh

As just discussed the initialisation is the stage at which the multiple meshes are determined, after that they operate as individual threads. The region-based initialisation is based on a multi-intensity mask, where some other *a priori* process determines the mask or can even be created by the user. There are several uses and reasons for using such a mask approach for the active mesh.

- In the case of a static camera fixed on a scene, image frame differencing (or indeed some form of dynamic updating image frame differencing, to account for global intensity changes etc.) makes it comparatively straightforward to determine the background region of the image frame. Intruder objects appear clearly (with the exception of some background 'print through') and can be identified as the location for an active mesh to be initialised. This 'small' mesh can then be tracked through image sequences. If multiple objects appear they can be tracked using the multiple mesh framework just outlined.

- Clustering and segmentation algorithms can determine region information. This information about the regions can be transferred to the initialisation of multiple active meshes, with the aim of tracking these regions individually. This is an application dependent implementation. This information could also define depth map information, determined by a 3-D structure from motion algorithm.

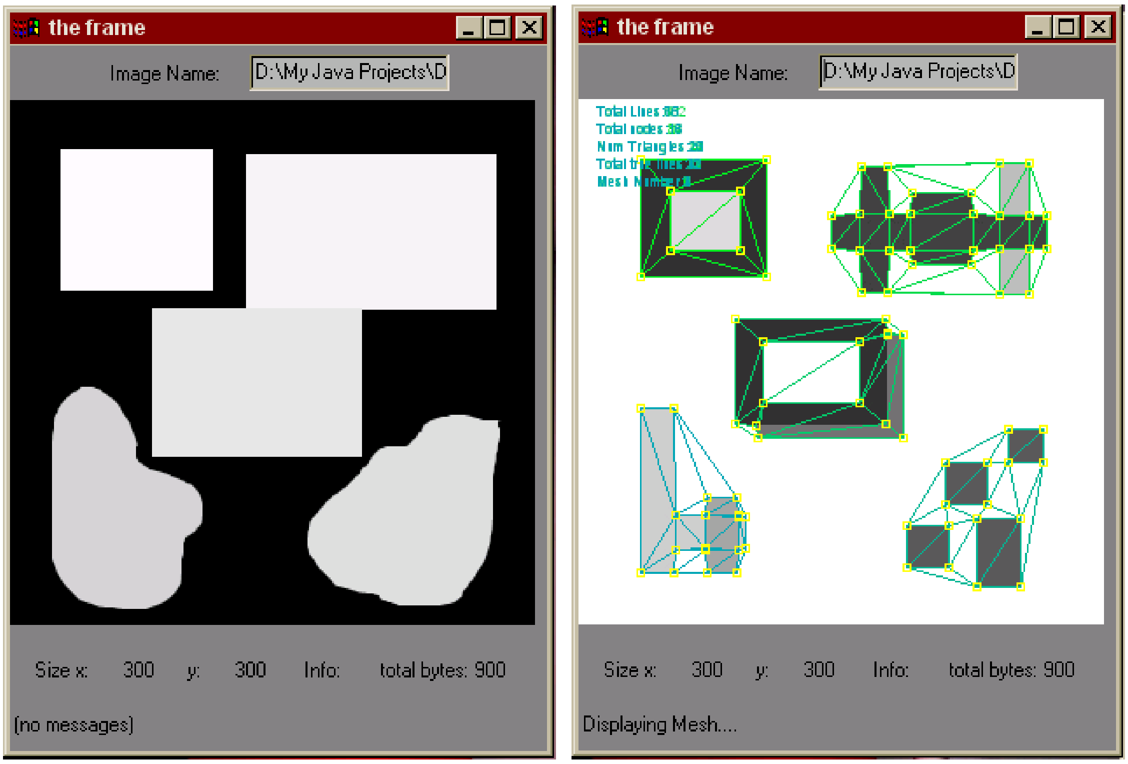


Image Name: D:\My Java Projects\D

Size x: 300 y: 300 Info: total bytes: 900

(no messages)

Image Name: D:\My Java Projects\D

Size x: 300 y: 300 Info: total bytes: 900

Displaying Mesh....

(a) (b)

Figure 4-26. The hand generated mask region image is shown as in (a) and the resultant multiple mesh output is shown in (b).

The way that the algorithm uses this information is straightforward. Assume that there is an image *I(x,y)* and a region map *R(x,y)*, where both images are of the same dimension. *R(x,y)* is structured as a 32-bit integer image (allowing 4.3 billion regions), with each integer level defining a different image region. Sort all the image features that have the same value of $R(x_f,y_f)$ into lists of features. For each list of features, create a new mesh. Figure 4-26 (a) shows the region image (that usually exists in memory only) that allows the definition of the different regions. Figure 4-26 (b) shows the resultant multiple meshes that are output from the algorithm. It is important to note that once the meshes have been created there is no dependence (or need) for the region image.

4.9 Notes on the Active Mesh

4.9.1 The Search Space

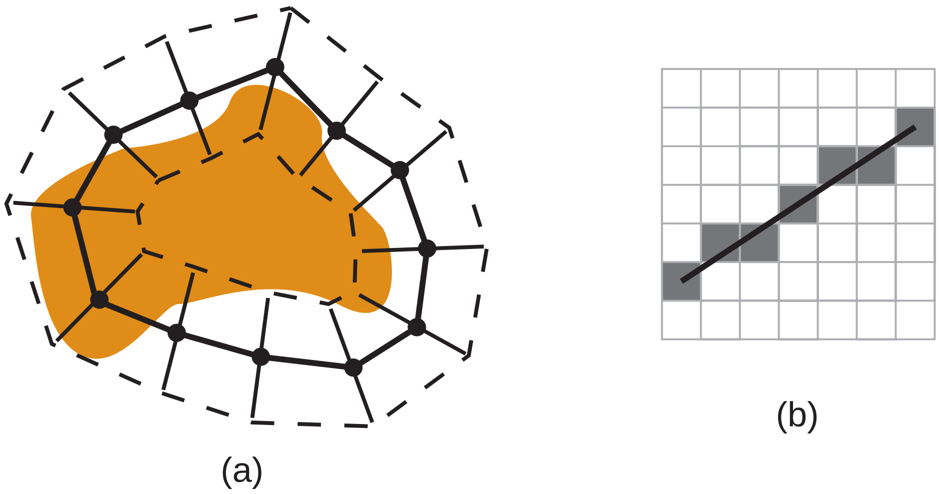
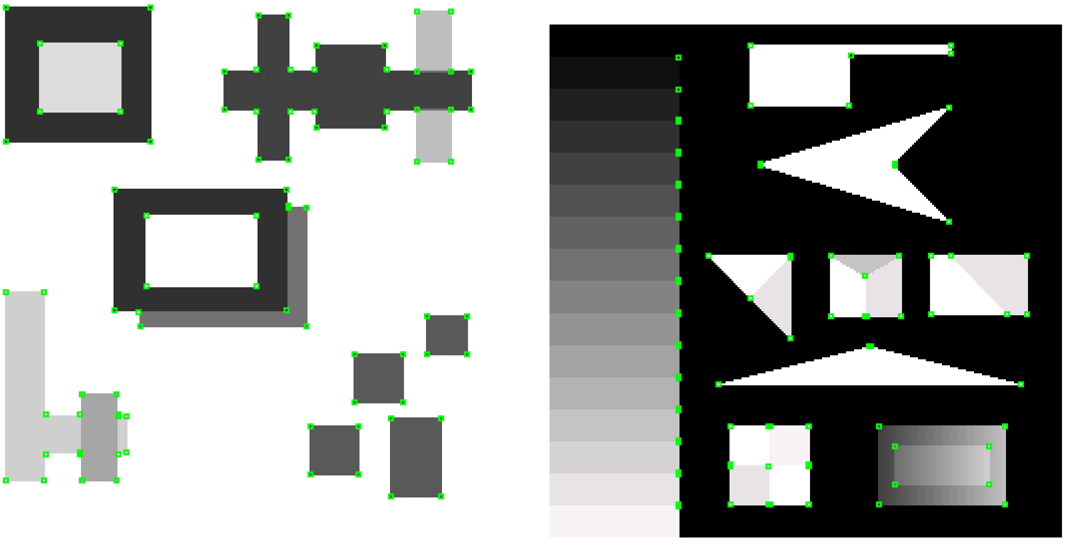


Figure 4-27. (a) Shows a common approach of linear search spaces for image edges when dealing with active contours (b) illustrates the discrete image form of a linear search.

In most active contour applications a linear search space is used as in Figure 4-27, with the region of interest bounded by the dashed lines. Lines drawn normal to the curve represent the search space where corresponding image features are likely to lie and may be detected using image filtering. This can be a very efficient way to perform image filtering, as it is only performed on a very small area of the image. Problems may however result from the way in which these search lines will intersect the pixels of the image, including sharp changes in intensity over the line. A possible solution to this problem is to use anti-aliasing sampling, where the intensity at a particular point is a weighted sum of four neighbouring pixels.

For dealing with this active mesh implementation, corners or feature points are used and linear search spaces are no longer suitable. First of all, there are no apparent linear paths to use and it is unlikely that a linear path would be robust in detecting discrete points. The use of feature points forces an exhaustive search over a region of interest, which is indeed more computationally intensive. However, it has been found that there is a limit to the computation involved as a feature detector may be applied to the entire image, rather than the individual search spaces. For example in an image of 256 x 256 pixels, and mesh nodes with a search space of 25 pixels in radius – only 34 mesh nodes are allowed before it is more efficient to perform the feature detector on the entire image, recording the *x* and *y* locations of each feature extracted. Some approaches and possible improvements to the search space, including the use of Kalman filtering are suggested in Section 6.3.3.



(a) (b)

Figure 4-28. The SUSAN algorithm as implemented in Java. In (a) tested on a test template frame and in (b) tested on the original test frame given by Smith (1992), to test consistency with the original algorithm.

4.9.2 The Terminating Condition

The terminating condition for this implementation can be one of two cases:

- A maximum number of iterations per frame *or*
- When the mesh nodes move less than a defined minimum distance

The ideal case is the second one where the mesh is allowed to settle to less than a certain defined threshold distance, thus allowing the mesh to settle quite accurately. For this implementation the value is set at 0.25 of one pixel. The only problems that can occur with this condition is that of an unstable settling where a mesh node could dither between a value of 0.2 to –0.2 and so prevents the mesh from settling. This is unusual, but can occur when a mesh node is connected to a large number of other mesh nodes. A maximum number of iterations threshold is also included to prevent this from occurring and ‘locking-up’ the algorithm. It was also found suitable to define a minimum number of iterations, to insure that the mesh at least attempts to search for a solution.

4.9.3 The SUSAN Corner Detection

The SUSAN Corner Detector as discussed in Section 2.6.2 was implemented in Java using the specialised *RGBImageFilter* class and altered to provide structured information about the corners. See Appendix D.

In Figure 4-29, the SUSAN corner detection algorithm is applied to the same image with differing thresholds to show how this has an effect on the number of corners detected. In (b) a threshold value of 16 is used. In this case 570 corners are detected. When this value is reduced to 8 the number of corners increases to 930 as in (c) and when this value is increased to 32 the number of corners reduces down to 305 as in (a). The quality of the corners at lower thresholds is poorer, in that the intensity difference that is required to classify a corner is reduced, with a small intensity difference being more affected by lighting conditions and noise. The corners that are detected with the highest difference are the strongest, i.e. they have a large image gradient on the corner and so are possibly the best corners for matching, in that with the exception of occlusion they are likely to be consistently detected from one frame to the next.

The system should provide a reasonable number of features for feature matching. What is a reasonable number of features in this case?

- The larger the number of features, the more computationally intensive the matching becomes, and the weaker the features become, in terms of consistently and reliably detectable features. Corners detected from texture information, or image quantisation effects are likely to be unreliable.
- Too few corners will cause regions in the image to contain no features for the feature-matching algorithm, resulting in sparse motion information. However, having no features detected in a region of uniform intensity may not be a problem for motion tracking, as the features surrounding this area will allow an estimation of motion to propagate towards the centre of the uniform region.
- A specific number of corners could be chosen, however this would not be suitable in many cases, as the corners could suffer from the same problems. The number of corners may be forced to be too high, giving poor features, or too low, giving a non-uniform image description.

A suitable number of corners would be a number of stable corners that are not excessively clustered in a particular region of the image. Choosing the correct number of corners was a problem also discussed in Charnley and Blissett (1989). In their application for surface reconstruction they manually select a corner threshold to give, typically 250 points in semi-structured images (8-bit 256x256) and a maximum of 500 points in images of natural vegetation. They found that this number of features allows an even distribution, except for the blander regions of the images.

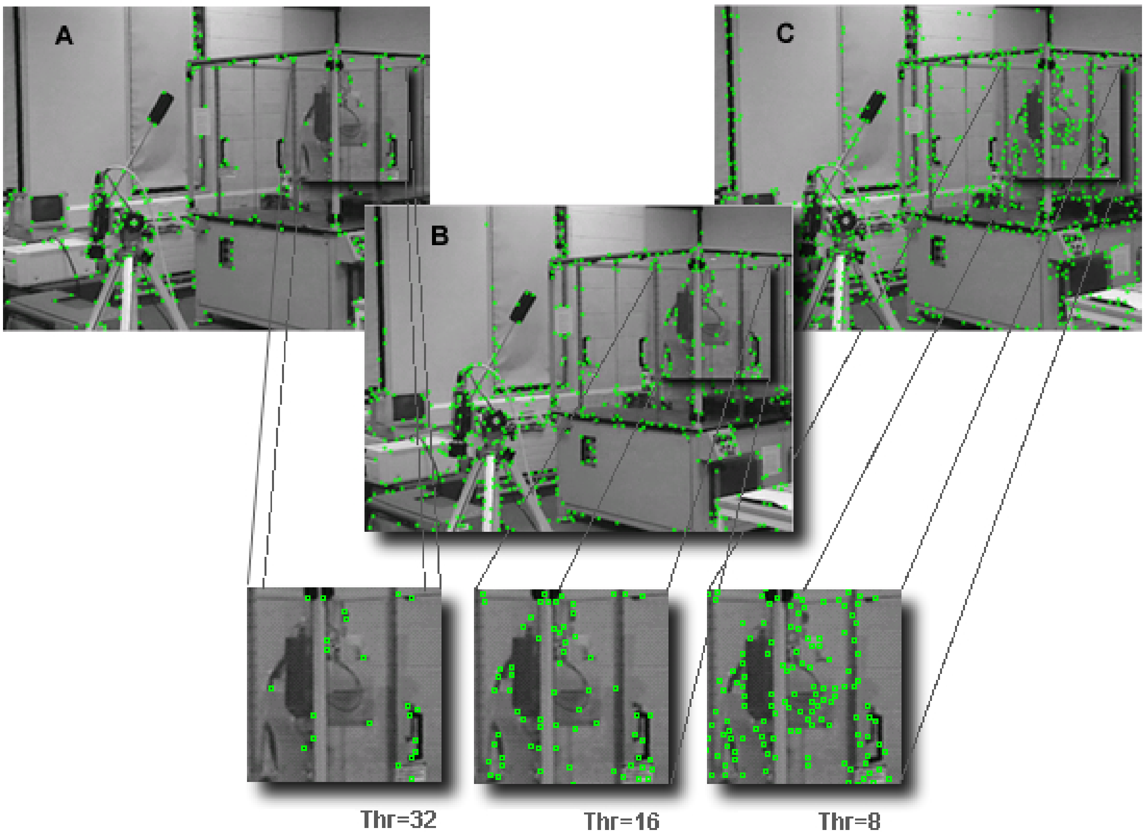


Figure 4-29. The SUSAN algorithm applied to the same image using different threshold values. In (a) Threshold = 32, (b) Threshold = 16, and in (c) Threshold = 8.

4.10 SUSAN Pre-filter modifications

The SUSAN corner detector of Smith (1992) was examined in detail and found to be a very efficient algorithm. Trajković (1998) developed a corner detector based on the SUSAN principle but involving calculating the CRF in many directions passing through the centre pixel, for the purpose of developing a faster algorithm.

It was found however, during the course of this research that a pre-filter could be written to further enhance the speed of the SUSAN algorithm. In Figure 4-29(b) at a brightness threshold of 16, approximately 570 corners are detected by the SUSAN algorithm, which is less than 0.5% of the total pixel area of the image. It makes sense therefore that the algorithm should be layered so that the more computationally expensive calculations be carried out only on the most likely set of corner candidates.

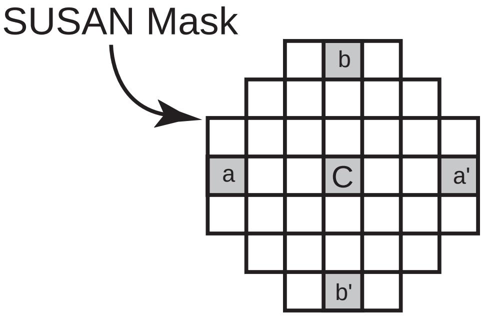


Figure 4-30. The SUSAN mask and mask points of the calculations.

The SUSAN algorithm of Smith (1992) agrees with this philosophy and performs the majority of computation in a multistage form, where the calculation of the USAN (USAN area) is the first and necessary step of the algorithm. This computation is in itself a computationally intensive calculation, involving 37 addition operations for each image pixel.

```
At each pixel:
If $|I_a - I_C| + |I_{a'} - I_C| < \alpha_s d$ Or $|I_b - I_C| + |I_{b'} - I_C| < \alpha_s d$
        Then do not continue
Else Contine to next SUSAN stage.
```

Algorithm 4-4. The SUSAN pre-filter algorithm.

The suggestion is to perform a pre-filter that tries to prevent ‘no hope’ candidate corners from passing on to a further stage for the USAN calculation. The concept is straightforward, because, examining points in the mask *a* and *a’*, shows that if the intensity difference between point *a* and point *C* is large then it is possible that a corner may exist at this point. This is performed similarly for the differences in the image intensities between *a’* and *C*, *b* and *C*, and *b’* and *C*. If there is no significant change then it is likely that the intensities of the mask are uniform – hence no corner. The algorithm takes the form of Algorithm 4-4, where *d* is the ‘usual’ user-defined brightness threshold of the SUSAN algorithm and I_a represents the image intensity at the point *a* in the mask. The user-defined constant value α_s is usually set at the value 1.0 (the reason for this is explained in Section 5.6.1)

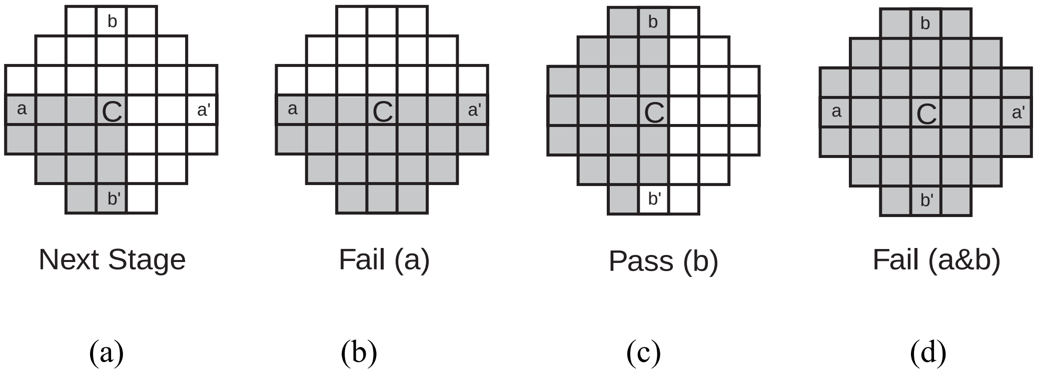


Figure 4-31. Showing some example cases for the pre-filter and whether they pass to the next stage or fail the test.

Figure 4-31, shows some example cases for the SUSAN pre-filter. The pre-filter prevents most ‘no hope’ candidate corners from being passed to the next stage of the SUSAN corner detection algorithm. In Figure 4-31(a) the intensity difference between the value at mask point *a’* and point *C* is large and so the corner candidate is passed to the next stage (this would also be the case for *b* to *C*). In Figure 4-31(b) the intensity difference between I_a - I_C and $I_{a'}$ - I_C is very low and so will fail the test – the corner will be dismissed as a ‘no hope’ match. In Figure 4-31(c) the current pixel location will

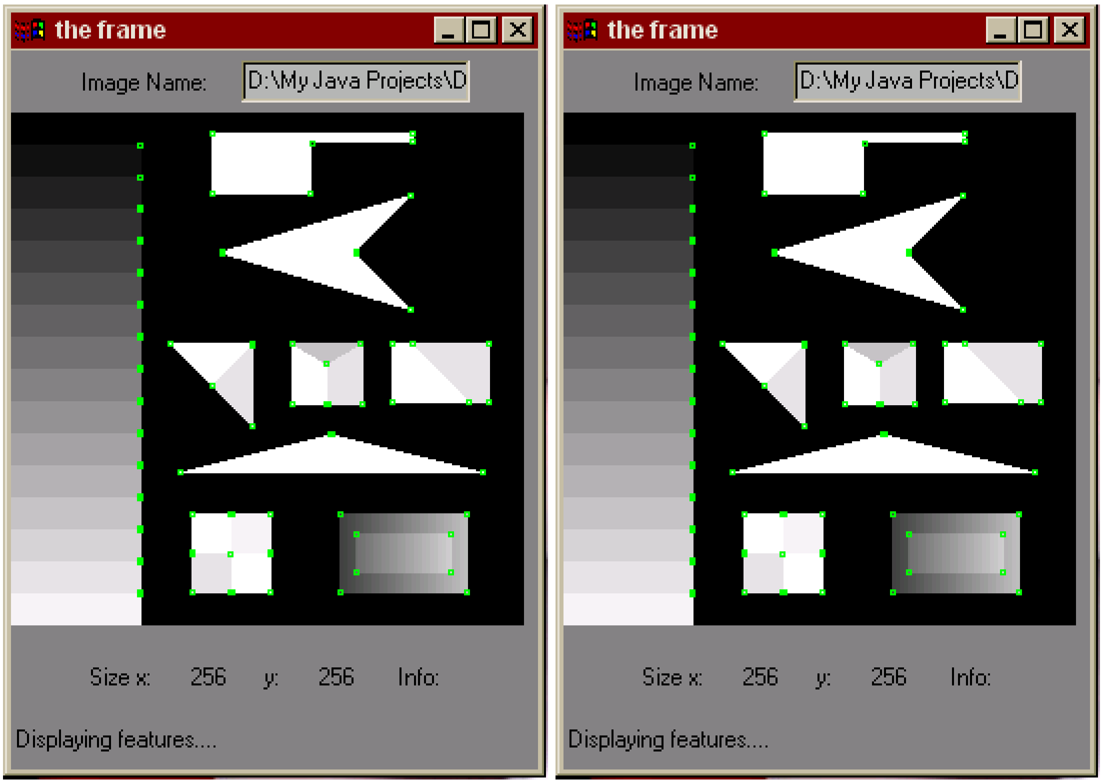


Image Name: D:\My Java Projects\D

Size x: 256 y: 256 Info:

Displaying features....

Image Name: D:\My Java Projects\D

Size x: 256 y: 256 Info:

Displaying features....

(a) (b)

Figure 4-32. The SUSAN test template used by Smith (1992) (a) shows the Test template after the SUSAN algorithm has been performed [280ms]. (b) Shows the template after the modified SUSAN has been performed [81ms].

pass the test as the intensity difference between $I_{b'}$ and I_C is large and so is passed to the next stage of the SUSAN algorithm, which will likely reject it. In Figure 4-31(d), the uniform intensity points that are most likely rejected without any further processing by the SUSAN algorithm. This type of point is most likely found in regions of uniform intensity, as the USAN area is > 2/5 of the total area. Results from this approach and comparisons are carried out in detail in Section 5.6. As shown in Figure 4-32, the modified algorithm identifies the exact same corners as the original algorithm in the test image used by Smith (1992).

4.10.1 Addition of Median Pre-Filtering

The median is a well-recognised filter for the reduction of image noise, especially for suppressing noise in regions of similar image intensity, especially important in edge detection techniques, since noise can confuse edge detection techniques into the detection of false edges. There are different definitions of the median filter, with the most common filter mask types being displayed in Figure 4-33. The median filter has long been recognised as an edge preserver, but not an edge enhancer, as it does not increase the contrast of the edge. It does degrade true edges to some extent, but the technique is superior to other linear filtering techniques. The reduction of image noise and thus false edges is a large benefit for minor edge degradation.

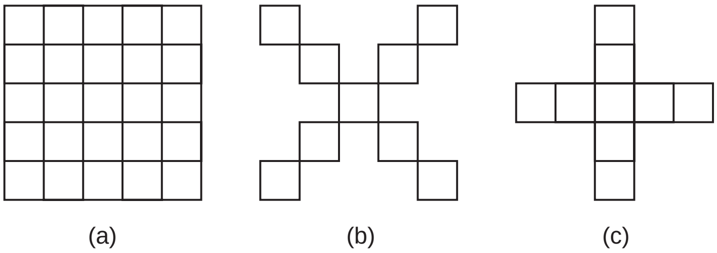


Figure 4-33. The most common median filter masks (a) square (b) X and (c) cross shapes.

Bovik *et al* (1987) perform a study on the different types of median filtering techniques available and on the effectiveness of these techniques. They conclude that the median filter can improve the performance of edge detectors, with the *X* and cross-shaped masks performing more favourably in the presence of a majority of horizontal, vertical and diagonal edges. The square masks allow the extraction of edge maps that are smoother but with larger amounts of edge displacement when all the orientations are considered.

It was determined during the course of this research that the median filter also played a role in extracting more stable corners using the SUSAN algorithm (See Section 4.2 on early techniques). The primary implementation was using a 3x3 square median mask, performed when each frame of the image sequence is loaded.

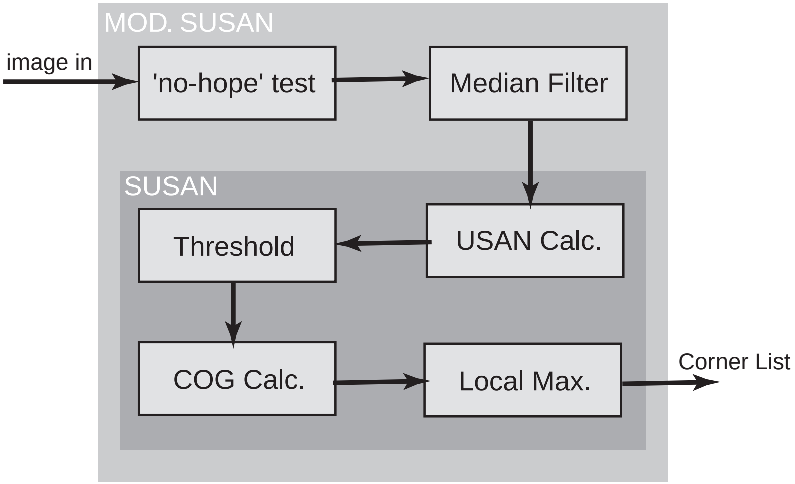


Figure 4-34. The SUSAN corner detector with pre-filters.

In the later implementation of the active mesh using the modified SUSAN corner detector as the primary feature detector it was discovered that an efficient implementation of the median filter could be performed, in much the same way as the SUSAN pre-filter was developed.

As discussed for edges the main use of the median filter is to reduce impulse noise that causes the detection of false edges. In much the same way this impulse noise causes the detection of false corners, especially in areas of uniform intensity. This means that most of the true corners will be a subset of the list of true and false corners. Another version of the SUSAN corner detector was developed with a median filter (with a choice masks) being performed after the initial reduction of ‘no-hope’ corners as discussed in Section 0). Other forms of noise reduction were not examined, such as noise modelling.

The overall algorithm is displayed in Figure 4-34, where the computationally intensive median filter is only performed on possible corners. This step is not as efficient as it might appear, as the median still must be calculated on the entire area of the mask. Each pixel is tagged once the median has been performed and so there is no duplication in the calculation.

4.11 Discussion

Several different active mesh approaches have been described – active mesh, adaptive active mesh and multiple active meshes. Each one of these approaches has different applications. The active mesh and adaptive active mesh approaches are very suitable for unconstrained motion tracking. The multiple region based approach is very suitable for fixed camera applications, as well as moving camera applications where multiple objects are involved. The results outlined in Chapter 5 will illustrate these approaches working in different situations.

Some of the advantages of the active mesh methods include:

- The initialisation of the mesh(es) is automatic and involves very little parameter changes, more tuning using the parameters available.
- It is based strongly on 2-D image features, which are shown to be less likely to suffer from local effects such as the aperture problem. These features have also been determined to be accurately defined and consistent between image frames. This algorithm takes the advantages of lower-level feature matching and provides a higher-level framework.
- The energy-based approaches of active contours has gained acceptance as a suitable framework for motion tracking, with this approach allowing a more structured model for dealing with objects than snakes or balloons.
- There are few assumptions made about the scene and no *a priori* information required.
- The system is modular where the features used, the matching algorithm and the image filters may be replaced with more suitable modules as they become apparent.
- The algorithm can deal with affine transforms of the object as well as deformations within the tracked object, to be shown in Chapter 5.
- The algorithm currently works very well only on a 2-frame basis. The use of multiple image frames and the addition of momentum to the mesh and Kalman filtering would certainly improve the quality of the tracking over multiple frames.
- The image frame step size can be quite large – typically of the order of 20 pixels, which is impressive in comparison to the approaches of optical flow, which requires very small image steps.

Primarily the number of mesh nodes and the number of features in the search space of each mesh node define the complexity of the active-mesh algorithm. The complexity can be defined as $O_1(n)$x$O_2(m)$ where n is the number of mesh nodes and m is the average number of features in each search space. O_1 is the mesh update operation, which includes the calculation of the internal and external forces. The reason that there is not a higher level of complexity is due to the fact that the forces to be applied at the connected nodes are only summed once for all nodes due to the node force combination algorithm. This would involve a level of complexity of $O_1(n^2)$ if each force was updated individually. O_2 is the feature-matching algorithm that is performed for each of the m features in each search space. Increasing the search space area will thus affect the complexity, as will increasing the number of features or the number of mesh nodes. In the multiple mesh approach this complexity will be required for each mesh, however the meshes will be smaller in size.

4.12 The Java Application Developed

To test the theoretical approach described in this research an application implementation was developed in Java. Initial implementations during the course of this research were in Khoros (Cantata), C, C++ for X-windows (using Motif) and then finally Java.

Khoros (version 2.0) was found to be a very suitable environment for the development of vision based algorithms, however it was found to be a difficult environment for developing applications that require advanced user interaction, such as dragging mesh nodes. Because such a specialised application was required involving low-level interaction with the image filters used it was decided to examine the newly emerging language of Java. The use of Java for vision applications development is discussed in a book to be released this year[21].

Figure 4-35 shows the latest implementation environment that was developed. The latest version uses Microsoft Window Foundation Classes (WFCs) for the creation of panes to store all the different options. Behind each tab there are options and controls for:

[21] P.F. Whelan, D. Molloy, "Machine Vision Algorithms in Java: Techniques and Implementation" to be published by Springer-Verlag, September 2000, 298 Pages. ISBN 1-85233-218-2.

- Main - loading main images,
- Filtering - filtering options as currently displayed,
- Options - general options,
- Mesh - mesh generation and parameters,
- Energies - set energy parameters and iterate,
- Frame - loading of frames options,
- History - to display motion paths etc.,
- Sequence - allows sequences to be loaded and operated on,
- Debug - for programming, dumps information as required,
- Matcher - parameters for the feature-matching algorithm.

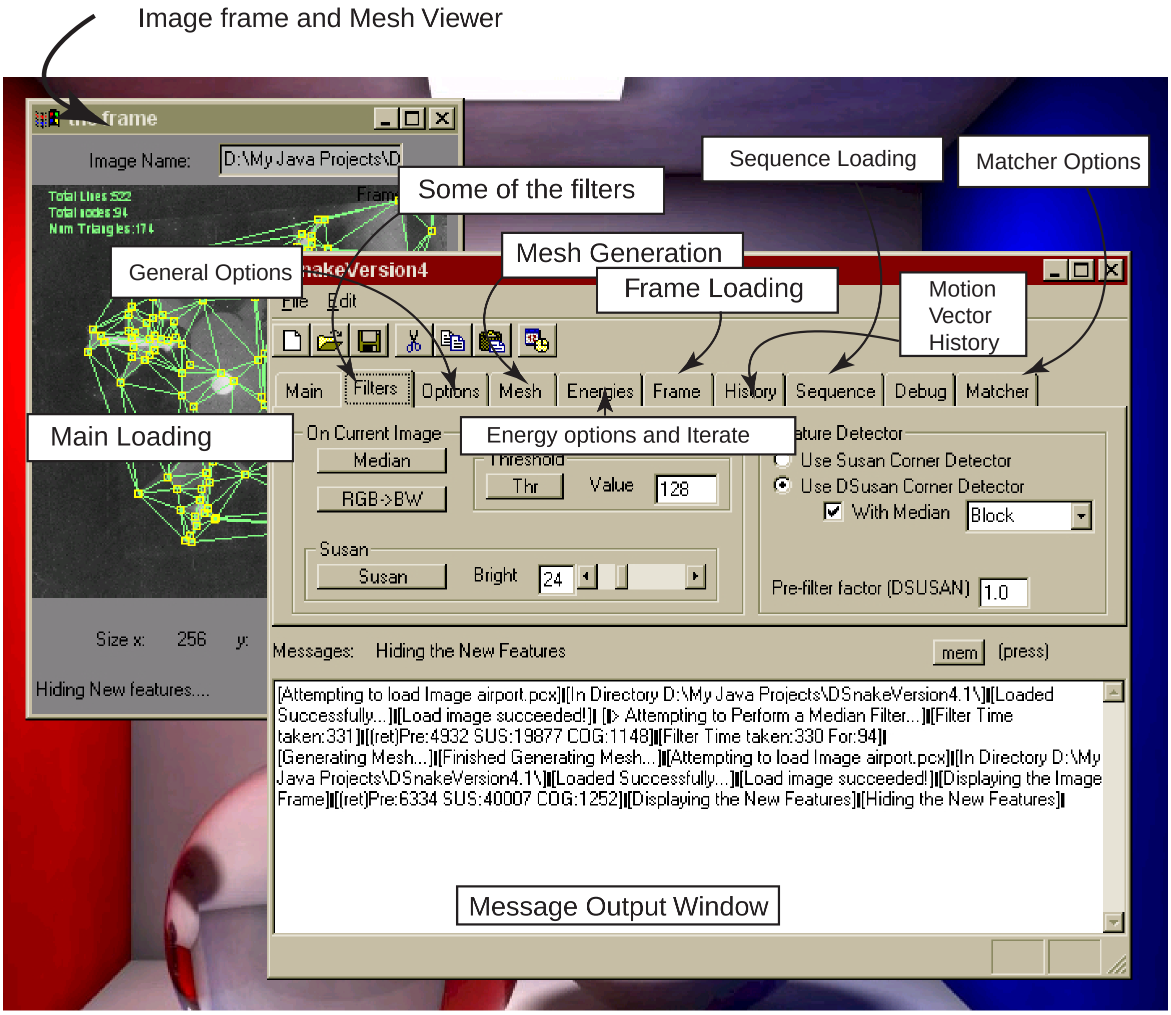


Image Name: D:\My Java Projects\D

Total Lines 522
Total nodes 94
Num Triangles:174

Size x: 256 y:

Hiding New features....

On Current Image

Threshold Value 128

Susan Bright 24

ature Detector
○ Use Susan Corner Detector
● Use DSusan Corner Detector
☑ With Median Block

Pre-filter factor (DSUSAN) 1.0

Messages: Hiding the New Features mem (press)

```
[Attempting to load Image airport.pcx]▮[In Directory D:\My Java Projects\DSnakeVersion4.1\]▮[Loaded
Successfully...]▮[Load image succeeded!]▮ [▮> Attempting to Perform a Median Filter...]▮[Filter Time
taken:331]▮[(ret)Pre:4932 SUS:19877 COG:1148]▮[Filter Time taken:330 For:94]▮
[Generating Mesh...]▮[Finished Generating Mesh...]▮[Attempting to load Image airport.pcx]▮[In Directory D:\My
Java Projects\DSnakeVersion4.1\]▮[Loaded Successfully...]▮[Load image succeeded!]▮[Displaying the Image
Frame]▮[(ret)Pre:6334 SUS:40007 COG:1252]▮[Displaying the New Features]▮[Hiding the New Features]▮
```

Figure 4-35. General form of the implementation environment.

This implementation allows sequences of images to be loaded and the algorithm to run on each frame for a certain number of iterations. The mesh update function is also included that updates the mesh on a frame-by-frame basis and the motion vectors may be plotted using the history functions.

An example Java Applet version of this implementation, containing a subset implementation of the main application developed is available at:

http://www.eeng.dcu.ie/~molloyd/phd/

Chapter 5 – Implementation, Testing and Results

5.1 Introduction

After a formulation was developed for using the active mesh structure, it was necessary to test its operation on different motion sequences, with varying conditions. Initially a simple generated scene was used to demonstrate the low-level operation of the mesh under varying conditions. Subsequently, more detailed simulated image sequences were used to demonstrate the motion and structural changes of the mesh under different image deformations. Finally, the mesh is tested on real-world image sequences, again under varying conditions.

5.2 Simple Operation of the Mesh

Initial testing of the mesh is concerned primarily with the low-level effects that occur within the mesh.

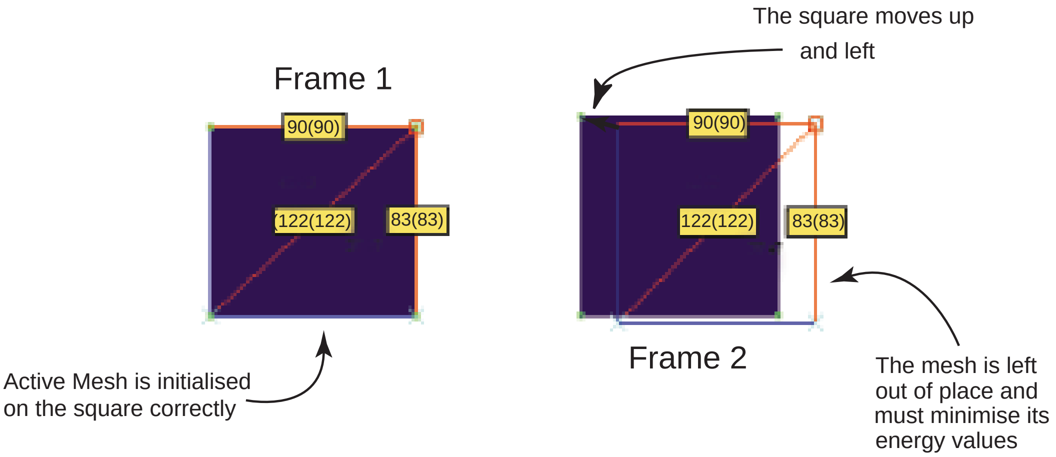

The square moves up and left
Frame 1
90(90)
122(122)
83(83)
90(90)
122(122)
83(83)
Frame 2
Active Mesh is initialised on the square correctly
The mesh is left out of place and must minimise its energy values

Figure 5-1. Showing the first and second frame of the translation. The image object and the mesh are captured directly from the active-mesh application.

5.2.1 Translation Results

This test will show the movement of the mesh purely using external forces. The object in the scene, a simple rectangular shape is translated left and slightly up after the mesh has been initialised in the first frame. The mesh internal lengths should be constant; with almost no internal forces being applied, as the *CurLenght* and *SetLength* values or the mesh lines are close to equal during the iterations. The mesh is generated using the Delaunay triangulation procedure as discussed previously and the determination of the mesh node positions is performed using the SUSAN corner detector. In Figure 5-1, frame 2, the mesh is displaced from its desired position in the image. The distance is relatively large, but still within the default search space of the mesh nodes (in this case a circular search space with a 25-pixel radius).

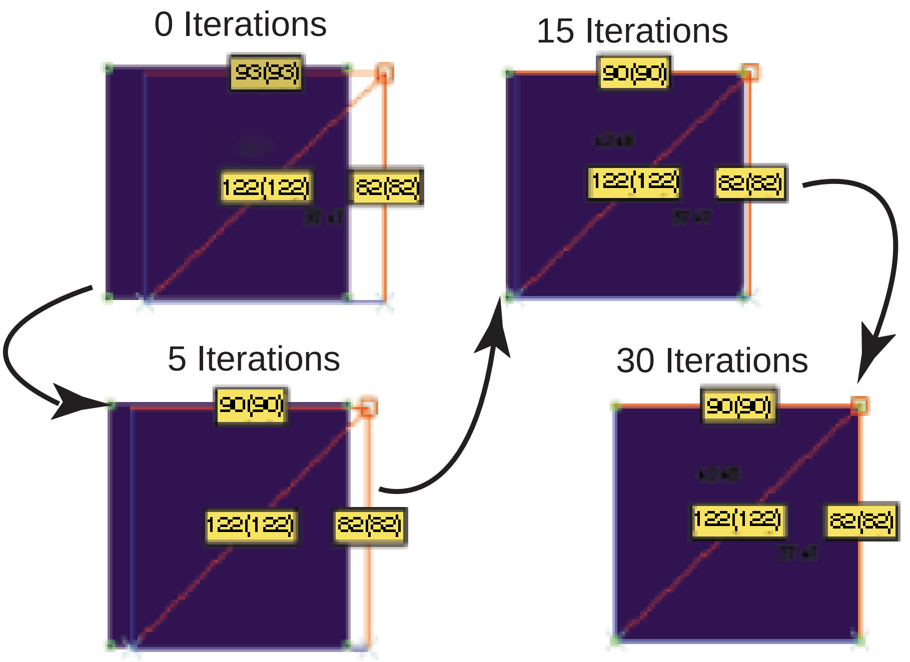
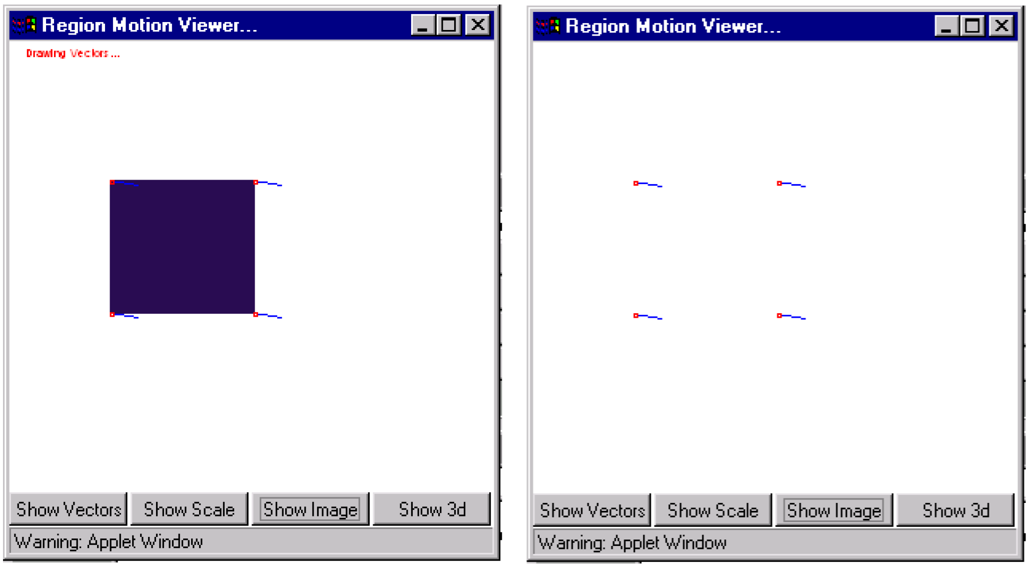


Figure 5-2. The sequence of iterations (at 0, 5, 15 and then 35 iterations) leading to a suitable solution.

Figure 5-2, shows the position of the mesh after different numbers of iterations. After 5 iterations the mesh is pulled almost half way towards the correct solution and by 15 iterations it has almost settled. After 15 more iterations, the mesh settles to the correct solution. The mesh is permitted to approach a correct solution slowly, so as to:

- Prevent the mesh from being pulled in the wrong direction by spurious data.
- Prevent the mesh from overshooting the correct solution, as such an overshoot can have a unstabilising ‘knock-on’ effect on the mesh, pushing correct nodes out of place and after sufficient iterations completely deforming the mesh.
- This rate of approach to a correct solution may also be varied for a particular application, such as security or biological cell tracking where the maximum velocity may be reasonably defined.



(a) (b)

Figure 5-3. The vector results from Figure 5-2, overlaid on the image in (a) and without the image in (b).

Figure 5-3, shows the results of this translation in vector form at each of the mesh nodes. In this image and in the rest of the results images the 3 x 3 pixel rectangular head on the vector represents the destination point of the vector hence showing the direction of the vector.

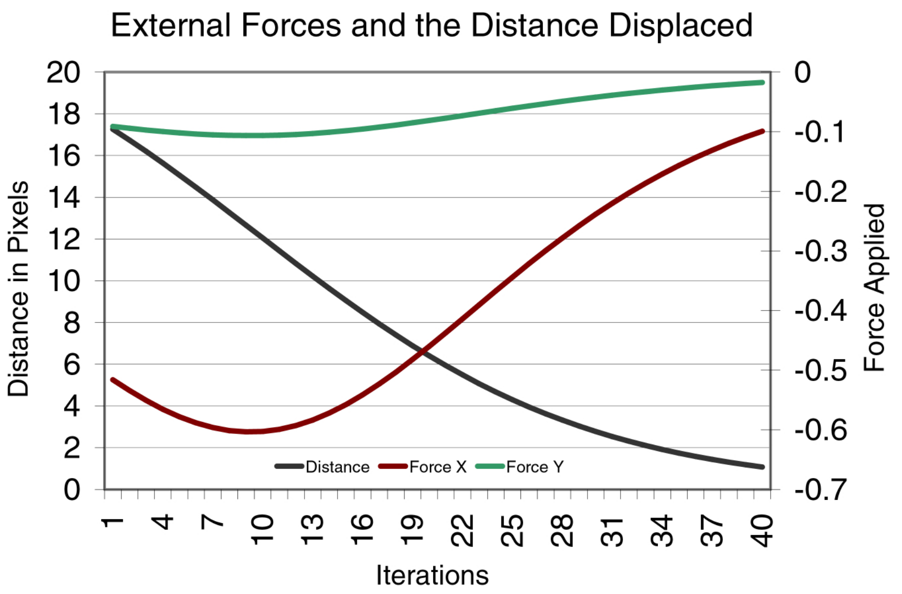


Figure 5-4. The external forces and the distance displaced at node 2.

