AN EXTENSION TO OPTIC FLOW ANALYSIS FOR THE GENERATION OF COMPUTER ANIMATED IMAGES

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF SCIENCE

AT THE UNIVERSITY OF CAPE TOWN

IN FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Ian Webb

November 1998

Supervised by

E.H. Blake

Abstract

This dissertation seeks to develop image based animation methods using the technique of optic flow analysis developed for a moving planar object.

Image based rendering is presented as a class of algorithm using two dimensional shortcuts to the problem of three dimensional animation. The optic flow field is used to develop an image based algorithm based on its use as a description of the differences between consecutive frames of an animation. A Taylor analysis of the optic flow field is the underlying tool used, breaking the field up into a hierarchy of terms.

For a moving planar object, we have considerably simplified the second order Taylor terms into a basis of just two independent terms, which can be related closely to a perspective transformation between frames. Perspective transformations capture exactly the optic flow of a moving planar object.

Using the simplified decomposition of the flow field for a moving plane, we decompose the frame to frame transformation into a hierarchy of terms of increasing accuracy and cost. Depending on their accuracy we may choose any of these as transformation on an image between frames, instead of rerendering. The errors in the approximation can be tracked via the Taylor series.

This dissertation develops the theory and presents an animation algorithm based on optic flow, and then presents the results of various tests of the algorithm in a variety of simple scenes. The results demonstrate the effectiveness of the algorithm and the time saving achieved in animation.

Acknowledgments

Karen Greaves, David Kossew and Oliver Saal, for their contribution in the form of a graduate project which tested the optic flow algorithm in the case of multiple objects.

Professor Edwin Blake, who supervised this dissertation.

Contents

Chapter 1

Introduction

1.1 Outline of the Introduction

In this introduction we set out the aims of our dissertation. We motivate the search for new techniques in animation, in terms of the demand for performance that may remain unachievable in hardware for some time. We introduce the mathematical technique of optic flow analysis which we have explored and developed, and discuss the development of an image based rendering algorithm using optic flow. We describe briefly the relation between the mathematics of our approach and a previous mathematical approach to animation called scene shifting. We introduce the main results of this dissertation, and finally provide a chapter-by-chapter outline of the dissertation.

1.2 Aims of this dissertation

The aim of this dissertation is the development of optic flow analysis as a technique for image based rendering, thereby developing mathematical techniques which can be used to speed up three dimensional animation using two dimensional transformations of the image. These techniques make use of various types of frame to frame coherence, which are grouped into a hierarchy of levels depending on their mathematical complexity. These levels relate closely to the terms in the Taylor series expansion of the optic flow field.

Our contribution here is in developing the mathematics underlying this relation, and then in extending previous work to relate the second order Taylor terms directly to perspective warping using homogeneous matrices. This involves an important simplification for the optic flow field of a moving planar surface, for which the six second order Taylor terms can be reduced to a new basis of just two independent terms.

This theory will be tested by implementing an algorithm for animating scenes consisting of planar surfaces. The intention is to demonstrate the applicability of optic flow analysis to animation, to test its performance, and to investigate whether it holds any advantages in practice.

1.2.1 Motivation

The increasing demand for interactive three dimensional graphics, for instance in virtual reality, places high demands on system performance. Inherent in the problem is the huge complexity of rendering a three dimensional scene, taking into account a sufficiently large number of polygons, occlusion of objects, shadows, lighting and reflection models, texture mapping and antialiasing.

Although high-end graphics workstations are capable of reasonable performance in this area, the problem is by no means solved. High quality real time animation remains beyond the reach of most desktop machines which do not have hugely powerful rendering engines, massively parallel architectures or enormous amounts of memory. Torborg and Kajiya [38] point out that high-end graphics systems require a memory bandwidth 20 times that currently obtainable on a PC, and current trends suggest that, despite hardware improvements, such bandwidth will remain unavailable for the PC for a long time to come.

In any case, even high-end Virtual Reality systems still experience latency problems. A delay of 100 ms is considered the maximum acceptable level, and yet in the graphics supercomputer Pixel Planes 5, the latency due to image generation alone use up more than half of this time [4]. Even high-end systems suffer from problems such as 'swimming', where a change in direction of motion means pipelined frames have to be recalculated, resulting in a disturbing lag.