Figure 5-4, shows the external forces that occur at node 2 and the distance of the node from the desired solution. The red and green lines represent the *x* and *y* forces respectively and the black line represents the distance of the mesh node from the desired solution. As can be seen in this case, a large -*x* and a smaller -*y* component were required to move the mesh to a suitable solution, i.e. left, (-*x*) and up, (-*y*) slightly. The distance is given as the black line starting at around 17 pixels from the ideal solution and minimising close to that solution after approximately 40 iterations. Figure 5-5 below shows the effect of varying the external energy variable value (α_E) from 0.01 to 0.9 and its effect on the settling time of the mesh on the correct result. Figure 5-5 illustrates that by choosing a value greater than the value previously chosen of 0.1 would have resulted in a faster settling time of the mesh on the desired solution, while choosing a smaller value would have increased the number of iterations required to settle on a suitable solution.

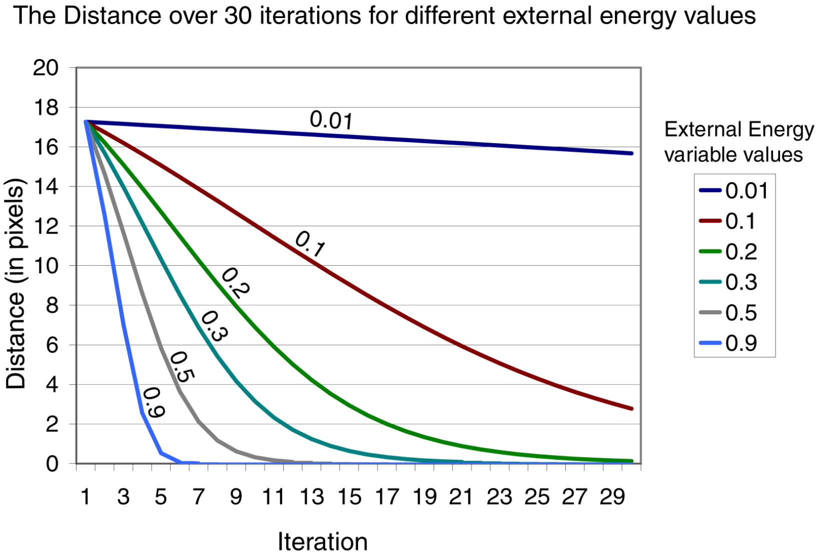


Figure 5-5. The distance over 30 iterations for different external energy values, as shown in the key on the right of the graph. The results were calculated based on node 2 in the previous example, where a value of 0.1 was used.

There is a 'trade-off' between α_E and the value of the internal energy α_I in the snake. If α_E is too large then the snake will be pulled in the direction of any external forces without being constrained by the internal forces. This choice of parameter values is very similar to the choice of λ and *(1-λ)* in the regularisation stage of active contour implementation (see Section 3.2.5). In effect these user-defined variables determine how fast the mesh will settle, traded off against stability and the effect of the internal forces on the mesh.

5.2.2 Point Displacement Results

This discussion will explain the operation and use of the internal forces in the mesh. The internal force formulations were developed to preserve the structure of the mesh and to allow a level of deformity in the mesh. In this test, a substantial impulse force is applied to a single node of the mesh, and its effects on the mesh are then examined. This kind of impulse force would be an extreme version of noise in an image sequence, where an incorrect close match for a mesh node might appear briefly in an image sequence. It is desired that the properties of the mesh would cause a force of this nature to have a very small impact on the structure of the mesh, allowing the mesh to be tolerant of incorrect node energy minimisations. The main influence of external forces should be restricted to cases where they are applied over several frames and are not localised to a single node, rather several similar forces being applied to a number of clustered nodes.

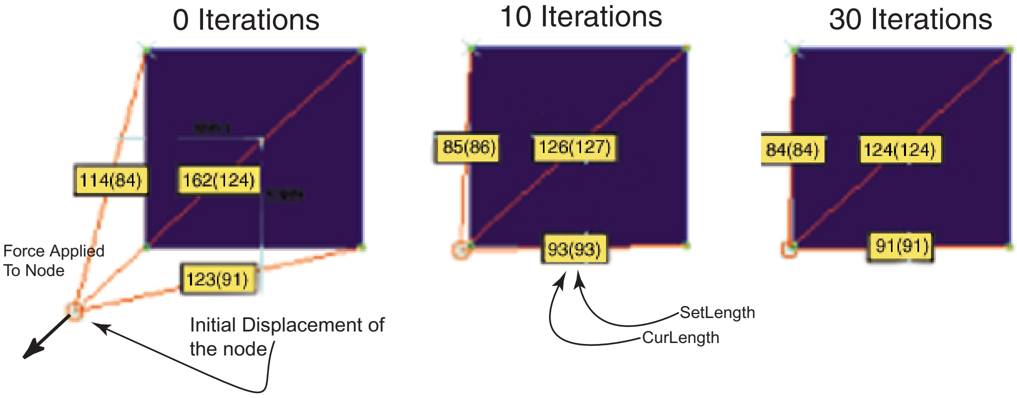


Figure 5-6. Illustrating the effect over 30 iterations of a force applied before iteration 0. The red lines represent the mesh lines. The boxes with the numbers give the *SetLength* and the *CurLength* as described previously.

In Figure 5-6, the force is applied at iteration 0. The interface that was developed allows a node to be dragged by the mouse to a set location. This is performed before iteration 0 causing the *CurLength* value of the displaced node connected mesh lines to be dramatically extended. The node is pulled a distance of around 40 pixels. As can be seen in Figure 5-6 (frame 0), the *SetLength* of the mesh lines is the same length, so this dragging has no effect before the iterations begin. Time is in effect stopped at this point and no force update iterations are performed until the user allows.

The mesh converges close to its original position after around 10 iterations and after 30 iterations it has almost completely converged to the original position of the mesh. The force that is applied is not constant over the sequence of frames, only applied before the first frame, so it should have no serious structural effect on the mesh. In this particular case the mesh should act exactly like a snake wrapped around the outside of the square, converging back to the square over a time period.

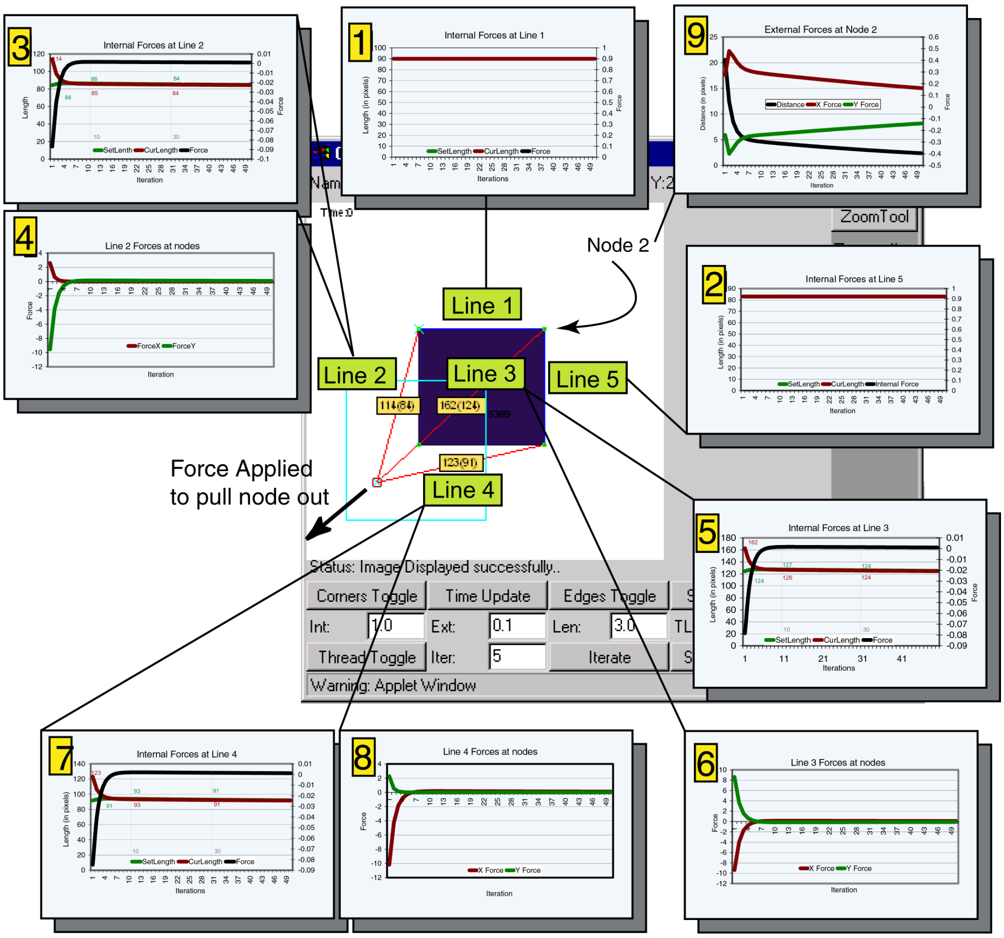


Figure 5-7. Showing a summary outline of the effects of the internal and external forces on the mesh lines and at the mesh nodes. The graphs are numbered so that they can be referred to in the following pages.

Figure 5-7, outlines the operation of the internal and external forces within the mesh. It shows each mesh lines association with its respective graphs. The graphs are sampled over 50 iterations. The results to be expected are that lines 2, 3 and 4 reduce in length to near their original length while lines 1 and 5 should try to conserve their current lengths. The external forces should also attempt to prevent this applied force from pulling the mesh away from the rectangle, by pulling the displaced nodes back to their current position on the image. The effect of having no external forces is explained later (in Section 5.23). The graphs labelled in Figure 5-7 are shown next and explained in detail.

Figure 5-8, shows Graph 1 and Graph 2 representing the changes occurring in the internal forces and lengths of line 1 and line 5 respectively. The *CurLength* and *SetLength* values are exactly equal for both graphs, with the *CurLength* being overlaid on top of the *SetLength* in both graphs. The *Internal Force* value in both graphs is also constant and has a value of zero. This is exactly as expected, as the changes in lengths of the lines (2, 3, 4) should not have an effect on the lengths of line 1 or line 5. Since there is no difference in length between the *CurLength* and the *SetLength* the internal force value is zero. If this force were applied to node 2 instead of the current node then there would be a similar change in length of the other lines in the mesh.

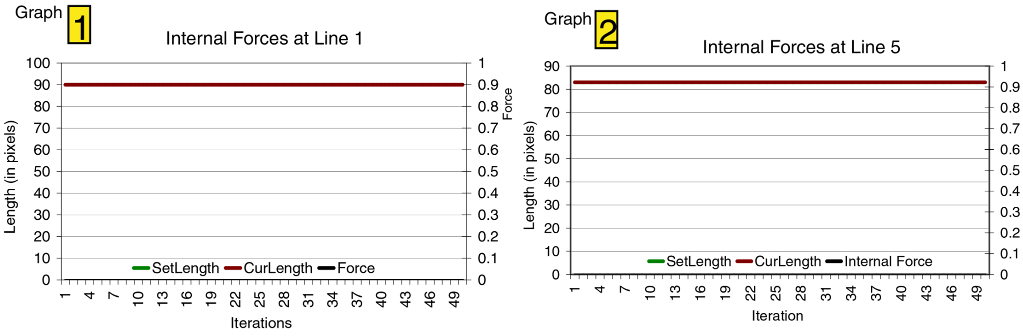


Figure 5-8. Graph 1 and 2; illustrate the effects of the internal forces at line 1 and at line 5 respectively.

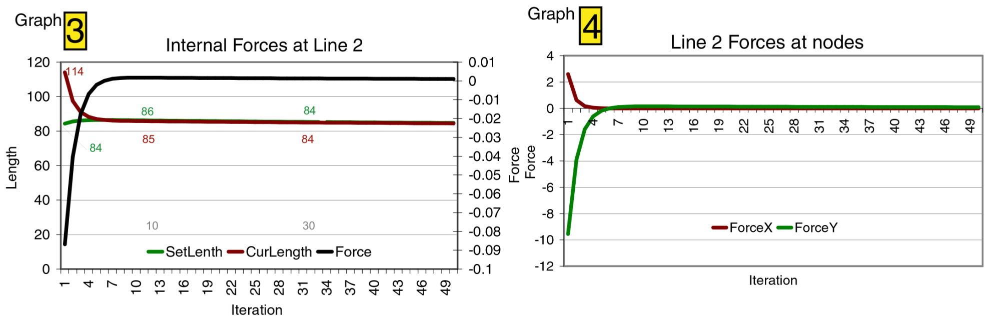


Figure 5-9. The internal forces at line 2 are shown in Graph 3 with the equivalent forces at the nodes shown in Graph 4.

Figure 5-9, shows graph 3 and graph 4 as summarised Figure 5-7, showing the internal forces and lengths and the external forces at the nodes respectively. Figure 5-9(a) shows the red line representing the *CurLength* value, reducing from an initial length of 114 down to a length of 84 pixels quite quickly (after 10 iterations). The green line representing the *SetLength* is forced to a value of 85 but due to external forces the value reduces back to 84.

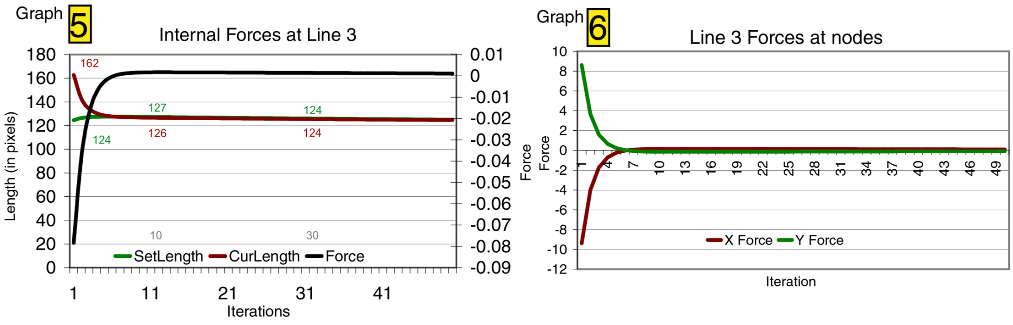


Figure 5-10. Graph 5, showing the internal forces at line 3 and Graph 6 showing the forces at both of the line 3 nodes.

The difference between the *CurLength* and the *SetLength* causes the magnitude to be initially large, falling off quite quickly after 10 iterations to a zero value. The graph in Figure 5-9(b) shows the forces that are resulting at the ends of the mesh line due to the force value from Figure 5-9(a). The large *x* and -*y* forces cause the mesh line to shrink in length. These forces fall off quickly (10 iterations) to a zero value, as no further change in the length of the mesh line is required when the *CurLength* and *SetLength* values are equal.

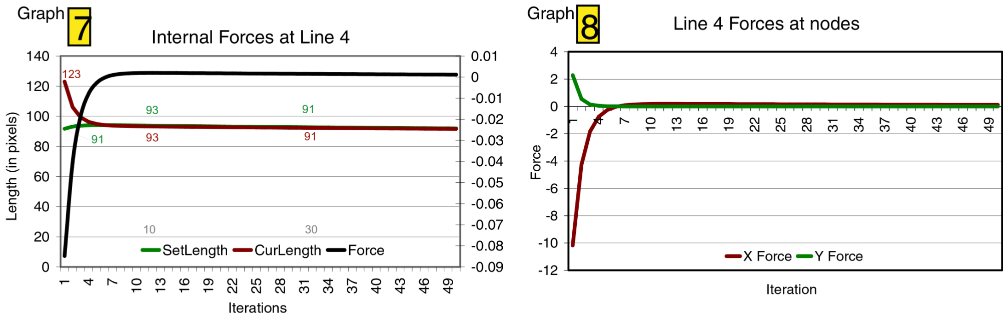


Figure 5-11. Graph 7, showing the internal forces at line 4 and Graph 8, showing the forces at both of the line 4 nodes.

Figure 5-10, shows the resulting graphs 5 and 6 showing the internal forces occurring at line 3. Graph 5, shows the large difference between the *SetLength* and the *CurLength*. This difference causes the resulting force to be also large. Since the mesh line is almost at a 45-degree angle the resulting *x* and *y* forces as shown in graph 6 are almost equal in magnitude for the iterations. This has the effect of pulling the displaced node up and right.

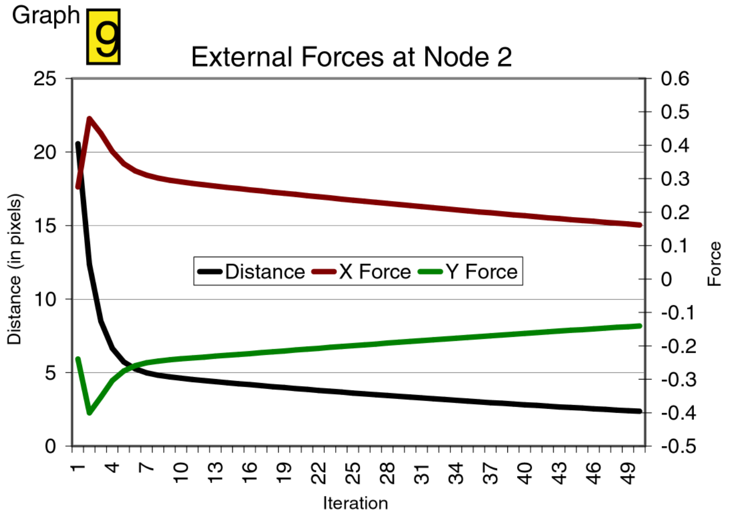


Figure 5-12. showing the external forces occurring at node 2 in Figure 5-7.

Figure 5-11 shows the resulting graphs 7 and 8. These graphs are very similar to those in Figure 5-9, except for the direction of the forces. Figure 5-12, shows Graph 9, an example of the external forces occurring at a node in the mesh. The *x* and *y* forces are pulling the mesh node towards the detected corner in the image. These external forces keep the mesh in place on the image features. Without these external forces, the entire mesh would be pulled in the direction of the applied force and displaced from the image features.

Figure 5-13, shows the vectors overlaid on the mesh (outlined). The vectors are an illustration of what occurs in the 40 iterations that took place. The internal forces are calculated for each mesh line within the mesh. If the mesh line is required to shrink or expand then the appropriate forces are added to the two nodes at the ends of the mesh line. The external forces are also calculated for each node and added to the node. When all the forces have been calculated at each node the forces are summed and the resulting force is applied. As can be seen in Figure 5-13, this has the form of a force pulling the displaced node back in towards the mesh, while keeping the remaining nodes located in the position of the detected best match corners.

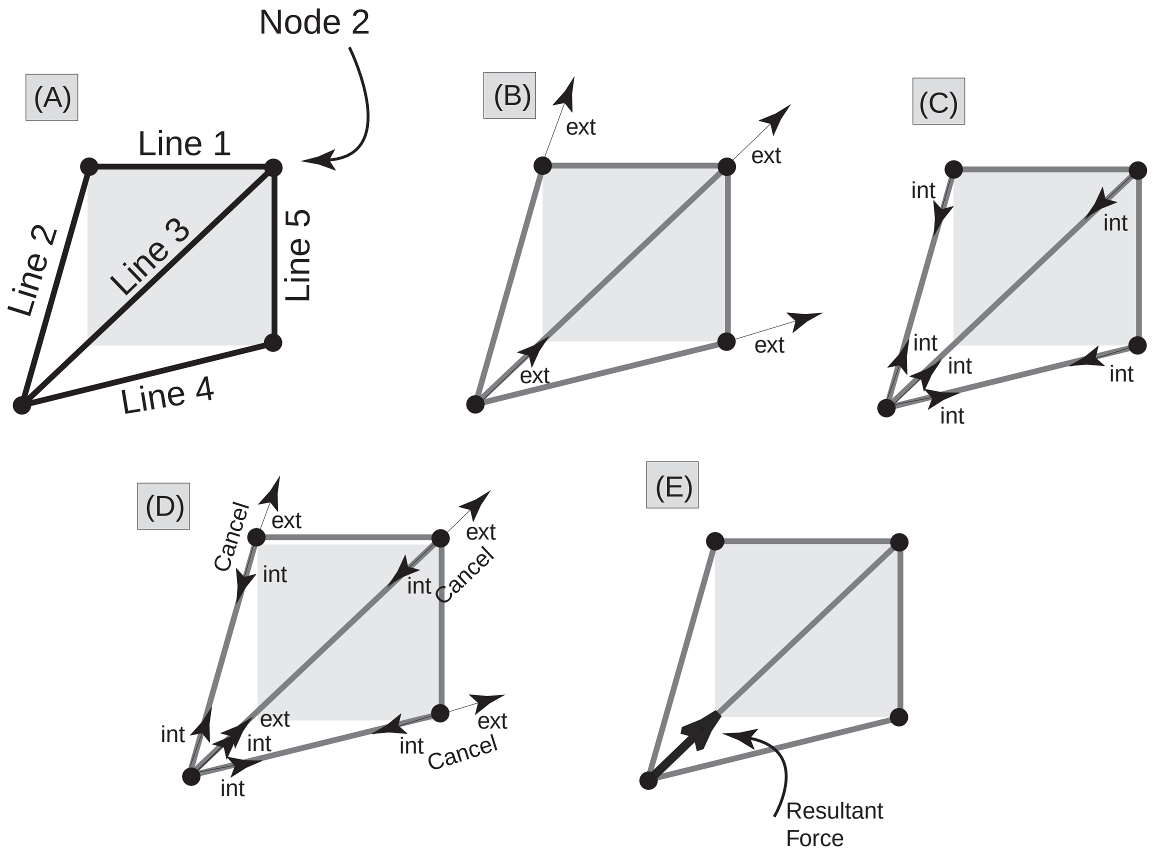


Figure 5-13. Illustrates the various forces occurring on the mesh at a particular time iteration. (A) Shows the mesh itself, (B) displays the external forces occurring on the mesh, due to the mesh nodes being pulled towards the detected corners in the image. (C) Shows the internal forces on the individual mesh lines, trying to minimise the difference between the *SetLength* and the *CurLength*. (D) Shows the different forces cancelling each other out and (E) displays the resulting force.

5.2.3 Point Displacement with no External Forces

To show the effects of the internal forces in the absence of external forces, the same point is displaced using the same forces as the previous case. However, in this case the second image frame is a blank image, with no corners detectable. The mesh has to minimise the internal energy within the mesh without any external forces to hold the mesh in place on the image features.

The mesh frames are shown as Figure 5-14, where the point is initialised on the object in the first frame. The mesh node is displaced and then the object disappears. Rather than the mesh shrinking to a point (or a circle) as occurs in many active contour implementations the mesh retains its overall structure, however as can be seen in Figure 5-15, the mesh is displaced from its original position due to the forces applied by the displaced node. This is a similar advantage to the properties of the dual active contour approach, discussed in Section 3.5.

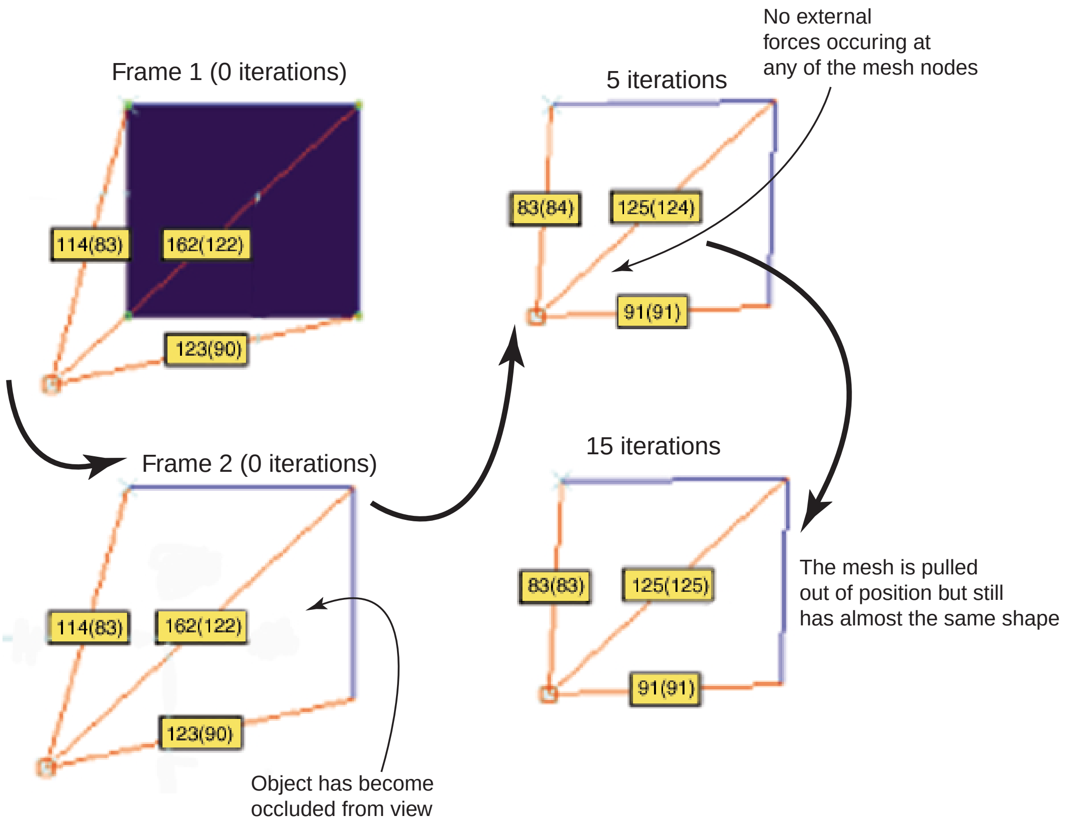

Frame 1 (0 iterations)
114(83)
162(122)
123(90)
Frame 2 (0 iterations)
114(83)
162(122)
123(90)
Object has become occluded from view
5 iterations
No external forces occuring at any of the mesh nodes
83(84)
125(124)
91(91)
15 iterations
83(83)
125(125)
91(91)
The mesh is pulled out of position but still has almost the same shape

Figure 5-14. The sequence of iterations when a point is displaced and the object has become occluded.

The mesh almost retains the same shape because the lengths of the mesh lines are still the same lengths as when previously initialised. However, the mesh is skewed slightly due to the forces created from the displaced node. If the object in the frame were to once again appear then the resulting external forces would pull the mesh again back to the location of the original mesh after only a few iterations.

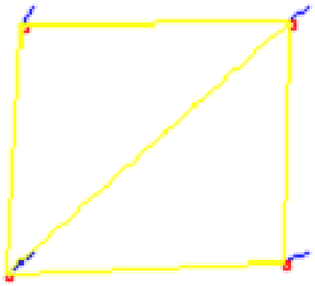
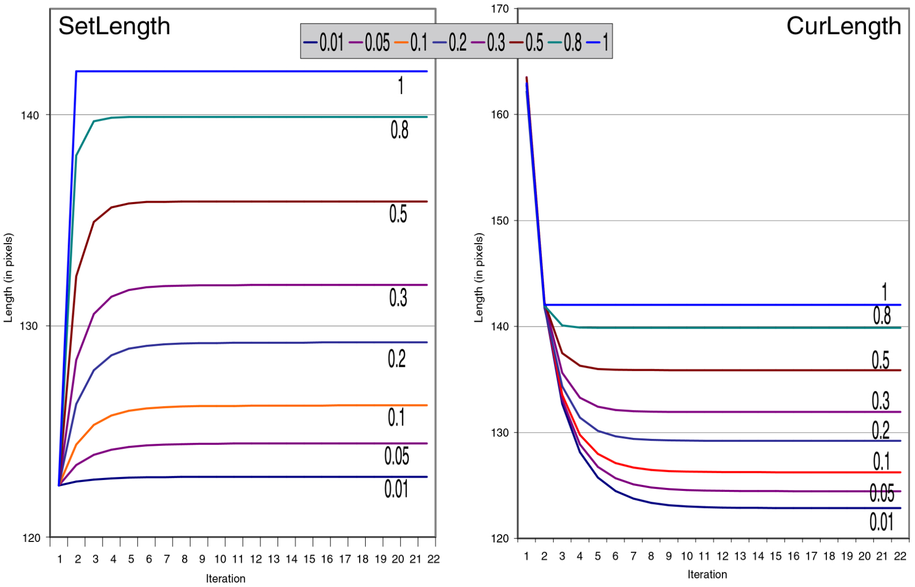


Figure 5-15. The results from the point displaced sequence when the object becomes occluded and there are no external forces possible. The vectors display the displacement between frames.

This property of the active mesh is especially useful in the presence of occlusion or random feature noise. The mesh is in equilibrium when the current mesh is similar to the stored mesh, in which case the energy is minimised. This prevents the mesh from collapsing down to a point or a line allowing a more stable implementation in real-world scenes.



(a) (b)

Figure 5-16. The parameter for setting the internal energy value as applied to the example in Section 5.2.2. with the displacement shown as in Figure 5-6, where (a) shows the *SetLength* of the mesh line, whereas (b) shows the *CurLength* of the mesh line.

Figure 5-16, illustrates the effect of altering the internal line length value (α_I) on the settling length of the *SetLength* and *CurLength* values, where the graph of a value in (a) has a corresponding value in (b) denoted by the same colour. If a large value is chosen then the *SetLength* changes quickly to the value of *CurLength*. If such a value was used the mesh would be very deformable, changing in shape with every force that is applied. If on the other hand a small value is chosen, as the threshold value then the mesh will be very rigid, with external forces having little or no effect on the structure of the mesh.

5.3 Results on Test Templates

The image shown in Figure 5-17 was used as a template to test the general structure of the snake being examined. The image was drawn with discrete grey level intensities with the corners perfectly defined (importantly, without any form of image aliasing). The corners in the initial frame are detected very accurately. Shapes were chosen to test the mesh with different movements, with the objects having the ability to move as a cluster or individually. From this image movements can be simulated and the structural deformations and movements of the mesh can be examined. The shapes were chosen to test complex and less complex structure sections of the active mesh. The shape on the top right results in a complex mesh while the shape on the top left allows a less dense mesh. The unstructured shape in the bottom right exercises the movement of uniform intensity corners. The shape in the centre is another shape that should have mesh nodes close together and spaced apart. The shape at the bottom right can be used to test the movement of independent uniform objects.

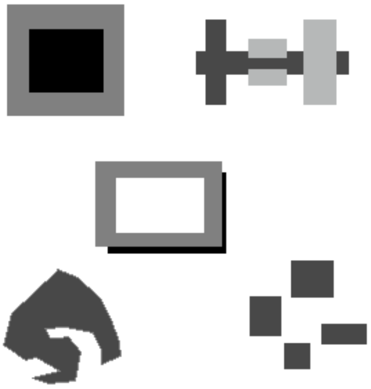
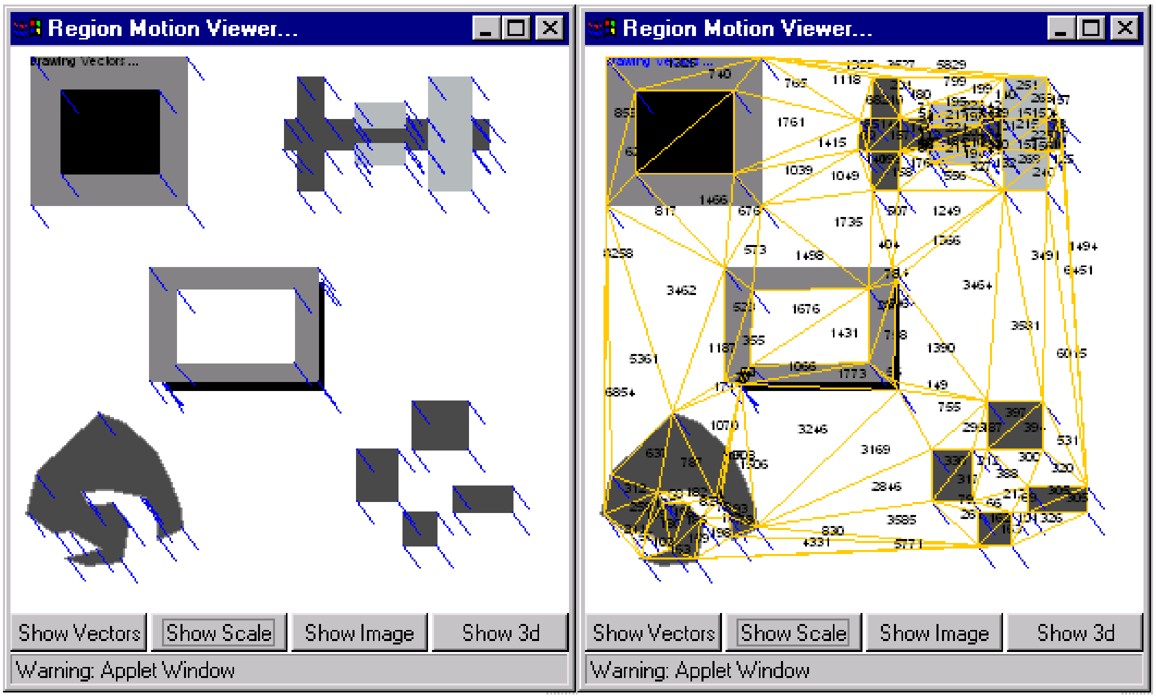


Figure 5-17. The example template used.

5.3.1 Test on Scene Translation

The first test involved a basic translation of the entire image from one point in the image to another point by a sizeable amount. All objects in the scene are moving in the same direction (left and up).



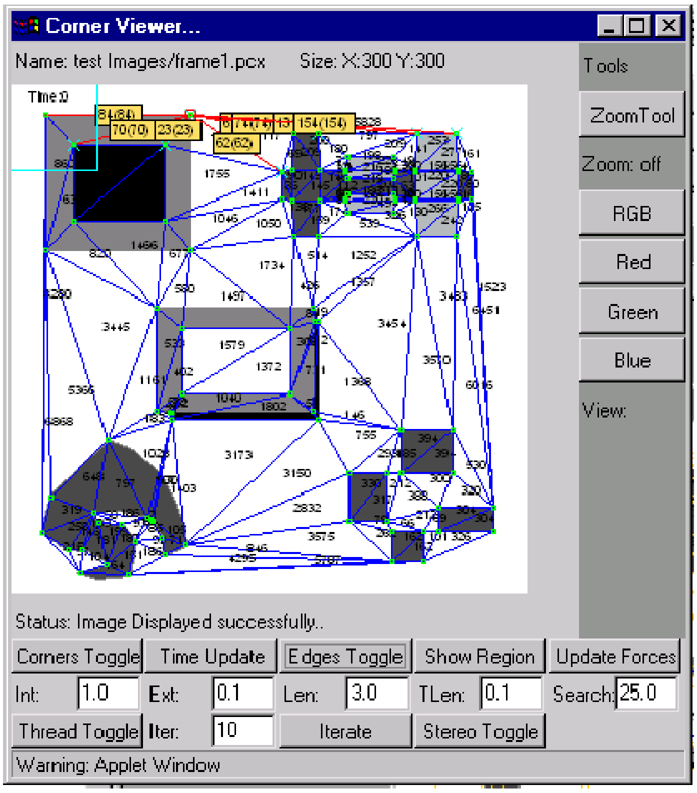
(a) (b)

Figure 5-18. (a) Displays the vector field overlaid on the image objects and (b) shows the same image with the addition of the mesh that was generated from the corner points. The numbers in (b) are for display purposes only showing the area of the triangle (in pixels) where that value is situated. The areas are used at a later staging to determine the scaling of a particular region.

As can be seen from the images in Figure 5-18 the motion is well described. There are very few incorrect vectors in the calculated vector field, as the mesh is preserving the match, even thought the frame translation jump is in excess of 10 pixels in length. This is a straightforward test for the mesh, as the mesh remains rigid, with no internal deformations. The only difficult aspect is with the search space of some nodes, where identical match corners are available due to the synthetic nature of the image. However, the overall movement of the mesh in a left/upward direction allows these nodes to locate the correct match feature.

5.3.2 Sub-Area Translation

The next step was to test a sub object translation of a section of the image frame. This is important in testing the internal deformity of the mesh. This type of deformation may be necessary, for instance, in the case where a single mesh is tracking the scene background as well as some objects moving independent of the background. The mesh in this case should be able to conserve its overall shape while allowing the determined sub-area to deform within the mesh. The mesh requires a level of deformity if it is to track independent objects in real-world scenes, where they are moving independent of the background or at different depths in the scene. This however causes problems in the trade-off with deformity against maintaining a rigid mesh. Even if the objects are rigid and the scene undergoes a scaling then the mesh must deform to maintain exact motion tracking.



Corner Viewer...

Name: test Images/frame1.pcx   Size: X:300 Y:300

Time:0

Zoom: off

Status: Image Displayed successfully..

Int: 1.0   Ext: 0.1   Len: 3.0   TLen: 0.1   Search: 25.0

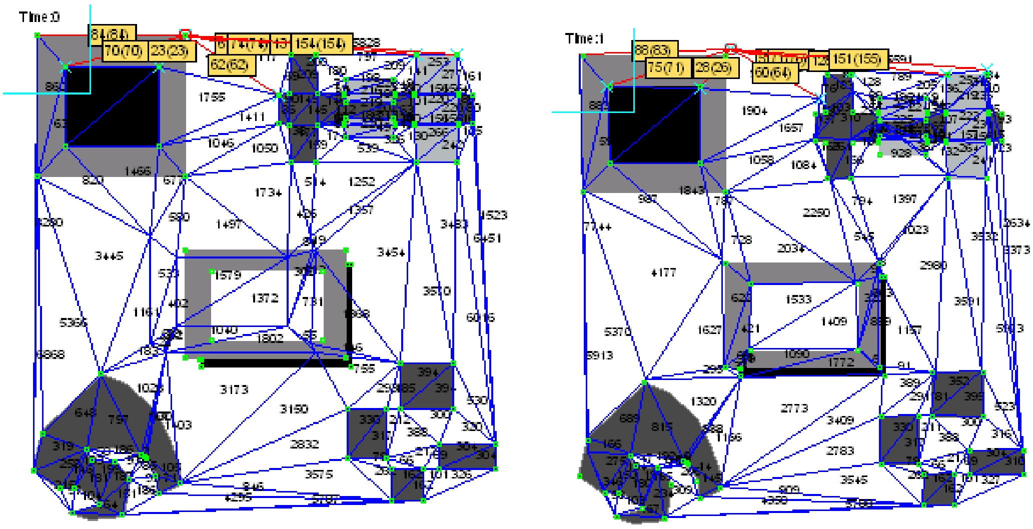
Iter: 10

Warning: Applet Window

Figure 5-19. Displaying the operation of the mesh-tracking algorithm. Where *Int* (α_I) refers to the value of the internal regularisation parameter, *Ext* (α_E) refers to the value of the external regularisation parameter, *Len* (α_{Line}) is the value of the length factor (usually 3.0), *Tlen* is the value of the length update value, *Search* represents the search area radius (r) of the snaxels (in pixels) and the *Iter* value is the number of iterations to perform using the Iterate button.

Figure 5-19, shows the active mesh Java application in operation, along with the parameters required. For each of these test examples the settings were set as above with about 60 iterations for the active mesh to settle to its final value, showing that while these parameters exist, the default values are good enough for the majority of applications. They are still available, to allow fine-tuning of the algorithm.

In Figure 5-20(a) the mesh is shown at its initial state, before any iterations have taken place. The majority of nodes are already settled on their best match points, except for the centre of the image where the translation has taken place. This translation is quite large (in the order of 15 to 20 pixels) and should involve substantial deformation of the mesh. In Figure 5-20(b) we see the mesh after 60 iterations, i.e. after it has settled on the image. This structural deformation is caused since the large external force is consistent throughout the 60 iterations, expanding the lengths and contracting the lengths of numerous mesh lines. Again, the same variable values are used as in the previous cases. The results from the sub-translation are shown as in Figure 5-21, where they are very



(a) (b)

Figure 5-20. (a) Shows the mesh overlaying on the new frame before any iterations have taken place. In (b) the mesh has undergone 60 iterations and has settled on the new image.

exact except for one vector in the bottom left corner of the sub-object. This deformation might be necessary in the case of small objects within the scene moving in a different way than the scene itself. The majority of the mesh still settles on the same image locations, as the new image position of the sub-object is pulling the nodes inwards, but the non-displaced nodes are maintaining their previous locations.

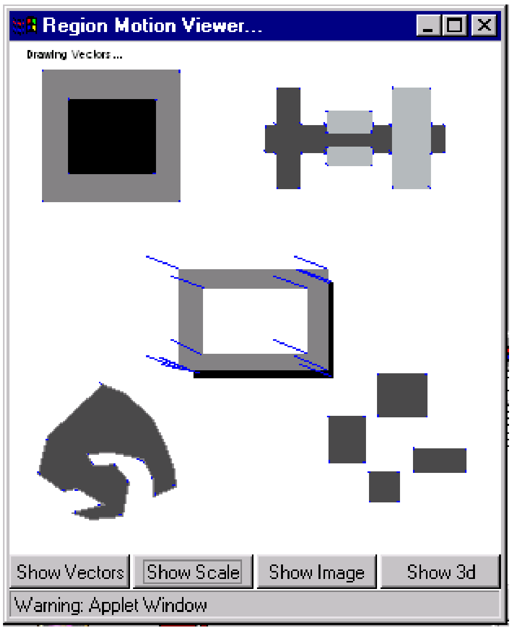
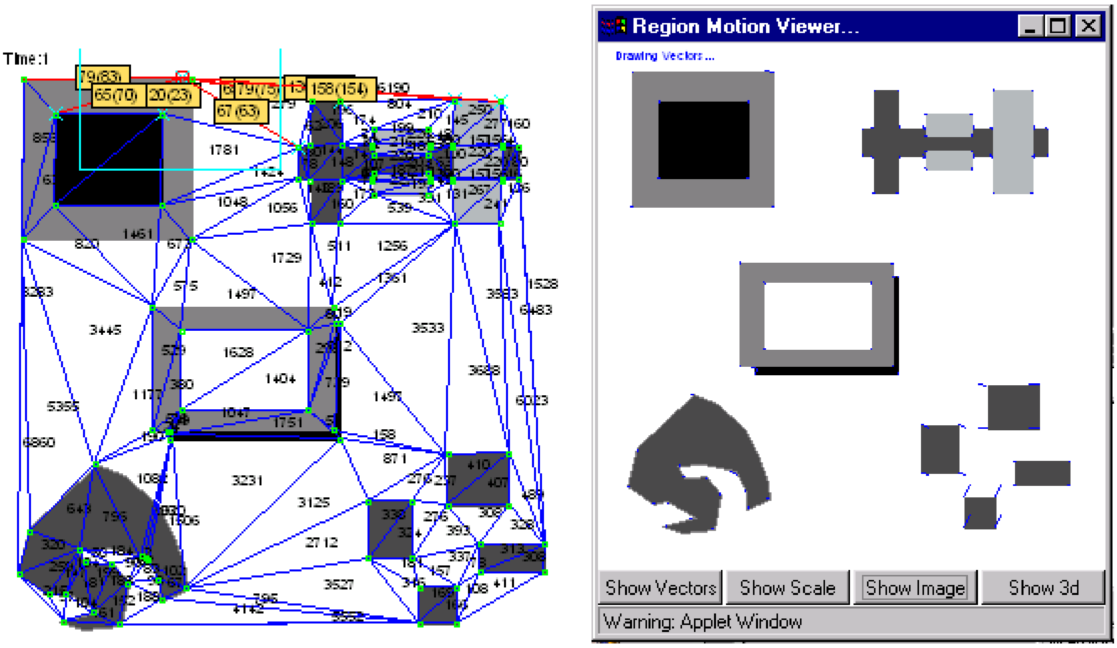


Figure 5-21. The resulting vectors overlaid on the second frame of the sequence, from Figure 5-20.

5.3.3 Multiple Sub-Region Translations



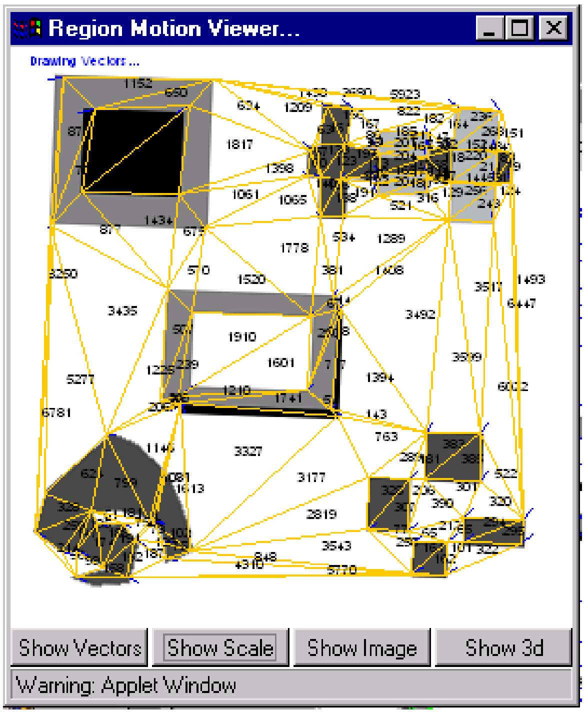
(a) (b)

Figure 5-22. Displays the effects of the movement of the objects in the bottom right of the scene. (a) Shows the mesh after it has settled on the scene and (b) displays the vectors overlaid on the second frame.

Another test was the sub-translation of another section of the image. The image in Figure 5-22(a) displays the mesh after it has finally settled. The squares in the right hand bottom corner have been displaced relative to each other. The results from this displacement are shown in Figure 5-22(b). This shows how the objects can move within the scene, without the mesh becoming dislocated, even though the corners are of similar intensity and spatial distance.

5.3.4 Rotation of the Scene

The next step was to test the mesh under a uniform scene rotation. The image was rotated by 2 degrees around the centre pixel of the image to test the meshes ability to remain fixed on image features. As can be seen in Figure 5-23 the mesh settled accurately after 60 iterations.

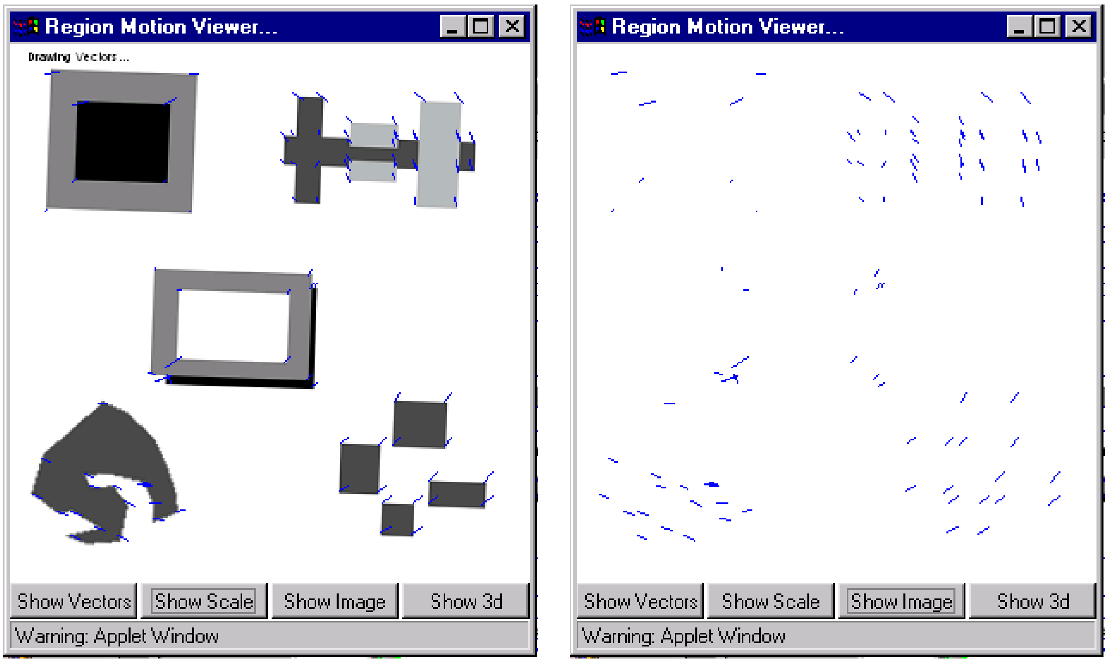


Drawing Vectors ...

Warning: Applet Window

Figure 5-23. Displays an example rotation of the scene, with the mesh and vectors overlaid on the second frame. The rotation was simulated using a standard image editing package.

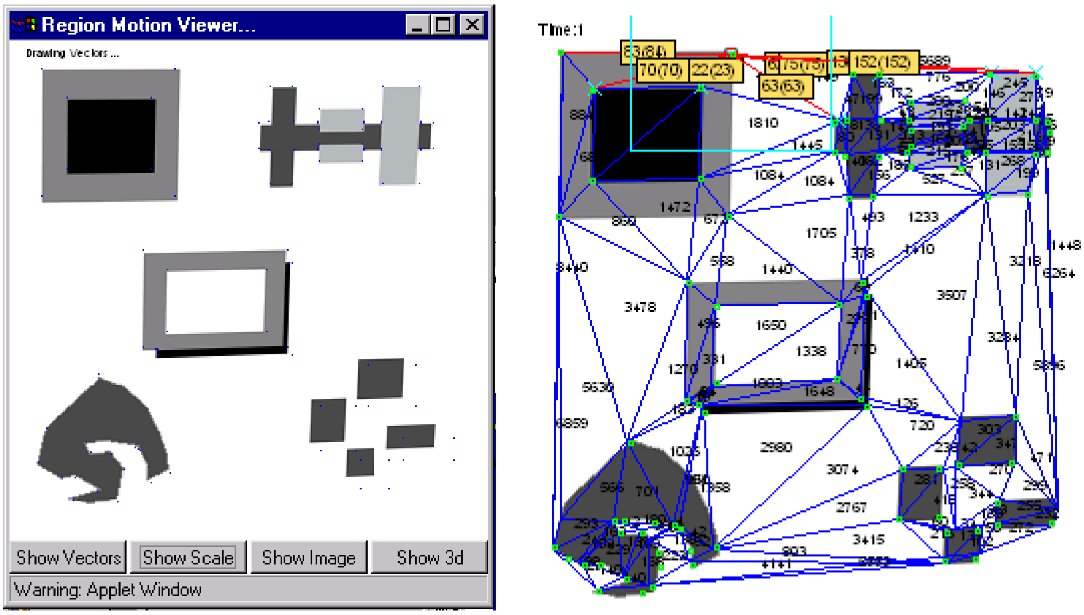
The images in Figure 5-24 show the rotation vector field with and without the image present. The results are very accurate except for some noise at the very centre of the image. The structure of the mesh is perfectly preserved through the rotation. The top left corner has been pulled out of shape slightly but this would settle if more than 60 iterations were allowed.



(a) (b)

Figure 5-24. (a) Shows the vectors overlaid on the image frame and (b) displays the vectors on the image without the image itself, showing a fairly consistent rotation pattern.

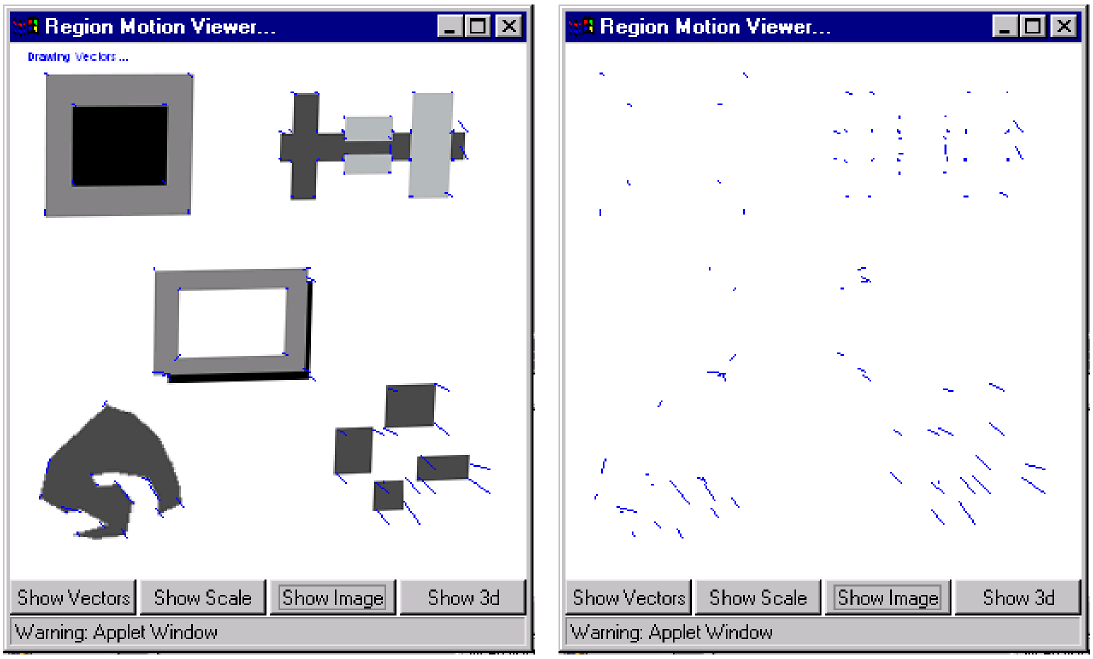
5.3.5 Distortion



(a) (b)

Figure 5-25. (a) Shows the distortion of the scene where the single pixels represent the position of the corners in the previous non-distorted image, and (b) displays the mesh after it has settled on the distorted image.

The next test was to use a non-uniform distortion, to test how the mesh would react when pulled in many different directions at the same time. Figure 5-25(a) shows the distorted image with the single pixels representing the corner points (and thus the position of the mesh nodes) prior to the distortion having taken place. The distortion was again simulated using an image processing tool[22], however a distortion of this form leads to the detection of new corners and the loss of previously detected corners, complicating the mesh energy minimisation. The algorithm was capable of dealing with this added complication, finding the most suitable corners in the search space. Figure 5-25(b) shows the mesh after it has settled on the image. Figure 5-26(a) and (b) show the final results overlaid on the image and without the image showing the final vector field.



(a) (b)

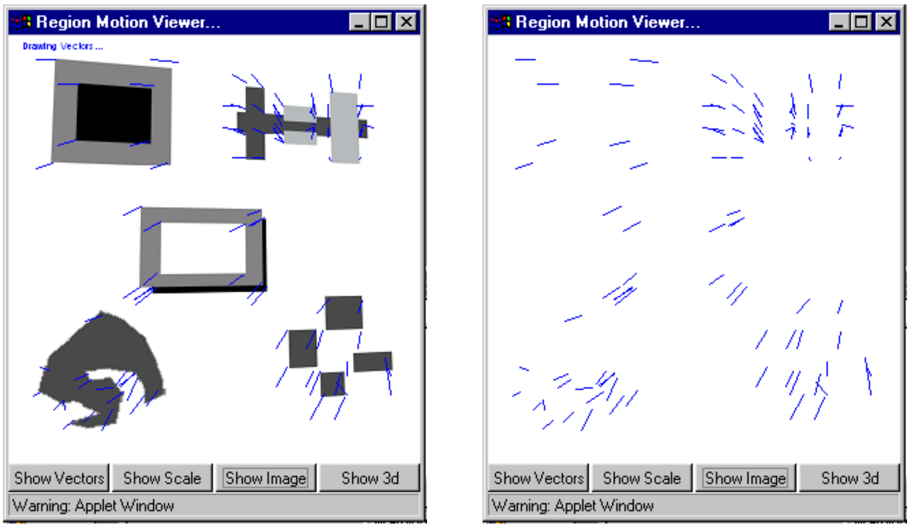
Figure 5-26. (a) Showing the vectors overlaid on the image and in (b) showing the vectors without the image.

Note that the noise at the top of the image in (b) was caused by the image distortion, rather than inaccuracies in the operation of the mesh.

[22] Adobe Photoshop 5.0 RGB warp image filters used.

5.3.6 Another Non-Linear Image Distortion

Another substantial distortion was also simulated. The largest distorted area is the top right hand corner of the image. Again for this test the parameters are left at the same values. Because of this distortion many new corners appear and some other corners disappear. This adds complexity to the task of finding a suitable solution but the mesh still minimises to a suitable solution after only 60 iterations. Figure 5-27 shows the minimised mesh super-imposed on the distorted image.



Drawing Vectors ...

Warning: Applet Window

Warning: Applet Window

(a)

(b)

Figure 5-27. The resulting vector field is super-imposed on the image in (a) and shown without the image in (b).

As can be seen in Figure 5-27 the results are promising for this case. The distortion is substantial throughout the entire image but the only area that is slightly problematic is the object in the bottom left of the image. The extra corners that are detected in the distorted image, along with the fact that the intensity values at these new corners will be very similar to the 'correct' corner matches at these points causes a little difficulty here. It is unlikely that such uniform intensity levels (to exact pixel value) will exist in real-world scenes. The vector field shown in Figure 5-27(b) gives a clear indication of the real motion of the scene, nearly like a 'black-hole' effect in the top right hand corner, with all objects being 'sucked in'.

5.3.7 Results from Scaling

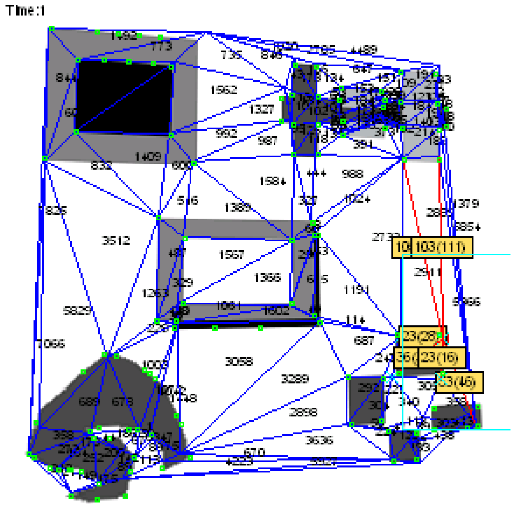
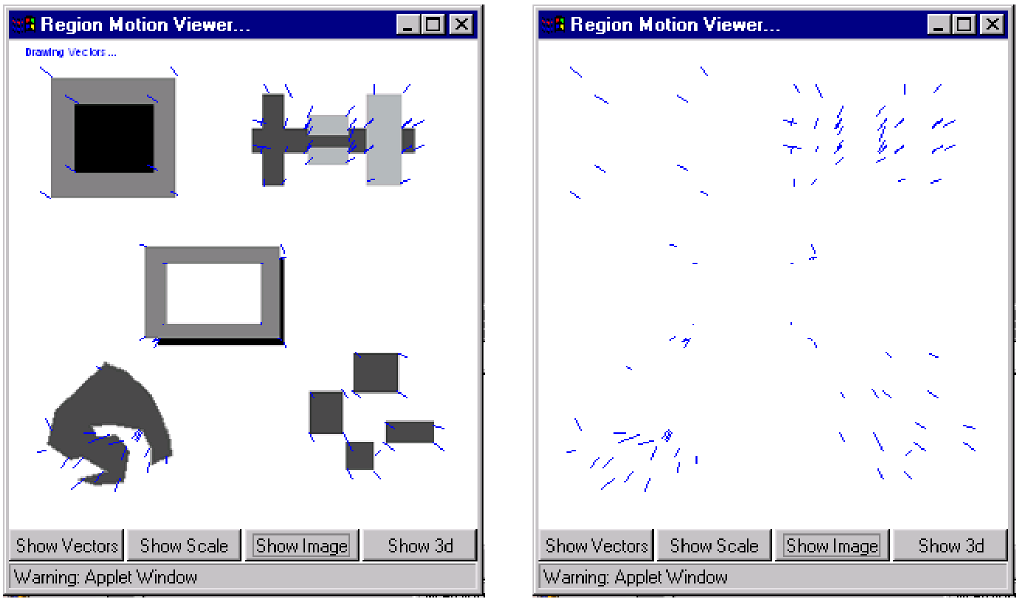


Figure 5-28. The scaled image and the resulting mesh that has converged to a solution after 60 iterations.

Another effect that had to be examined was that of uniform scaling on the mesh. In this test the external forces should force the mesh lines to shrink uniformly across the mesh, decreasing the overall size of the mesh. In Figure 5-28 the mesh is shown after it has settled on the scaled image, showing that it is quite exact. In Figure 5-29(a) and (b) the resulting vector field is displayed from this operation.



(a) (b)

Figure 5-29. (a) Displays the vector field overlaid on the image after scaling and (b) shows the same field without the image.

It is necessary that the mesh must have the ability to scale when travelling towards, or away from the scene, this however causes difficulties in the mesh as it pulls the structure of the mesh totally out of shape against the internal forces that will be produced in the mesh.

5.4 Results in Real World Sequences

5.4.1 The Corridor 1 Sequence

After testing the algorithm for simulated distortions and simulated images it was necessary to also test it using real-world images, adding the problems of noise and real-world distortions.

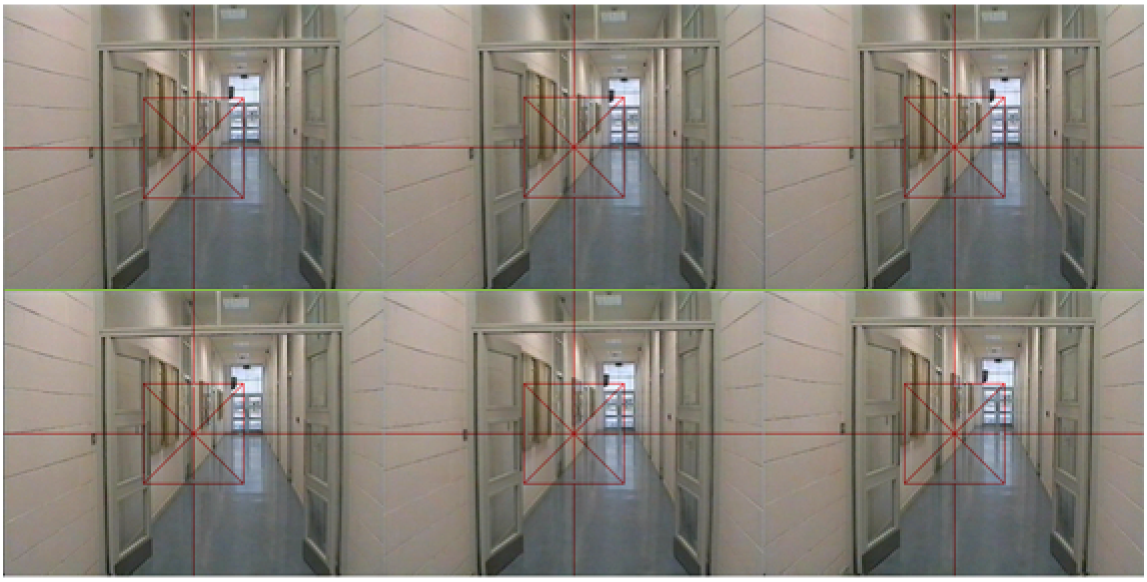


Figure 5-30. Showing 6 images from an image sequence used for testing. The images are laid out left-to-right from top-to-bottom. The grid is superimposed over the image to allow a better impression of the movement between frames.

The images in Figure 5-30 were captured using a Sun Ultra II with a Sun Video camera and capture card. The camera was mounted on a Glidcam 2000 (a camera stabilising gyroscope structure) to minimise bounce due to walking, that travelled down a corridor in DCU. The sequence was captured as an MPEG movie, so some difficult image compression effects appear in the image frames.

As would be expected the meshes can become quite detailed when dealing with real-world sequences, and it was important for the algorithm to deal with the resulting complexity. As can be seen in Figure 5-31 the mesh has ~600 mesh nodes and can still exist as a Delaunay mesh. The mesh is shown in dark blue and the new features (to be matched with) are shown as small green squares. The areas of the individual triangles are shown as small numbers. The area of the triangles can give useful information about the movement in the mesh. The areas of the triangles will increase or decrease according to scaling within the image. Similarly the triangle areas will remain constant under *x* and *y* translation.

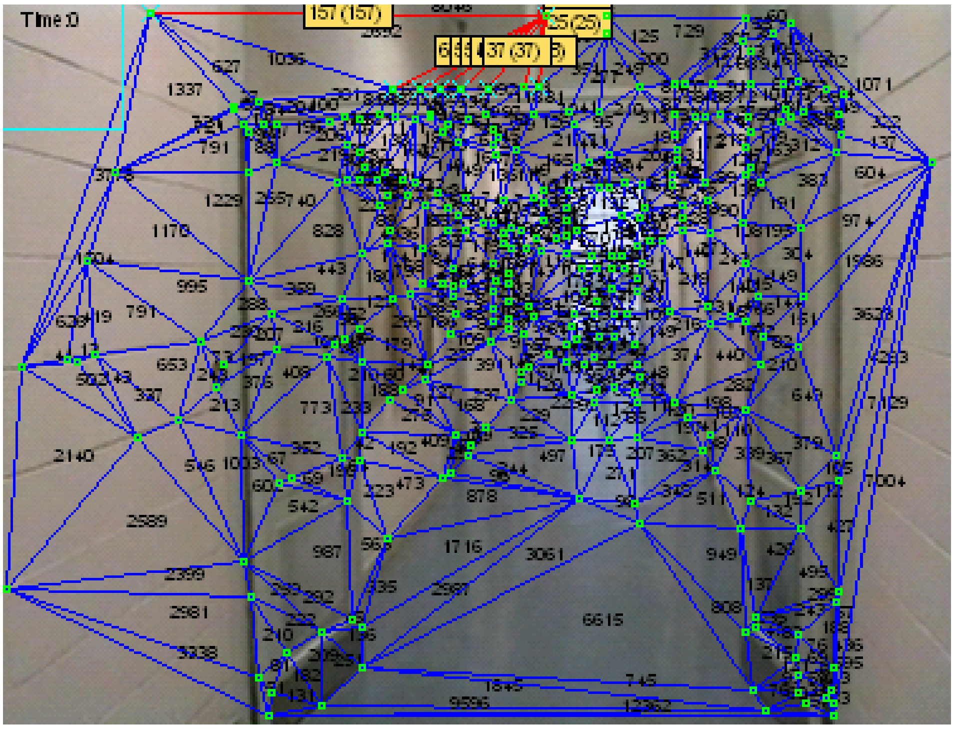
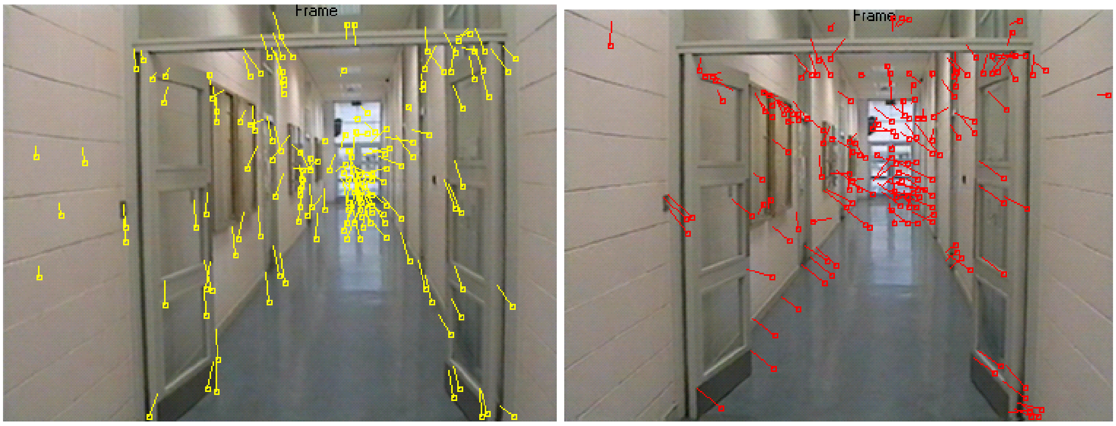


Figure 5-31. A detailed complex mesh for the corridor sequence. This mesh has 600 mesh nodes.

The algorithm is once again applied using the normal user-defined values. The higher the concentration of mesh nodes, the more structured that area becomes. In Figure 5-31, the yellow boxes give the *setLength* and *curLength* values of those particular mesh lines. It was also possible to add/remove features from the mesh while still conserving a true Delaunay triangulation, showing the Delaunay mesh creation and mesh initialisation to be stable.

Figure 5-32 shows two frames from the corridor sequence. The vector fields that are calculated are based on an intensity threshold of 20 with 30 iterations of the algorithm being performed in each frame. As can be seen the field is quite well defined in both frames, with a substantial spatial translation and scaling taking place (approx 12 pixels on average). The effect of the mesh is to globally smooth and correct the matching algorithm. Some of the vectors can be seen to be slightly incorrect, but are caused by mesh nodes not being able to locate suitable features in the subsequent frames. The active mesh approach attempts to help solve this problem, in that weakly matched mesh nodes are in effect ‘carried’ with the mesh to estimated suitable locations. In the next frame they will likely be in suitable positions for future iterations. It is important to remember that these images are affected by the ‘blocking’ effect of the MPEG encoder.



(a) (b)

Figure 5-32. Two tracked frames from the sequence in Figure 5-30.

5.4.2 The Laboratory Sequence

Another sequence that was used was the Laboratory Sequence, captured using RAW image captures. The images are cluttered, but do not suffer from any lossy compression effects. This sequence uses only two frames and the motion in the frames are being estimated only using the information from these frames.

Figure 5-33, shows the vectors calculated from two frames of the laboratory sequence, again captured within DCU. It is important to note that the vector fields shown in all these comparisons are not filtered in any way to remove erroneous vectors. In Figure 5-33(top) the pure vector fields are displayed, showing a well defined vector field. Figure 5-33(bottom) shows the same vector field with the previous feature points shown in green and the new frame feature points shown in dark red.

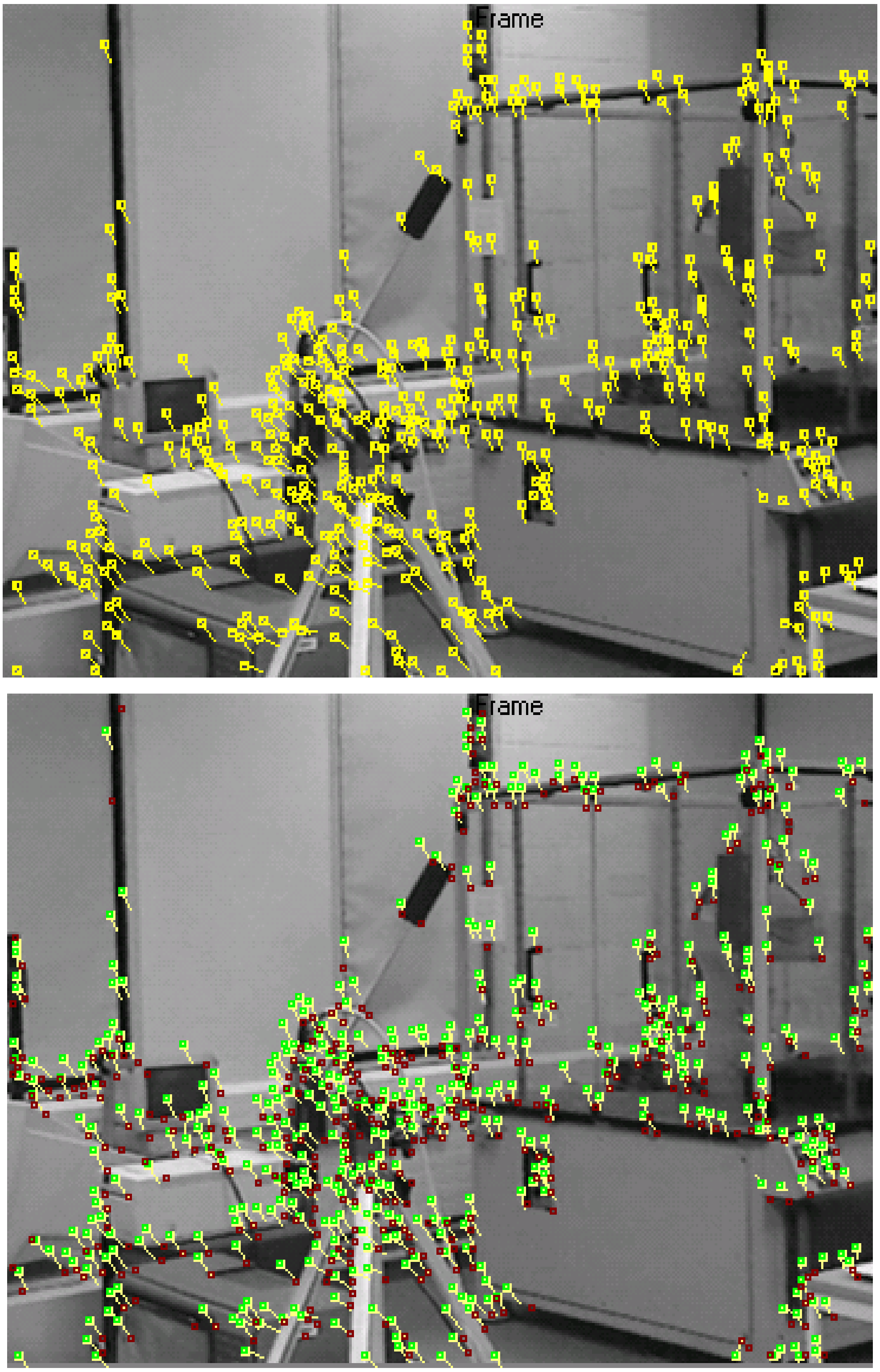


Figure 5-33. The vector fields calculated from two frames. The top image shows the pure vector field, while the bottom image shows the features from the previous frame in green and the current frame feature points in dark red. Note that several vectors are missing either a previous or current feature point.

If the image is closly examined it can be seen that the majority of vectors have settled on, or close to a new frame feature point (in most cases, what human obervers would determine to be suitable matches). In Figure 5-34, the feature detector threshold value is reduced, causing an increase in the number of mesh node points, also causing an increase in the density of the vector field. This field is not significantly different from the field in Figure 5-33, but is more densly filled.

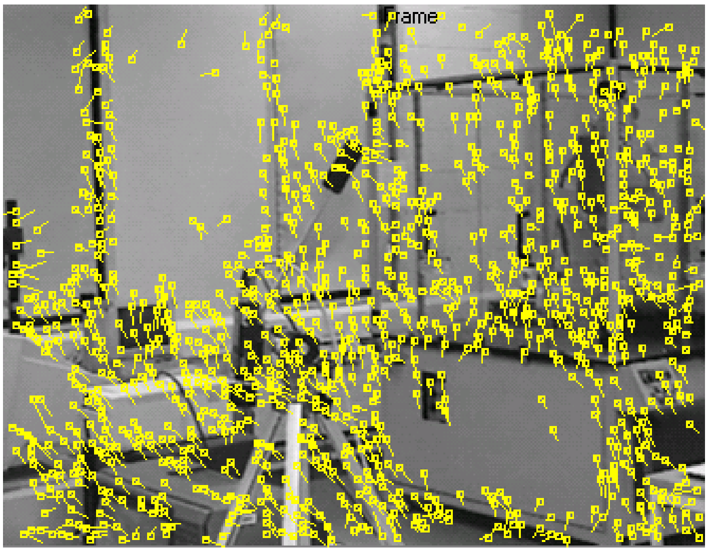
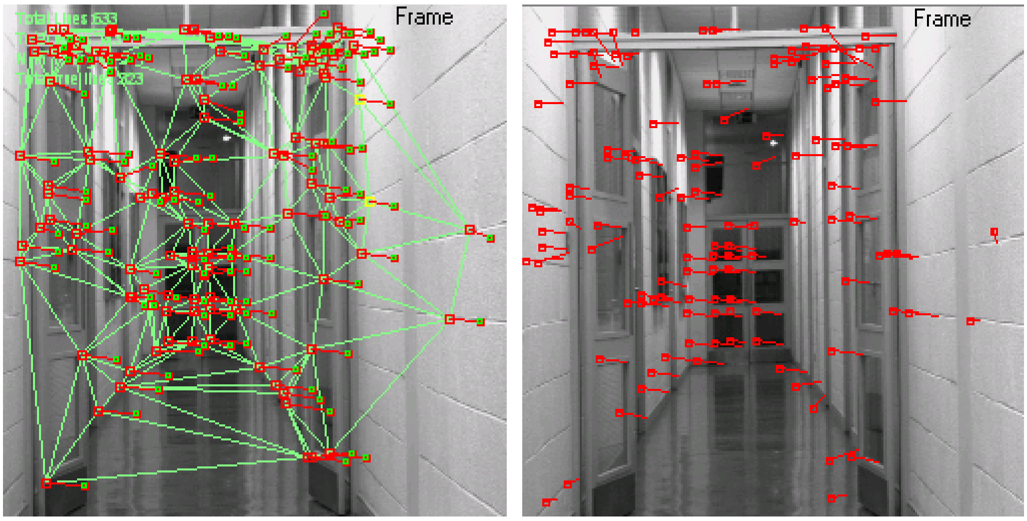


Figure 5-34. The effect of decreasing the feature detection threshold value, thus increasing the number of mesh nodes

5.4.3 The Corridor 2 Sequence



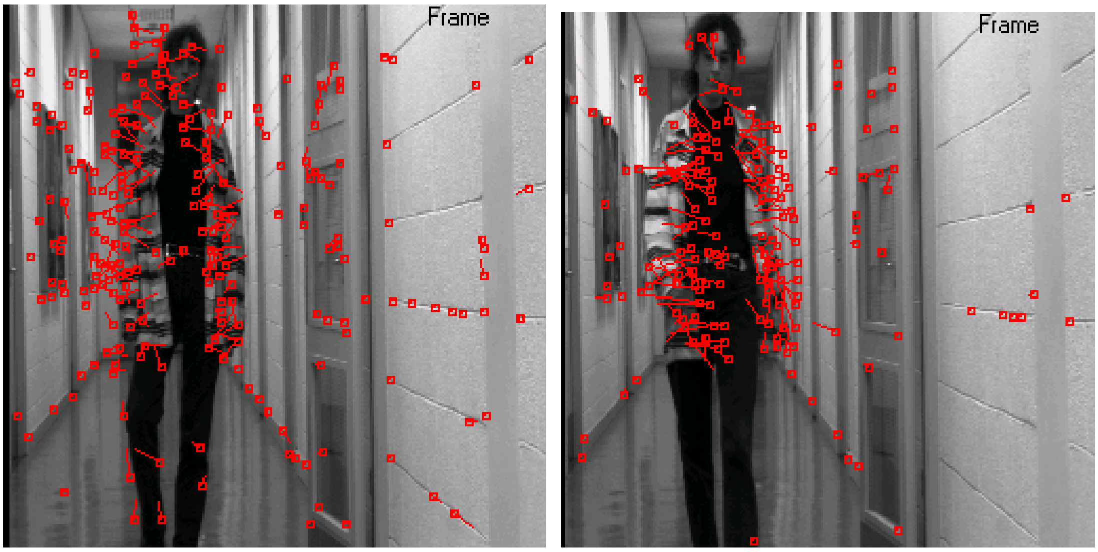
(a)

(b)

Figure 5-35. Two frames from the corridor 2 sequence.

The algorithm is also applied to the corridor 2 sequence (as discussed in Section 4.2) in the hope of obtaining better results. Figure 5-35, shows two frames from the corridor 2 sequence. In (a) the mesh is overlaid on the frame to show how well the mesh covers the entire frame, and includes the vectors at the node points. In (b) another frame is shown, with similar clear results.

Figure 5-36, shows two frames where an object (a person) has entered the frame. The camera is almost stationary in this case, but no form of image differencing is used between the frames. The stationary nodes are well defined within the image and so remain fixed, wheras the mesh nodes defining the person move from one frame to the next. In (a) the person moves slightly to the right and in (b) the person moves to the left and forward. In both cases the vector field is more clearly defined about the textured shirt, whereas the non-textured trousers do not attract the same attention. Indeed this is a difficulty with this approach, as with most motion tracking algorithms, where low textured areas are difficult to track.



(a) (b)

Figure 5-36. Two frames where a person is moving relative to the scene.

The areas that are moving with similar velocities as easily grouped using the triangle structure of the mesh. If each mesh node on a triangle has a similar velocity then that triangle is tagged with the average velocity. All connected triangles with similar velocities are grouped together as the same object.

5.4.4 Testing the Adaptive Active Mesh

The adaptive active mesh also needed to be tested on standard image sequences. The adaptive mesh adds features to the mesh as the sequence progresses as well as removing mesh nodes that are no longer locating suitable features. The first standard sequence to be used is from the CMU image on-line database[23].

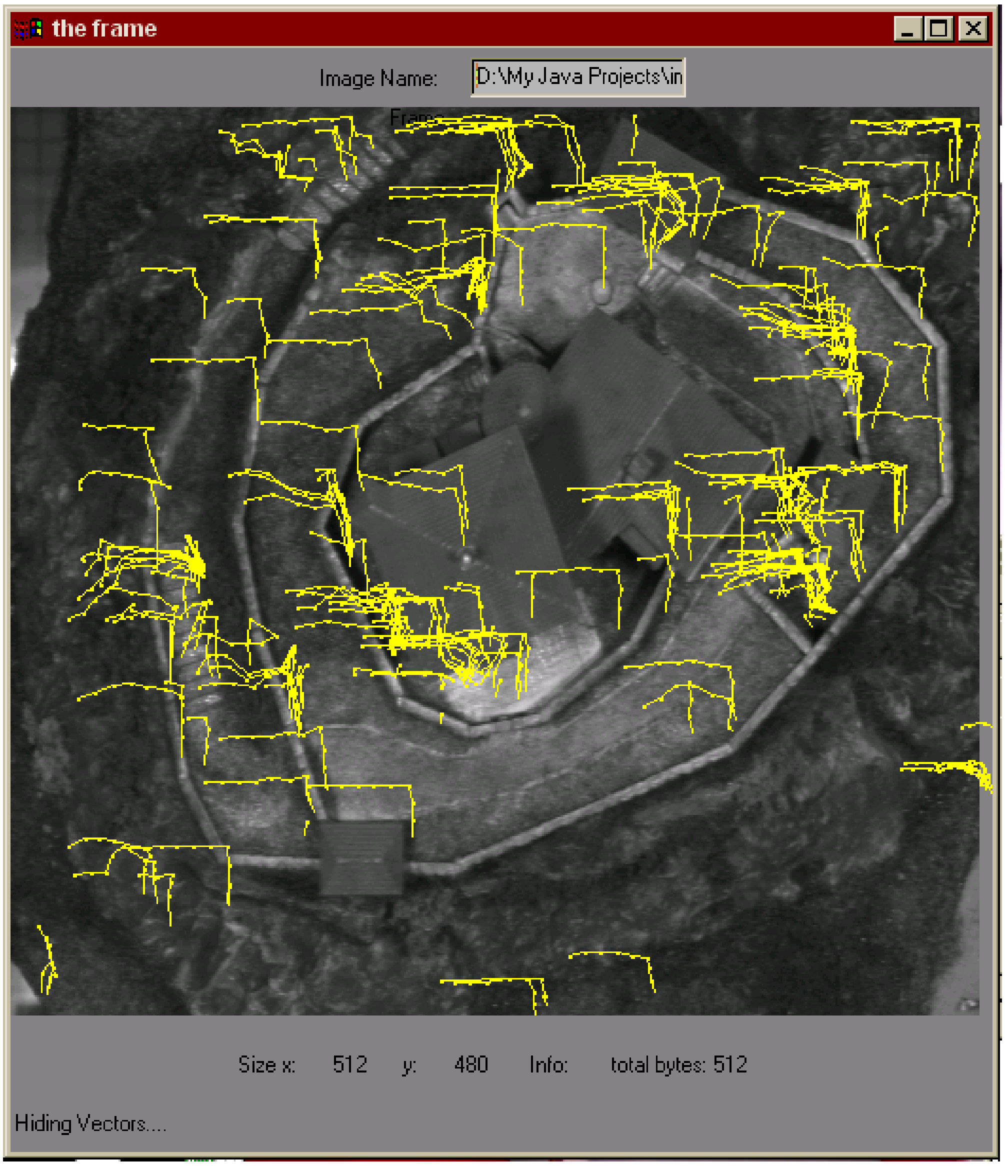


Figure 5-37. 12 frames from the Castle image sequence (SUSAN threshold 35).

Figure 5-37, shows 12 frames from the castle image sequence. The vector paths show the castle image being tracked with the ‘dots’ on the paths representing image frames.

[23] Obtained from an Internet Computer Vision Image Database – Vision and Autonomous Systems Centre’s Image Database (http://www.ius.cs.cmu.edu/idb/)

The vector paths show the view moving right and down relative to the observer. The results from this sequence are well defined, with the only problem cases existing around the outside edges of the image. The results shown are for a threshold value of 35, to allow an outline view of the vectors.

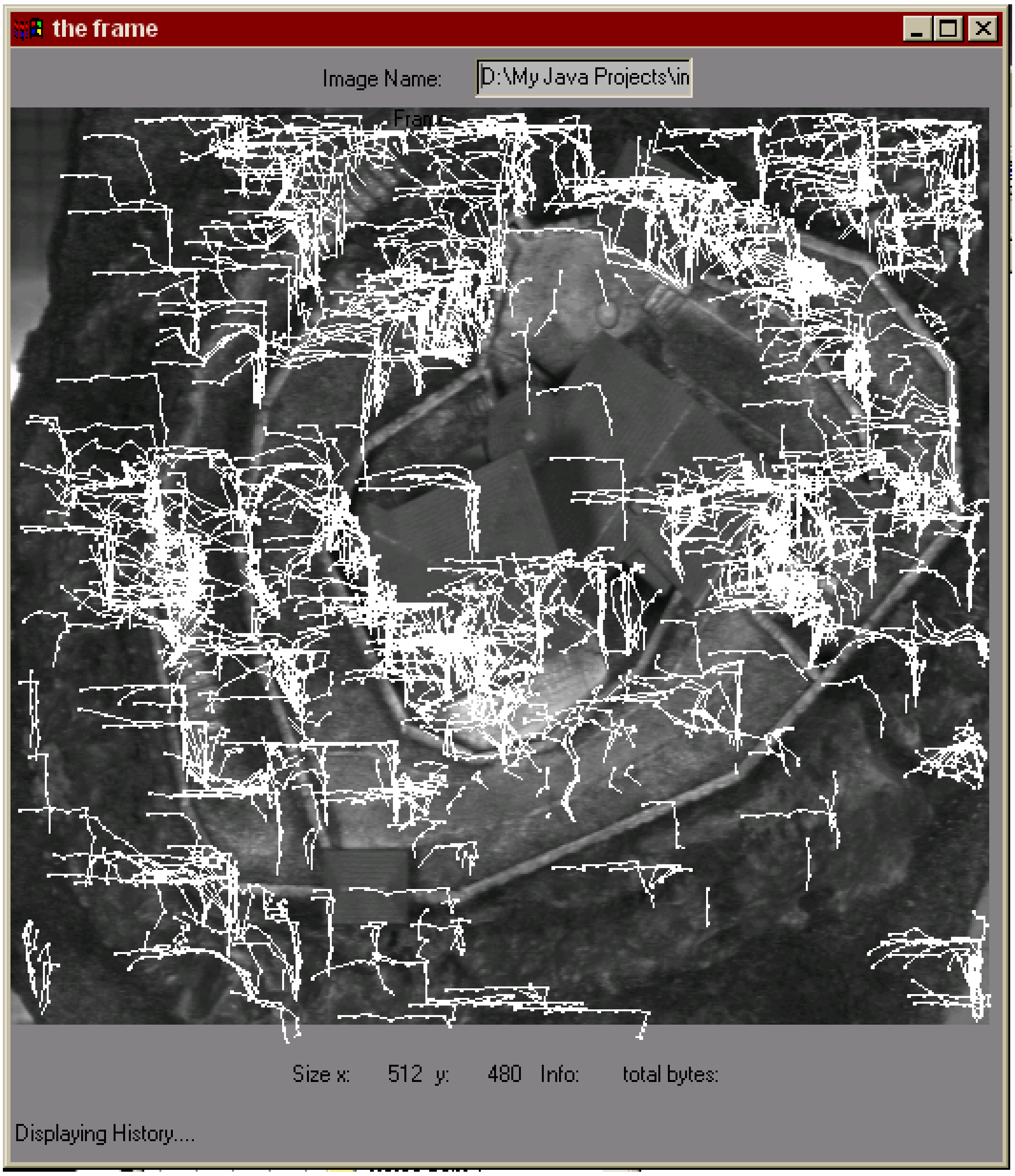


Figure 5-38. 12 frames from the Castle image sequence (SUSAN threshold 24).

The same sequence is also used with a threshold of 24 to show the more normal, dense field that will result. However the paths are more cluttered and less distinguishable to the human eye. The shorter paths are new paths that are detected during the course of the image sequence. These paths are then tracked for the remainder of the sequence.

5.4.5 The Laboratory 2 Sequence (CMU Database)

Another test sequence that was used for testing was also from the CMU on-line image database. Figure 5-39 (top) and (bottom) show two frames of the motion sequence.

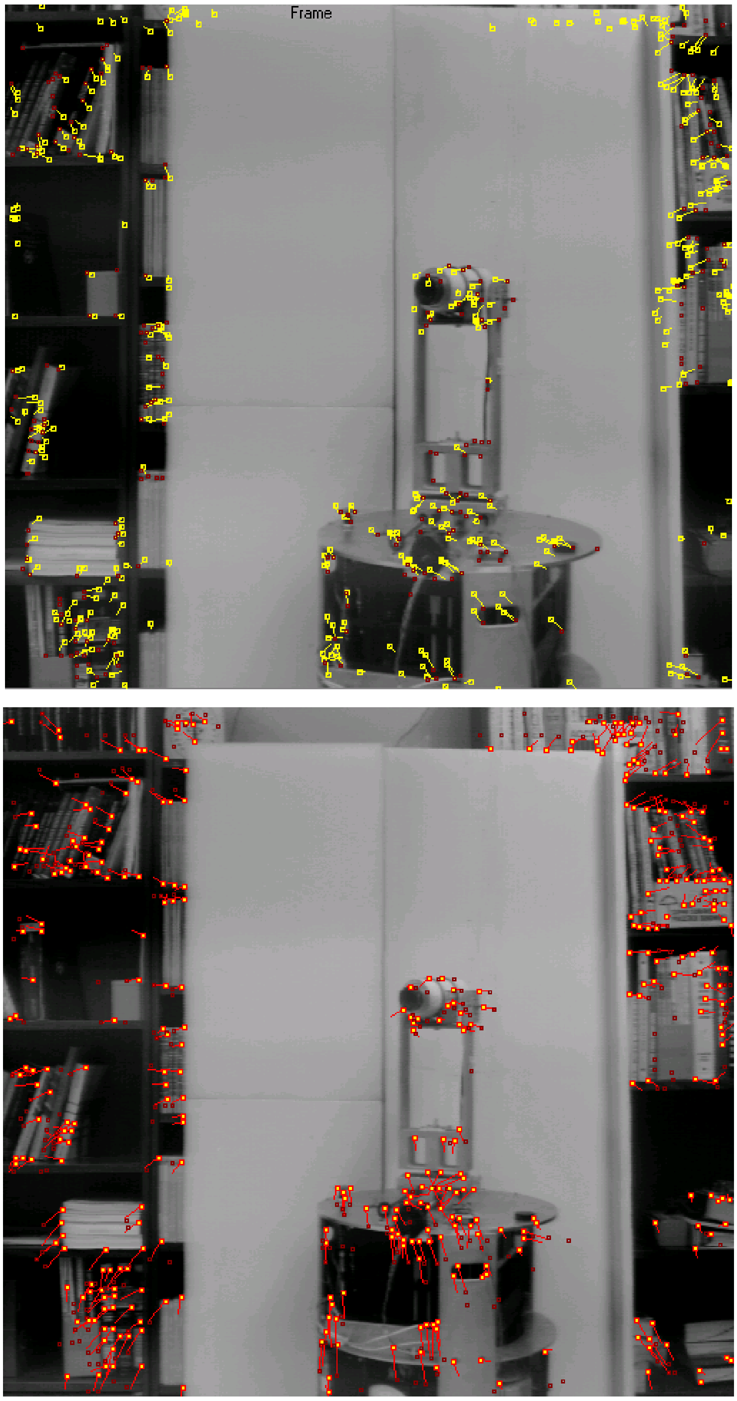


Figure 5-39. Two frames with calculated flow vectors.

As can be seen in both cases the vectors suggest that the camera is moving towards the object in the centre of the image. The largely uniform areas surrounding the object at the centre have almost no mesh nodes and subsequently no motion vectors. Figure 5-40 shows 5 frames of the laboratory 2 image sequence as the observer aproaches the object in the centre of the image. The vector paths are again well defined, with the shorter paths representing new features that have been added to the mesh. Vectors that have left the field of view have been removed from the frame.