One study on motion sickness in VR systems found that 8 out of 150 subjects experienced such severe nausea in a trial that they had to opt out [35]. It is not clear how much of this problem is caused by other factors such as focus, poor stereoscopic calibration, a heavy visor and so on, but the lag between making a movement and seeing the effect is a contributing factor.

System latency inherent in the traditional graphics pipeline seems likely to remain significant. While there have been many advances in rendering quality, reductions in system latency are rarer: perhaps because they have fewer obvious benefits [4]. But reducing lag is vital to developing immersive environments such as VR or Augmented Reality (AR) [19].

1.3 Optic Flow Analysis

In Chapter 2 we will describe the background to image based rendering, and introduce the *optic flow field*. Optic flow will be used to describe the difference between consecutive frames in an animation sequence, in terms of a field of velocity vectors for each image point. In Chapter 4 we will develop the mathematics of *optic flow analysis* which underlies our approach to animation, allowing previous frames to be reused after applying transformations which are computed from the optic flow field.

The optic flow field is first analysed in terms of a Taylor Series, and the terms from the Taylor Series related to a hierarchy of image transformations:

1. no update, if all orders of Taylor terms are close to zero

2. translation of the previous, related to the zero order Taylor terms

3. affine transformation, related to first order terms

4. perspective transformation, related to second order terms
5. rerendering the object, if terms higher than second order are significant.

Our contribution is in exploring these relationships mathematically, particularly among the second order terms for a moving planar object. In this case the six second order Taylor terms are reduced to just two independent terms, which give a good approximation to the actual perspective transformation between frames. This is important because this transformation can then be found directly, rather than by an indirect method such as a least squares approximation.

1.4 Outline of this dissertation

1.4.1 Background and related work

In Chapter 2, we summarise related work in the field of image based rendering. We begin by motivating the need for techniques of animation other than physical simulation, and then discuss a number of image based methods, which are categorised into three areas:

- interpolation techniques, using image morphing and warping to produce intermediate views from previously computed or stored views.
- image layer techniques, where the image is split into multiple layers, which are then manipulated independently and composited together to form the final image. Our algorithm using optic flow analysis falls into this category, as well as the important Talisman architecture.
- light field rendering, where a four dimensional field captures the complete optic information in the region surrounding an object. Views can be produced by taking two dimensional 'slices' of this field.

1.4.2 Scene shifting by optic flow analysis

Before we proceed to develop our own theory and algorithm for image based rendering, we explore an approach by Hofmann [17] which puts a mathematical basis on a technique of layered animation. The technique is widely used, but the mathematical analysis presented in [17] is not well known. We show that this approach does not produce any results that can't be produced using optic flow analysis, and that optic flow analysis covers more motions; thus optic flow is a more general approach. This comparison is based on reformulating the mathematics of [17] in the same optic flow notation we have used elsewhere.

1.4.3 Optic flow analysis

In Chapter 4 we present a theory of optic flow analysis, including both previous work and our own contribution to this theory for animation using optic flow. We describe how a Taylor analysis of the optic flow field relates to the frame to frame transformations in an image based animation algorithm.

Concentrating on a moving planar object, we then introduce an extension of the analysis up to second order terms. We introduce a new basis for the second order terms which relates directly to a perspective warp between frames. This basis is used later to produce an algorithm for animating using frame to frame transformations up to and including perspective transformations.

1.4.4 Algorithm for animation using optic flow

In Chapter 5 we make the relationship between the derived Taylor Series terms and the frame to frame image transformations explicit, and develop two algorithms for animation using optic flow.

The first algorithm is presented for a single planar object; and the second for multiple objects.

1.4.5 Main results

Results from three experiments using optic flow for animation are presented in Chapter 6.

The first experiment uses affine optic flow for animating a single planar object. The algorithm is tested against several different motion scripts.

The second experiment for animating a single planar object using perspective optic flow. The same scripts are used as before, and the implementation is similar to the previous experiment.