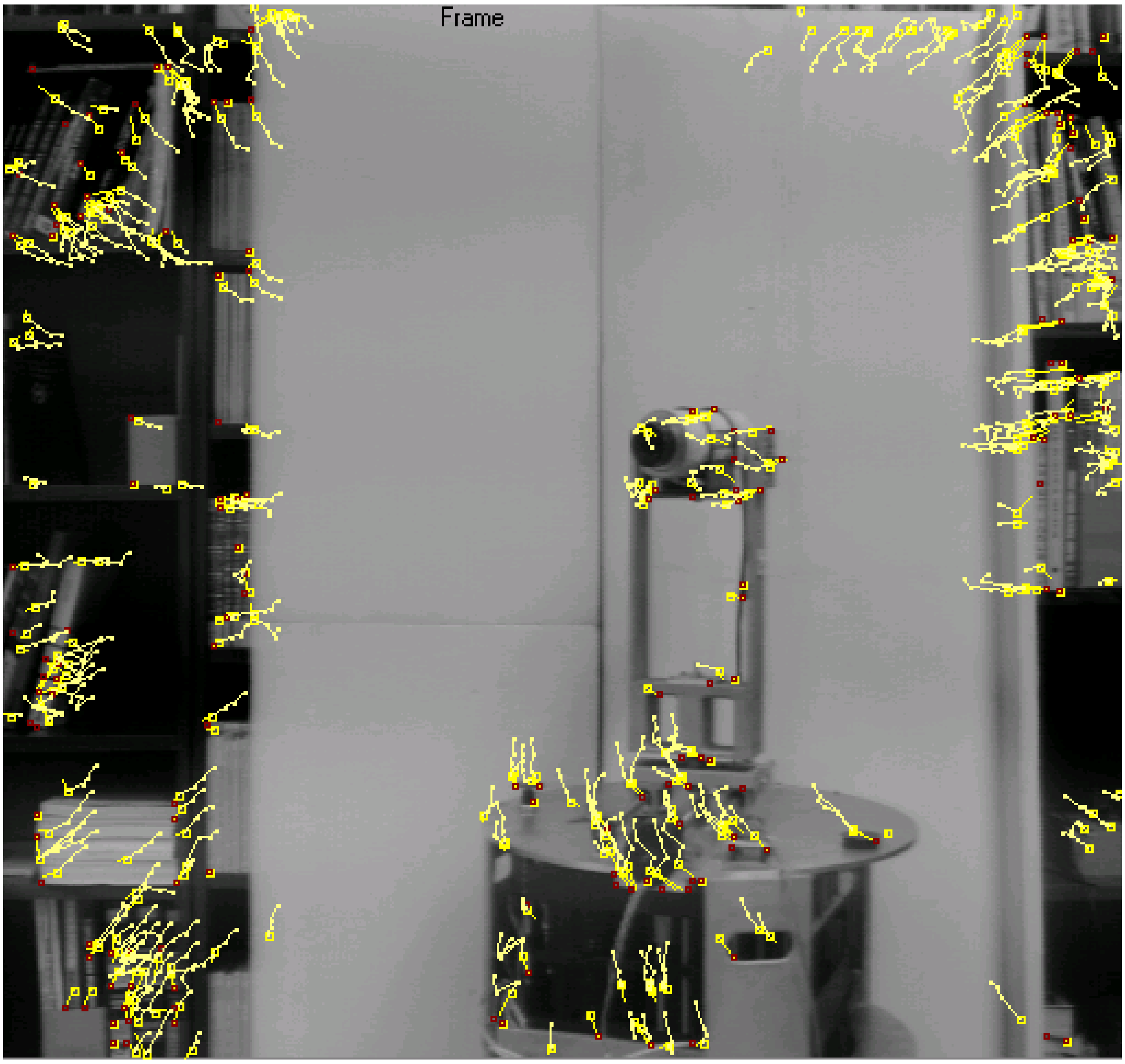
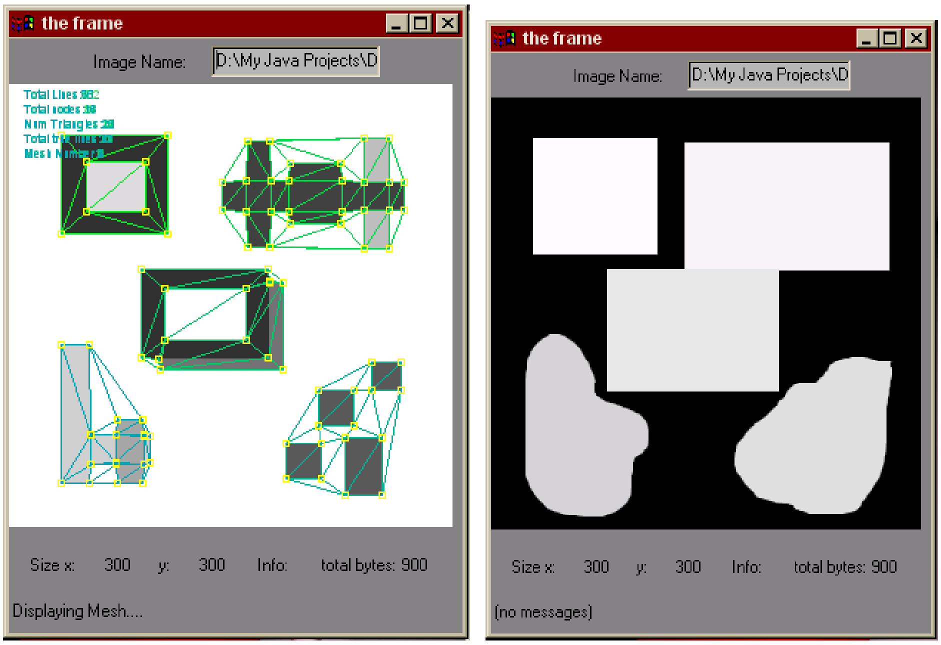


Figure 5-40. 5 frames of the laboratory 2 image sequence.

5.5 The Region and Multiple Mesh Approaches

5.5.1 The Test Template Results

As discussed in Section 4.7 the use of multiple meshes for tracking different objects in image sequences can have several benefits. The multiple meshes are constructed using a region map as in Figure 5-41 (b) where different intensities define different mesh regions.



D:\My Java Projects\D

Size x: 300 y: 300 Info: total bytes: 900

Displaying Mesh....

D:\My Java Projects\D

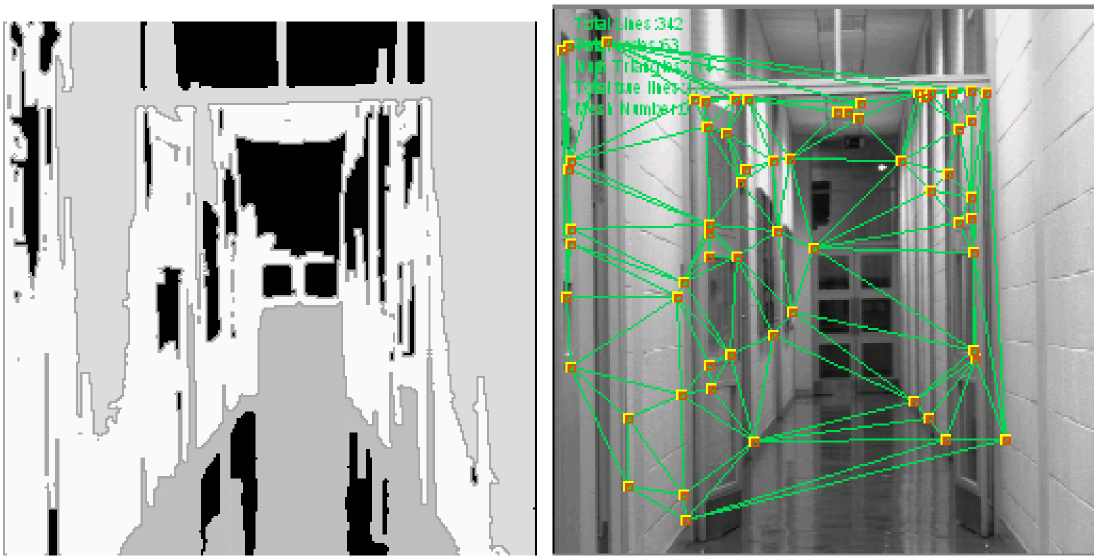
Size x: 300 y: 300 Info: total bytes: 900

(no messages)

(a) (b)

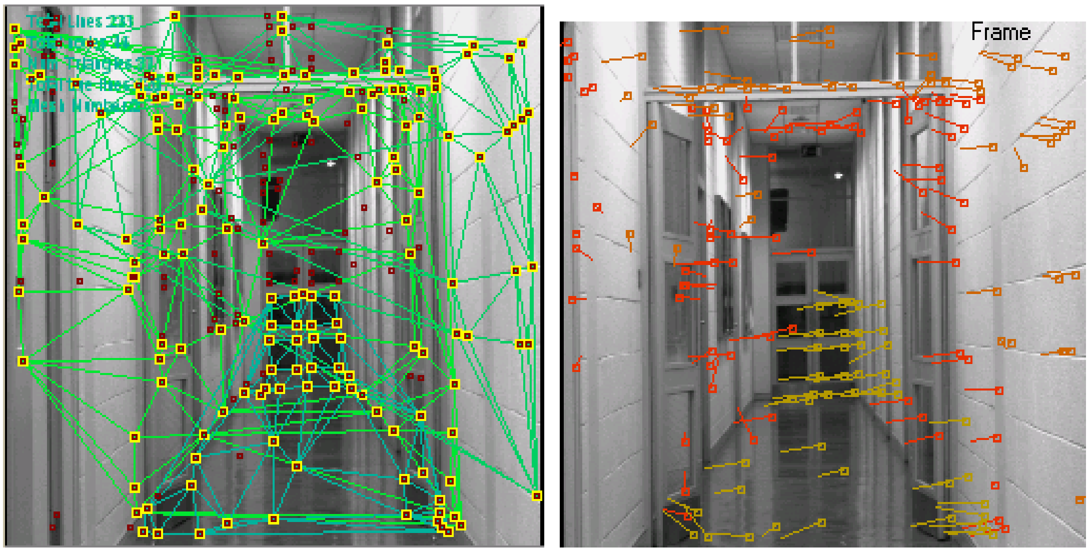
Figure 5-41. The test template multiple meshes in (a) created using the test region map (created by hand) in (b).

The algorithm creates one mesh for each of the different intensities in the region map. It is important to note that this region map can be constructed dynamically and so does not necessarily have to be a stored image. Each one of the meshes is displayed in Figure 5-41(a) and are not connected in any way. These meshes can then track the objects individually. Figure 5-42(a) shows the region map extracted using the k-means clustering algorithm, merged with a watershed algorithm. The 3 biggest 'blobs' were chosen as the regions. This clustering technique is not ideal for the task required, however it is performed as a demonstration of the use of clustering. Figure 5-42(b) shows the mesh created from using the first region in the region map. The three meshes are shown in Figure 5-43(a) and the equivalent mesh vectors are shown in different colours in (b). This approach of using multiple meshes is especially useful in scenes with multiple objects.



(a) (b)

Figure 5-42. (a) The region map for the corridor determined using the k-means algorithm and (b) the mesh created using region 1, the brightest region.



(a) (b)

Figure 5-43. The three meshes are shown in (a) and the vectors calculated for each of the meshes, shown in different colours in (b).

5.5.2 The Junction Image Sequence (CMU Database)

Figure 5-44, shows a junction image sequence frame with the different objects initialised in the scene. The object areas are determined using straightforward image frame subtraction (and thresholding) from a reference frame of the scene. Once the multiple region maps are determined and the meshes are initialized, no more image frame subtraction is required. The independent meshes, in this case one per vehicle, then search for image features.

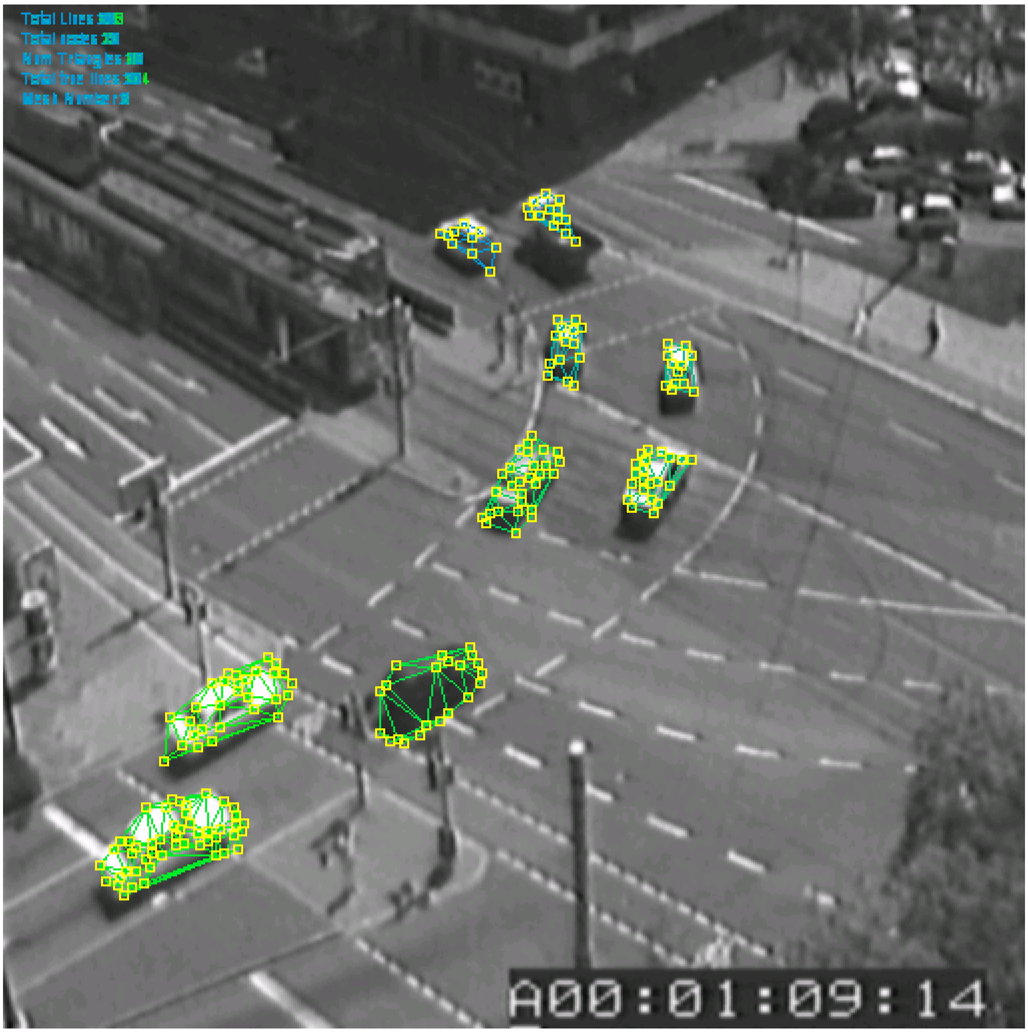
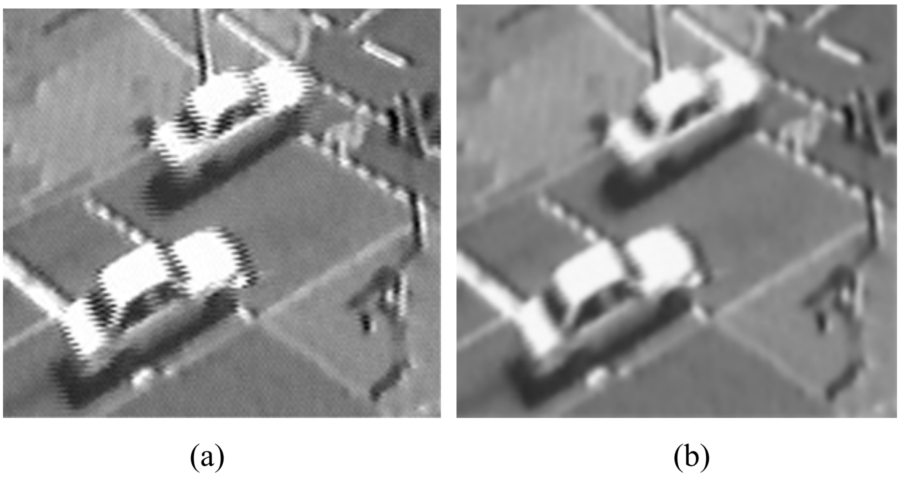


Figure 5-44. The junction image sequence with initialised meshes.

This sequence is more difficult to track than initially expected due to problems with the speed of the image capture routine. Figure 5-45 (a) shows the main problem associated with the captured images. The cars to be tracked are distorted along the vertical scan lines of the image capture, which can badly effect the detection of stable corner feature points. This difficulty has been reduced, by Gaussian smoothing filtering as in Figure 5-45 (b). This is not an ideal solution as it smooths over image features, a better image capture routine is required.



(a) (b)

Figure 5-45. The problem with the captured images in the junction sequence in (a) and in (b) a Gaussian smoothed version of the same frame.

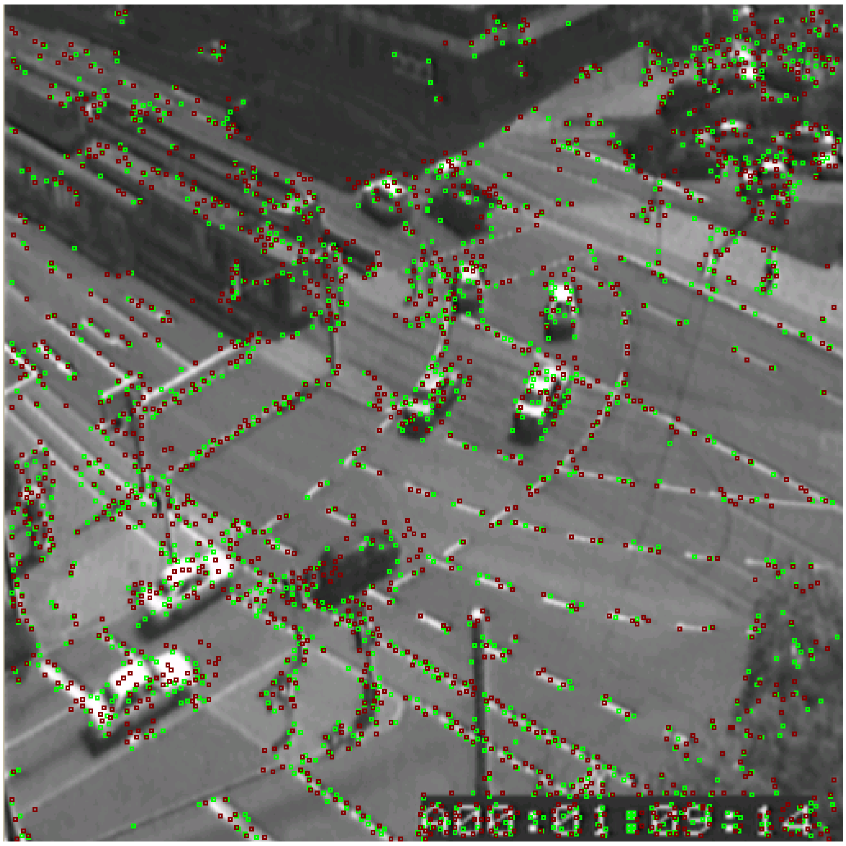


Figure 5-46. The clutter in the junction scene, with the new feature points displayed in red.

The tracking algorithm is also hampered by the vast number of determined features in the subsequent frame. This is shown in Figure 5-46, where there are a large number of features in the search space of each mesh node. This clutter is distracting to the mesh nodes and can cause some of them to go astray. However, the results from the tracking algorithm is quite successful, and are displayed in Figure 5-47.

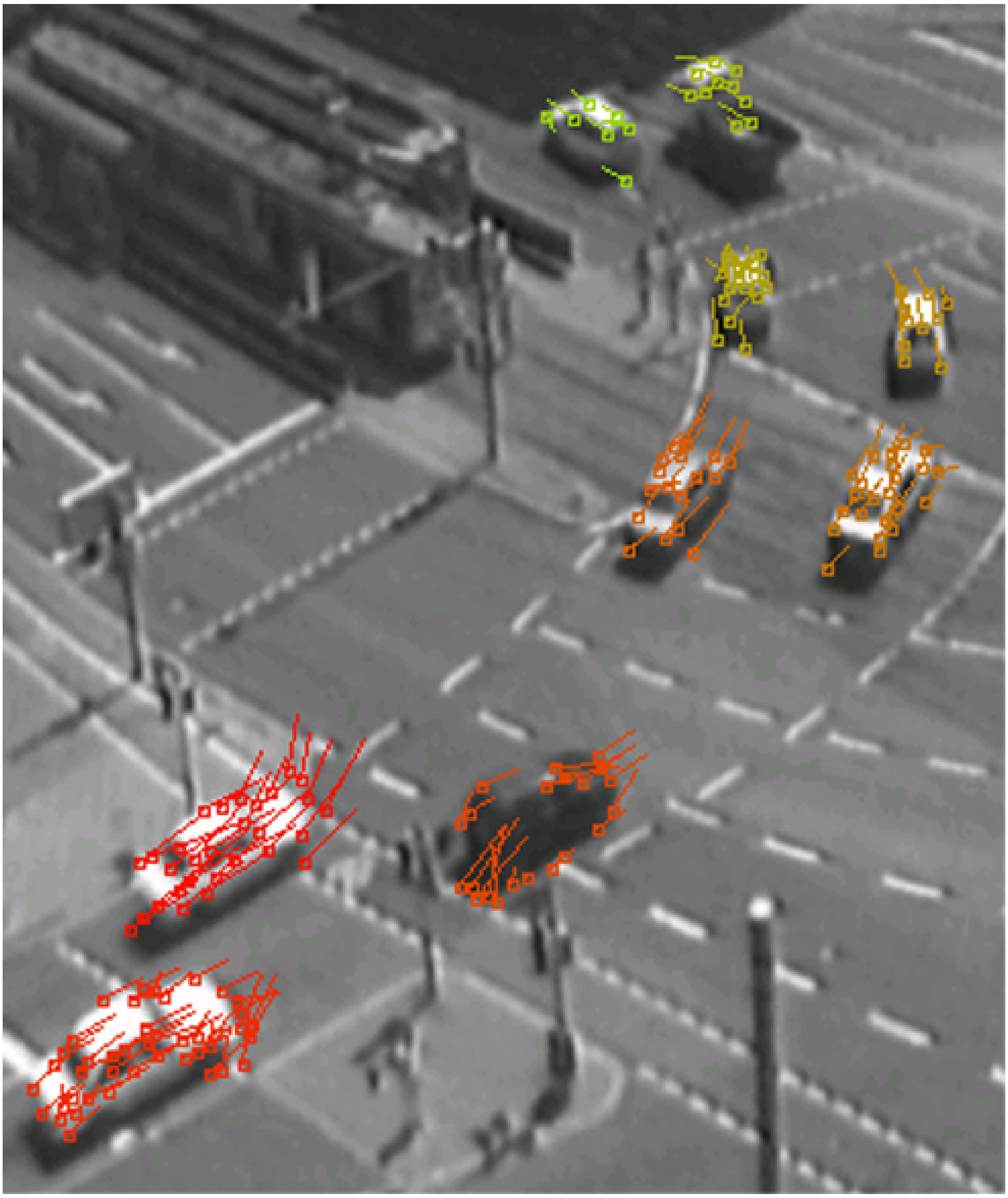


Figure 5-47. The junction results (zoomed by 1.5), with the vector heads located on the position of the objects in the current frame, pointing towards the exact position of that feature in the next frame. Each set of vectors (clustered to mesh) are represented using different colours.

Each colour represents the vectors on the same object. This sequence will be discussed later in Section 6.3.4, in further directions for research. As discussed in Section 4.4.2, the choice of feature matching algorithm was chosen to be based on feature parameters rather than intensity patches, due to problems with the object boundaries. In this case the object boundary features are on the convex hull of the mesh. Inside the mesh the feature points can be based on an intensity patch approach.

5.6 Other Standard Test Sequences

This section will discuss more results calculated from standard test sequences that are available on-line for testing motion based algorithms.

5.6.1 The Cup Sequence

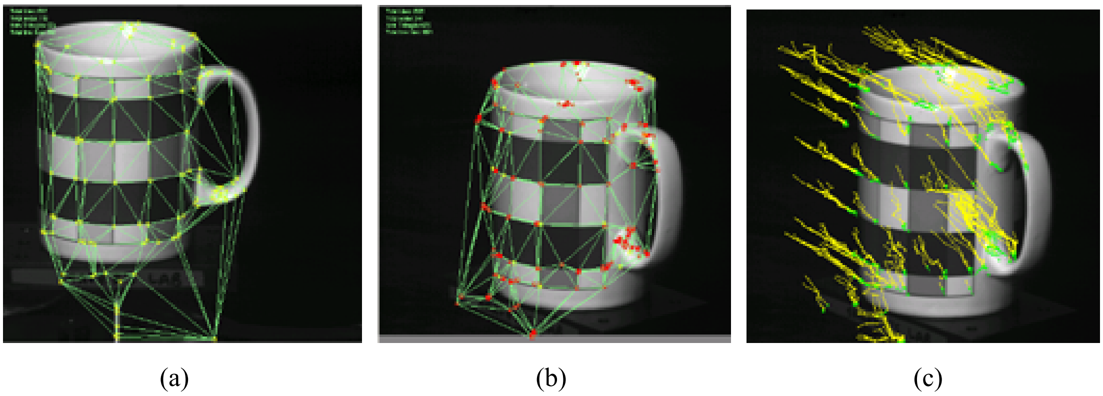


Figure 5-48. The mug sequence (a) the mesh as initialized on the first frame, (b) the mesh resulting on the 12th frame of the sequence and (c) the vector paths shown over 12 frames. These images are shown in detail in Appendix C (Figure C-1, Figure C-2, Figure C-3).

Figure 5-48 displays the mug sequence with (a) showing the initial mesh initialised from the corner feature points and (b) displays the mesh after the 12 frames had taken place. The mug rotated clockwise and translated right and down in the image. The resulting vector paths in (c) describe the motion between the intermediate image frames. Some interesting comments about this sequence include, that the points detected in (a) at the bottom of the image (on the metal stand) bunch together as that part of the object leaves the image space. The points detected on the text of the stand are also stable, 'pulling out' from the mug in (b). The pattern on the mug greatly aids the determination of the motion on the mug surface. If the surface was completely free from texture it would be difficult to track, however, as shown in this sequence the handle and rim of the cup as well as the stand also provide geometrical features for motion tracking.

5.6.2 The Parking Sequence

The parking sequence was obtained from a vehicle undergoing parallel parking manoeuvres. There are 12 frames in this sequence, which is a standard test for motion tracking algorithms. The active mesh algorithm required only the first and last frames of the sequence to estimate the motion occurring in the sequence. The initialised mesh is shown as in Figure 5-49 (a) where the majority of feature points exist in the area of interest at the lower half of the image. The resulting vectors are shown in (b) displaying very little motion of the building or the left-most bush. The parking meters, the rest of the bushes and the other vehicle display motion as the vehicle rotates about its axis.

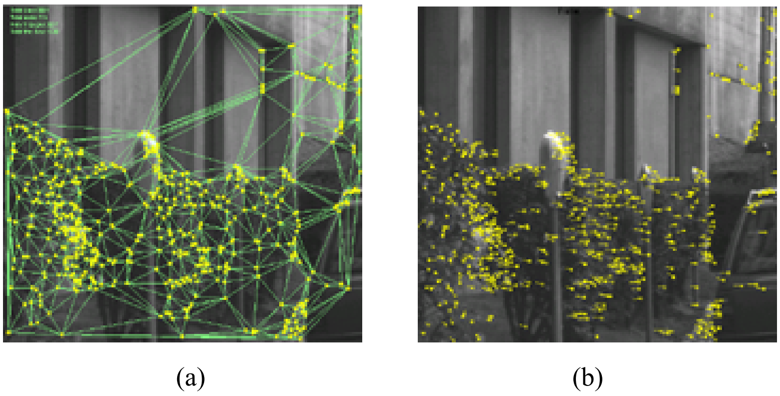


Figure 5-49. The parking sequence (a) the mesh as initialized on the first frame, (b) the resulting vectors calculated from the two image frames. These images are shown in detail in Appendix C (Figure C-4, Figure C-5).

5.6.3 The Shrubbery Sequence

The shrubbery sequence displays another moving vehicle example. This sequence in Figure 5-50 consists of 24 frames of a vehicle observer moving right and pivoting down (reversing up a hill). Image (a) displays the yellow vectors representing the vector paths tracked over 24 image frames.

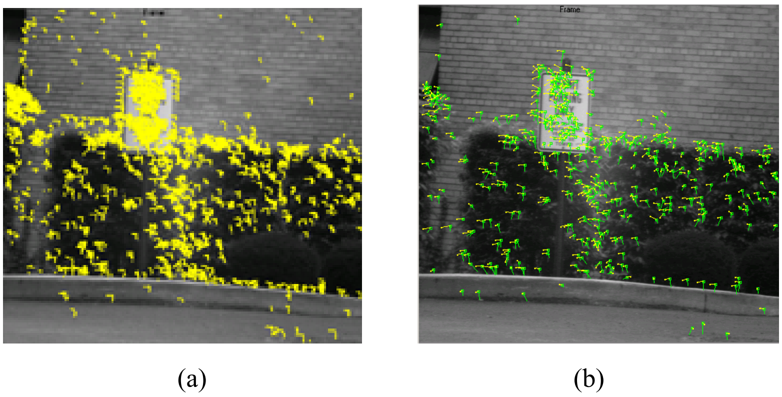


Figure 5-50. The shrubbery sequence (a) the vector paths calculated using 24 image frames and (b) the vector paths calculated using only the first, middle and last frames. These images are shown in detail in Appendix C (Figure C-6, Figure C-7).

The motion is clearly described by the vectors at most of the areas in the image. In (b) only 3 frames are used, the first, last, and middle frame. This provides an accurate estimation of the motion occurring in the sequence using only 2 frame updates rather than the 23 frame updates required in (a). The only significant difference is that in (a) new features have been detected in the intermediate frames (e.g. on the brickwork) and added to the mesh, whereas these features were not located in (b).

5.6.4 The Town Sequence

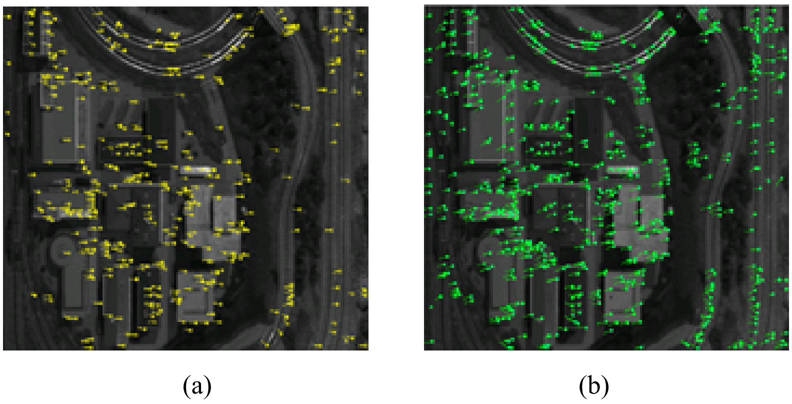


Figure 5-52. Two sections of the town sequence (a) the vector paths calculated using only 2 image frames and (b) the vector paths calculated using only 2 image frames. These images are shown in detail in Appendix C (Figure C-8, Figure C-9).

Figure 5-52, shows two sections of the town sequence. In (a) 6 frames are available but only the first and last frames are used. The resulting vectors again describe the motion in the scene skipping 4 updates. In (b) 7 frames are available and again only the first and last frames are used, again increasing the scene motion step that the algorithm must describe. The active mesh has little difficulty in calculating the motion vectors for such a feature rich scene.

5.6.5 The Building Sequence

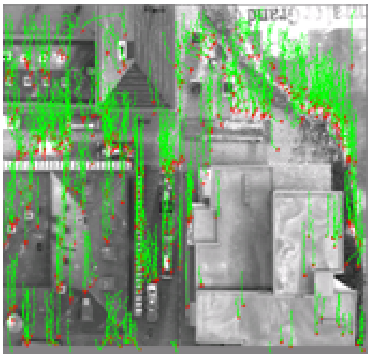


Figure 5-51. The building sequence showing the vector paths calculated over 20 image frames. This image is shown in detail in Appendix C (Figure C-10).

Figure 5-51 shows 20 frames of the building sequence tracked using the adaptive active mesh algorithm. The red nodes represent the endpoints in the sequence that was tracked and the green paths represent the motion that has taken place.

5.6.6 The SRI Laboratory Sequence

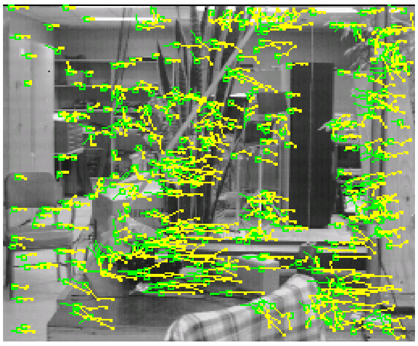


Figure 5-53. The SRI laboratory sequence resulting vectors. This image is shown in detail in Appendix C (Figure C-12).

Figure 5-53, displays the SRI laboratory sequence and resulting vectors. Note that the vector paths are longer in the foreground than in the background parts of the image. The back of the room is almost stationary, providing some degree of information about the depth of the scene (this will be examined in Section 5.9.2). The sequence is very cluttered and the active mesh still produces motion information. The green vectors represent the last vector in the yellow sequence.

5.6.7 The College Sequence

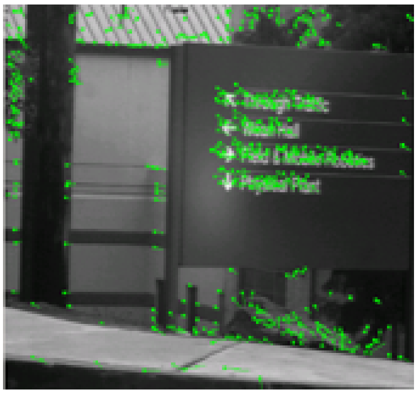


Figure 5-54. The college sequence resulting vectors. This image is shown in detail in Appendix C (Figure C-13).

Figure 5-61, displays the college sequence. There are actually 20 frames in this sequence but the active mesh algorithm generates the motion vectors using only the first and last images, ignoring the intermediary frames. The estimation is good in detailed areas but has trouble with the non-textured geometric featureless image regions.

5.6.8 The 'Trouble' Teddy Bear Sequence.

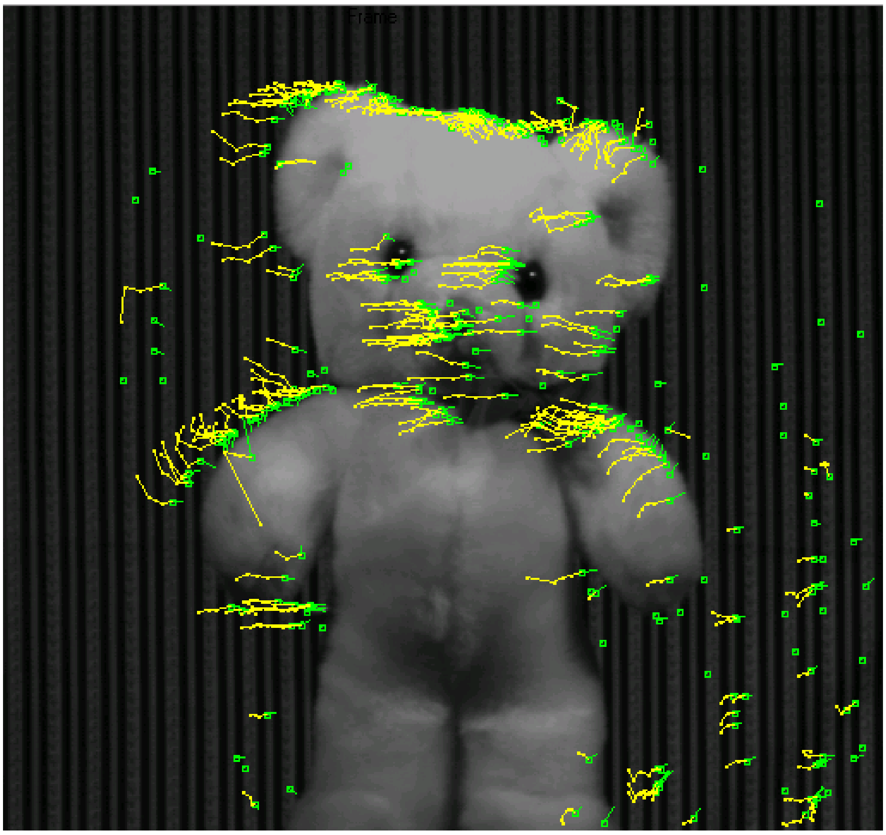


Figure 5-55. The teddy bear sequence resulting vectors.

Figure 5-55 displays the teddy bear sequence last frame and resulting vectors. This sequence provided many difficulties for the active-mesh algorithm and indeed provides difficulty for many other motion tracking algorithms. The camera translates across the front of the bear causing the bear to 'move' in the image at a faster apparent velocity than the wallpaper.

This sequence provides problems for many reasons:

- The wallpaper has very few corner features and the motion step is larger than the strip width, causing problems in estimating the motion due to the uniformity of the pattern. This is a difficult problem for even a human observer giving a discrete sampling rate.
- The uniform texture and geometry of the object makes corner detection within the object pointless, with the exception of the facial features.
- The intersection of the vertical lines of the wallpaper and the bear provide very strong 'unsuitable' corner points. These points are unsuitable as the wallpaper is moving independent of the bear.

It is difficult to determine a suitable motion determination approach for this scene. Optical flow algorithms would also have difficulty with the wallpaper step-size and the uniform texture of the bear. Edge detection based algorithms would have difficulty with the wallpaper stripes. The region based active mesh would be a more suitable approach for tracking the bear object provided that a good depth map could be determined. The active mesh does provide an accurate estimation of the motion of the facial features of the bear.

5.7 The Modified SUSAN (DSUSAN) Results

To test the accuracy of this approach and to determine the advantages of this approach, numerous tests were performed. As shown previously in Section 4.10, Figure 4-35 displays the SUSAN test template after the normal and modified versions have been applied. The regular SUSAN algorithm executes in 280ms and detects 83 corners. The modified version executes in 81ms and detects the same 83 corners. This represents a factor of improvement of 3.4 in the execution speed of the algorithm. The modified version works particularly well in this case as there are plenty of uniform areas that are immediately rejected by the pre-filter. To this end the pre-filter was also examined on some real-world images such as the airport and laboratory images.

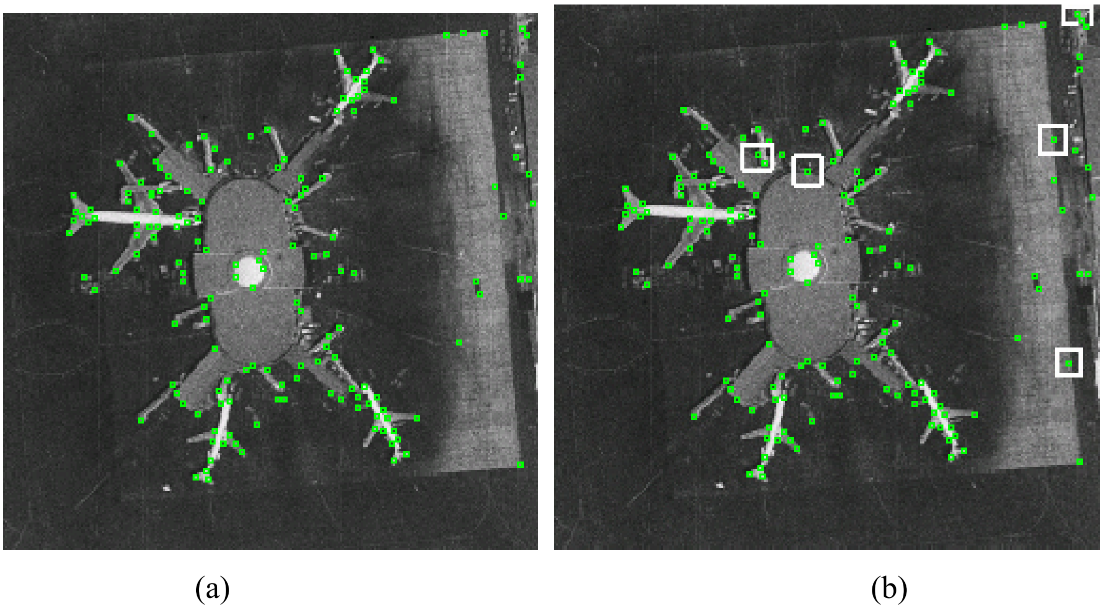
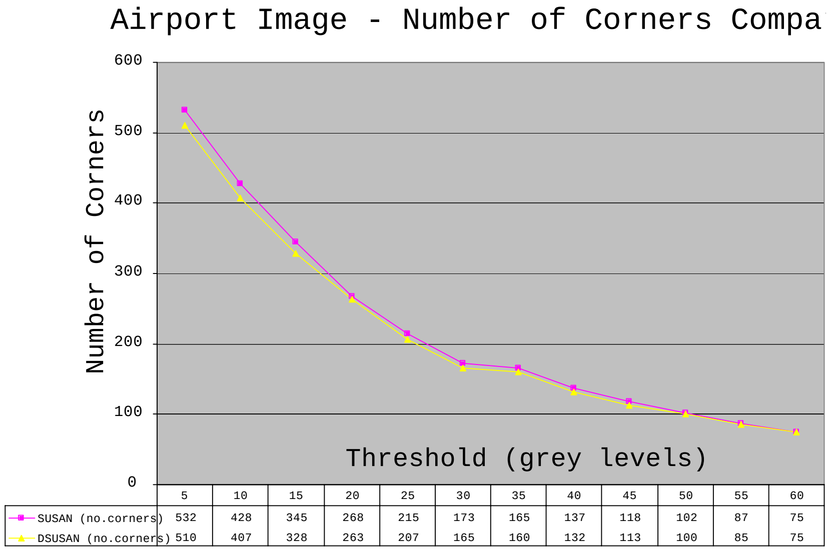


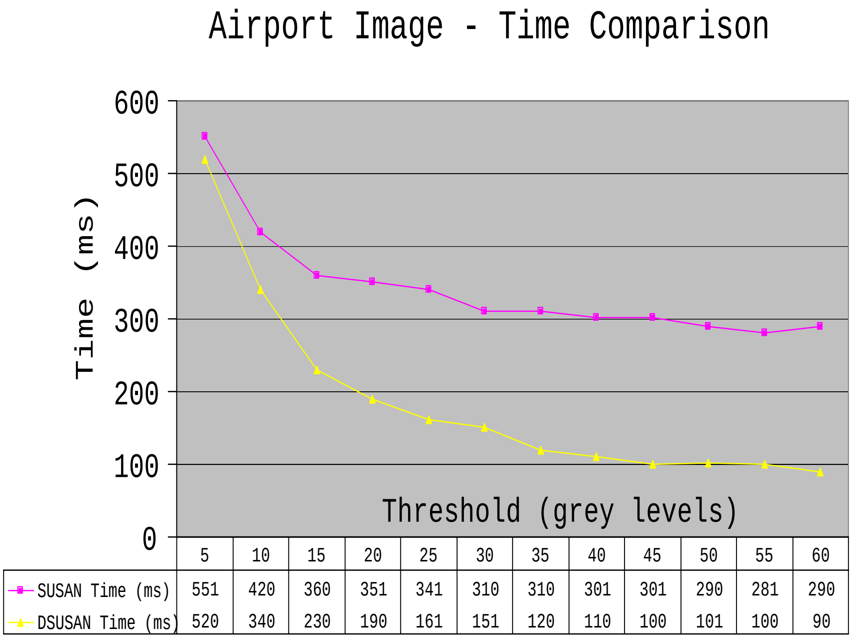
Figure 5-56. The airport image test (a) the airport image after the SUSAN corner detector has been performed at a threshold value of 35 taking 310ms and (b) the airport image after the modified SUSAN corner detector has been performed at a threshold value of 35 taking 120ms.



	5	10	15	20	25	30	35	40	45	50	55	60
SUSAN (no.corners)	532	428	345	268	215	173	165	137	118	102	87	75
DSUSAN (no.corners)	510	407	328	263	207	165	160	132	113	100	85	75

Figure 5-57. Showing the number of corners detected by the SUSAN and DSUSAN (SUSAN with pre-filter) filters (α_s=1.0).

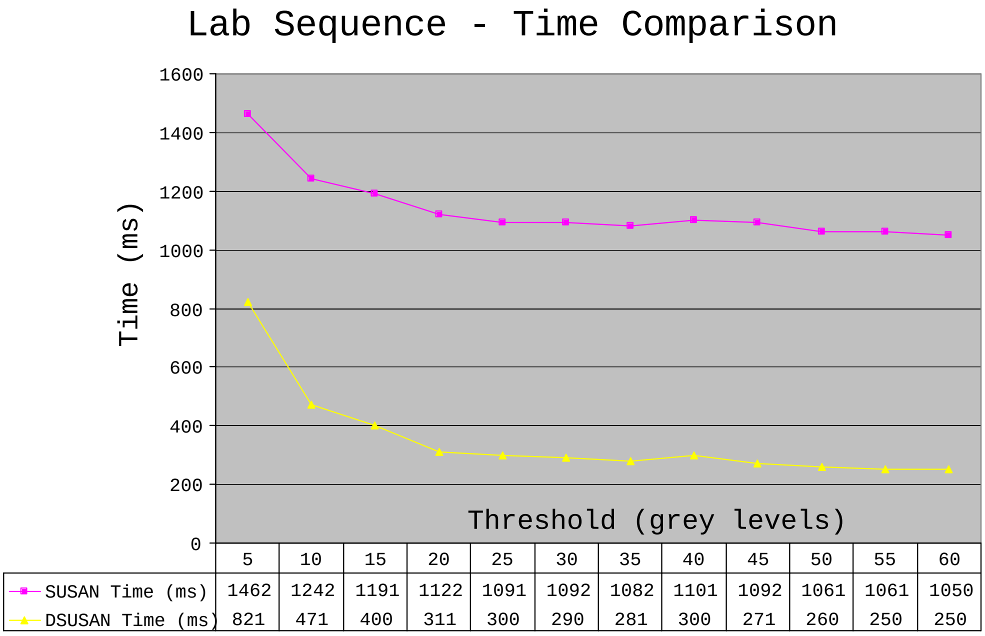
Figure 5-56 shows an example of the SUSAN corner detector in (a) and the modified SUSAN corner detector in (b). The threshold value is the same for (a) and (b) but the modified detector is 2.6 times faster. The pre-filter rejects 5 corners that are detected using the original detector. The corners that are rejected are very weak corners



	5	10	15	20	25	30	35	40	45	50	55	60
SUSAN Time (ms)	551	420	360	351	341	310	310	301	301	290	281	290
DSUSAN Time (ms)	520	340	230	190	161	151	120	110	100	101	100	90

Figure 5-58. The Airport Image, time comparison graph, showing the time in milliseconds[24] that the algorithms take to execute (α_s=1.0).

Figure 5-58, shows the execution speed of both filters, the SUSAN and DSUSAN (SUSAN with pre-filter). It can be seen that for low grey-level thresholds the factor of improvement of execution speed is not substantial, but for higher threshold values the factor of improvement is substantial. Typically the SUSAN detector will operate in the 20-25 brightness threshold ranges to obtain a generous number of stable features. At this point the DSUSAN detector operates 2.1 times faster. Figure 5-57, shows the graph of the number of corners detected with SUSAN and DSUSAN. As the grey-level threshold increases the number of corners detected by both detectors becomes very similar (higher threshold implies stronger grey-level corner). This difference is at a maximum at the lower value of the threshold (5-10), where the features are weakest.



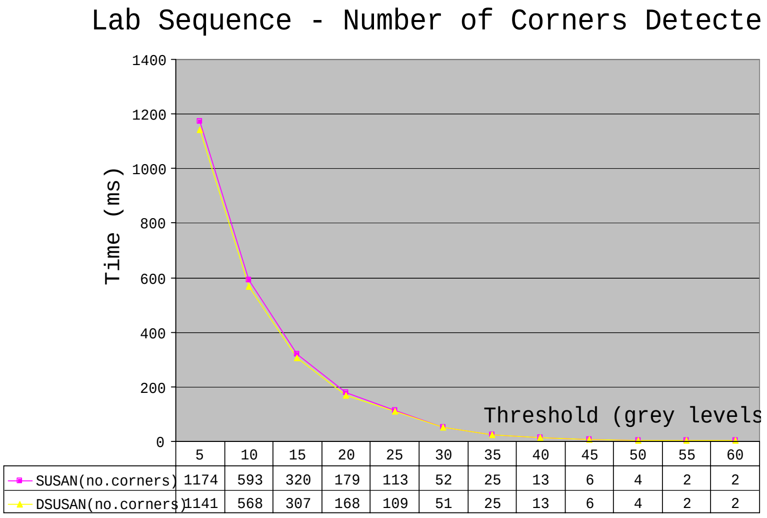
	5	10	15	20	25	30	35	40	45	50	55	60
SUSAN Time (ms)	1462	1242	1191	1122	1091	1092	1082	1101	1092	1061	1061	1050
DSUSAN Time (ms)	821	471	400	311	300	290	281	300	271	260	250	250

Figure 5-59. Time comparison on the laboratory 2 sequence – the SUSAN (regular) and DSUSAN (modified SUSAN) are shown. There is a consistent improvement in the execution speed of the corner detector (α_s=1.0).

Another image was examined from the Laboratory 2 Sequence and compared with the SUSAN and the modified DSUSAN corner detectors. Figure 5-59, shows a comparison in the execution of the SUSAN and DSUSAN corner detectors. The DSUSAN operates up to 3.6 times faster at the usual brightness threshold range of 20-25. The results are

[24] The time in milliseconds was calculated on the Microsoft Java ++ Just In Time (JIT) compiler running under Windows NT on a portable Pentium II (200 MHz) with 128MB RAM. The execution time in milliseconds is dependent on the hardware architecture, but the factor of improvement is unlikely to change.

better than those for the airport image. This is likely due to more non-textured areas in the image and also due to the fact that the lab image is 512 x 512 pixels in area whereas the airport image is 256 x 256 pixels in area. This helps the operation improvement, as the pre-filter will reject uniform intensity areas as ‘no hope’ corner candidates.



	5	10	15	20	25	30	35	40	45	50	55	60
SUSAN(no.corners)	1174	593	320	179	113	52	25	13	6	4	2	2
DSUSAN(no.corners)	1141	568	307	168	109	51	25	13	6	4	2	2

Figure 5-60. The comparison between the number of corners detected by the SUSAN and DSUSAN corner detectors (α_s=1.0) in the Laboratory 2 image sequence.

Figure 5-60, shows the number of corners detected by the SUSAN and DSUSAN corner detectors for different brightness threshold values. The difference in the numbers decreases, as stronger corners are required at higher threshold levels. In the normal operating range of the SUSAN corner detector the modified version detects 4 less corners than the original detector.

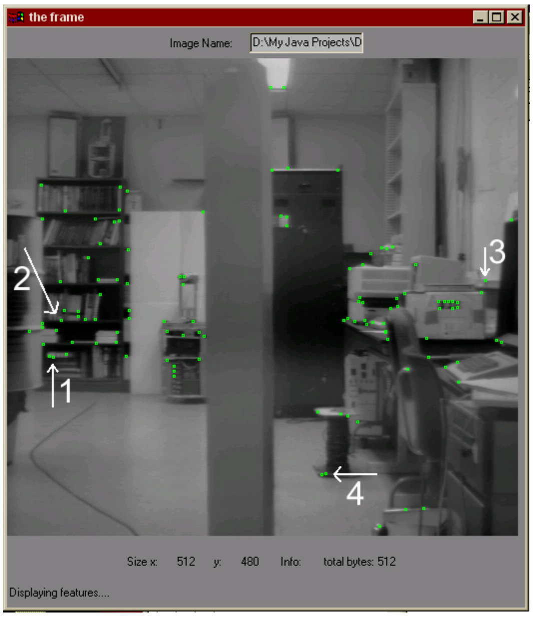

the frame

Image Name: D:\My Java Projects\D

Size x: 512 y: 480 Info: total bytes: 512

Displaying features....

Figure 5-61. Displays a frame from the laboratory 2 image sequence at a threshold of 25. There are 4 more corners detected by the original SUSAN corner detector and these are shown here.

This is shown as in Figure 5-61, where the arrows point at the corners that are not detected in the equivalent DSUSAN image output. A further examination of these missing corners was performed to check the importance of their values. Figure 5-62, shows a 500% zoom image of the corners and corner locations that have disappeared from the pre-filtered versions. Pair 1 and Pair 4 are not immediately recognisable as corners, whereas Pairs 2 and 3 show possible corner points. These points are rejected by the corner detector as the gradient level is very uniform along the *a* to *a'* direction of the mask as described in Section 4.1. These corners are however quite weak and when the brightness threshold is increased to 30 in the original SUSAN algorithm these corners disappear.

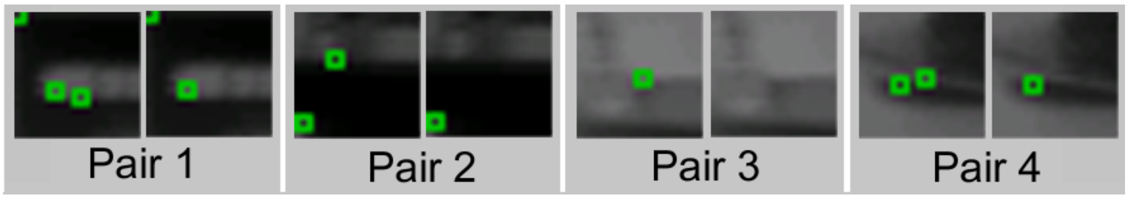
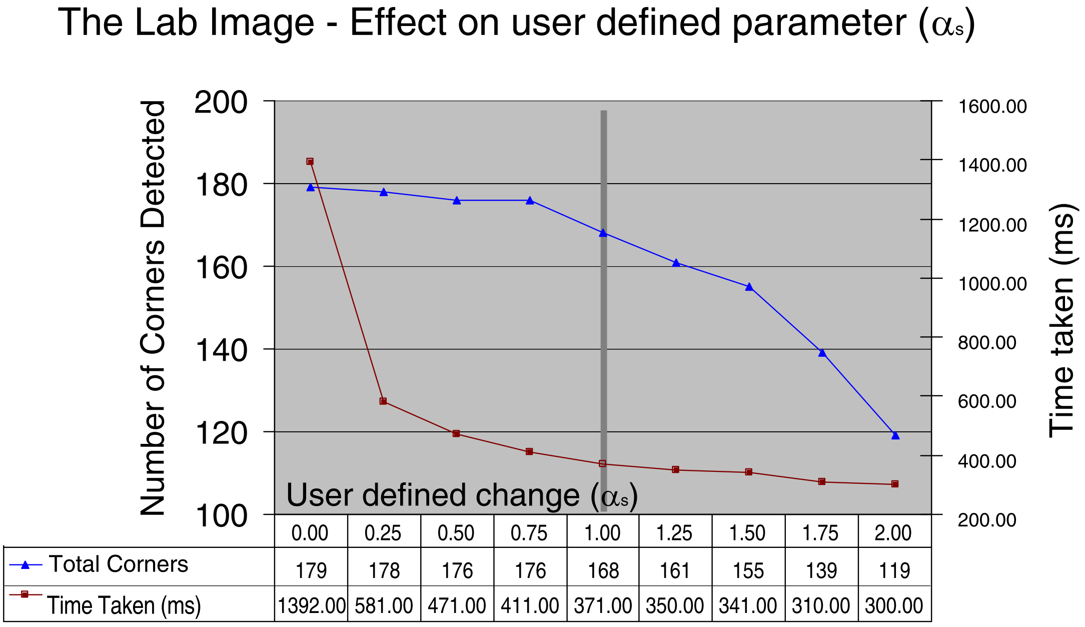


Figure 5-62. A zoomed version of the four missing corners.

5.7.1 The Effect of modifying the user-defined parameter α_S.

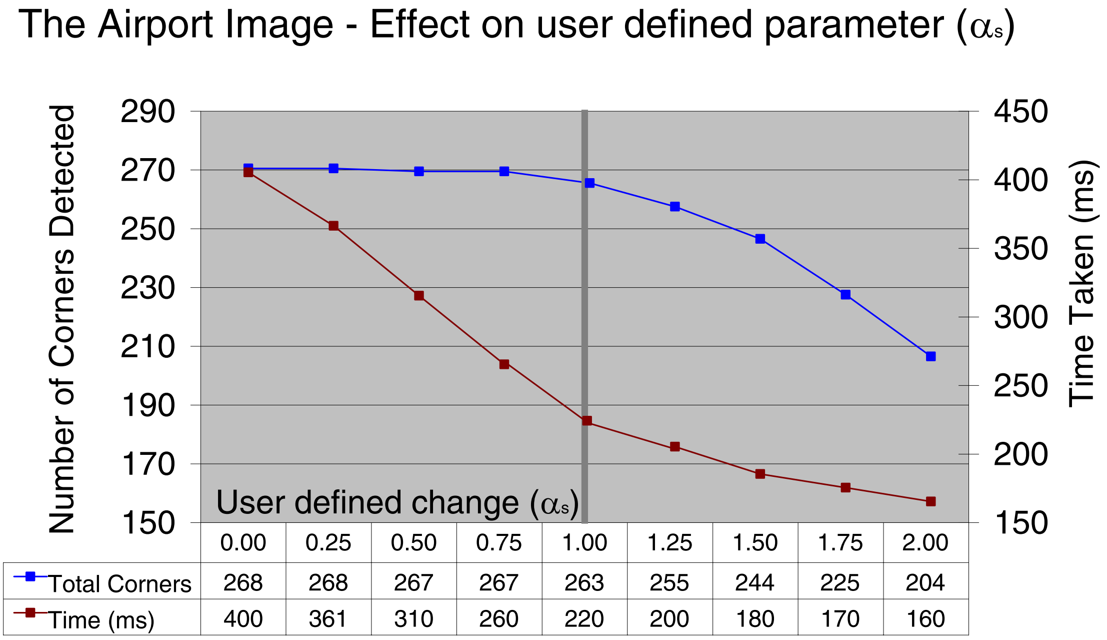


	0.00	0.25	0.50	0.75	1.00	1.25	1.50	1.75	2.00
Total Corners	179	178	176	176	168	161	155	139	119
Time Taken (ms)	1392.00	581.00	471.00	411.00	371.00	350.00	341.00	310.00	300.00

Figure 5-63. The Laboratory 2 Image and the effect of the user defined parameter α_s.

α_S is the user-defined constant that allows the user to choose the effect of the brightness threshold on the SUSAN pre-filter. An α_S*=0.0* means that the threshold value of the

SUSAN pre-filter is set to allow all candidate corners to proceed to the next stage of the algorithm. This is the least efficient point of the detector and takes the longest time. This is equivalent to having no pre-filter at all.



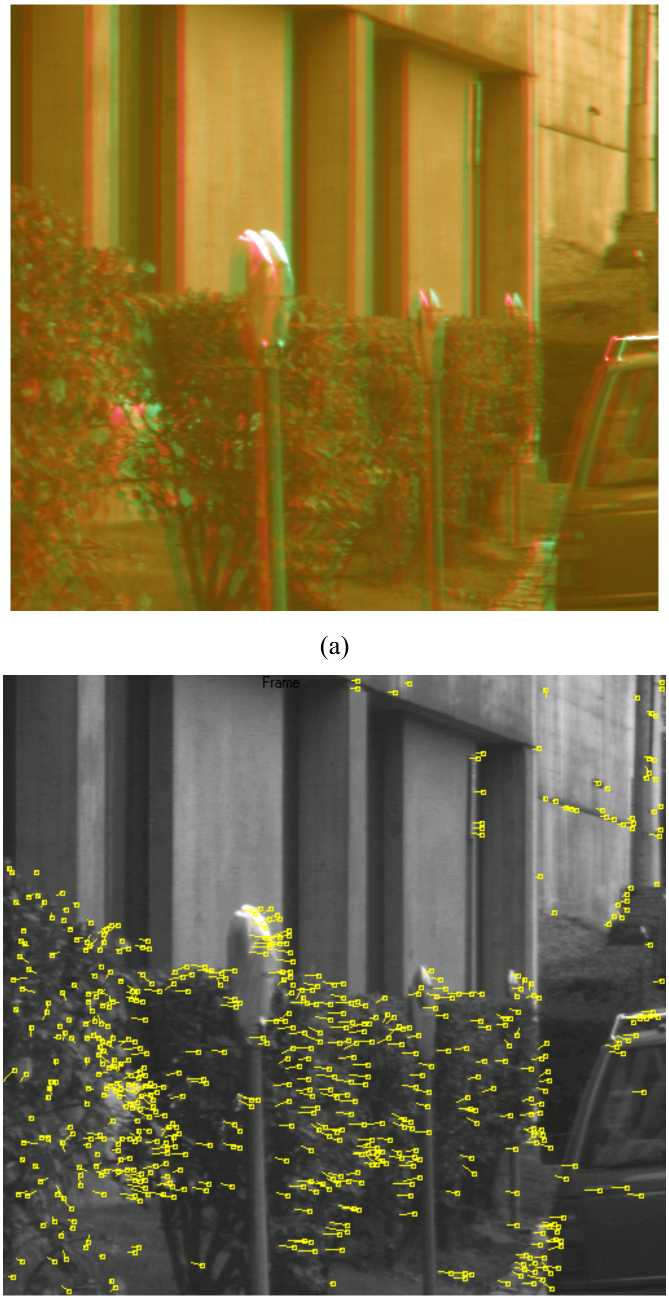
	0.00	0.25	0.50	0.75	1.00	1.25	1.50	1.75	2.00
Total Corners	268	268	267	267	263	255	244	225	204
Time (ms)	400	361	310	260	220	200	180	170	160

Figure 5-64. The Airport Image and the effect of the user defined parameter $\alpha_{s.}$

Figure 5-64 and Figure 5-63 show the effect of modifying the user defined α_s parameter from 0 to 2.0. In both cases it is apparent that the higher the value of α_s the faster the detector operates, but the lower the number of corners detected. It should be clear from these graphs why a value of α_s*=1.0* has been chosen as this value provides a good trade-off between the time taken and the number of corners detected.

5.8 Hand Comparison of Results

It is difficult to determine the accuracy of this approach, as it is an active approach, converging to the correct solution, but providing a reasonable estimate of the motion in the scene. As with most other active approaches, if the frame changes before the mesh has fully converged, it should become concerned with the new frame.

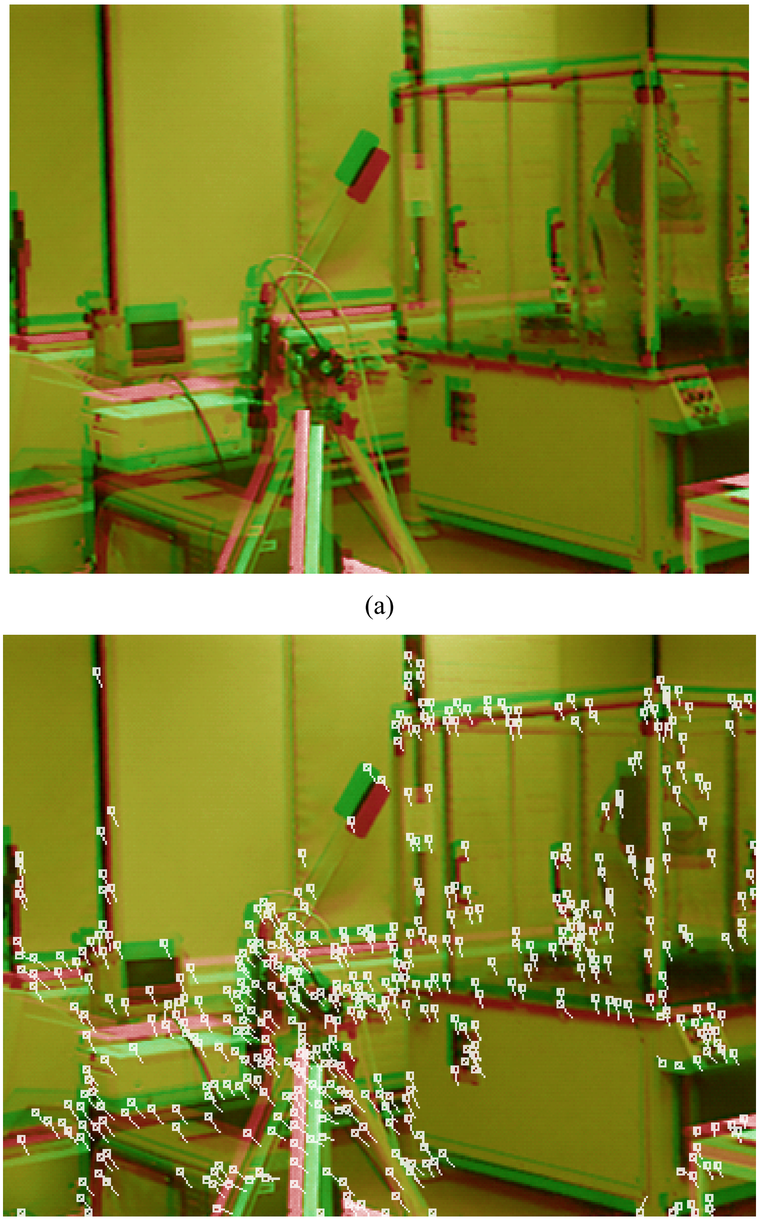


(a)

(b)

Figure 5-65. (a) Illustrates the motion in the scene and (b) shows the vectors calculated in the scene using the active mesh algorithm.

Figure 5-65 (a) illustrates the motion in the parking sequence of images (over 12 frames) as discussed in Section 5.6.2 . The first frame has been altered from a greyscale image to becoming the green plane of an RGB colour image, and the last frame has been altered to becoming the red plane of the same colour image. This causes ghosting that displays the movement between the two frames. A hand drawn comparison was attempted between these two frames to point out the problems in the active-mesh results, but was found to be less accurate than the obtained results.



(a)

(b)

Figure 5-66. (a) Illustrates the motion in the scene and (b) shows the vectors calculated in the scene using the active mesh algorithm overlayed on (a).

Figure 5-66 (a) illustrates using the same technique, the motion between the two frames of the laboratory sequence as discussed in Section 5.4.2 Once again the calculated vector field in (b) represents correctly the vector field which a human would determine from the difference image, as shown in (a).

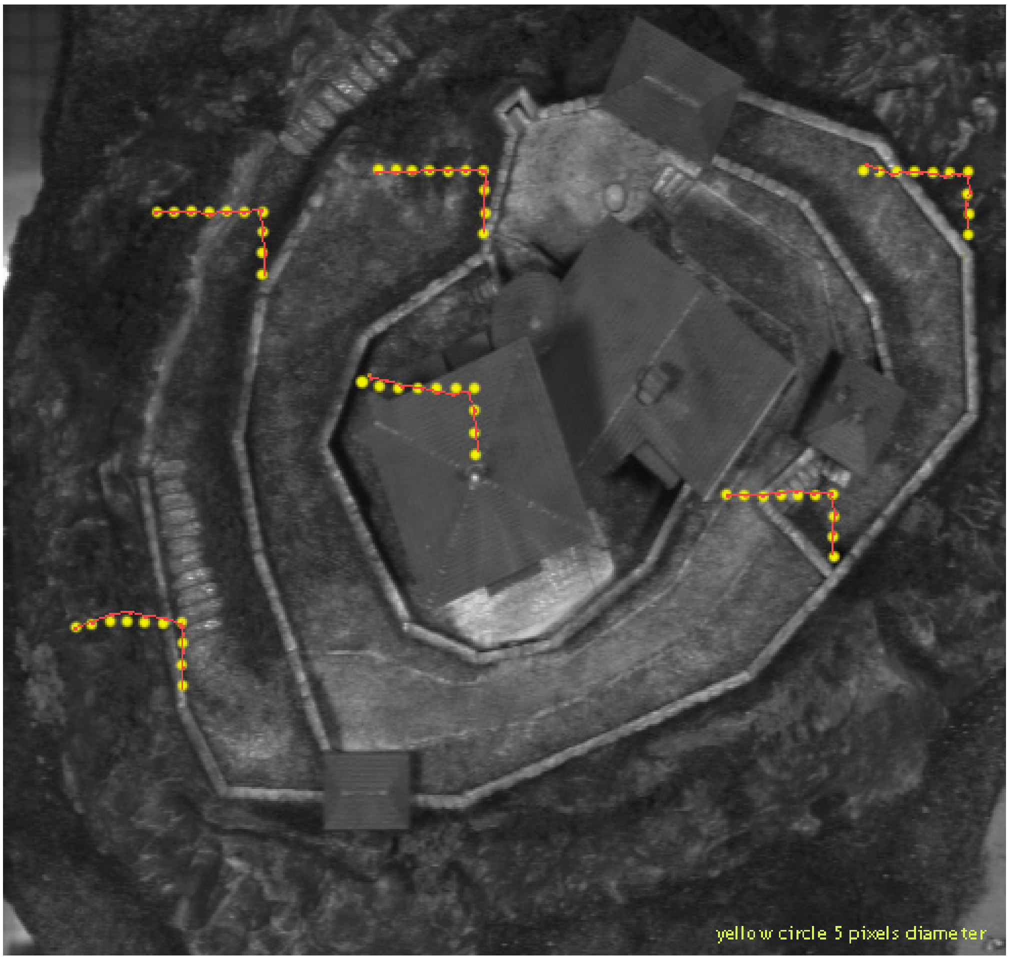


Figure 5-67. Hand-drawn estimate of motion in the castle sequence[25].

Figure 5-67 displays a hand-drawn estimate of the motion in the castle sequence (as in Section 5.4.4) over the 10 frames. A yellow dot was placed on strong features through the image frames. The features chosen were easy to locate for a human observer (such as the apex of the roof) and so were not strong image features. The equivalent calculated paths using the active mesh algorithm are overlaid in red, following very closely the expected locations (usually within ± 2 pixels, allowing that the human choice is accurate to within ± 1 pixels). The surrounding mesh nodes, weakly pulling away or towards the correct path, usually cause any deviation of a node from the path. This can cause a slight deviation in the path of the node, but can be traded off against internal stability for stronger external forces, if desired.

[25] A useful 'animated gif' is available at http://www.eeng.dcu.ie/~molloyd/phd/ that displays the movement in this sequence with the mesh overlaid, and the vector fields growing to follow the sequence.

5.9 3-D Re-Construction using Active Meshes

5.9.1 Stereo Imaging

The key problem area in stereo vision is the determination of the correspondences between features in the two camera views. These features may be image intensity values, edges, points, and other primitives. The fundamental sections of stereo imaging are (1) how the geometry and calibration of the stereo system calibrated (2) what primitives are to be matched between the two images (3) what *a priori* assumptions are made about the scene to determine the disparity, and (4) the estimation of depth from the disparity. Before solving the correspondence problem the multiple cameras must be properly calibrated (Geiger *et al*, 1992).

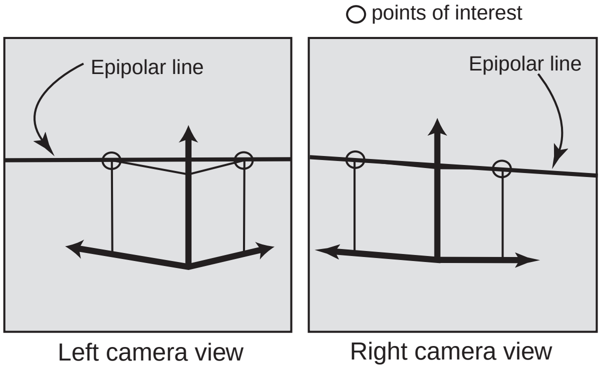


Figure 5-68. Calibrating two cameras.

Calibration is necessary before it is possible to perform any 3-D reconstruction of any form, as if the camera was not properly calibrated the structure constructed, would have little real-world correspondence. This is usually performed by the use of a *calibration pattern* (Faugeras, 1993). This calibration pattern is one where the *calibration points* are extracted at the intersection of the vertical and horizontal lines, usually very easily.

Epipolar lines are calculated that go through the pixels at the two upper corners of the grids in the left and right image (see Figure 5-68). This may be repeated for many points in the images and it becomes apparent that all the epipolar lines converge to a point, the *epipole*, located outside the image. From this method, and more complicated mathematical expressions it is possible to use this epipole for the reconstruction of 3-D points and the calibration of the cameras.

5.9.2 Stereo Correspondence

This is the key problem in the stereo approach; matching features between the two camera views. This correspondence is in general under-determined and heuristics are often required. These heuristics are generally derived from the properties of the real world, e.g. the *ordering constraint*, since most surfaces in the real-world are smooth, points lie in the same epipolar lines in both views. One famous case where this constraint is violated is the double nail illusion (Yuille, 1990), as in Figure 5-69.

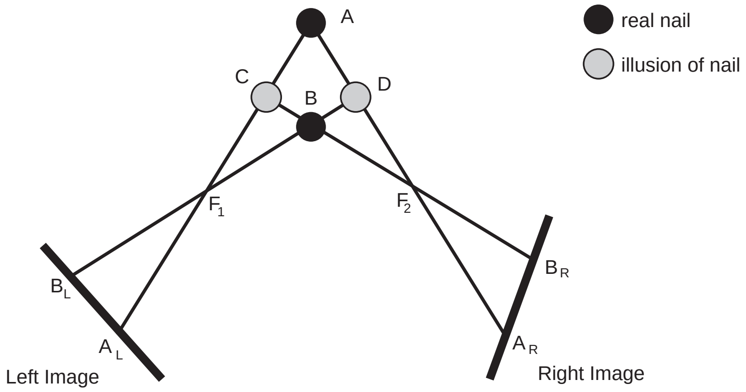


Figure 5-69. The double nail illusion.

When Humans look at the nail *B*, at a close distance with one eye on either side they see 3 nails in total, at points *C*, *B* and *D*. This has important consequences for the ordering constraint, as the observer moves left to right in the Left image B_L then A_L will be met. However going left to right in the Right Image A_R then B_R will be met, violating the ordering constraint. The *epipolar constraint* is another constraint that is of good use in the correspondence problem. It states that points on the epipolar line of one camera can be matched to the points on an epipolar line of another camera directly. This changes the problem of point matching to the matching of epipolar lines that are in parallel in each individual view.

As discussed in Section 3.7.1, active contour models can be applied to the problem of stereo matching where a pair of corresponding contours is available in a pair of stereo images. Kass *et al* (1987) uses a constraint added to the model energy formulation to pass information between the pair of contours while Terzopoulos *et al* (1987) uses sequence of images of the silhouettes of objects for 3-D reconstruction.

The active mesh formulation that is described in this work may also be modified for use in 3-D reconstruction. Brady and Wang's (1992) DROID approach, as discussed in Section 2.9, uses feature matching for a structure from motion algorithm based on stereopsis (the use of two cameras at different positions to examine the one scene). The correspondence calculation is the most difficult and erroneous part of the algorithm.

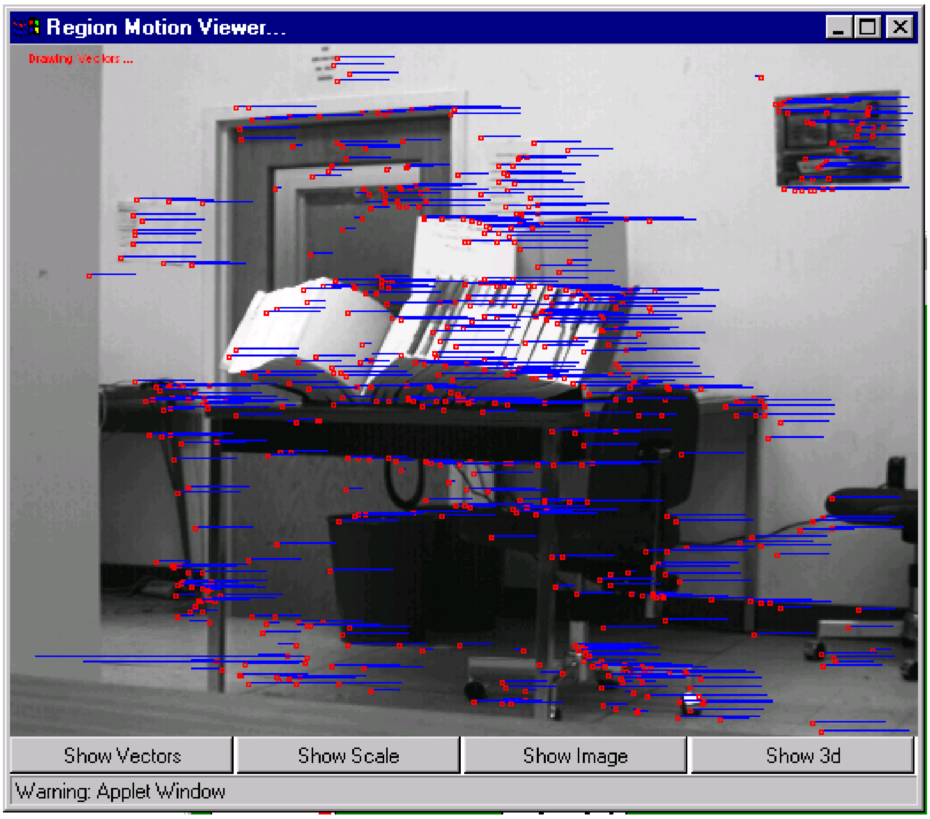


Figure 5-70. The stereo matching results of the active mesh

The active mesh algorithm has been tested on pairs of epipolar-constrained images for feature matching for 3-D reconstruction. It has been found that no ordering or uniqueness constraints are required as the structured feature point format of the mesh automates this to an extent. Rather, in these epipolar-constrained images, the feature search space of the mesh nodes can be constrained to be linearly (or almost linearly) aligned with the principal *x*-axis of the constrained image. This constraint on the search space improves the quality of the matching algorithm. The results from the active-mesh approach to stereo matching are shown in Figure 5-70.

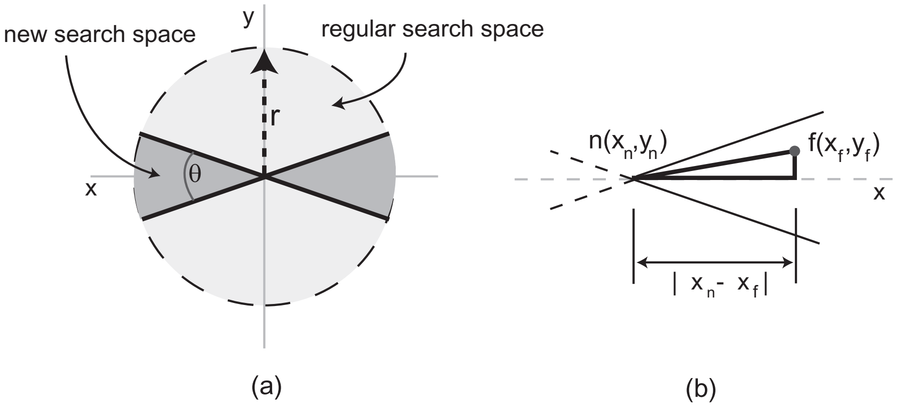


Figure 5-71. Stereo correspondence search space (a) shows the search space and (b) illustrates a close up of the measurements made.

Figure 5-71(a), shows the stereo correspondence search space that is used. The new search space is within the regular search space, inside an angle *θ*, where *θ* is usually small (~5 degrees). The reason for the angular search space is to deal with positional errors in the *y* direction that occurs with the location of 2-D feature points in calibrated image pairs. Figure 5-71(b) shows the calculation of distance between the mesh node point *n* and the feature point *f*, using only the *x*-component of the distance. Only the *x*-component of the force is applied to the mesh node, to prevent the mesh node from 'passing on' *y*-forces to connected nodes. This prevents mesh nodes from being pushed out of their *y*-located epipolar lines.

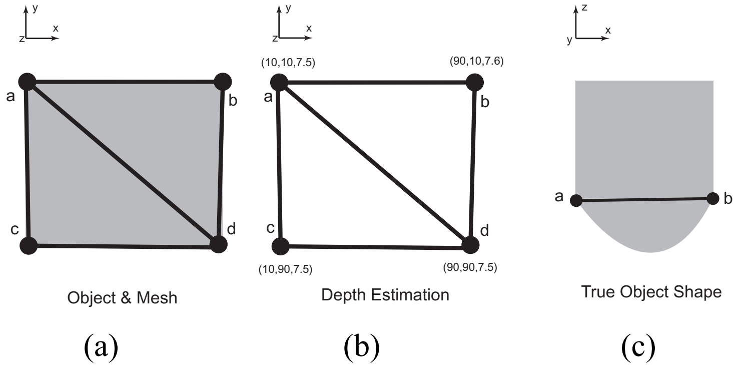


Figure 5-72. Depth estimation (a) illustrates a straight on view of what appears to be a cube. In (b) the depth is estimated at the corner points. However in (c) it is apparent from the view from above that the true shape of the cube curves out towards the observer.

It is important to note that depth information calculated using the active mesh implementation is only an estimate of the depth of the scene. For example, in Figure 5-72 the object is uniformly textured and there are no feature points detected other than at the silhouette points of the object. However, the true shape of the object is quite different from the estimated shape but is not determined by the stereo algorithm. This is true of many depth stereo algorithms.

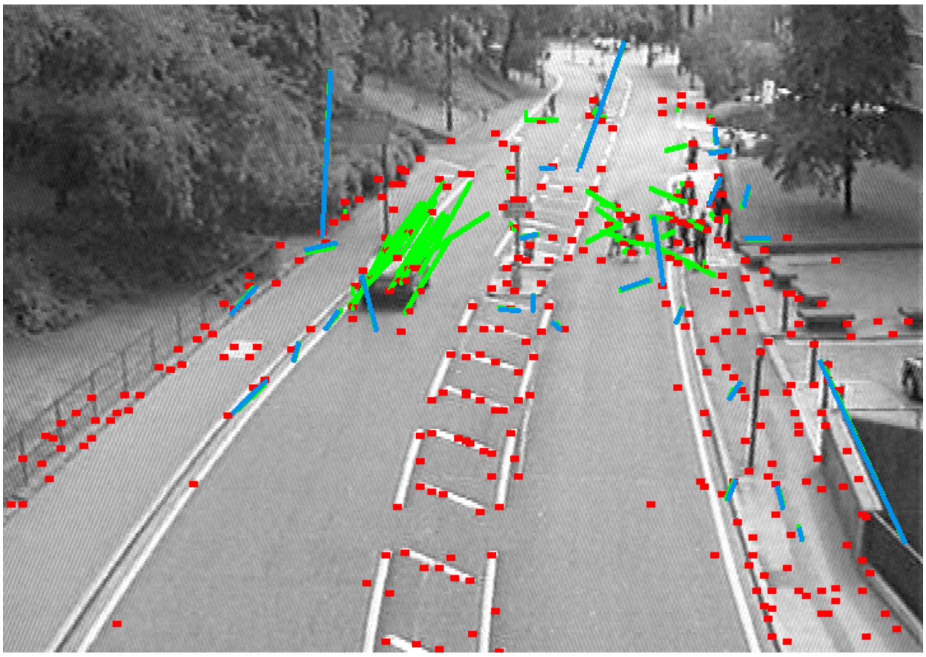


Figure 5-73. The ASSET-2 problem vectors (in blue), labelled by hand.

5.10 Comparison with the Active Mesh Technique

There are several differences between the active mesh approach and the other approaches as described. In comparison to the ASSET-2 (Smith, 1995) approach:

- The clustering technique is used to identify regions of similar flow vectors and the feature matching is not constrained in any way other than feature matching.
- The active mesh approach uses constrained feature matching, with the mesh structure greatly aiding the matching process.
- Increasing the number of features in the image sequence will result in a greater number of poor feature matches, which will be determined by the clustering algorithm as separate clusters, or indistinct clusters.
- Increasing the number of features in the active mesh approach increases the density of points on an object and further constrains the feature matching.
- In the ASSET-2 approach, weak matches will be discarded, whereas in the active mesh approach the weak matches will have a weak effect on the mesh.

- As can be seen in Figure 5-73, the ASSET-2 algorithm has problems with vector outliers, giving, in this case some very strange vectors (top, left and right of the road). There is no significant higher-level process that can prevent this.

The active mesh approach provides a higher-level approach to feature matching. The spatial relationship of features in the primary frame is used to constrain the matching process in the subsequent frame. The more features that are used, the better described the motion becomes. It provides a much less computationally expensive approach than optical flow:

- Feature points are used, involving calculation at only a small proportion of image points.
- Larger motion steps are possible, as shown in the results; the algorithm could skip 12 frames of available standard optical flow test sequences, and still provide a reasonable estimate of the motion.
- It integrates local and global information, the local search space information of the mesh nodes (external forces), with the global mesh structure constraints (internal forces).
- It is robust to image noise, avoiding the use of second order derivatives and preserving the mesh structure through occlusion.

The active mesh approach provides a relatively computationally inexpensive approach to robustly determining motion information in image sequences, using no *a priori* scene information, initializing and adapting as necessary.

Chapter 6 - Summary and Future Work

This research has demonstrated a method that allows unconstrained motion tracking in image sequences. It combines a high-level active mesh approach with the lower-level aspects of feature detection to generate a template for feature matching that is active in nature. This template can remove the ambiguity in the lower-level feature matching approach by allowing a more globally constrained solution to a usually local problem.

The active mesh algorithm is a modular approach that allows components of the algorithm, such as the feature detectors, filters and feature matching modules to be replaced by other techniques.

The active mesh approach provides several advantages for motion tracking, such as:

- Self-Initialisation, based on image features, providing a suitable starting point for the algorithm.
- Self-adaptive, as features may be added or removed from the mesh as frames progress, to prevent the mesh from becoming redundant.
- No *a priori* knowledge required about the scene, but can be added to provide an even more structured, multiple mesh and region approaches.
- The high-level information is captured in the mesh model that uses lower-level feature information. The strength of the matches also lends information to the motion of the mesh.
- The algorithm can determine the quality of match at each node.
- The algorithm is provided with parameters that are optimised for tuning rather than having to be accurately determined by the user for useful performance.

An application was successfully developed in Java to implement this approach as described in Section 4.12, showing different possible implementation scenarios.

.

6.1 Research Contributions

6.1.1 Major Contributions

The immediate contribution of this work is the overall algorithm that provides an innovative approach to helping solve the motion-tracking problem. It uses self-initialising, automatic modelling using unstructured meshes in an active weak model approach. This faced the difficult challenge of providing an unstructured mesh model that would function as an active contour model, while still preserving its structure. Suitable force handling was developed to combine the forces at the mesh nodes and energy formulations were developed to deal with the determined internal and external forces that arise. This active mesh approach was studied under numerous test sequences to refine the approach and to determine suitable feature matching techniques and parameters for the internal and external energy formulations. The overall algorithm was applied to numerous real-world image sequences, providing proof of the practical application of this approach. The active mesh was shown to be a higher-level approach than feature matching, where the spatial relationship of features in the primary frame is used to constrain the matching process in the subsequent frames. This algorithm was shown to perform well on real-world images, at a fraction of the computational cost of an optical flow algorithm.

The further extension of this technique to multiple meshes, region meshes and depth estimation was performed and provided new application areas for the algorithm. The multiple meshes were applied to multiple independent object tracking and the suggestion was made for the combination of an overall global mesh and numerous smaller meshes, to provide local and global information within the scene. These applications have been shown worthy of future attention for both applications and further improvements. Indeed the long-range contribution of this work should act as the basis for future applications in long sequence motion tracking, 3-D reconstruction and other vision applications.

6.1.2 Minor Contributions

Other contributions of this work include the implemented use of the Java language for computer vision applications, finding useful aspects of the language structure, ideal for vision applications. Some of these include the development of Java classes for image

.

processing, active contours and active meshes that can be used directly by future researchers. Other useful aspects of this research include:

- The modification of the SUSAN algorithm, by adding ‘no-hope’ candidate prefilters for improved computational performance of the algorithm, by taking advantage of its multistage approach.
- The implementation of the SUSAN algorithm with an active contour approach was also performed.

6.2 Generality of the Method

The method can be generalised to other machine vision applications, but for optimum performance a scene with strong feature points is required. The modularity of the approach can further aid the generality of the method, for example, by replacing the corner feature detector by a region centre location algorithm.

The choice of parameters for tuning also helps in the generality, allowing the user interaction with the low-level algorithm parameters. For example, allowing the adjustment of the regularisation parameters can define a mesh that is strongly affected by external forces, or indeed indifferent to them.

As few domain specific assumptions as possible were made, to allow the algorithm to be as generic as possible. Only a few assumptions were made for the optimum operation of this method, including:

- The features used are consistent across image frames.
- The spatial difference of the motion between frames is within the mesh node search space at a number of mesh nodes.
- At least 3 feature points are available for mesh generation.

Generality is determined by the first two conditions, while the last assumption is a categorical condition. Further testing is required to assess the sensitivity of the assumptions for other application implementations.

6.3 Future Work

As is true for many works of research there are many opportunities for extensions and refinements to the methodology presented. The usefulness of the methodology could be enhanced in the following areas:

.

- Further image pre-filtering for noise reduction.
- Feature detector improvement by the use of detection methods that become available.
- Improvement of the feature matching technique to provide a more suitable technique for the active mesh implementation.
- Search space choice automation, using time filtering.

The extent to which these improvements will be explored is a function of the applications to be developed that will depend on this method. The general technology trends of improvement of the image capture quality and microprocessor performance will also have an effect of the quality and execution speed of the algorithm. Some future improvements to the algorithm include: 3-D active meshes, the use of B-Splines, the use of Kalman filtering, as well as improvements to the region based active mesh algorithm.

6.3.1 3-D Active Meshes

As discussed previously one of the major difficulties in the general approach is the accurate matching of feature points. If a stereo view was used for the implementation stage then it should be possible to use 3-D depth information to allow the mesh exist in the 3-D plane, rather that only in the image plane. Most of the structure of the mesh would be the same, except that node connections would have mesh nodes with *(x,y,z)* positions and the forces would also be of that form. The idea would be that as well as providing the motion estimation in the scene the algorithm would also provide an estimate of the 3-D scene structure. The mesh would also be bound more tightly to the scene structure and would be less likely to drift in location. For this approach to work there would be certain constraints:

- The 3-D location information would have to be accurate and consistent on a frame-by-frame basis.
- A 3-D version of the Delaunay algorithm would have to be used.

.

6.3.2 Use of B-Splines as the mesh lines

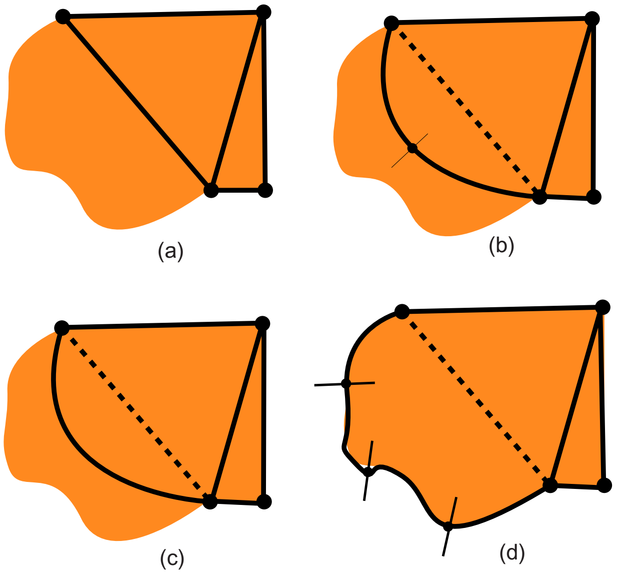


Figure 6-1. The B-spline active mesh approach.

In the current implementation of the active mesh, the mesh lines have no relationship with the image other than being the direct connection between image features. It should be possible to relate the mesh to image edges, to allow the integration of edge-based information into the mesh. The most likely way to link these image edges into the mesh would be to use B-splines to represent those mesh line connections that can be related to image edges. This would be especially true in the multiple mesh implementation, where the meshes might be representing objects, such as vehicles (as in Section 5.5.2) where the mesh lines on the convex hull of the mesh could certainly be replaced by ‘edge-loving’ B-splines. Figure 6-1 shows an example of this idea, where in (a) the true edge of the object is related to the mesh, except to the curved left area of the object. If this convex hull’s mesh line is replaced in (b) by a B-spline that performs an edge constrained expansion, perhaps using the step-by-step initialisation method of Neuenschwander *et al* (1994)(see Section 3.2.2). This edge determination of the object area would provide further energy forces (edge based) to help in the motion determination process as well as defining the true object boundary. If the true boundary is known then the motion information could be propagated along the edge (and indeed estimated along the edge) to help in motion description.

6.3.3 Use of Kalman Filtering for Active Meshes

As discussed in Section 3.4 on the use of Kalman filtering for active contours, it should also be possible to apply that technique to the active mesh implementation. This would

.

allow not only prediction of position of subsequent frames, but it should be possible to apply the Kalman filtering to the determination of the search space around the mesh nodes. Baumberg (1996) uses a Kalman approach in choosing the search direction for the Leeds people tracker, using prior knowledge that is available about the location of the image features, using the spatial covariance of the estimated contour points. The constrained feature search that is suggested was found to improve the performance of the tracking algorithm.

Another idea for improving the use of the search space that has been considered is the use of a general search space that grows, until the match uncertainty is at an acceptable threshold. For example, if the search space begins at a fixed value radius and only 25% of the mesh nodes have located suitable match features, the search space could be increased in radius until 50%+ of the mesh nodes have located suitable match features. This idea could also be coupled with the Kalman filtering approach as a prediction of the next frame search space.

6.3.4 Improvements to the Region Based Active Mesh

There are many improvements that are possible to the region based active mesh. The calculation of the region information, with the exception of the image subtraction, is being performed outside of the algorithm. This should be integrated into the main algorithm and developed to provide a more advanced region segmentation algorithm.