The third experiment involved a scene with multiple planar objects animated using optic flow. Various different scripts were implemented and tested for various sample scenes of up to 20 polygons.

This chapter concludes with various comparisons and conclusions about the usefulness and drawbacks of our optic flow animation algorithm.

1.4.6 Conclusions

Chapter 7, the Conclusion, begins by summarising the theory developed here, the comparison between optic flow and scene shifting, and the algorithms developed. The final results are analysed and conclusions drawn from them. Finally the chapter discusses the limitations of this work and makes suggestions for further research.

Chapter 2

Background and related work

2.1 Introduction and outline of this chapter

In this chapter we first motivate the search for new techniques in animation, and the need to avoid physical simulations. Image based rendering is introduced as a shortcut which makes use of many types of coherence in animation.

Related work in the field of image based rendering is then discussed and the different approaches categorized into three broad areas: image interpolation techniques, image layer techniques, and light field. Section 2.6 then summarises the different techniques and discusses their usefulness in relation to our approach of using optic flow analysis as a tool for animating moving planar surfaces.

2.1.1 A motivation for avoiding physical simulations (or: Why is interactive 3D graphics such a hard problem?)

'Nature laughs at the difficulties of integration'
Pierre-Simon de Laplace (1749 – 1827) [24]

As mentioned in Section 1.2.1 of the Introduction, the demand for interactive three dimensional graphics puts high demands on system performance which are likely to remain unattainable on desktop machines for some time. The reasons for this are rooted in the immense power of the human visual system.

The human eye has a total retinal area of 5cm^2, with over 160 000 cone cells per square millimetre. Rod cells are capable of detecting a single photon, and the eye can detect flicker at up to 70 Hz. The eye can can cope with brightness over 8 orders of magnitude, from 10^{-3} to 10^5 cd/m^2 [39]. The sensitivity of the eye varies over the visual field, from the high spatial accuracy and colour sensitivity of the fovea, to the outer areas of the retina, where a predominance of rod cells gives great sensitivity in low light levels and the ability to detect very rapid motion.

Graphical systems therefore have a hard task in trying to reproduce the visual experience available to the human visual system using a graphical display of limited resolution, with limitations on palette and refresh rates. Significant progress has been made in some of these areas: high end systems now make use of more colours and faster refresh rates than we can distinguish.

However a much bigger problem lies in the huge complexity of a real scene. The view while traveling down a leafy avenue involves the interaction of literally billions of surfaces. These interactions take place in nature due to vast numbers of photons traveling and reflecting virtually instantaneously – a natural parallel computer on a huge scale. We are still many orders of magnitude away from a photo-realistic walkthrough of a virtual forest, even on the fastest parallel graphics supercomputers, so animation remains a hard problem.

Instead, the trend in animation has began to turn away from simulating physical processes to finding new shortcuts that make the problem more tractable. One approach is to make use of *coherence* in the moving image.

2.1.2 Coherence

Coherence, or unity between adjacent parts, is a handle often used to simplify potentially complex tasks. By exploiting such unity or similarities, coherent data may be processed using shortcuts and approximations not available in a generalised data set.

Coherence is particularly important for moving image sequences, where several forms of coherence are present. We have categorised them as follows:

- **Model coherence**
  Objects in the three dimensional world are generally contiguous (in one piece) and continuous in time (they stay put). These trivial observations have important consequences for vision and animation.
- **Image coherence**
  As a consequence of model coherence, the projected two dimensional image exhibits its own coherence. Nearby image points often come from the same object, and as a consequence have similar colour, surface orientation, shading, motion, etc.
- **Viewpoint coherence**
  The viewer's position and orientation tend to change continuously and smoothly over time. Over a short time interval, the view is seen from nearby viewpoints in similar orientations. The viewer's velocity also tends to vary smoothly, so image motion is similar over intervals.
- **Frame to frame coherence**
  An important consequence of viewpoint coherence (and model coherence, to an extent) is that consecutive frames are very similar. It is this type of coherence that we will be addressing throughout this dissertation.

It seems certain that the human visual system makes use of many types of coherence. Patterns in a moving image which are related to specific motions or familiar objects, are quickly understood by the visual system which then extracts detailed information relating to the three dimensional world.