Currently the individual meshes have no relationship to each other. It should be possible to combine the individual meshes with a general scene active mesh to provide information about the relationship of these objects. The use of multiple meshes should also allow the determination of scene occlusion information, when the objects have intersecting paths. The object that has the strongest match information after an occlusion is the forefront object. This method should prove useful in a wide variety of research and educational endeavours.

.

Bibliography

A.A. Amini, T.E. Weymouth, R.C. Jain (1990), "Using Dynamic Programming for Solving Variational Problems in Vision", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12, 9, 855-867.

N. Ancona (1992), "A fast Obstacle Detection Method based on Optical Flow", *ECCV'1992*, 267-272.

D.H. Ballard, C.M. Brown (1982), *Early Processing. In: Computer Vision*, Prentice-Hall (Englewood Cliffs, New Jersey 07632), 63-118.

J.L. Barron, D.J. Fleet, S.S. Beauchemin, T.A. Burkitt (1992), "Performance of Optical Flow Techniques", *Proceedings of the 1992 Computer Vision and Pattern Recognition*, 236-242

B. Batchelor (1991), *Intelligent Image Processing in Prolog*, Springer-Verlag, London.

A. Baumberg (1996), "Hierarchical shape fitting using an iterated linear filter", *in proc. British Machine Vision Conference, volume 1*, 313--323. (Also: A.Baumberg (1996), "Hierarchical shape fitting using an iterated linear filter", *Internal Technical Report*, School of Computer Studies, University of Leeds, 19 pages)

B. Bhanu, O.D. Faugeras (1984), "Shape Matching of Two-Dimensional Objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, 137-156.

A. Blake, A. Yuille (1992) *Active Vision*, The MIT Press, Cambridge, Massachusetts.

A. Blake, M. Isard (1998), *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion*, Springer-Verlag Press.

P. Bouthemy, E. Francois (1993), "Motion Segmentation and qualitative dynamic scene analysis from an image sequence", *International Journal of Computer Vision*, 10, 157-182.

A.C. Bovik, T.S.Huang, D.C. Munson, (1987) “The Effect of Median Filtering on Edge Estimation and Detections”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9, 2, 181-193.

R. Bowden, T.A. Mitchell, M. Sahardi (1997), " Real-Time Dynamic Deformable Meshes for Volumetric Segmentation and Visualisation", BMVC, 1997. 13 pages.

M. Brady, H. Wang, (1992) “Vision for mobile Robots”, Phil. Trans. R. Soc. London B. 337, 341-350.

K. Brisdon, G.D. Sullivan, K.D. Baker (1988), “Feature Aggregation in Iconic Model Matching”, *Proceedings of the Alvey Vision Conference*, AVC-88, Manchester, 19-24.

N. Byrne (1992), *Motion Estimation in Computer Vision*, Ph.D. Transfer Requirement Report, School of Electronic Engineering, Dublin City University.

J. Canny (1986), “A computational approach to edge detection”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8, 6, 679-698.

D. Cédras, M. Shah (1995), "Motion-based recognition: a survey", *Image and Vision Computing*, 13, 129-155.

F. Chabat, G.Z. Yang, D.M. Hansell (1999), “A corner orientation detector”, *Image and Vision Computing*, 17, 761-769.

S. Chandran, A.K. Potty (1998), "Energy Minimisation of Contours Using Boundary Conditions", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 5, 546-549.

.

D. Charnley, R. Blissett (1989), “Surface Reconstruction from outdoor image sequences”, *Image and Vision Computing*, 7, 1, 10-16.

L.D. Cohen (1991), "NOTE On Active Contour Models and Balloons", *Computer Vision Graphics Image Processing: Image Understanding*, 53, 2, 211-218.

L.D. Cohen, I. Cohen (1993), "Finite-Element Methods for Active Contour Models and Balloons for 2-D and 3-D Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15, 11, 1131-1147.

T.F. Cootes, C.J. Taylor (1992), "Active Shape Models - 'Smart Snakes' *", British Machine Vision Conference 1992*, Leeds, 22-24 Sep 1992, 266-276.

I.J. Cox (1993), "A Review of Statistical Data Association Techniques for Motion Correspondence", *International Journal of Computer Vision*, 10, 53-66.

I.J. Cox, S.L. Hingorani (1996), “An efficient Implementation of Reid’s Multiple Hypothesis Tracking algorithm and its evaluation for the purpose of visual tracking”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18, 2, 138-150.

R. Curwen, A. Blake (1992), "Dynamic Contours: Real-time Active Splines", *Active Vision*, The MIT Press, Cambridge, Massachusetts, 39-57.

L.S. Davis (1979), "Shape Matching Using Relaxation Techniques", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1, 60-72.

R. Deriche (1987), “Using Canny’s criteria to derive a recursively implemented optimal edge detector”, *International Journal of Computer Vision,* 167-187.

E.E. Dickmanns, V. Graefe (1991), “Applications of Dynamic Monocular Machine Vision”, *Computer Vision – Advances and Applications*, IEEE Computer Society Press, 661-681.

.

Z. Duric, E. Rivlin, A. Rosenfeld (1998), "Understanding Object Motion", *Image and Vision Computing*, 16, 785-797.

M. Etoh, Y. Shirai, M. Asada (1993), "Active Contour Extraction Based on Region Descriptions Obtained from Clustering", *Systems and Computers in Japan*, 24, 11, 55-65. (Translated from: Denshi Joho Tsushin Gakkai Tonbushi, vol. J75-D-II, 7, 1111-1119.)

J.Q. Fang, T.S. Huang (1984), "Some Experiments on Estimating the 3-D Motion Parameters of a Rigid Body from Two Consecutive Image Frames", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, 545-554, 1984.

O.D. Faugeras, S. Maybank (1990), "Motion from Point Matches: Multiplicity of Solutions", *International Journal of Computer Vision*, 4, 225-246.

O. Faugeras (1993), *Projective Geometry. In: Three-Dimensional Computer Vision (A Geometric Viewpoint)*, Cambridge, Massachusetts: The MIT Press, 7-33.

Y.S. Fong, D.H. Brown (1985), "A Centroid Tracking Scheme in a Weighted Coordinate System", ECE Dept. Clarkson University, New York, Report. 219-221.

S. Geman, D. Geman (1984), "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images", *IEE Transactions or Pattern Analysis and Machine Intelligence,* 6, 721-741.

E. Grosso, M. Tistarelli, G. Sandini (1992), "Active/Dynamic Stereo for navigation", *Computer Vison, ECCV, Italy May'92*, Lecture notes in Computer Science 588, 516-525.

W. Guan, M. Songde (1998), "A List-Processing Approach to Compute Voronoi Diagrams and the Euclidean Distance Transform", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 7, 757-761.

S. Gunn, M.S. Nixon (1997), "A Robust Snake Implementation; A Dual Active Contour", *IEEE Transactions Pattern Analysis and Machine Intelligence*, 19, 1, 63-68.

C.G. Harris (1987), “Determination of Ego-Motion from matched points”, *3rd Alvey Vision Conference Proceedings*, 189-192.

M. Hashimoto, H. Kinoshita, Y. Sakai (1994), "Sampled Active Contour Model, *Transactions of the Institute of Electronics, Information and Communication Engineers*, j77D-2, 2172-2178.

T. Heap, D. Hogg (1996), “3D Deformable Hand Models”, *Internal Technical Report*, School of Computer Studies, University of Leeds, 10 pages.

D.J. Heeger (1998), “Optical Flow Using Spatio-Temporal Filters”, *International Journal of Computer Vision*, 1, 4, 279-302.

E.C. Hildreth (1984), "Computations Underlying the Measurement of Visual Motion", *Artifical Intelligence*, 23, 309-354.

B.K.P. Horn and B.G. Schunck (1981), "Determining Optical Flow", *Artificial Intelligence*, 17, 185-203

H.H.S. Ip, D. Shen (1998), "An affine-invariant active contour model (AI-Snake) for model-based segmentation", *Image and Vision Computing*, 16, 135-146.

A.K. Jain (1989), *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ, USA: Prentice-Hall.

M. Kass, A. Witkin, and D. Terzopulos (1987), "Snakes: Active contour models", *Proceedings of the 1st Internal Conference on Computer Vision*, 259-268.

.

J.K. Kearney, W.B. Thompson, D.L. Boley (1987), "Optical Flow Estimation: An Error Analysis of Gradient-Based Methods with Local Optimization", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9, 229-244.

J. Kim, C. Joo, J. Choi (1990), "Improved block matching algorithm considering edge components", *Applications of Digital Image Processing XIII.* SPIE Press, 181-191.

I. Kosaka, G. Nakazawa (1995), "Vision-Based Motion Tracking of Rigid Objects Using Prediction of Uncertainties", *Proceedings of the 1995 International Conference on Robotics and Automation*, Nagoya, Japan, 9 pages.

K.F. Lai (1994), *Deformable Contours: Modelling, Extraction, Detection and Classification*, Ph.D. Thesis at the University of Wisconsin-Madison.

K.F. Lai, R.T. Chin (1998), "On modelling, extraction, detection and classification of deformable contours from noisy images", *Image and Vision Computing,* 16, 55-62.

F. Leymarie, D. Levine (1993), "Tracking Deformable Objects in the Plane Using an Active Contour Model", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15, 6, 617-633.

K.V. Mardia, W. Qian, D. Shah, K.M.A. deSouza (1997), "Deformable Template Recognition of Multiple Occluded Objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19, 9, 1035-1042.

D. Marr, H.K. Nishihara (1978), "Visual Information Processing: Artificial Intelligence and the Sensorium of Sight", *Technology Review*, Oct, vol. 81, pt.1, 28-49.

J. Matas, J. Kittler (1992), "Contextual Junction Finder", *British Machine Vision Conference 1992*, Leeds, 22-24 Sep 1992, 119-128.

.

I.S. McQuirk, B.K.P. Horn, H.-S. Lee, and J.L. Wyatt (1998), “Estimating the Focus of Expansion in Analog VLSI”, *International Journal of Computer Visions*, Vol. 28, No. 3, 261-277.

G. Medioni, R. Nevatia (1984), "Matching Images Using Linear Features", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, 675-685.

F. Meyer, P. Bouthemy (1992), “Region-Based Tracking in an Image Sequence”, *Proceedings of the Second European Conference on Computer Vision*, 476-484.

B.R. Mitchell, T.A. Sahardi (1998) "Real-Time Dynamic Deformable Meshes for Volumetric Segmentation and Visualisation", Machine Vision and VR Group, Dept. M &ES, Brunel Universtiy. 1-13, *On-Line publication*: http://www.brunel.ac.uk/~empgrrb/publications/bmvc97/paper.html

F. Mokhtarian, R. Suomela (1998), "Robust Image Corner Detection Through Curvature Scale Space", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 20, 12, 1376-1381

D. Molloy, T. McGowan, K. Clarke, C. McCorkell, P.F. Whelan (1994), "Application of machine vision technology to the development of aids for the visually impaired", in *Machine Vision Applications, Architectures, and Systems Integration III*, Proc. SPIE 2347, 31 Oct-2 Nov, Boston USA, 59-69.

D. Molloy, P.F. Whelan (1996), "Motion Discrimination in Applied Vision Systems", Dublin City University, School of Electronic Engineering, *Internal Technical Report*, 26th March 1996.

D. Molloy, P.F. Whelan (1997a), "Robotic navigation using self-initialising active contours", *Intelligent Robots and Computer Vision XVI: Algorithms, Techniques, Active Vision, and Materials Handling*, Pittsburgh, Oct. 1997, Proc. SPIE Vol.3208.

.

D. Molloy, P.F. Whelan (1997b), "Self-Initialising Active Contours for Motion Discrimination", *Proc. IMVIP'97 - Irish Machine Vision and Image Processing Conference 1997*, 10 - 13 September 1997, University of Ulster, Northern Ireland, pp 139 - 146.

D. Molloy, P.F. Whelan (1999a), "Motion Tracking using Self-Initialising Active Meshes", *The I.E.E. Seventh International Conference on Image Processing and its Applications*, University of Manchester, UK: 12-15 July.

D. Molloy, P.F. Whelan (1999b), "Active-Mesh Self-*Initialisation", IMVIP'99 Irish Machine Vision and Image processing Conference 1999*, September 1999, Dublin City University. pp 116 - 129.

H.P. Movavec (1977), "Towards Automatic Visual Obstacle Avoidance", *Proceedings of the International Joint Conference on Artificial Intelligence*, Cambridge MA, 584.

D.W. Murray, B.F. Buxton (1987), "Scene Segmentation from Visual Motion Using Global Optimization", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9, 220-228.

H.H. Nagel (1987), "On the estimation of Optical Flow: Relations between Different Approaches and Some New Results", *Artificial Intelligence*, 33, 299-324.

H.H. Nagel (1990), "Extending the 'Oriented Smoothness Constraint' of optical flow into the temporal domain and the estimation of derivatives", *Proceedings of the 1st European conference on Computer Vision Proceedings*, 139-148.

F. Nashashibi, P.Fillatreau, B.D. Wright, T. Simeon (1994), "3-D Autonomous Navigation in a Natural Environment", *IEEE International Conference on Robotics and Automation*, California, May'94, 433-439.

.

C. Nerriec, A. Diez, I. Alvarez, J.A. Cancelas, R. Gonzalez (1994), "A Syntactic and Contextual Edge Detector and Application of a Rewriting Rules Set to Improve Detection" *Proceedings of IECON '94, The 20th international Conference on Industrial Electronics Control and Instrumentation*, IEEE Press, 2, 994-997.

W. Neuenschwander, P. Fua, G. Szekely, O. Kubler (1994), "Initialising Snakes", *Proceedings of the 1994 IEEE Computer Society on Computer Vision and Pattern Recognition*, Seattle, 658-663.

J.A. Noble (1989), *Descriptions of Image Surfaces*, Ph.D. thesis, Department of Engineering Science, Oxford University, 215 pages.

J.S. Park, J.H. Han (1998), "Contour matching: a curvature-based approach", *Image and Vision Computing*, 16, 181-189.

N. Peterfreund (1999), “Robust Tracking of Position and Velocity With Kalman Snakes”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **21**, 6.

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery (1992) *Numerical Recipes in C: The Art of Scientific Computing,* 2nd Edition, Cambridge University Press.

J.W. Roach, J.K. Aggarwal (1979), "Computer Tracking of objects moving in space", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1, 127-135.

L. Rosenthaler, F. Heitger, O. Kubler, R. von der Heydt (1992), "Detection of General Edges and Keypoints", *Computer Vision - ECCV '1992 (Second European Conference on Computer Vision).* Springer-Verlag, 78-86.

S. Soatto, P. Perona (1995), “Dynamic Rigid Motion Estimation From Weak Perspective”, ICV’95, 321-328.

J.A. Schnabel, S.R.Arridge (1999), "Active Shape Focusing", *Image and Vision Computing,* 17, 419-428.

.

S. Sclaroff, J. Isidoro (1998), "Active-Blobs", Internal Technical Report, Computer Science Department, Boston University (BU CS TR97-008), 9 pages.

B.G. Schunck (1989), "Image Flow Segmentation and Estimation by Constraint Line Clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11, 10.

I.K. Sethi and R. Jain (1987), "Finding Trajectories of Feature Points in a Monocular Image Sequence", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9, 56-73.

L.S. Shapiro, H. Wang, J.M. Brady (1992), "A Matching and Tracking Strategy for Independently Moving Objects", *British Machine Vision Conference 1992,* Leeds, 306-315.

J.R. Shewchuk (1997), *Delaunay Refinement Mesh Generation*, Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, CMU-CS-97-137.

E.P. Simoncelli, E.H. Adelson (1991), "Computing Optical Flow Distributions Using Spatio-temporal Filters", *MIT Media laboratory Vision and Modeling technical Report #165*, 20 pages.

D. Sinclair, A. Blake, S. Smith, C. Rothwell (1992), "Planar Region Detection and Motion Recovery", *British Machine Vision Conference 1992*, Leeds 22-24 Sep., 59-68.

S.M. Smith (1992), "A New Class of Corner Finder", *British Machine Vision Conference 1992,* Leeds, 22-24 Sep 1992, 139-148.

S.M. Smith 1992b), "Edge Thinning Used in the SUSAN Edge Detector", *Defense Research Agency Technical Report TR95SMS5*, 4 pages.

S.M. Smith (1992c), "Feature Based Image Sequence Understanding", *Ph.D. Thesis Submission*, Oxford University, Trinity Term 1992.

S.M. Smith, (1995), “ASSET-2: Real-Time Motion Segmentation and Object Tracking”, *Internal Technical Report TR95SMS2b*, Department of Clinical Neurology, Oxford University, 28 pages. {Also published as: S.M. Smith (1995), “ASSET-2: Real-Time Motion Segmentation and Shape Tracking”,*ECCV’1995,* 237-244.}

S.M. Smith, (1995b), “ALTRUISM: Interpretation of Three-Dimensional Information for Autonomous Vehicle Control”, *Internal Technical Report TR95SMS3*, Department of Clinical Neurology, Oxford University, 19 pages.

P. Soille, (1999), *Morphogical Image Analysis – Principles and Applications*, Springer-Verlag. Berlin Heidelberg, ISBN 3-540-65671-5 , April 1999.

L.H. Staib, J.S. Duncan (1992), "Boundary Finding with Parametrically Deformable Models", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14, 11, 1061-1075.

G.D. Sullivan, K.D. Baker (1992), "Active Contours using Finite Elements to Control Local Scale", *British Machine Vision Conference 1992*, Leeds, 22-24 Sep 1992, 481-488.

D. Terzopoulos, A. Witkin, M. Kass (1987), "Symmetry-Seeking Models for 3D Object Reconstruction*", Proceedings of the 1st Internal Conference on Computer Vision*, 269-279.

D. Terzopoulus, R. Szeliski (1992), "Tracking with Kalman Snakes", *Active Vision*, The MIT Press, Cambridge, Massachusetts, 3-20.

A. Tesei, C.S. Regazzoni (1994), “Local Density Evaluation and Tracking of Multiple Objects from Complex Image Sequences”, *Proceedings of IECON 1994.* IEEE Press, 744-748.

.

W.B. Thompson, S.T. Barnard (1981), "Lower Level Estimation and Interpretation of Visual Motion", COMPUTER, 14, 20-28.

C. Thorpe, M.H. Hebert, T. Kanade, S.A. Shafer, (1988) “Vision and Navigation for the Carnegie-Mellon Navlab”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 10, 3, 362-367. {Also published as: C. Thorpe, M.H. Hebert, T. Kanade, S.A. Shafer, (1991) “Vision and Navigation for the Carnegie-Mellon Navlab”, Computer Vision – Advances and Applications, IEEE Computer Press, 690-701}

C. Tomasi, T. Kanade (1990), “Shape and Motion without depth”, *In Proceedings of Image Understanding Workshop.*

P.H.S. Torr, D.W. Murray (1992), "Statistical Detection of Independent Movement from a Moving Camera", *British Machine Vision Conference 1992,* Leeds, 22-24 Sep 1992, 79-88.

M. Trajkovic, M. Hedley (1998), “Fast Corner Detection”, *Image and Vision Computing*, 16, 75-87.

R.Y. Tsai, T.S. Huang (1981), "Estimating Three-Dimensional Motion Parameters of a Rigid Planar Patch", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29, 1147-1152.

S. Ullman (1979), *The interpretation of Visual Motion*, MIT Press.

D. Vernon (1999), "Computation of instantaneous optical flow using the phase of Fourier components", *Image and Vision Computing*, 17, 189-199.

H. Wang, M. Brady (1995), “Real-time corner detection algorithm for motion estimation”, *Image and Vision Computing*, 13, 9, 695-703.

.

H. Wang, Lee (1994), "Active Mesh - A Feature Seeking and Tracking Image Sequence Representation Scheme", *IEEE Transactions on Image Processing*, 3, 5, 611-624.

C.S. Wiles, M. Brady (1995), “Closing the Loop on Multiple Motions”, ECCV’95, 308-313.

A.D. Worrall, R.F. Marslin, G.D. Sullivan, K.D. Baker, “Model-based Tracking”, Intelligent Systems Group, University of Reading, Technical Report, 9 pages.

Y.S. Yao, R. Chellappa (1995), “Tracking a Dynamic Set of Feature Points”, *IEEE Transactions on Image Processing*, 4, 10, 1382-1395.

D. Young (1995), "Active Contour Models (Snakes)", *Sussex Computer Vision: Teach Vision 7*. On-Line http://www.cogs.susx.ac.uk/, 18 pages.

A.L. Yuille, D.S. Cohen, P.W. Hallinan (1989), "Featue Extraction from Faces using Deformable Templates", *In Proceedings of Computer Vision and Pattern Recognition*, 104-109. {Repeated in: A.Yuille, P.Hallinan (1992), “Deformable Templates”, *Active Vision*, The MIT Press, 21-38}

F. Yuille, N. M. Grzywacz (1989), "A Mathematical Analysis of the Motion Coherence Theory", *International Journal of Computer Vision*, 3, 155-175.

Q. Zheng, R. Chellappa (1995), “Automatic Feature Point Extraction and Tracking in Image Sequences for Arbitrary Camera Motion”, *International Journal of Computer Vision*, 15, 31-76.

S.C. Zhu, T.S. Lee, A.L. Yuille (1995), "Region Competition: Unifying Snakes, Region Growing, Energy/Bayes/MDL for Multi-band Image Segmentation", *International Conference on Vision 1995*, IEEE press, 416-423.

Appendix A – Vision Techniques

A.1 Estimating the partial derivatives

For practical implementation of the optical flow algorithm it is necessary to be able to estimate the partial derivatives. Each of the estimates is the average of the four first differences taken over adjacent measurements in the image cube.

$$E_x \approx \frac{1}{4}\left\{E_{i,j+1,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+1} - E_{i+1,j,k+1}\right\}$$

$$E_y \approx \frac{1}{4}\left\{E_{i+1,j,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i,j+1,k} + E_{i+1,j,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+1} - E_{i,j+1,k+1}\right\}$$

$$E_t \approx \frac{1}{4}\left\{E_{i,j,k+1} - E_{i,j,k} + E_{i+1,j,k+1} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j+1,k} + E_{i+1,j+1,k+1} - E_{i+1,j+1,k}\right\}$$

A-1

The three partial derivatives of image brightness at the centre of the cube are each estimated from the average of first differences along four parallel edges of the cube. In this case *i* corresponds to the *y*-direction, *j* to the *x*-direction and *k* to the time direction. Here each unit of *x* and *y* is the *grid spacing* and each interval of *t* is the *frame sampling period*.

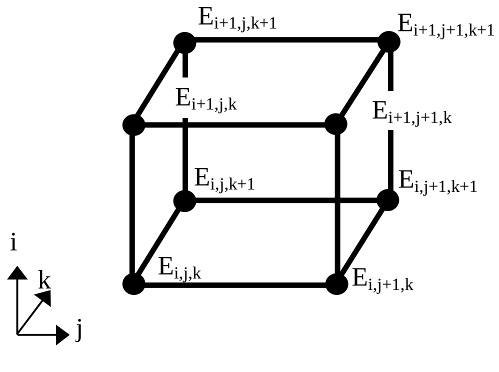


Figure A-1. Estimating the Laplacian of the Flow Velocities

1/12	1/6	1/12
1/6	-1	1/6
1/12	1/6	1/12

Figure A-2. A typical weighting system

The most common way of doing this is in the following form

$$\nabla^2 u \approx \kappa\left(\bar{u}_{i,j,k} - u_{i,j,k}\right) \text{ and } \nabla^2 v \approx \kappa\left(\bar{v}_{i,j,k} - v_{i,j,k}\right) \qquad \text{A-2}$$

Where the local averages of $\bar{u}$ and $\bar{v}$ are defined as

$$\bar{u}_{i,j,k} = \frac{1}{6}\left\{u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k}\right\} + \frac{1}{12}\left\{u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k}\right\}$$

$$\bar{v}_{i,j,k} = \frac{1}{6}\left\{v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\right\} + \frac{1}{12}\left\{v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k}\right\}$$

A-3

The proportionality factor κ equals 3 if the average is computed as shown and there is uniform grid spacing.

The Laplacian is estimated by subtracting the value at a point from a weighted average of the values at neighbouring points. Shown in Figure A-2 are suitable weights by which values can be multiplied.

A.2 Minimising Errors

The next step is to minimise the sum of the errors in the equation for the rate of change of image brightness given by, $\xi_b = E_x u + E_y v + E_t$ and the departure from smoothness in the flow by:

$$\xi_c^2 = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \qquad \text{A-4}$$

Horn & Schunck (1981), let the error to be minimised be:

$$\xi^2 = \iint \left(\alpha^2 \xi_c^2 + \xi_b^2\right) dxdy \tag{A-5}$$

Since in practice there will be quantisation error and noise, ξ_b will not be zero. This will have an error magnitude that is proportional to the noise in the measurement. From this fact Horn & Schunck choose a suitable weighting factor α^2. This minimisation is to be performed by finding suitable values for *(u,v)* the optical flow velocity, obtaining:

$$\begin{aligned} \left(\alpha^2 + E_x^2 + E_y^2\right)u &= +\left(\alpha^2 + E_y^2\right)\bar{u} - E_x E_y \bar{v} - E_x E_t \\ \left(\alpha^2 + E_x^2 + E_y^2\right)v &= -E_x E_y \bar{u} + \left(\alpha^2 + E_x^2\right)\bar{v} - E_y E_t \end{aligned} \tag{A-6}$$

where $\bar{u}$ and $\bar{v}$ are defined on the previous page, and then rewriting these equations as:

$$\begin{aligned} \left(\alpha^2 + E_x^2 + E_y^2\right)\left(u - \bar{u}\right) &= -E_x\left[E_x \bar{u} + E_y \bar{v} + E_t\right] \\ \left(\alpha^2 + E_x^2 + E_y^2\right)\left(v - \bar{v}\right) &= -E_y\left[E_x \bar{u} + E_y \bar{v} + E_t\right] \end{aligned} \tag{A-7}$$

and for an iterative solution, Horn & Schunck compute a new set of velocity estimates $\left(u^{n+1}, v^{n+1}\right)$ from the estimated derivatives and the average of the previous velocity estimates $\left(u^n, v^n\right)$ by:

$$\begin{aligned} u^{n+1} &= \bar{u}^n - E_x\left[E_x \bar{u}^n + E_y \bar{v}^n + E_t\right] \Big/ \left(\alpha^2 + E_x^2 + E_y^2\right) \\ v^{n+1} &= \bar{v}^n - E_y\left[E_x \bar{u}^n + E_y \bar{v}^n + E_t\right] \Big/ \left(\alpha^2 + E_x^2 + E_y^2\right) \end{aligned} \tag{A-8}$$

These estimates at a new point do not depend directly on the previous estimates at the same point.

In the parts of the image where the brightness gradient is zero (e.g. uniform regions), the velocity estimates are the averages of the neighbouring velocities. Eventually the values around such a region will propagate inwards. The number of iterations should be larger than the number of pixels across the region of largest area.

A.3 Centroid Tracking

The centroid of an object in a scene is a very useful measurement. It is defined as: *the point in a set whose coordinates are the mean values of the coordinates of the other points in the set.* On a 2-D object it is invariant through rotation and proportional scaling. One particular implementation of the centroid calculation suggested by Fong *et al* (1985) is especially applicable to real-time implementation. It is calculated in a similar form to Batchelor (1991):

$$I \leftarrow \frac{\sum_j \sum_i \{a(i,j) \times i\}}{N_{i,j}} \text{ and } J \leftarrow \frac{\sum_j \sum_i \{a(i,j) \times j\}}{N_{i,j}} \qquad \text{A-9}$$

Where $N_{i,j} \leftarrow \sum_j \sum_i a(i,j)$, in general $a(i,j) = 1$ or 0 for a binary image.

This is a rectangular coordinate system weighted with a monotonically increasing function of distance between the origin and the points in the coordinate system. It is useful in that it provides a very simple description of the motion that the object is undergoing between two frames, by the use of a single vector, providing that it has been calculated correctly. Since the object shape must be rigid, this causes problems in real-life scenes, as the tracked point may chance on the object, but it will however still be the centroid.

A.4 Region Based Representation

For region based tracking, we generally try to extract a representation of the region to be tracked. Some of the commonly used representations include:

Run length codes: Any region or a binary image can be viewed as a sequence of alternative strings of 0’s and 1’s. Run-length codes can represent these strings, or runs. For raster scanned images these images are then represented by the *(x,y)* coordinate of the first pixel in the run and the length of the run. *See Ballard (1982).*

Quad-Tree Method: In the quad-tree method, the given region is enclosed in a rectangular region. This area is divided into four quadrants, each of which is marked if it is totally black or totally white. The quadrant that is both black as well as white is marked as grey and is further sub-divided into four quadrants. It appears that quad-tree coding would be more efficient than run-length coding from a data compression viewpoint,

.

however computation of shape measurement such as perimeter and moments as well as image segmentation may be more difficult.

Projections: A 2-D shape or region *R* can be represented by its projections. A projection *g(s,θ)* is simply the sum of the run-lengths along a straight line of orientation theta and placed at a distance *s*. This is a reasonable method as the centroid and the maximum and minimum distance from the centroid can be used for calculating the angle *θ*.

Watersheds: When looking at the problem of multiple overlapping objects in a scene, morphological watersheds are extremely useful. When for example, two circles become partially overlapped they seem like one object. Watersheds may be used to break these circles into individual regions. See Dougherty (1992).

The use of regions as features allows handling of consistent object-level entities. With the other algorithms of tracking feature points, it is common to group these points together into a region at a later stage. It seems possible to examine regions as the feature points, thus removing the need to group feature points. Regions are however difficult to describe, so that they may be accurately matched. Descriptors such as the centre of gravity change position within the region as the area varies. Generating an approximation to the convex hull is common approach, where the positions of the vertices of the approximation are used. The problem then becomes matching the vertices of the convex hull so that they can be corresponded in the following frames (Meyer & Bouthemy, 1992).

A.5 Edge Detection

For a continuous edge in an image *I(x,y)*, the derivative of the image will result in a local maximum in the direction of the edge. This means that one simple edge detection technique would be as follows: (see Figure A-3).

Measure the gradient of *I* along *r* in a direction *θ*,

$$\frac{\partial I}{\partial r} = \frac{\partial I}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial I}{\partial y}\frac{\partial y}{\partial r} = I_x \cos\theta + I_y \sin\theta \qquad \text{A-10}$$

The maximum value of $\frac{\partial I}{\partial r}$ is when $\frac{\partial}{\partial \theta}\left(\frac{\partial I}{\partial r}\right) = 0$

.

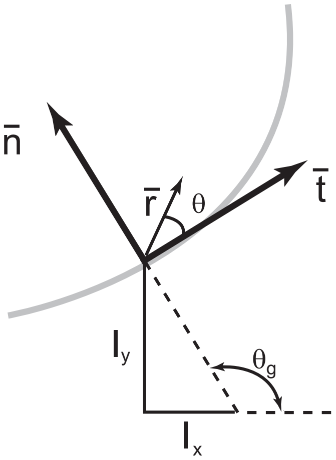


Figure A-3. Edge detection.

Thus $-I_x \sin\theta_g + I_y \cos\theta_g = 0$ A-11

$$\Rightarrow \theta_g = \tan^{-1}\left(\frac{I_x}{I_y}\right) \text{ and } \left(\frac{\partial I}{\partial r}\right)_{MAX} = \sqrt{I_x^2 + I_y^2} \quad \text{A-12}$$

Where θ_g is the direction of the edge and $\partial I/\partial r$ is the directional gradient. Based on these concepts the following two types of detection operators are introduced, gradient operators and compass operators.

A.5.1Gradient Operators

For the gradient operators a pair of masks is used, **H**$_1$ and **H**$_2$, which measure the gradient in two orthogonal directions. These *bi-directional* gradients are defined as $g_1(m,n)\underline{\underline{\Delta}}\langle\mathbf{U},\mathbf{H}_1\rangle_{m,n}$[1] and $g_2(m,n)\underline{\underline{\Delta}}\langle\mathbf{U},\mathbf{H}_2\rangle_{m,n}$ thus the gradient vector magnitude and direction are given by:

$$g(m,n) = \sqrt{g_1^2(m,n) + g_2^2(m,n)} \text{ or equivalently} \quad \text{A-13}$$

$$g(m,n)\underline{\underline{\Delta}}|g_1(m,n)| + |g_2(m,n)| \quad \text{A-14}$$

[1] For an image ***U*** the inner product at the location *(m,n)* is given as the correlation

$$\langle\mathbf{U},\mathbf{H}\rangle_{m,n}\underline{\underline{\Delta}}\sum_i\sum_j h(i,j)u(i+m,j+n) = u(m,n)*h(-m,-n)$$

where ***H*** is a $p\times p$ mask

.

$$\text{and } \theta_g(m,n) = \tan^{-1}\frac{g_2(m,n)}{g_1(m,n)} \qquad \text{A-15}$$

The table that follows has some of the more commonly used masks:

	H1	H2		H1	H2
Roberts	$\begin{bmatrix} \mathbf{0} & 1 \\ -1 & 0 \end{bmatrix}$ bolded figure is the position of the origin	$\begin{bmatrix} \mathbf{1} & 0 \\ 0 & -1 \end{bmatrix}$	Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Prewitt	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 \end{bmatrix}$	Isotropic	$\begin{bmatrix} -1 & 0 & 1 \\ -\sqrt{2} & \mathbf{0} & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$

Table A-1. Commonly used edge detection masks.

The pixel location is an edge point if *g(m,n)* exceeds a threshold *t*. Thus the edge map *ε(m,n)* is given by:

$$\varepsilon(m,n) = \begin{cases} 1, & (m,n) \in I_g \\ 0, & \text{otherwise} \end{cases} \quad \text{where } I_g \underline{\underline{\Delta}} \{(m,n); g(m,n) > t\} \qquad \text{A-16}$$

The value *t* is commonly set from the cumulative histogram of *g(m,n)*, so that typically 5-10% of the pixels with the largest gradient are used. Nerriec *et al* (1994) apply a *re-writing rules* based approach at this stage, the idea of which is to reinforce the continuous edges and corners, while removing noisy regions in which there are many changes in edge direction. They define rule types of the form:

- Type-1 rules: eliminate non-realistic configurations, mainly in the intersection of two edges.
- Type-2 rules: remove double edges, detected by several directions of analysis.
- Type-3 rules: improve quality of detections, e.g. by closing edges in a 5x5 neighbourhood and removing edges where the length is below a predefined threshold.

Rosenthaler *et al* (1992) and Bouthemy (1989) use very similar methods.

.

The Laplacian is an edge detection operator that is an approximation to the mathematical Laplacian $\nabla^2 f = \partial^2 f / \partial x^2 + \partial^2 f / \partial y^2$, in the same way as the gradient is an approximation to the first partial derivatives. One simple version for the discrete version of the Laplacian is given by:

$$\mathbf{L}(x,y) = f(x,y) - \frac{1}{4}\left[f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) \right] \qquad \text{A-17}$$

The Laplacian has however some disadvantages:

- There is no useful directional information.
- Since it is a second order derivative, it enhances the high frequency noise in the image (even more than the gradient method).
- The thresholded magnitude of the Laplacian also produces double edges.

For these reasons, the Laplacian is not commonly used. Probably the most commonly used and most documented edge detection algorithm is the Canny edge detector (Canny, 1986). Many Canny implementations convolve the image with a Gaussian filter, to smooth it and then search for maxima in the first partial derivatives of the resulting signal. The image is then convolved with 4 masks, each looking for horizontal, vertical and diagonal edges. Whichever mask provides the largest result at each pixel is marked and the direction is recorded. Non-maximal suppression is performed and any gradient value that is not a local peak is set to zero. Connected sets of points are sorted into lists and then hysteresis thresholding is performed to eliminate weak edges and retain connected edge points. The Canny detector fails at junctions and sharp corners and the hysteresis thresholding fails at dense edges or low noise as it is weighted to either the top or bottom of the scale.

A.5.2 Compass Operators

Compass operators measure gradients in a selected number of directions. In the case of the Kirsch operator,

$$\begin{bmatrix} 5 & 5 & 5 \\ -3 & \mathbf{0} & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

represents the north-going edge. Each of the directions is extracted by rotating the values in the mask clockwise through 45°. To apply this method, let $g_k(m,n)$ denote the

.

compass gradient in the direction θ_k. So starting at north and then rotating by *k* times 45°, where *k=0,1,…,7*. Thus the gradient at *(m,n)* may be given by:

$$g(m,n) \underline{\underline{\Delta}} \max_k \left\{ \left| g_k(m,n) \right| \right\} \qquad \text{A-18}$$

and thus the direction is given by $\theta_k = \frac{\pi}{2} + k\frac{\pi}{4}$, where *k* is a maximum.

A.6 Sequences of Images

The task of estimating 3-D motion parameters and the structure of a scene from a long image sequence is divided into three main subtasks (Cédras *et al*, 1995):

- The first subtask is to establish a feature correspondence between two images at different time instants.
- The second is to compute the structure of the scene and determine the inter-frame motion parameters.
- The third subtask is to perform motion and structure estimation from a sequence of many views taken at many time instants, which allows the repeated observations of the same part of the scene.

Knowledge about the object dynamics and thus temporal coherence of motion can be used further constrain the estimation processes.

A.7 Kalman Filtering

Kalman filtering is a method for computing weighted least-squares solution for a dynamic system. It is a *sequential* method in the sense that observed data is sequentially fed into the algorithm and new estimates are calculated from the previous estimates and the current observation. A *batch* method on the other hand is where all observations are processed together in a batch fashion and estimates are determined directly from all observations. In the dynamic system that is required for this approach the batch method cannot be used, as information must be available at each frame. See Saotta *et al* (1996) for information on dynamic systems.

The recursive Kalman Filtering approach has some desirable properties over the batch approach:

- Firstly all old observations are discarded once they have been used for estimation. This is very useful if the number of estimates is ‘infinite’, as the batch

.

method would have to operate on an 'infinite' number of values, being both slow and requiring 'infinite' storage.

- Secondly the Kalman approach is efficient as computation with only a small number of estimates is required.
- Thirdly, the next estimate may be calculated at the time of observation, instead of in the batch method of having to wait for all data to be collected before receiving estimates.

For a more complete description and a derivation of the Kalman filter see Jain (1989) and Weng *et al* (1993).

However, sequential methods have relatively poor performance for non-linear problems. Early observations are especially inaccurate since the estimated system matrices (Jacobian matrices) are not updated initially, meaning that parameter values close to the initial guess are used. These inaccurate estimates require several frames before they begin to converge to the actual state trajectory. This accounts for the Kalman approach, in general requiring 20 to 40 frames before it converges to acceptable solution. This is not a problem in the linear case as the system Jacobian matrix is constant. In general by properly defining the states and observations, it is possible to convert the non-linear Kalman filtering problem into a linear Kalman filtering problem, dramatically improving the performance of the algorithm. See Tesei & Regazzoni (1994).

Weng *et al* (1993) developed a *recursive-batch* approach that achieves good performance without suffering from excessive computational cost. This is achieved by using relatively small batch sizes. This approach is recursive since it estimates, based on the current batch of data and the previous estimate, where the processing algorithm uses a certain length (batch size) of the sequence. This method has the advantages:

- It can process virtually infinitely long image sequences with a limited physical memory.
- The algorithm is efficient since only a relatively small amount of computation is required to update the estimates.
- The algorithm is claimed to out-perform the Kalman approach, as the data is calculated in a batch fashion in which each overlapping batch sufficiently covers the interaction among data.

.

The algorithm of Weng *et al* is a straight combination of the recursive and batch approach. If an infinite batch length is used in the recursive-batch approach then it becomes the batch approach. On the other hand if a batch size of 1 is used then the recursive-batch approach becomes the recursive approach. The choice of a good batch size is thus vital to the recursive-batch approach.

A.8 Motion Events and Image Sequences

In general image sequences are captured at a constant frame interval, but not necessarily so when using motion events. When using motion events it has been found that using sequences of images between motion events (i.e. with low activity & discontinuities) are experimentally better for extracting information than standard time sampled image sequences. (Cédras *et al*, 1995)

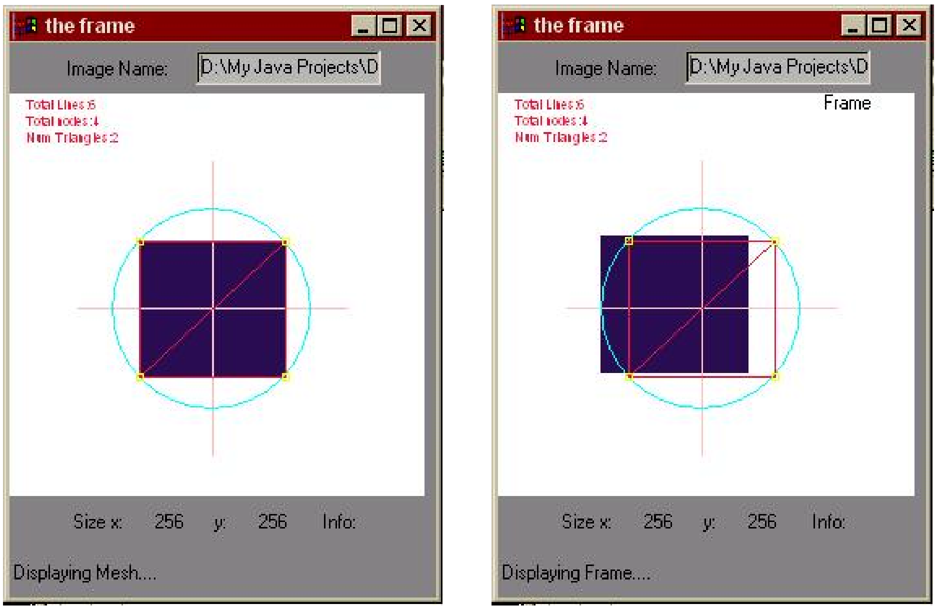
Motion events are defined as: *significant changes or discontinuities in motion*. It has been found that the changes in direction and speed of motion of an object can give significant information about an object. Events can be extracted by evaluating the *scale-space*[2] of the *x*-velocity curves and *y*-velocity curves at different scales extracted from a trajectory. The shape of the curves and other features can help distinguish the type of motion that has been undergone, e.g. translation can be classified as either straight line or curved. Motion events can also be applied to more complicated sequences, like human motion. They are widely applicable and can be used in conjunction with other types of features.

[2] Non-regular sampling time means we must transform the data to a scale-space to get meaningful results.

.

Appendix B – Implementation Issues of the Self-Adaptive Active Mesh

The initial implementation of the active mesh made no provision for allowing mesh nodes to be added or removed from the mesh, once the mesh had been created. It became clear that this facility would have to be introduced to the mesh so that it could operate over multiple image frames, where feature points may become occluded for a number of frames, or new strong feature points may become apparent that were not present in the initial frame.



the frame

Image Name: D:\My Java Projects\D

Total Lines:5
Total nodes:4
Num Triangles:2

Size x: 256 y: 256 Info:

Displaying Mesh....

the frame

Image Name: D:\My Java Projects\D

Total Lines:5
Total nodes:4
Num Triangles:2

Frame

Size x: 256 y: 256 Info:

Displaying Frame....

(a) (b)

Figure B-1. The initial frame is loaded in (a) and in (b) the new frame is loaded.

So, some of the advantages of a self-adaptive mesh would include:

- Features that became occluded in the mesh could be removed if required.
- New strong features that appeared in the sequence could be added to the mesh as new mesh nodes
- Other facilities could be implemented to add and remove nodes from the mesh – a higher-level process.
- The features go out of view of the camera could also be removed from the mesh.

.

It was decided that the mesh be constructed as an iterative Delaunay mesh. The reasoning behind this was clear. An iterative Delaunay mesh is always a Delaunay mesh, even as it is being constructed. As mentioned previously the structure of the mesh must be conserved to some degree for the combination of mesh forces and the implementation of the internal energies. Following this idea, nodes could be added or removed from the mesh as the sequence progressed and the mesh should always conform to the Delaunay principles.

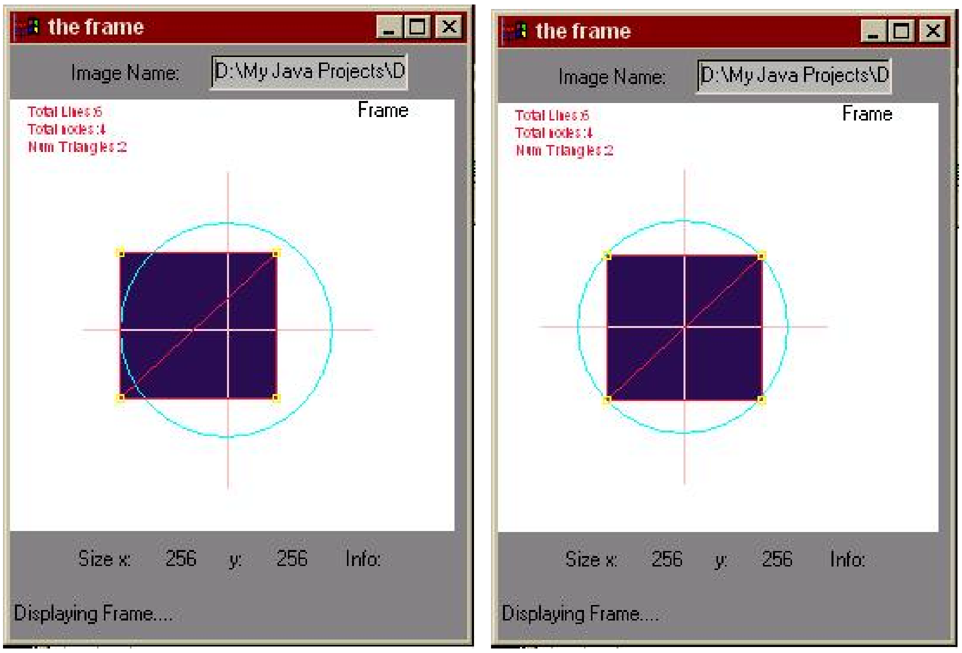


Image Name: D:\My Java Projects\D
Total Lines:6
Total nodes:4
Num Triangles:2
Size x: 256 y: 256 Info:

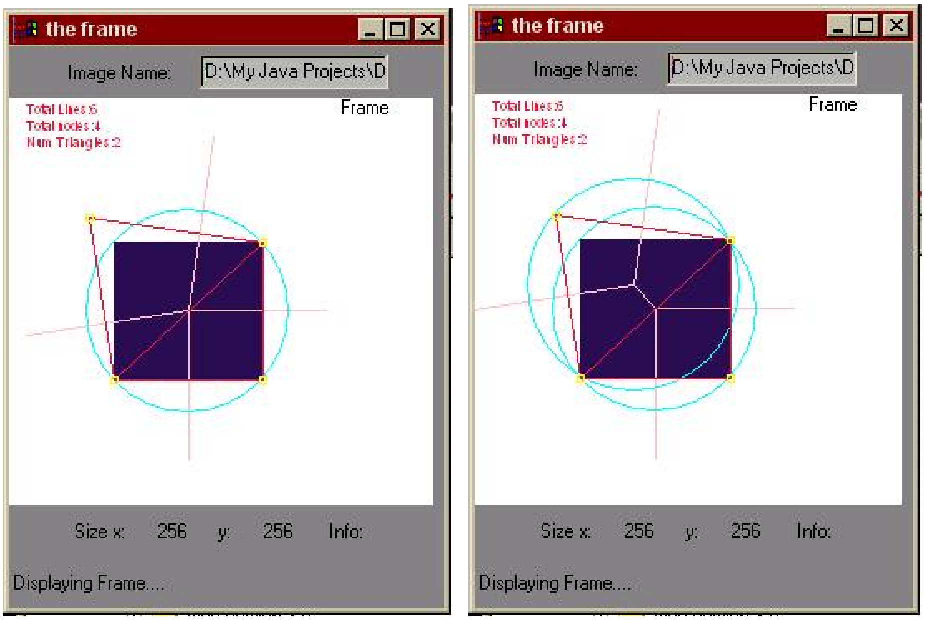
Image Name: D:\My Java Projects\D
Total Lines:6
Total nodes:4
Num Triangles:2
Size x: 256 y: 256 Info:

(a) (b)

Figure B-2. The mesh settles on a new location after 10 iterations and in (b) the Circumcircle parameters must be updated for it to remain associated with the mesh.

However the implementation evolved to be much more difficult than was first expected. The first issue was keeping the mesh up-to-date as the sequence evolved. Figure B-1(a) shows the initial frame as it is loaded and the mesh is initialised on the image. The Mesh is shown around the square, the circumcircles is shown circling the object and the Voronoi regions are shown as a ‘cross’. Figure B-1(b) shows a new frame loaded in the sequence. The Mesh remains in the same location initially and so do the Voronoi and circumcircle.

.



the frame

Image Name: D:\My Java Projects\D

Total Lines:5
Total nodes:4
Num Triangles:2

Frame

Size x: 256 y: 256 Info:

Displaying Frame....

the frame

Image Name: D:\My Java Projects\D

Total Lines:5
Total nodes:4
Num Triangles:2

Frame

Size x: 256 y: 256 Info:

Displaying Frame....

Figure B-3. Top left node of the mesh moved in (a) and the new mesh update performed to update the circumcircles and the Voronoi diagram as in (b).

Figure B-2(a) shows what occurs when the mesh moves from one frame to the next after the mesh has been initialised in the first frame. The mesh location updates to the new position of the object, but the Delaunay circumcircles stays stationary in the initial location at which it was initialised. It was necessary to update the location of the circumcircles to maintain the Delaunay structure of the mesh for the addition or removal of future points, as shown in Figure B-2 (b). The addition of code to update the circumcircles in the algorithm solved many problems with the mesh becoming unstable during the addition of mesh nodes, but problems still resulted from time to time as described below.

In Figure B-3 a force is applied to the top left node of the mesh. This force pulls the node from being on the circumcircles of the other three points. The mesh is no longer correct as discussed previously and the update algorithm is performed giving the image as in Figure B-3 (b). As can be seen, the Voronoi diagram and the circumcircles have been corrected to reflect the changes in the mesh. In Figure B-4(a) the problem case is starting to emerge. When the node is now forced in towards the centre of the mesh, the Voronoi diagram can be seen to be visibly incorrect. The update algorithm was developed to solve this problem and the resulting mesh is shown as in Figure B-4(b). Note that the mesh diagonal line has been swapped to conform to the min-max angle criterion. The mesh is now in a correct form.

Figure B-5 shows the problem case re-occurring if the top-left mesh node is once again pulled away from the mesh. The mesh lines must once again be swapped. However in (b) not the problems that can occur if a mesh node is added to the triangulation inside the second Voronoi region after the mesh has been pulled, before the update. This mesh node causes a mesh to be created that is a non-Delaunay mesh, certainly leading to problems in later stages.

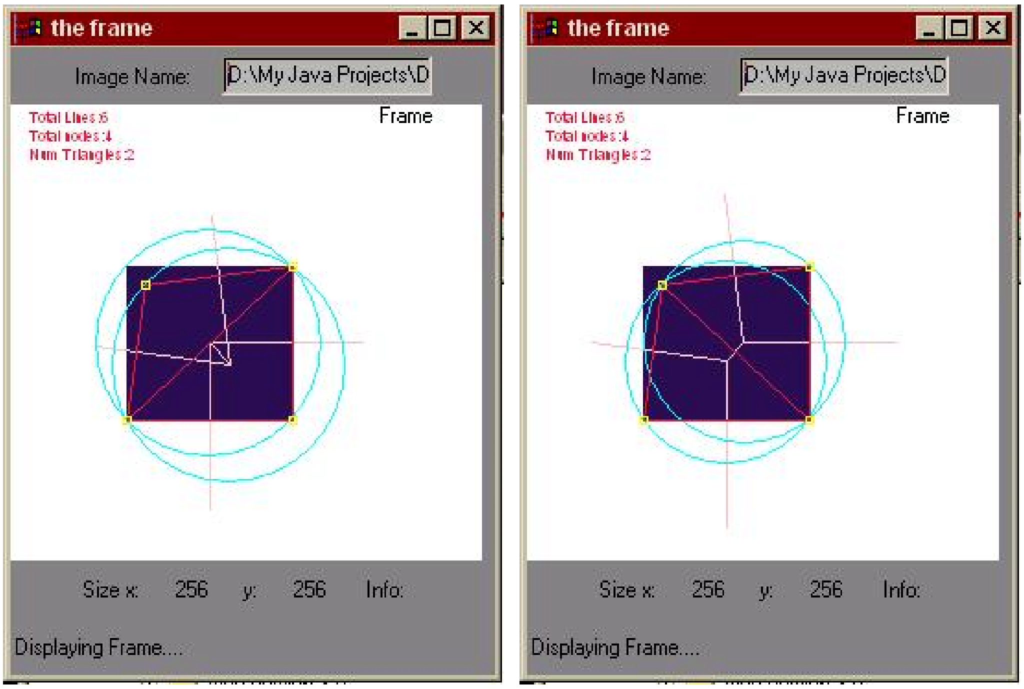


Image Name: D:\My Java Projects\D

Total Lines:5
Total nodes:4
Num Triangles:2

Size x: 256 y: 256 Info:

Displaying Frame....

Image Name: D:\My Java Projects\D

Total Lines:5
Total nodes:4
Num Triangles:2

Size x: 256 y: 256 Info:

Displaying Frame....

Figure B-4. Problem case starting to occur as in (a) and in (b) the update algorithm swaps the vertices to account for this problem. The mesh is correct again.

These problems start to emerge in Figure B-6 (a) when the top left node is removed from the mesh. Once this node is removed the centre node is connected to the top right corner of the mesh, but once the node is added again to the mesh the connection is established again to the that node. The mesh is not performing as a Delaunay mesh and in Figure B-7 the mesh eventually breaks down and cannot be used in the Active mesh algorithm, as there no longer exists a triangular structure that is required. The centre node has been removed from the mesh, but does not remove completely, leading to a single line connection existing without an associated triangle.

.

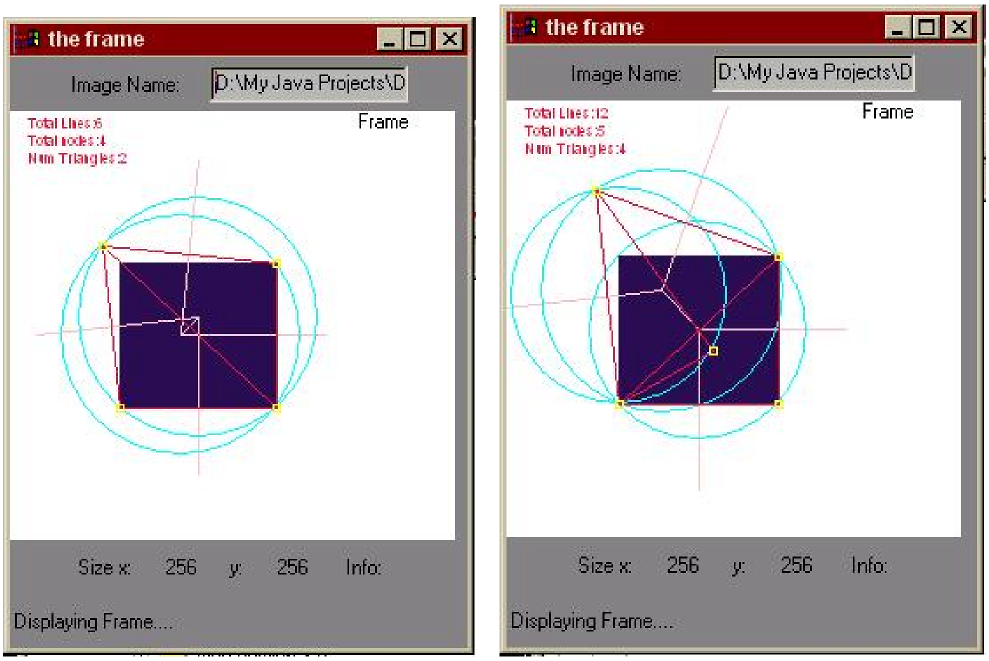


Image Name: D:\My Java Projects\D

Total Lines:6
Total nodes:4
Num Triangles:2

Size x: 256 y: 256 Info:

Displaying Frame....

Image Name: D:\My Java Projects\D

Total Lines:12
Total nodes:5
Num Triangles:4

Size x: 256 y: 256 Info:

Displaying Frame....

Figure B-5. If the node is once again pulled then the problem case re-occurs and the vertices must be swapped again. In (b) however a node is added to the mesh, the mesh is not conforming to the Delaunay principles and the mesh is incorrect.

Problems within the mesh were difficult to determine on the update algorithm and difficult to fix at this point. The update algorithm was therefore re-written to delete and then re-construct the part of the mesh that was estimated to be problematic.

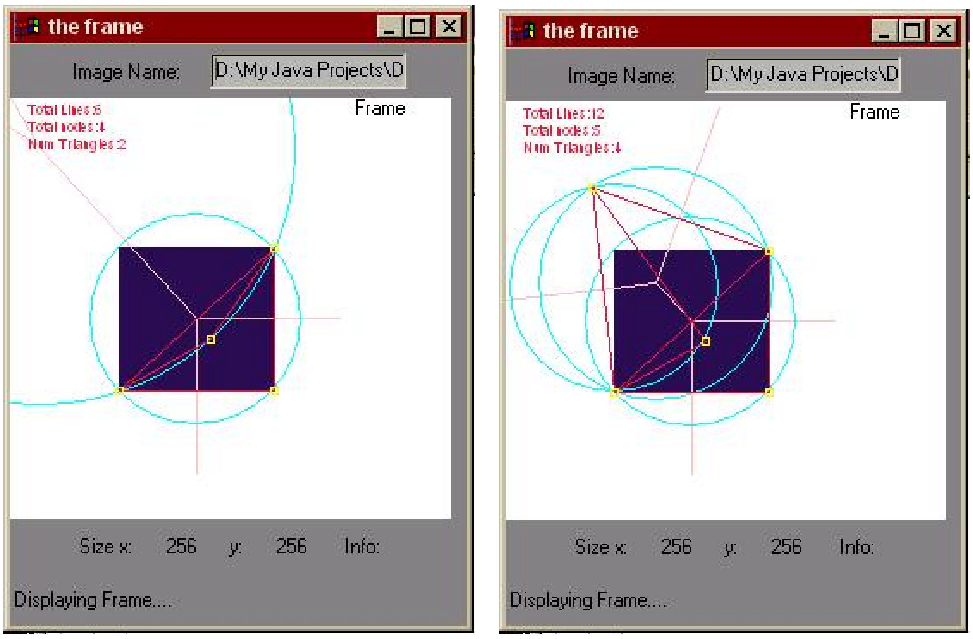


Image Name: D:\My Java Projects\D

Total Lines:6
Total nodes:4
Num Triangles:2

Size x: 256 y: 256 Info:

Displaying Frame....

Image Name: D:\My Java Projects\D

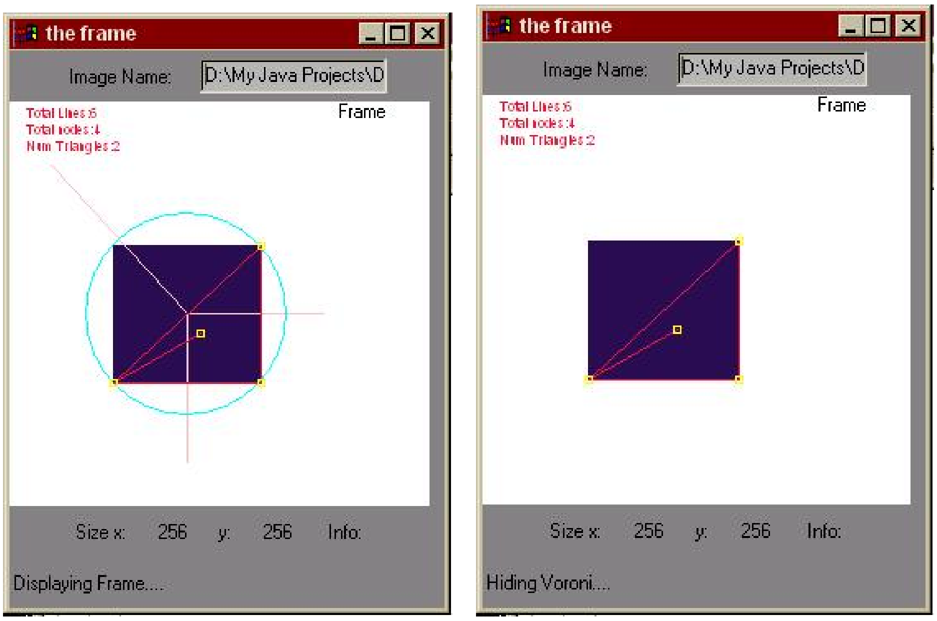
Total Lines:12
Total nodes:5
Num Triangles:4

Size x: 256 y: 256 Info:

Displaying Frame....

Figure B-6. In (a) if the top left node is deleted and in (b) if it is once again added to the mesh.

.



D:\My Java Projects\D

Total Lines :5
Total nodes :4
Num Triangles :2

256 256

Displaying Frame....

D:\My Java Projects\D

Total Lines :5
Total nodes :4
Num Triangles :2

256 256

Hiding Voroni....

Figure B-7. In (a) serious problems occurring in the mesh when the centre node is removed from the mesh and in (b) the same image just displaying the mesh.

Appendix C – Full Image Result Sets.

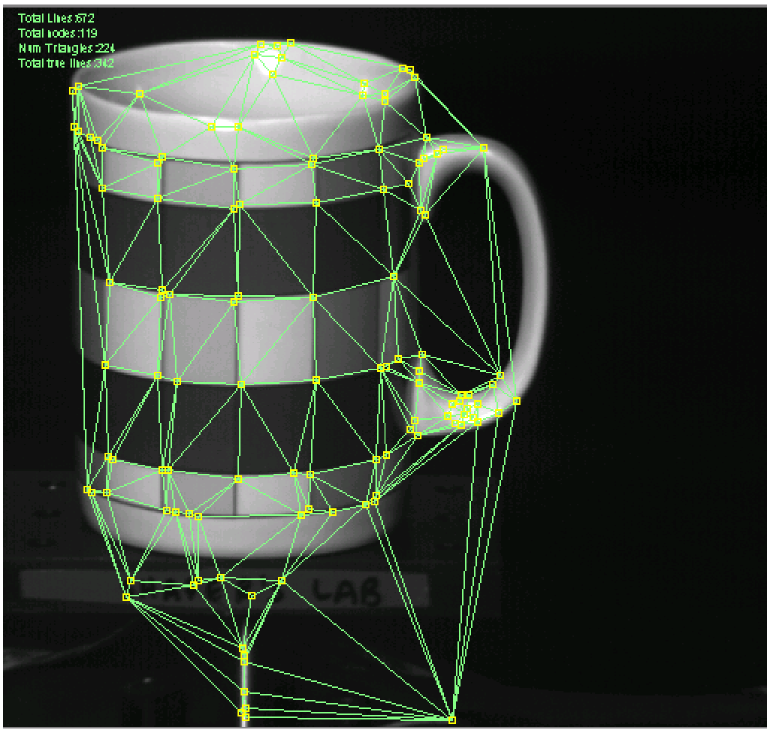


Figure C-1. The mug sequence Frame 1 the initialised mesh.

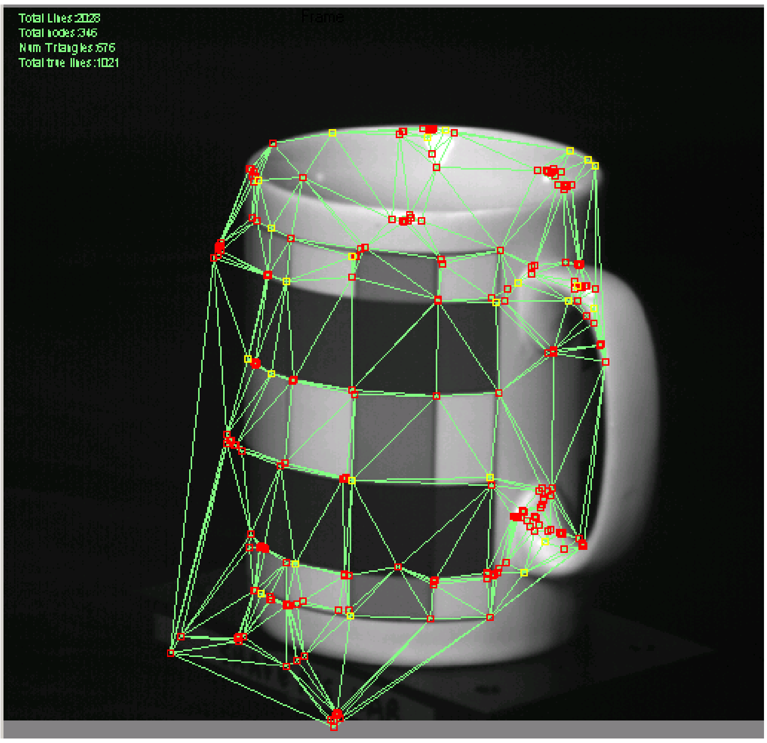


Figure C-2. The mug sequence Frame 12 the mesh has followed the mug.

.

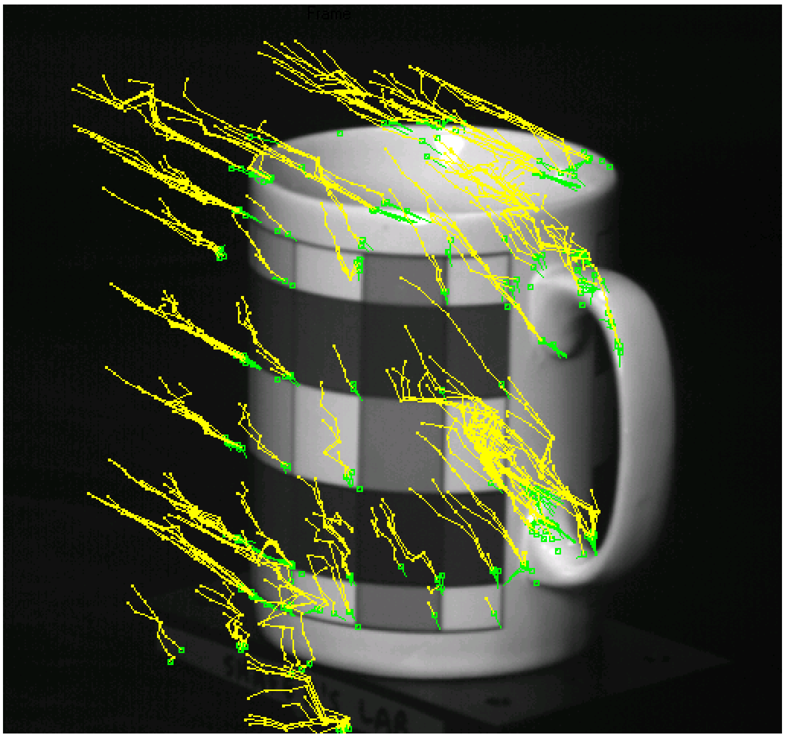


Figure C-3. The mug sequence vector tracked over 12 frames

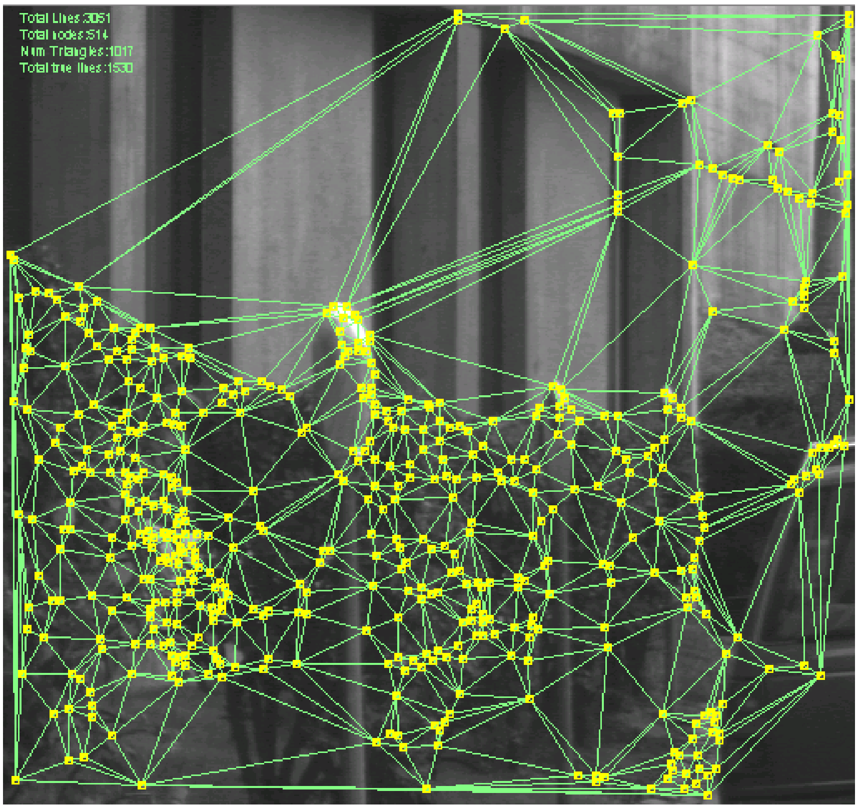


Figure C-4. The parking sequence showing the mesh initialised on Frame 1.

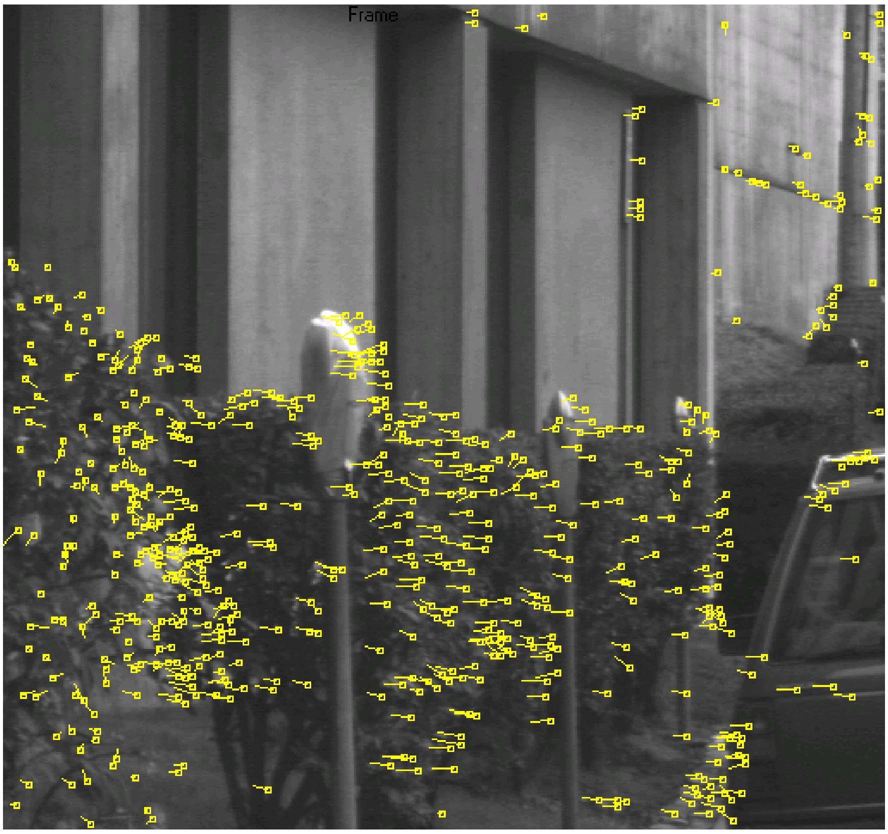


Figure C-5. The parking sequence has 12 frames. This result was calculated using only the first and last image frame.

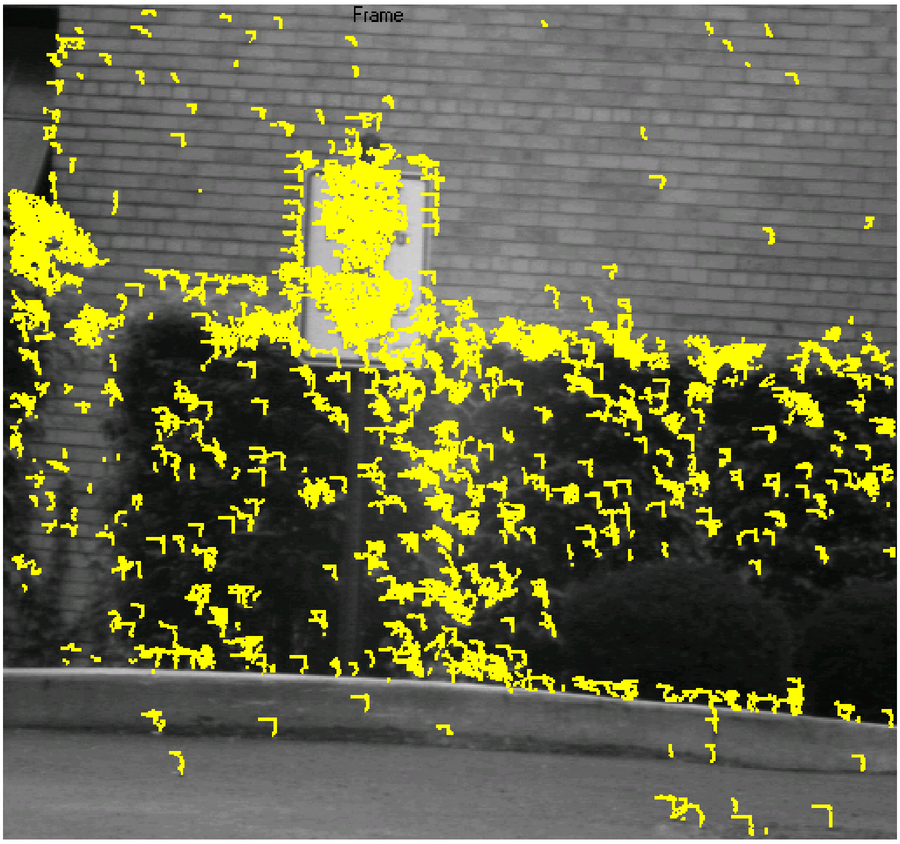


Figure C-6. Shrubbery Sequence using all 24 frames.

.

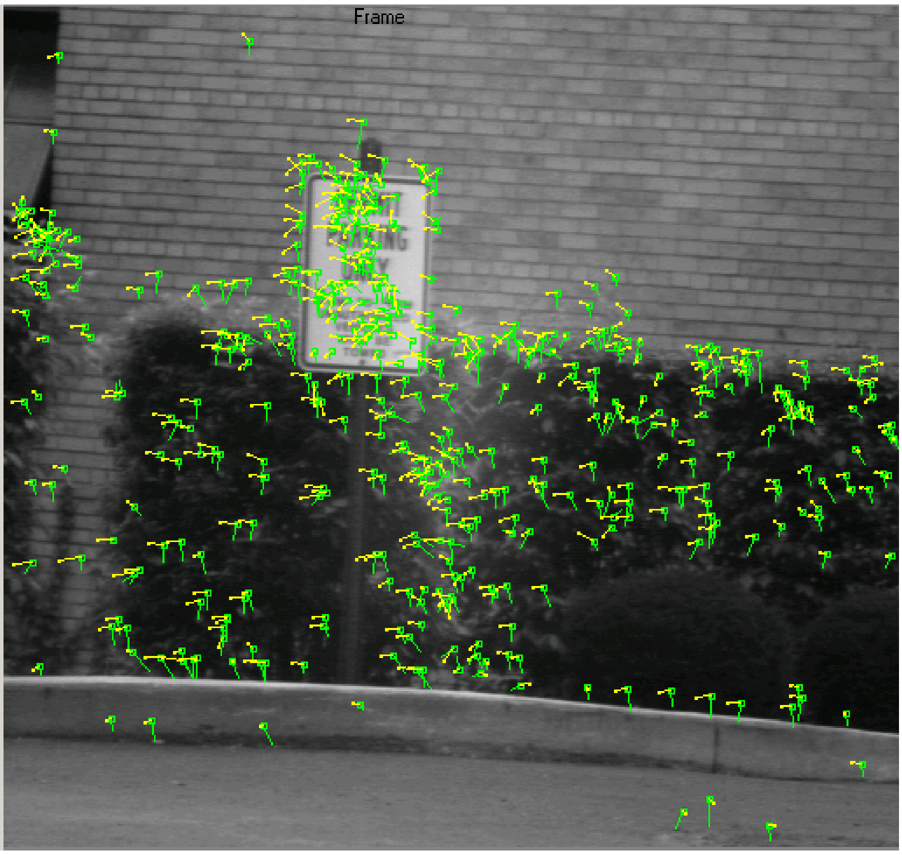


Figure C-7. Shrubbery Sequence using only first, middle and last Frames.

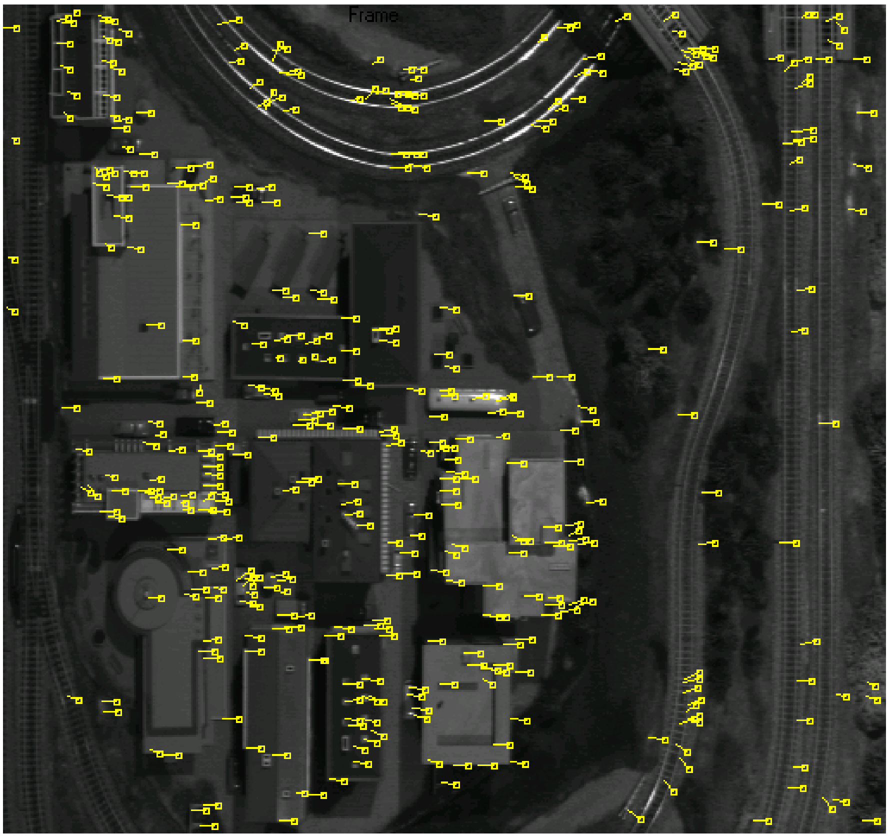


Figure C-8. Town Sequence (6 frames) using only the first and last frames

.

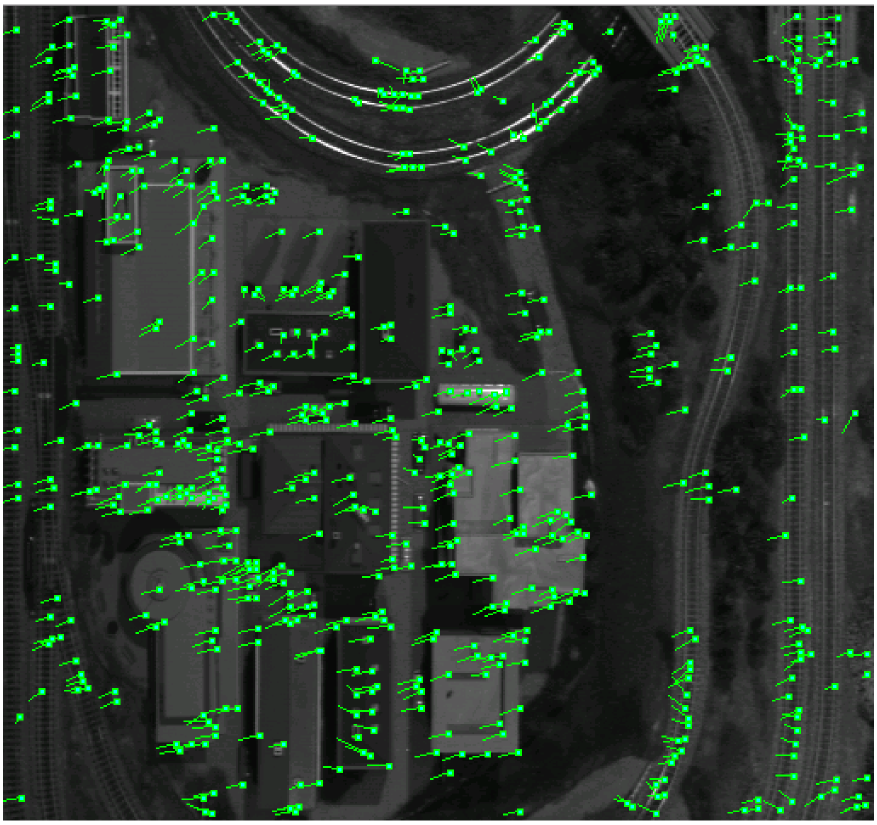


Figure C-9. Another 6 frames from the town sequence, again skipping 4 frames, using only the first and last frames.

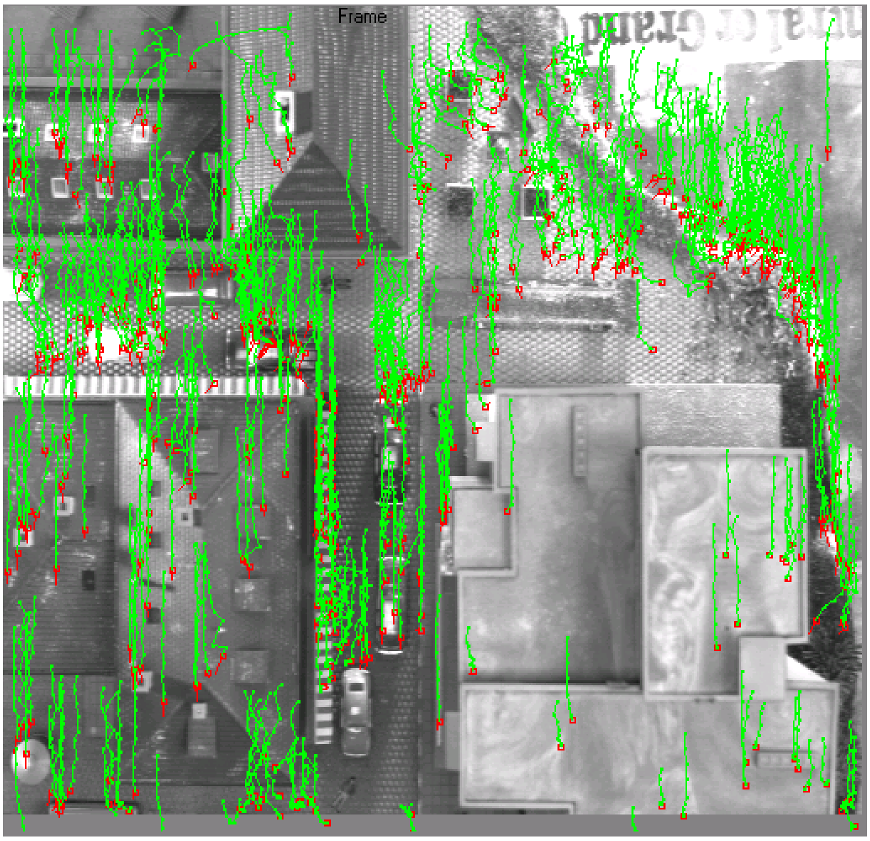


Figure C-10. A building sequence using 20 frames.

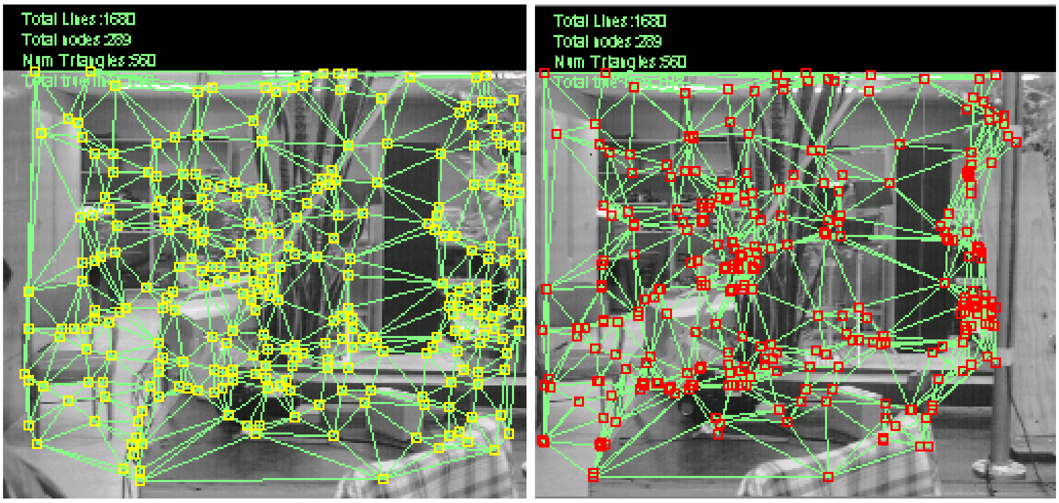


Figure C-11. The SRI Laboratory Sequence. In (a) the start frame and the mesh and in (b) the end frame and mesh

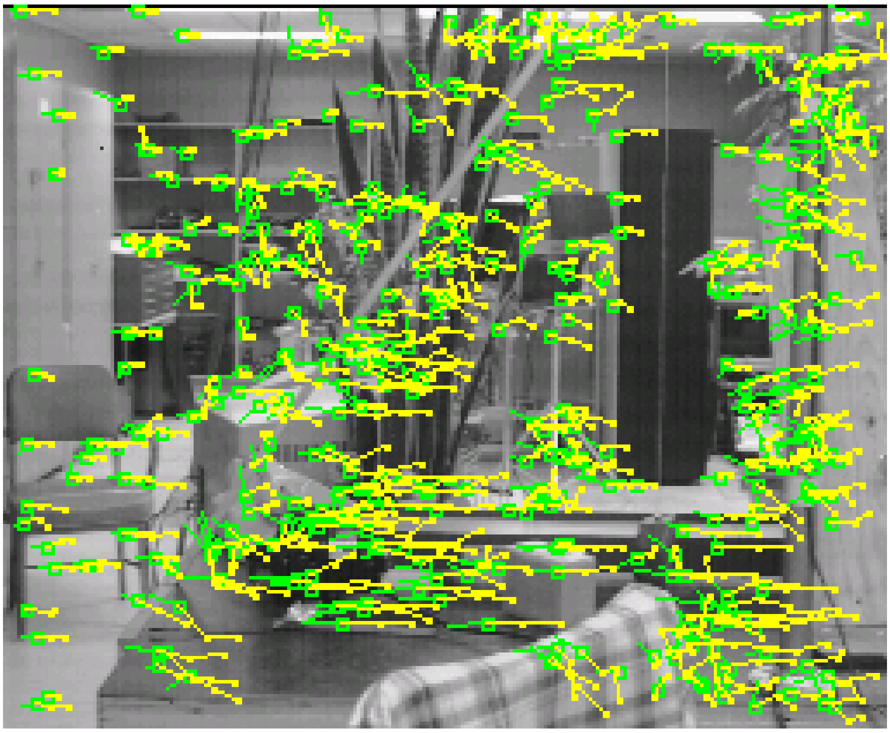


Figure C-12. The SRI Laboratory Sequence resulting vectors. Note that the vector paths are longer in the foreground than in the background parts of the image.

.

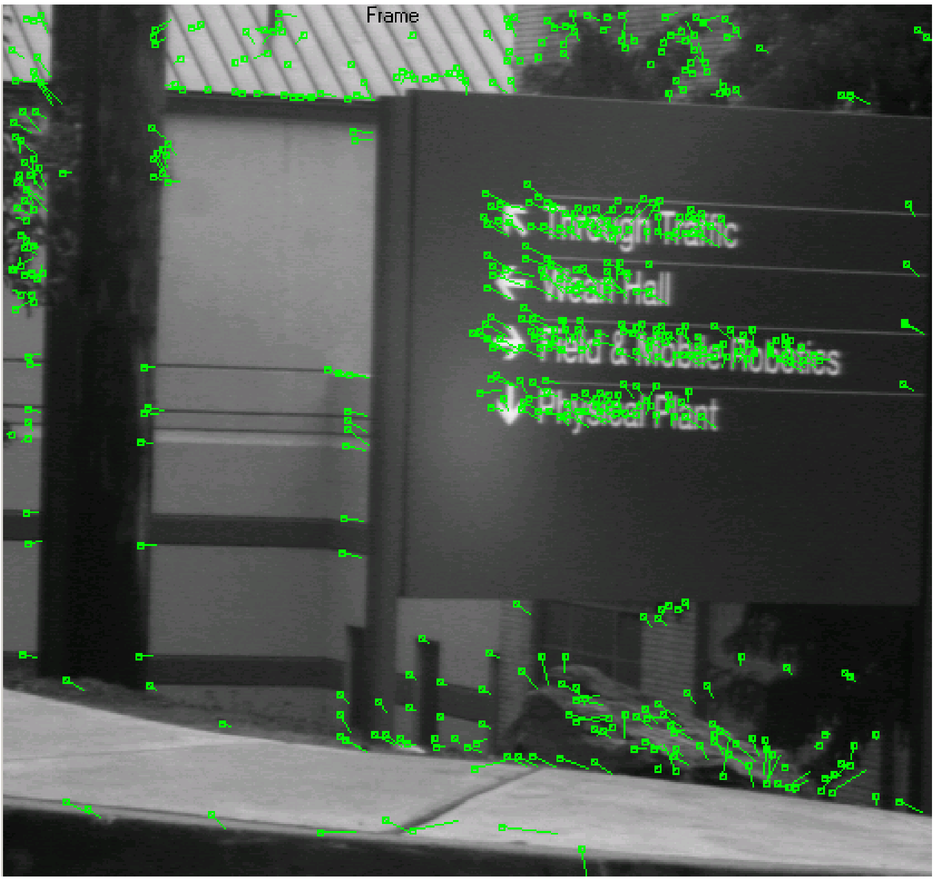


Figure C-13. The College sequence results calculated using only the first and 20th frame.

Appendix D - Algorithm Class Diagrams

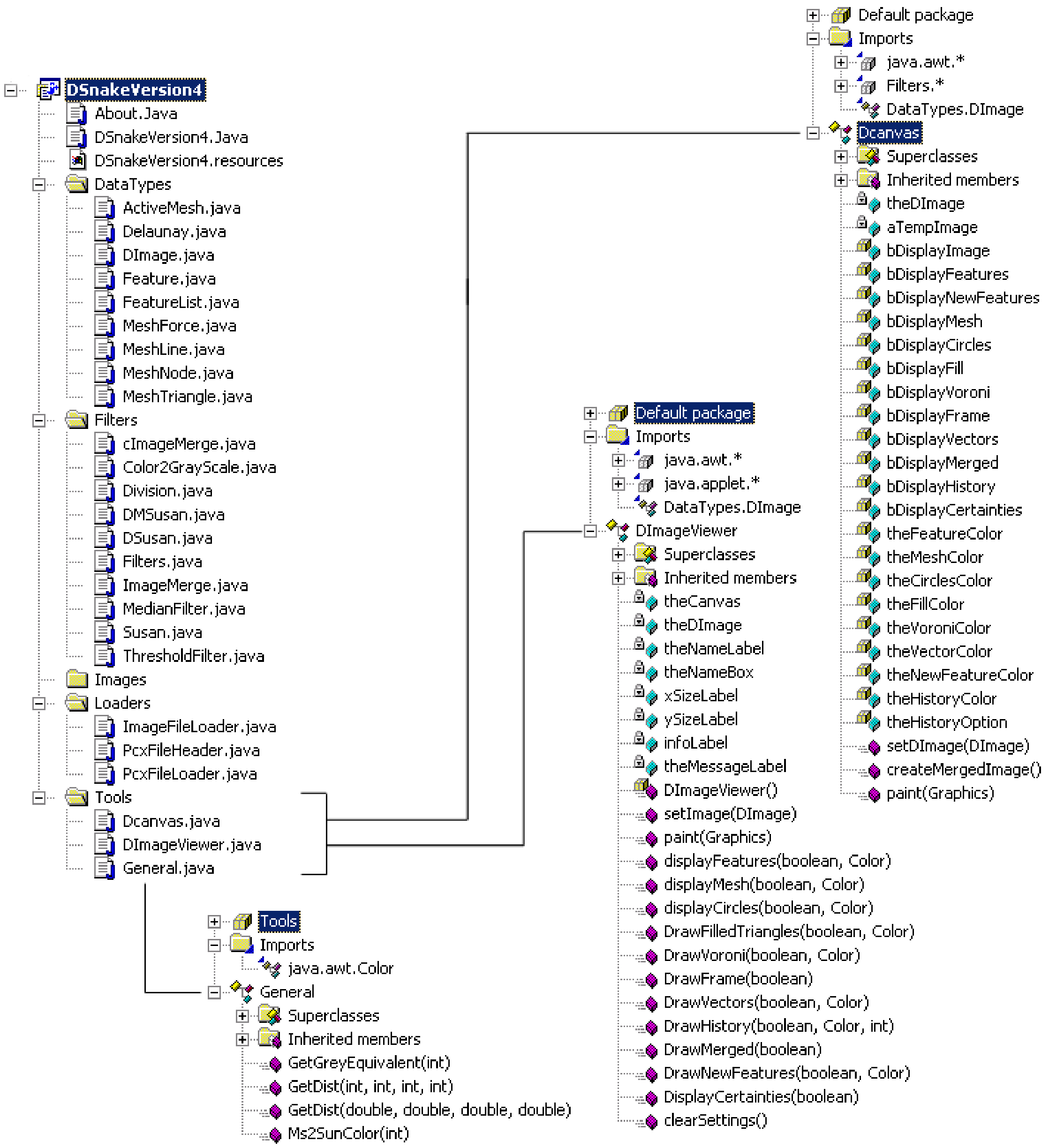


Figure D-1. The main structure and some of the tools classes.