The field of Computer Vision aims to extract the same information, using coherence in the moving image. Various forms of coherence have also been exploited in image sequence compression, such as in the MPEG standard. In this dissertation we will be discussing and developing ways to exploit coherence in animation.

2.2 Image based rendering

Existing rendering using a graphics pipeline has been accused of overkill, essentially because such a renderer needs to be capable of rendering an entirely different scene at every frame [38]. While this claim is exaggerated (in the sense that caching of textures and model data does make use of frame to frame coherence), it is true that rendering consecutive frames involves performing the same calculations on a similar view of the same model, and obtaining very similar output images. Exploiting these many layers of coherence is the main motivation for *image based rendering*.

Image based rendering refers to a broad category of animation techniques where image processing techniques are substituted for physical simulation.

The term encompasses techniques which use a limited set of existing photographic images to generate new views. This type of image based rendering was surveyed by Kang in [22], but this survey did not encompass a further class of image based rendering, where approximations are used to simplify the generation of images without significant loss of realism, concentrating on the viewer's perception and interpretation of the image.

For example, instead of radiosity and ray tracing (which are simulations of physical interactions of light and the scene), image based techniques such as texture mapping and bump mapping can be added to provide extra perceptual detail to a purely geometric model.

More powerful image based rendering techniques can be applied for a moving image. Recently a number of new approaches have been developed to speed three dimensional animation by performing transformations as much as possible on the two dimensional image. Because consecutive frames of an animation are very similar, these transformations in image space are much simpler than rerendering the scene at every step, and the reduced demands for processing power allow for faster frame rates, extra realism and more interactivity.

Kang's *Survey of image-based rendering techniques* [22] identified four basic categories, based on the pixel indexing scheme. *Non-physically based image mapping* incorporates techniques such as morphing which do not consider 3D geometry. *Mosaicking* represents a scene by a number of images stitched together. *Interpolation from dense samples* uses many images of a scene from different viewpoints which are stored and interpolated to produce new views). Finally, *geometrically valid pixel projection* calculate geometric constraints which can be used to calculate the changing location of pixels. Kang's categorisation is not the only possible division, and we have developed a different scheme below.

We have divided the image based systems discussed in the remainder of this chapter into three broad categories, as follows:

1. *Image interpolation techniques*
   includes work by Chen and Williams on view interpolation for image synthesis [6], McMillan and Bishop on plenoptic modeling [32], which are examples of 'mosaic' systems where a whole scene is represented from a collection of separate views, and Seitz and Dyer [34, 33] on view morphing and view interpolation for producing intermediate views of an object.

2. *Image layer techniques*
   incorporating Hofmann's scene-shifting [17], depth maps, Torborg and Kajiya's Talisman architecture [38], and optic flow based animation, which will be developed further in Chapter 4.

3. *Light field*
   incorporating Levoy and Hanrahan's light field rendering [29] and Gortler, Grzeszczuk, Szeliski and Cohen on the lumigraph [12].

2.3 Image interpolation techniques

Various techniques have been developed to interpolate between different views of a scene to produce intermediate views, using image morphing. A key aspect of these techniques is that they rely on establishing a pixel correspondence between two or more key images which is then used to interpolate between the images.

In a synthetic animation the correspondences can be obtained from the object, but for natural scenes user input is often required to produce a sparse set of corresponding points or outlines, for instance the features of a face. These are used to automatically generate a complete correspondence between pixels in the two images. The pixels are then interpolated to produce intermediate images.

Similar methods can also be used to extrapolate from a single image, although problems arise with holes due to occlusion.

2.3.1 View Synthesis and View Morphing

Image morphing can also be applied to different views of the same object, producing realistic three dimensional effects purely from the three dimensional images. However there is no guarantee that the intermediate images are realistic intermediate views of the object, and much relies on the artist or user to ensure that good results are achieved. An alarming effect is that under simple linear interpolation of pixels, straight lines can appear bent in intermediate images. So the image information alone is not sufficient.