.

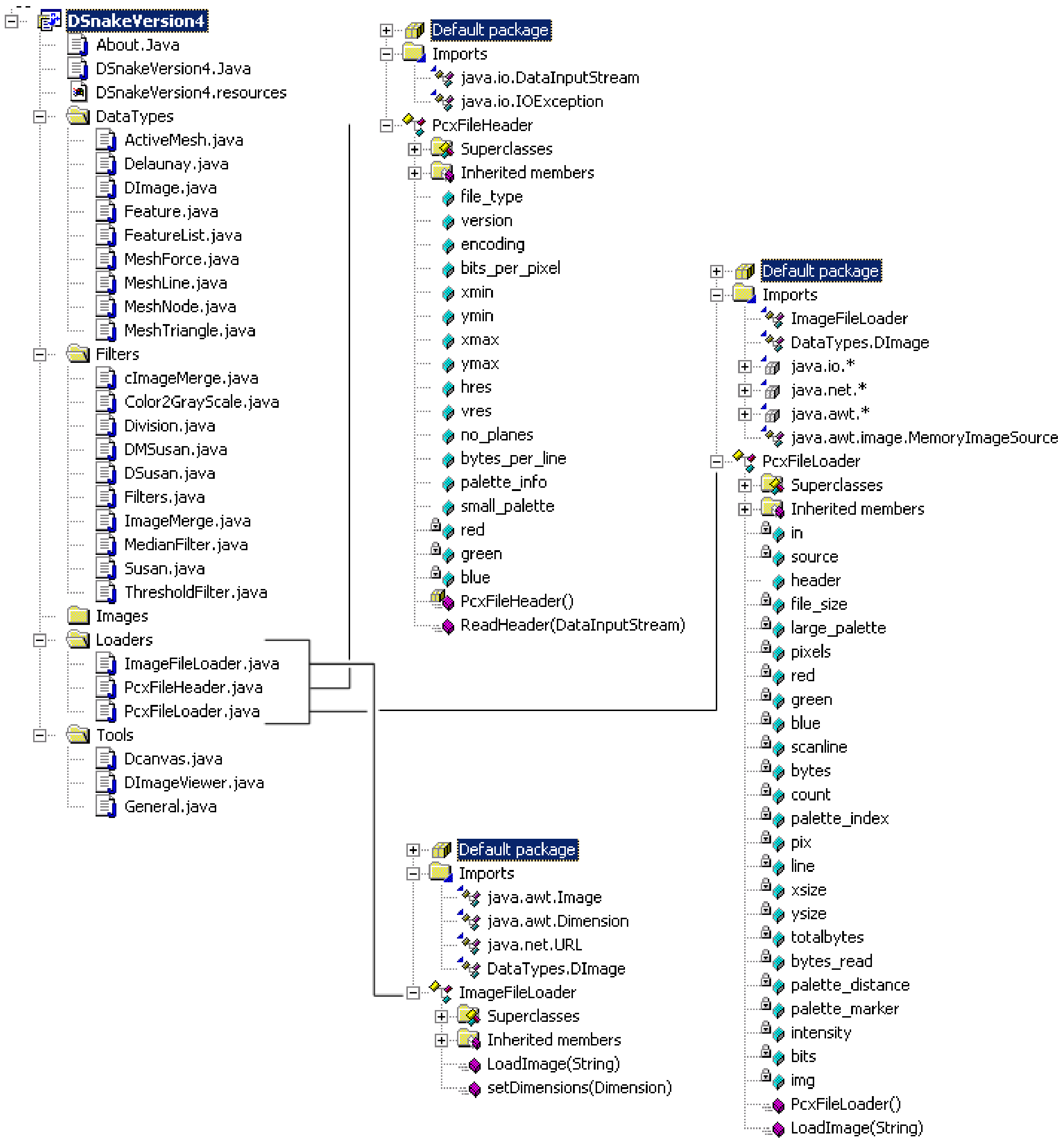
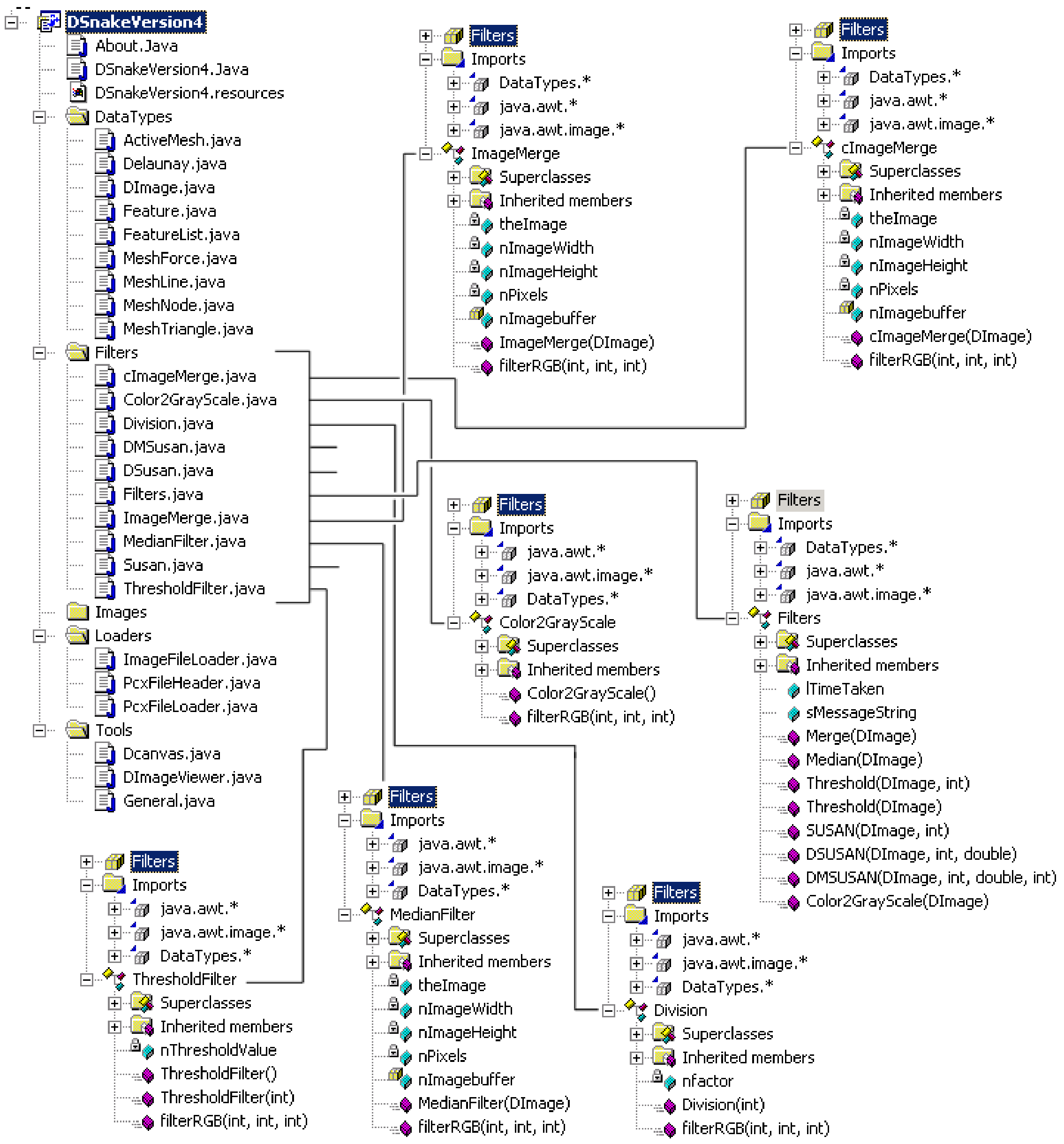


Figure D-2. The main structure and some of the loaders classes.



```
DSnakeVersion4
    About.Java
    DSnakeVersion4.Java
    DSnakeVersion4.resources
    DataTypes
        ActiveMesh.java
        Delaunay.java
        DImage.java
        Feature.java
        FeatureList.java
        MeshForce.java
        MeshLine.java
        MeshNode.java
        MeshTriangle.java
    Filters
        cImageMerge.java
        Color2GrayScale.java
        Division.java
        DMSusan.java
        DSusan.java
        Filters.java
        ImageMerge.java
        MedianFilter.java
        Susan.java
        ThresholdFilter.java
    Images
    Loaders
        ImageFileLoader.java
        PcxFileHeader.java
        PcxFileLoader.java
    Tools
        Dcanvas.java
        DImageViewer.java
        General.java
```

```
Filters
Imports
    DataTypes.*
    java.awt.*
    java.awt.image.*
ImageMerge
    Superclasses
    Inherited members
    theImage
    nImageWidth
    nImageHeight
    nPixels
    nImagebuffer
    ImageMerge(DImage)
    filterRGB(int, int, int)
```

```
Filters
Imports
    DataTypes.*
    java.awt.*
    java.awt.image.*
cImageMerge
    Superclasses
    Inherited members
    theImage
    nImageWidth
    nImageHeight
    nPixels
    nImagebuffer
    cImageMerge(DImage)
    filterRGB(int, int, int)
```

```
Filters
Imports
    java.awt.*
    java.awt.image.*
    DataTypes.*
Color2GrayScale
    Superclasses
    Inherited members
    Color2GrayScale()
    filterRGB(int, int, int)
```

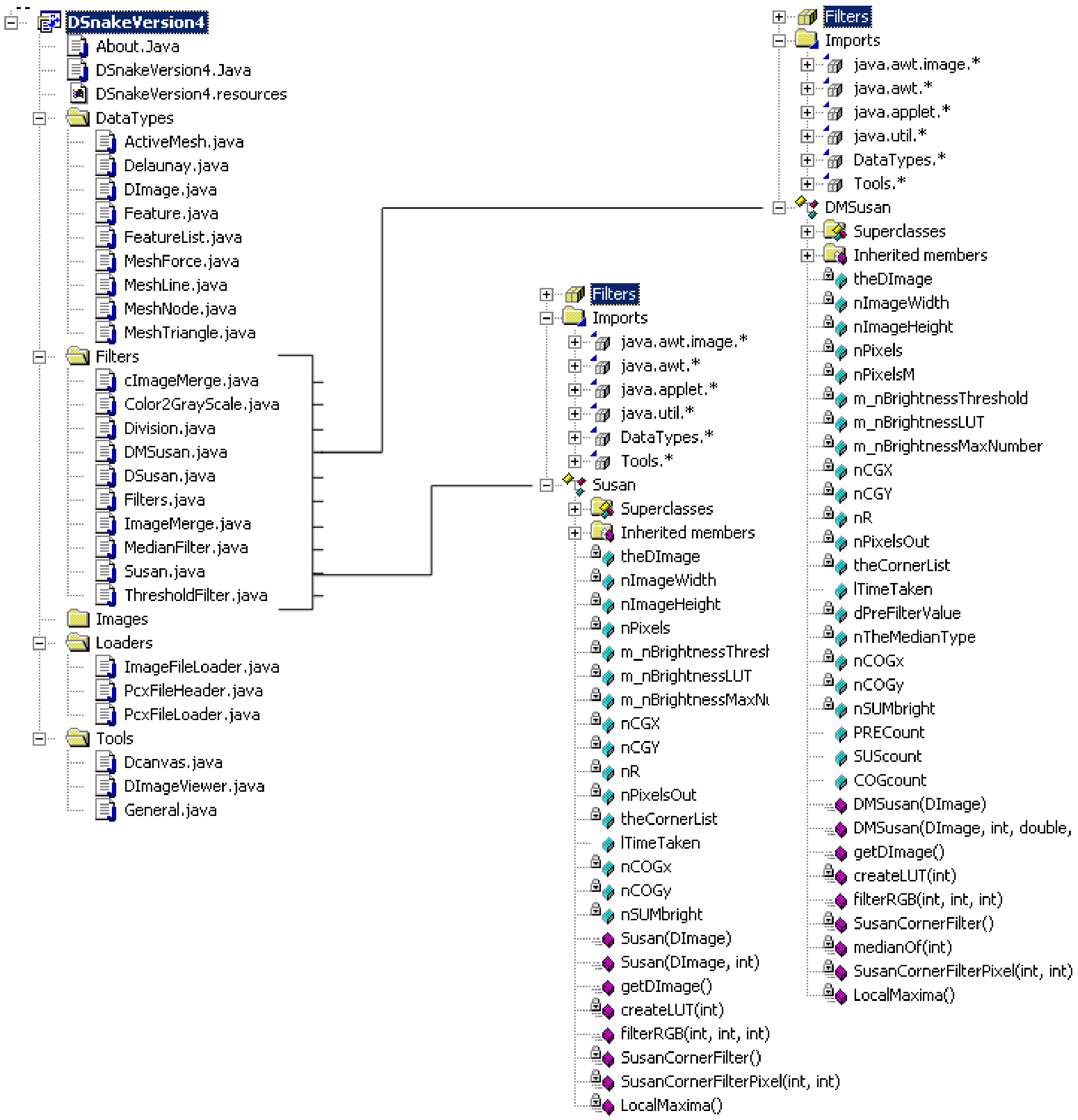
```
Filters
Imports
    DataTypes.*
    java.awt.*
    java.awt.image.*
Filters
    Superclasses
    Inherited members
    lTimeTaken
    sMessageString
    Merge(DImage)
    Median(DImage)
    Threshold(DImage, int)
    Threshold(DImage)
    SUSAN(DImage, int)
    DSUSAN(DImage, int, double)
    DMSUSAN(DImage, int, double, int)
    Color2GrayScale(DImage)
```

```
Filters
Imports
    java.awt.*
    java.awt.image.*
    DataTypes.*
ThresholdFilter
    Superclasses
    Inherited members
    nThresholdValue
    ThresholdFilter()
    ThresholdFilter(int)
    filterRGB(int, int, int)
```

```
Filters
Imports
    java.awt.*
    java.awt.image.*
    DataTypes.*
MedianFilter
    Superclasses
    Inherited members
    theImage
    nImageWidth
    nImageHeight
    nPixels
    nImagebuffer
    MedianFilter(DImage)
    filterRGB(int, int, int)
```

```
Filters
Imports
    java.awt.*
    java.awt.image.*
    DataTypes.*
Division
    Superclasses
    Inherited members
    nfactor
    Division(int)
    filterRGB(int, int, int)
```

Figure D-3. The main structure and some of the filter classes.



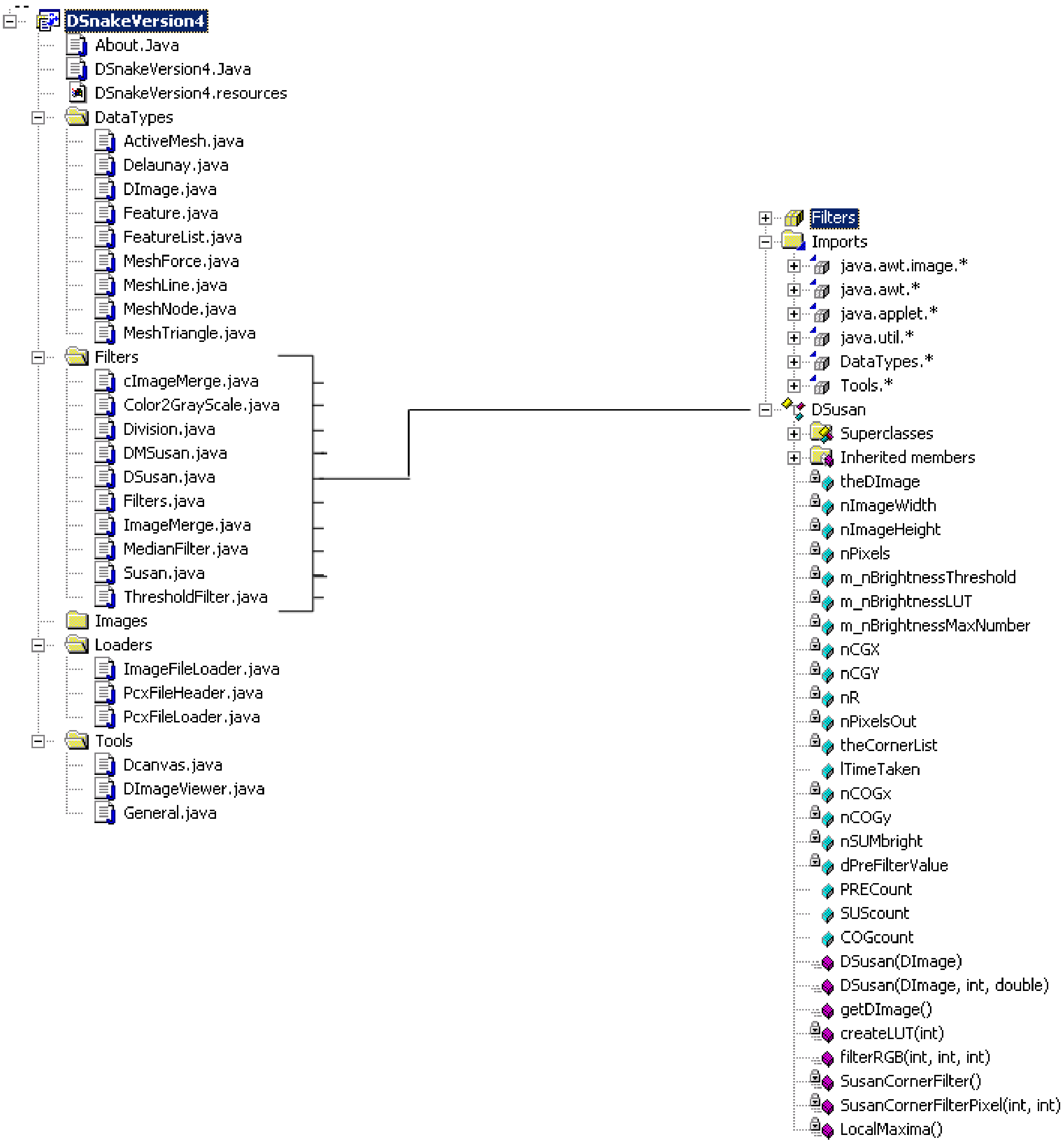
```
DSnakeVersion4
    About.Java
    DSnakeVersion4.Java
    DSnakeVersion4.resources
    DataTypes
        ActiveMesh.java
        Delaunay.java
        DImage.java
        Feature.java
        FeatureList.java
        MeshForce.java
        MeshLine.java
        MeshNode.java
        MeshTriangle.java
    Filters
        cImageMerge.java
        Color2GrayScale.java
        Division.java
        DMSusan.java
        DSusan.java
        Filters.java
        ImageMerge.java
        MedianFilter.java
        Susan.java
        ThresholdFilter.java
    Images
    Loaders
        ImageFileLoader.java
        PcxFileHeader.java
        PcxFileLoader.java
    Tools
        Dcanvas.java
        DImageViewer.java
        General.java
```

```
Filters
    Imports
        java.awt.image.*
        java.awt.*
        java.applet.*
        java.util.*
        DataTypes.*
        Tools.*
    Susan
        Superclasses
        Inherited members
        theDImage
        nImageWidth
        nImageHeight
        nPixels
        m_nBrightnessThresl
        m_nBrightnessLUT
        m_nBrightnessMaxNu
        nCGX
        nCGY
        nR
        nPixelsOut
        theCornerList
        lTimeTaken
        nCOGx
        nCOGy
        nSUMbright
        Susan(DImage)
        Susan(DImage, int)
        getDImage()
        createLUT(int)
        filterRGB(int, int, int)
        SusanCornerFilter()
        SusanCornerFilterPixel(int, int)
        LocalMaxima()
```

```
Filters
    Imports
        java.awt.image.*
        java.awt.*
        java.applet.*
        java.util.*
        DataTypes.*
        Tools.*
    DMSusan
        Superclasses
        Inherited members
        theDImage
        nImageWidth
        nImageHeight
        nPixels
        nPixelsM
        m_nBrightnessThreshold
        m_nBrightnessLUT
        m_nBrightnessMaxNumber
        nCGX
        nCGY
        nR
        nPixelsOut
        theCornerList
        lTimeTaken
        dPreFilterValue
        nTheMedianType
        nCOGx
        nCOGy
        nSUMbright
        PRECount
        SUScount
        COGcount
        DMSusan(DImage)
        DMSusan(DImage, int, double,
        getDImage()
        createLUT(int)
        filterRGB(int, int, int)
        SusanCornerFilter()
        medianOf(int)
        SusanCornerFilterPixel(int, int)
        LocalMaxima()
```

Figure D-4. The main structure and some of the filter classes.

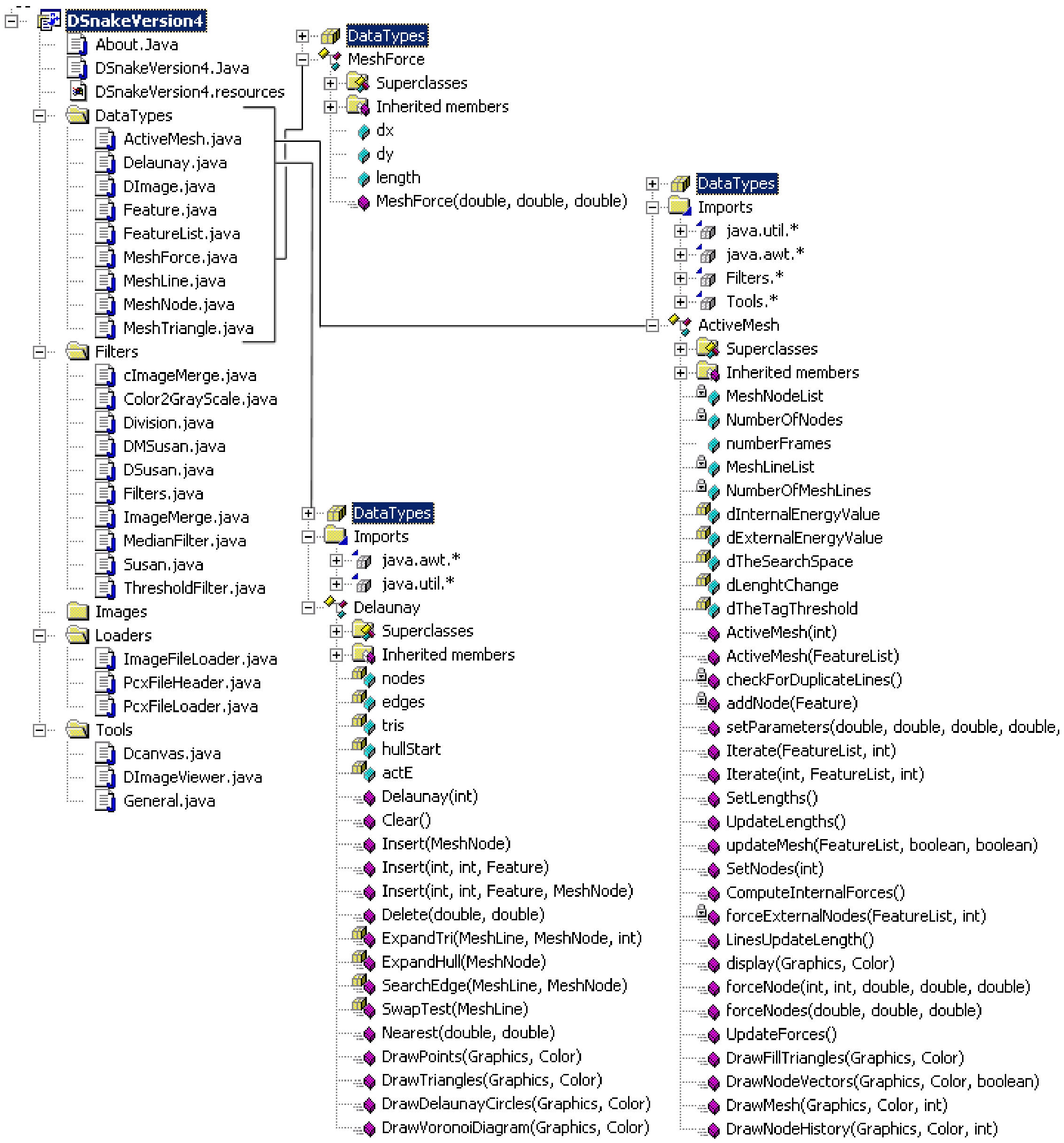
.



```
DSnakeVersion4
    About.Java
    DSnakeVersion4.Java
    DSnakeVersion4.resources
    DataTypes
        ActiveMesh.java
        Delaunay.java
        DImage.java
        Feature.java
        FeatureList.java
        MeshForce.java
        MeshLine.java
        MeshNode.java
        MeshTriangle.java
    Filters
        cImageMerge.java
        Color2GrayScale.java
        Division.java
        DMSusan.java
        DSusan.java
        Filters.java
        ImageMerge.java
        MedianFilter.java
        Susan.java
        ThresholdFilter.java
    Images
    Loaders
        ImageFileLoader.java
        PcxFileHeader.java
        PcxFileLoader.java
    Tools
        Dcanvas.java
        DImageViewer.java
        General.java
```

```
Filters
Imports
    java.awt.image.*
    java.awt.*
    java.applet.*
    java.util.*
    DataTypes.*
    Tools.*
DSusan
    Superclasses
    Inherited members
    theDImage
    nImageWidth
    nImageHeight
    nPixels
    m_nBrightnessThreshold
    m_nBrightnessLUT
    m_nBrightnessMaxNumber
    nCGX
    nCGY
    nR
    nPixelsOut
    theCornerList
    lTimeTaken
    nCOGx
    nCOGy
    nSUMbright
    dPreFilterValue
    PRECount
    SUScount
    COGcount
    DSusan(DImage)
    DSusan(DImage, int, double)
    getDImage()
    createLUT(int)
    filterRGB(int, int, int)
    SusanCornerFilter()
    SusanCornerFilterPixel(int, int)
    LocalMaxima()
```

Figure D-5. The main structure and some of the filter classes.



```
DSnakeVersion4
    About.Java
    DSnakeVersion4.Java
    DSnakeVersion4.resources
    DataTypes
        ActiveMesh.java
        Delaunay.java
        DImage.java
        Feature.java
        FeatureList.java
        MeshForce.java
        MeshLine.java
        MeshNode.java
        MeshTriangle.java
    Filters
        cImageMerge.java
        Color2GrayScale.java
        Division.java
        DMSusan.java
        DSusan.java
        Filters.java
        ImageMerge.java
        MedianFilter.java
        Susan.java
        ThresholdFilter.java
    Images
    Loaders
        ImageFileLoader.java
        PcxFileHeader.java
        PcxFileLoader.java
    Tools
        Dcanvas.java
        DImageViewer.java
        General.java
```

```
DataTypes
MeshForce
    Superclasses
    Inherited members
    dx
    dy
    length
    MeshForce(double, double, double)
```

```
DataTypes
Imports
    java.awt.*
    java.util.*
Delaunay
    Superclasses
    Inherited members
    nodes
    edges
    tris
    hullStart
    actE
    Delaunay(int)
    Clear()
    Insert(MeshNode)
    Insert(int, int, Feature)
    Insert(int, int, Feature, MeshNode)
    Delete(double, double)
    ExpandTri(MeshLine, MeshNode, int)
    ExpandHull(MeshNode)
    SearchEdge(MeshLine, MeshNode)
    SwapTest(MeshLine)
    Nearest(double, double)
    DrawPoints(Graphics, Color)
    DrawTriangles(Graphics, Color)
    DrawDelaunayCircles(Graphics, Color)
    DrawVoronoiDiagram(Graphics, Color)
```

```
DataTypes
Imports
    java.util.*
    java.awt.*
    Filters.*
    Tools.*
ActiveMesh
    Superclasses
    Inherited members
    MeshNodeList
    NumberOfNodes
    numberFrames
    MeshLineList
    NumberOfMeshLines
    dInternalEnergyValue
    dExternalEnergyValue
    dTheSearchSpace
    dLenghtChange
    dTheTagThreshold
    ActiveMesh(int)
    ActiveMesh(FeatureList)
    checkForDuplicateLines()
    addNode(Feature)
    setParameters(double, double, double, double,
    Iterate(FeatureList, int)
    Iterate(int, FeatureList, int)
    SetLengths()
    UpdateLengths()
    updateMesh(FeatureList, boolean, boolean)
    SetNodes(int)
    ComputeInternalForces()
    forceExternalNodes(FeatureList, int)
    LinesUpdateLength()
    display(Graphics, Color)
    forceNode(int, int, double, double, double)
    forceNodes(double, double, double)
    UpdateForces()
    DrawFillTriangles(Graphics, Color)
    DrawNodeVectors(Graphics, Color, boolean)
    DrawMesh(Graphics, Color, int)
    DrawNodeHistory(Graphics, Color, int)
```

Figure D-6. The main structure and some of the dataType classes.

.

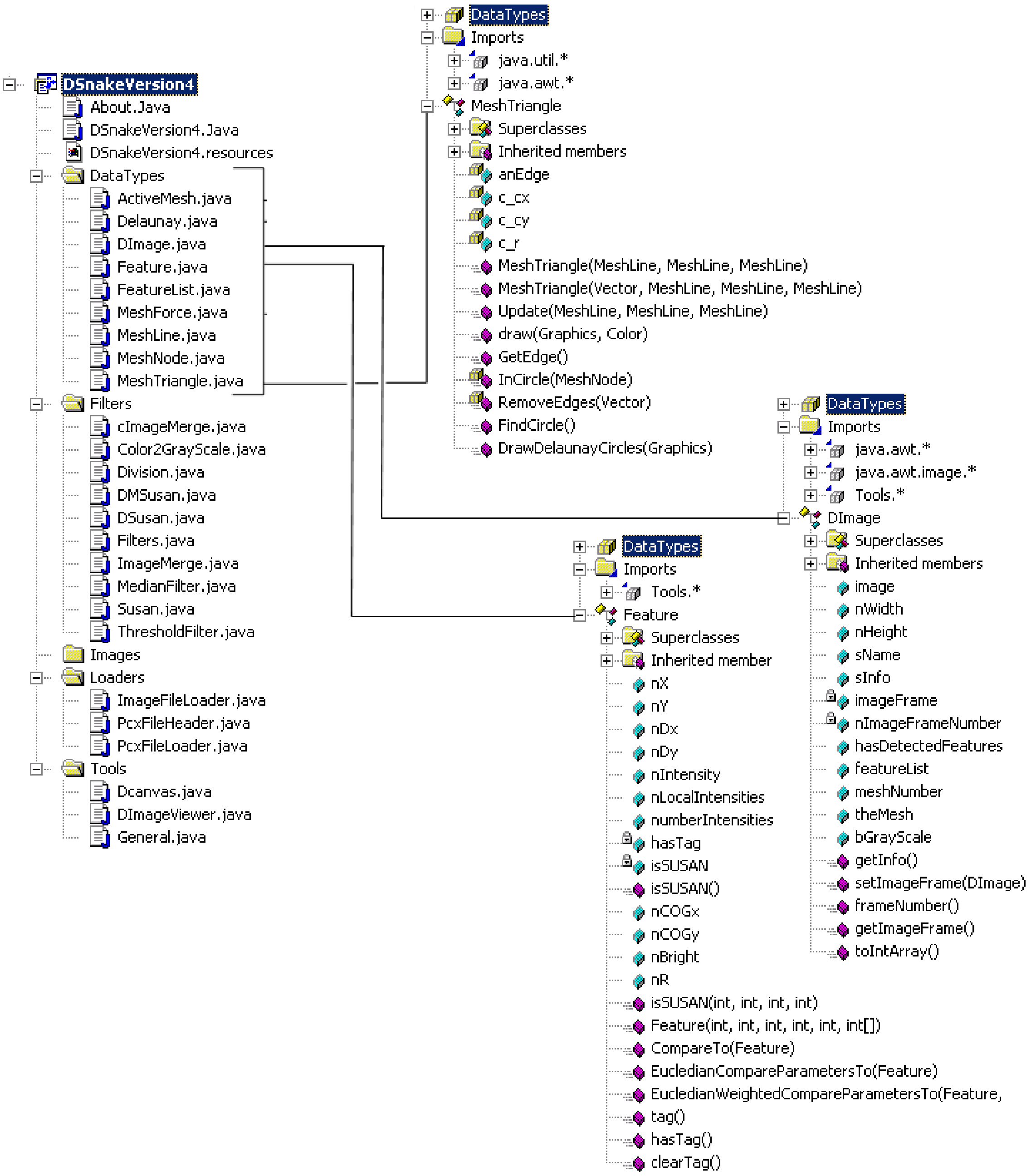
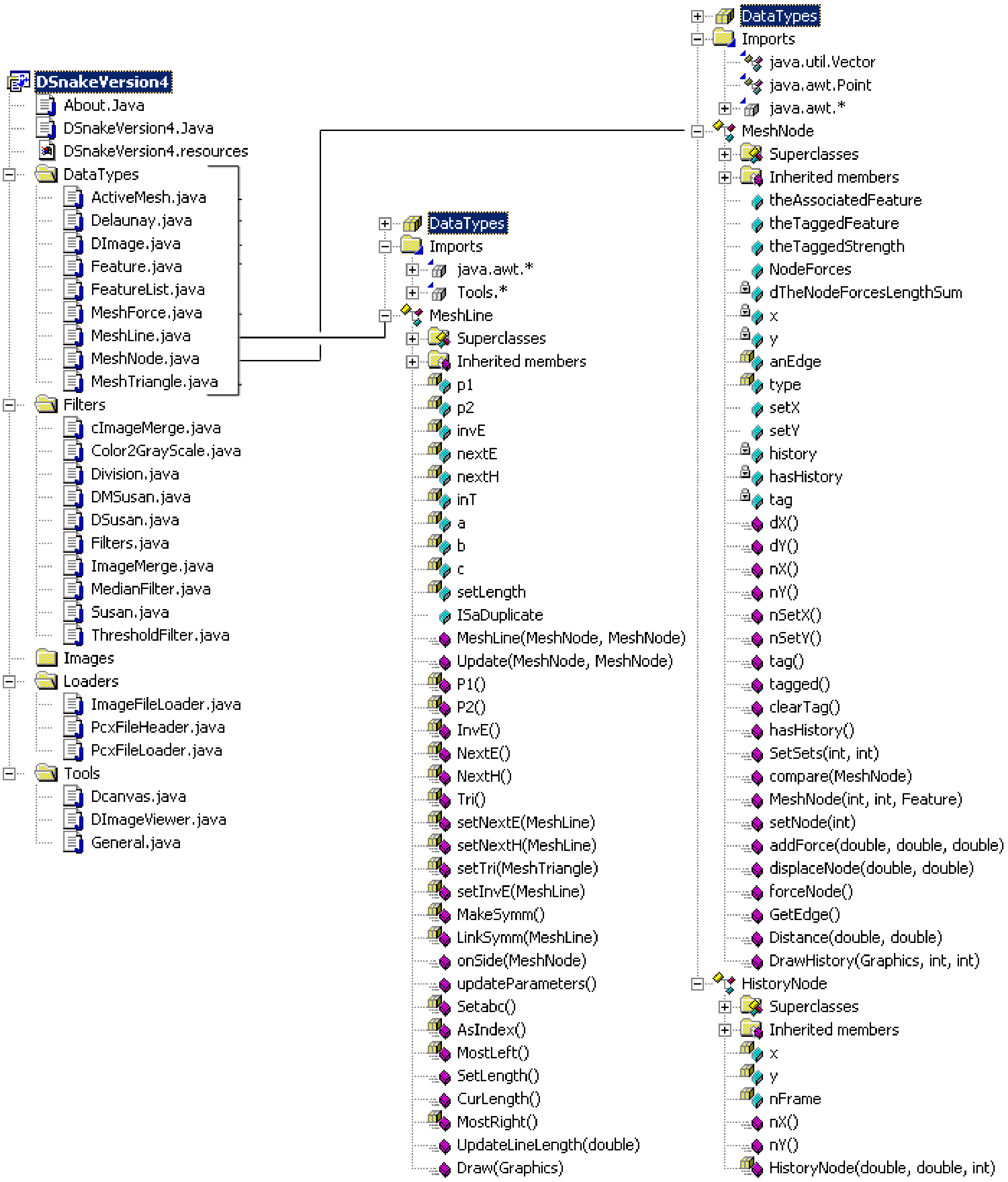


Figure D-7. The main structure and some of the dataType classes.

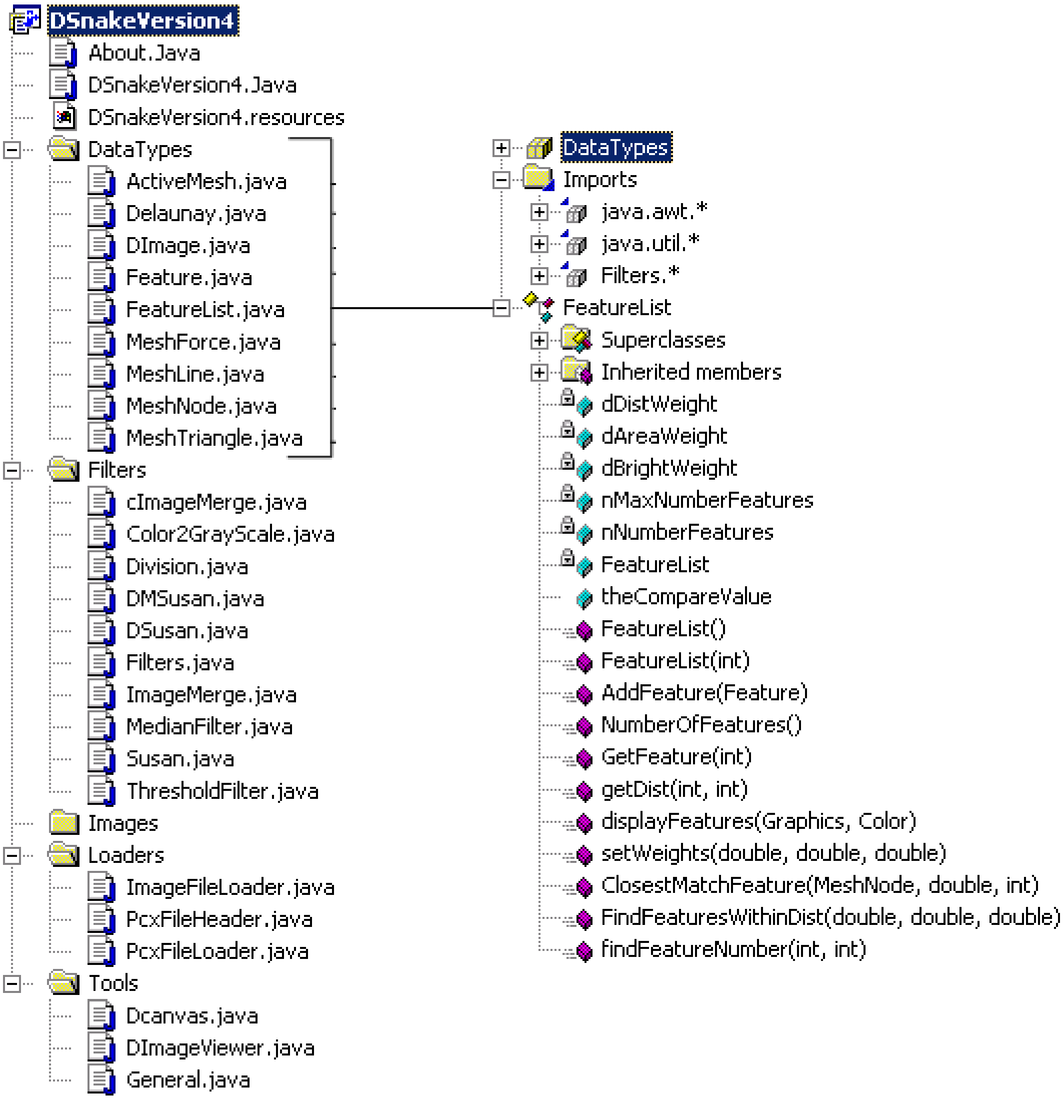


```
DSnakeVersion4
    About.Java
    DSnakeVersion4.Java
    DSnakeVersion4.resources
    DataTypes
        ActiveMesh.java
        Delaunay.java
        DImage.java
        Feature.java
        FeatureList.java
        MeshForce.java
        MeshLine.java
        MeshNode.java
        MeshTriangle.java
    Filters
        cImageMerge.java
        Color2GrayScale.java
        Division.java
        DMSusan.java
        DSusan.java
        Filters.java
        ImageMerge.java
        MedianFilter.java
        Susan.java
        ThresholdFilter.java
    Images
    Loaders
        ImageFileLoader.java
        PcxFileHeader.java
        PcxFileLoader.java
    Tools
        Dcanvas.java
        DImageViewer.java
        General.java
```

```
DataTypes
Imports
    java.awt.*
    Tools.*
MeshLine
    Superclasses
    Inherited members
    p1
    p2
    invE
    nextE
    nextH
    inT
    a
    b
    c
    setLength
    ISaDuplicate
    MeshLine(MeshNode, MeshNode)
    Update(MeshNode, MeshNode)
    P1()
    P2()
    InvE()
    NextE()
    NextH()
    Tri()
    setNextE(MeshLine)
    setNextH(MeshLine)
    setTri(MeshTriangle)
    setInvE(MeshLine)
    MakeSymm()
    LinkSymm(MeshLine)
    onSide(MeshNode)
    updateParameters()
    Setabc()
    AsIndex()
    MostLeft()
    SetLength()
    CurLength()
    MostRight()
    UpdateLineLength(double)
    Draw(Graphics)
```

```
DataTypes
Imports
    java.util.Vector
    java.awt.Point
    java.awt.*
MeshNode
    Superclasses
    Inherited members
    theAssociatedFeature
    theTaggedFeature
    theTaggedStrength
    NodeForces
    dTheNodeForcesLengthSum
    x
    y
    anEdge
    type
    setX
    setY
    history
    hasHistory
    tag
    dX()
    dY()
    nX()
    nY()
    nSetX()
    nSetY()
    tag()
    tagged()
    clearTag()
    hasHistory()
    SetSets(int, int)
    compare(MeshNode)
    MeshNode(int, int, Feature)
    setNode(int)
    addForce(double, double, double)
    displaceNode(double, double)
    forceNode()
    GetEdge()
    Distance(double, double)
    DrawHistory(Graphics, int, int)
HistoryNode
    Superclasses
    Inherited members
    x
    y
    nFrame
    nX()
    nY()
    HistoryNode(double, double, int)
```

Figure D-8 The main structure and some of the dataType classes.



```
DSnakeVersion4
    About.Java
    DSnakeVersion4.Java
    DSnakeVersion4.resources
    DataTypes
        ActiveMesh.java
        Delaunay.java
        DImage.java
        Feature.java
        FeatureList.java
        MeshForce.java
        MeshLine.java
        MeshNode.java
        MeshTriangle.java
    Filters
        cImageMerge.java
        Color2GrayScale.java
        Division.java
        DMSusan.java
        DSusan.java
        Filters.java
        ImageMerge.java
        MedianFilter.java
        Susan.java
        ThresholdFilter.java
    Images
    Loaders
        ImageFileLoader.java
        PcxFileHeader.java
        PcxFileLoader.java
    Tools
        Dcanvas.java
        DImageViewer.java
        General.java
```

```
DataTypes
Imports
    java.awt.*
    java.util.*
    Filters.*
FeatureList
    Superclasses
    Inherited members
    dDistWeight
    dAreaWeight
    dBrightWeight
    nMaxNumberFeatures
    nNumberFeatures
    FeatureList
    theCompareValue
    FeatureList()
    FeatureList(int)
    AddFeature(Feature)
    NumberOfFeatures()
    GetFeature(int)
    getDist(int, int)
    displayFeatures(Graphics, Color)
    setWeights(double, double, double)
    ClosestMatchFeature(MeshNode, double, int)
    FindFeaturesWithinDist(double, double, double)
    findFeatureNumber(int, int)
```

Figure D-9. The main structure and some of the dataType classes.

.

Author’s publications related to this work.

- D. Molloy, T. McGowan, K. Clarke, C. Mc Corkell, P.F. Whelan (1994), "Application of machine vision technology to the development of aids for the visually impaired", in *Machine Vision Applications, Architectures, and Systems Integration III*, Proc. SPIE 2347, 31 Oct-2 Nov, Boston USA, 59-69.

- D. Molloy, P.F. Whelan (1996), "Motion Discrimination in Applied Vision Systems", Dublin City University, School of Electronic Engineering, *Internal Technical Report*, 26th March 1996.

- D. Molloy, P.F. Whelan (1997a), "Robotic navigation using self-initialising active contours", *Intelligent Robots and Computer Vision XVI: Algorithms, Techniques, Active Vision, and Materials Handling*, Pittsburgh, Oct. 1997, Proc. SPIE Vol.3208.

- D. Molloy, P.F. Whelan (1997b), "Self-Initialising Active Contours for Motion Discrimination", *Proc. IMVIP'97 - Irish Machine Vision and Image Processing Conference 1997*, 10 - 13 September 1997, University of Ulster, Northern Ireland, pp 139 - 146.

- D. Molloy, P.F. Whelan (1999a), "Motion Tracking using Self-Initialising Active Meshes", *The I.E.E. Seventh International Conference on Image Processing and its Applications*, University of Manchester, UK: 12-15 July.

- D. Molloy, P.F. Whelan (1999b), “Active-Mesh Self-*Initialisation”, IMVIP’99 Irish Machine Vision and Image processing Conference 1999*, September 1999, Dublin City University. pp 116 - 129.

Submitted for Pattern Recognition Letters 28th December 1999.

- As: PATREC 1629: Active-meshes (Molloy, D. and P. F. Whelan).

Book in progress:

- P. F. Whelan, D. Molloy, “Machine Vision Algorithms in Java: Techniques and Implementation”. To be published by Springer-Verlag, September 2000.