Laveau and Faugeras [25] show how to dispense with a model and instead represented a scene by a collection of images and their 'fundamental matrices' which describe the projection of the scene onto that image. The fundamental matrices help provide a correspondence between pixels in two images in terms of so-called *epipolar lines.* A pixel in one image is known to lie on a particular epipolar line in the second image. The epipolar line is formed from the triangle of the two viewpoints and the object point, intersected with the viewing plane. Corresponding pixels can be identified by a one dimensional search along epipolar lines (see Figure 1).

A similar technique is used by Seitz and Dyer [33] to produce valid intermediate views of an object, from initial views I_0 and I_1. They assume an orthographic projection and 'monotonicity', which means that the order of points along epipolar lines is

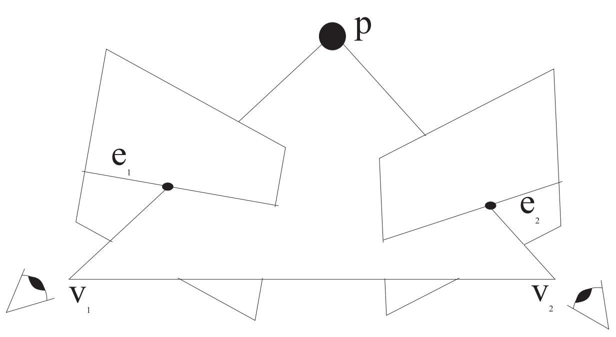


Figure 1: Viewpoints v_1 and v_2 of point p define epipolar lines e_1 and e_2 respectively. Given viewpoint v_1 and the image of p in this view, the epipolar line e_2 can be computed, and the new image of p is constrained to lie on e_2.

not changed. Essentially this means no occlusions occur between the two views, a constraint which applies to many solid objects, particularly convex objects, but not usually to entire scenes. Their algorithm first produces two prewarped 'rectified' views, $\hat{I}_0$ and $\hat{I}_1$, which aligns the epipolar lines of the two images. A straightforward interpolation of these rectified views produces a valid intermediate view $\hat{I}_{0.5}$, which is then warped back to produce the intermediate image $I_{0.5}$ (See Figure 2). If the scene does not satisfy the monotonicity constraint, the intermediate views typically include blurring where incorrect or incomplete point correspondences were computed.

Seitz and Dyer [34] also developed a simple extension called *view morphing* to produce very convincing three dimensional effects such as changes in viewpoint, guaranteeing that that the intermediate images are valid views of the same object. Here they no longer relied on orthographic projection, but allowed more complex image morphing techniques on the rectified images to produce the intermediate image. This also allowed images of different objects to be morphed from different viewpoints, in a strikingly three-dimensional fashion.

Avidan and Shashua's work on the *trilinear tensor* [3] showed how to synthesise new views of a scene given three or more views and a dense correspondence between their image points. This work showed how a tensor $< 1, 2, 3 >$ relating the given images 1, 2 and 3 could be used to generate the tensor $< 1, 2, \psi >$ relating images 1 and 2 with some new view ψ for which the camera location and rotation are known. This tensor

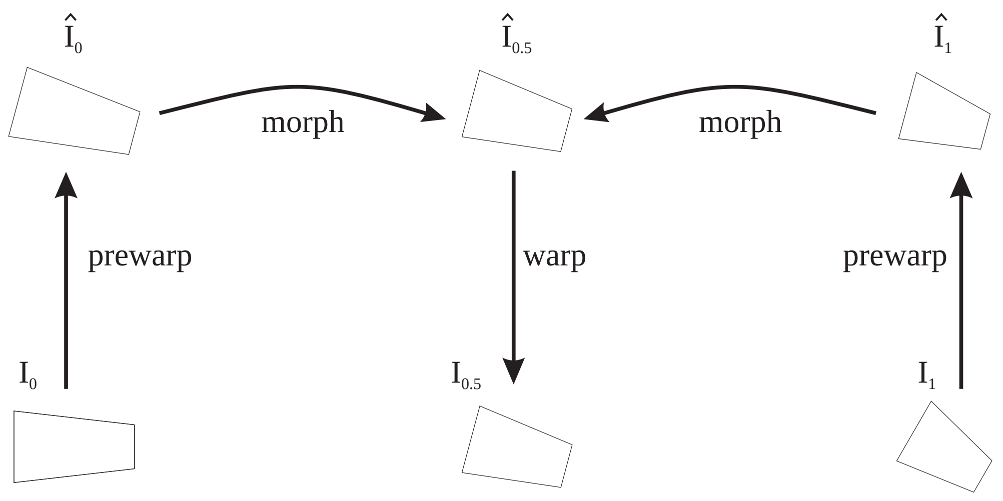


Figure 2: Seitz and Williams' method of view morphing relies on producing two rectified views $\hat{I}_0$ and $\hat{I}_1$ which are then morphed to produce an intermediate view $\hat{I}_{0.5}$. The view $\hat{I}_{0.5}$ is then warped to produce the correct intermediate view $I_{0.5}$.

is then used as a warping function to generate the new image at ψ from images 1 and 2. This approach works for arbitrary scenes except where points which are invisible in both seed images – these of course cannot be synthesised.

2.3.2 View Interpolation for Image Synthesis

S.E. Chen and L. Williams [6] proposed a system which uses image morphing to produce interpolated intermediate images from a limited number of existing views of the scene. It relies on depth information being stored for each pixel in a large set of precomputed images, and allows a viewer to move interactively through a large environment. Their method does not rely on having two key images, except that a second image reduces the number of 'holes' due to visibility changes.

View Interpolation relies on a large set of precomputed *environment maps*. An environment map is a complete description of the view from a given point, also known as the *plenoptic function* at that point. Ideally it would be represented as the projection of the scene onto a sphere centred at the point, but because of difficulties with spherical coordinates, it is usually represented as a cube of views or a cylinder about the point.

For each prestored image, the range data for each pixel (depth away from the camera) is stored. To compute an intermediate image, the range data and new camera position are used to determine a pixel-by-pixel correspondence between the prestored source images and the destination.

This correspondence is implemented as a spatial lookup table in which each pixel is given an offset for image warping. A quadtree approach is used, since neighbouring pixels tend to move in a similar manner. Within individual blocks of the quadtree, this approach is very similar to Hofmann's scene-shifting [17] (Section 2.4.1 and Chapter 3).

Since only image information is stored, problems arise with occlusion and reappearance of parts of the scene. The holes are identified as regions where the background colour is still present after the morph has been performed. Chen and Williams propose interpolating adjacent pixels to fill in the holes, or in a synthetic environment, using ray tracing for the missing pixels. Neither method is particularly elegant and the first is inadequate since holes may be fairly large when a nearby object moves past the viewer, and objects which are invisible in the prestored images may become visible in intermediate frames.

Benefits and drawbacks of View Interpolation

Morphing provides a simple way of exploiting frame to frame coherence while the quadtree decomposition exploits image coherence. Because the only information which is required is the image and range data, view interpolation is independent of scene complexity. Chen and Williams claim an interactive speed achieved on personal computers.

Morphing of images is not necessarily realistic since only straight-line interpolation of pixels is possible. Interpolation along curved arcs would be a better approximation for rotating objects.

View interpolation handles both natural and synthetic scenes, although the depth information is more difficult to obtain in a natural scene. Only view-independent shading is possible, so reflection and specular reflection are not supported, although an additional rendering step could include these features. The scene is considered to be static.

View interpolation requires a large overhead in terms of precomputing and storing of key images. This means it is unsuitable for very large environments where interactivity is required. However it is suitable for predetermined walkthroughs, for instance of an architectural environment, where key-frame animation can be used.

2.3.3 Plenoptic Modeling

Another image based rendering system was proposed by McMillan and Bishop in [32], using a similar method to that of Chen and Williams [6] to reconstruct new views from prestored images. Plenoptic modeling falls into a class of 'mosaic' systems such as Quicktime VR, where the object or scene is represented by a number of different views.

McMillan and Bishop used real-world scenes obtained by panning a camera through 360° to produce a cylindrical environment map, which is a representation of the plenoptic function at that point.

Given two cylindrical views from different points, it is possible to obtain point correspondences between them using *epipolar curves* which are pairs of curves on the two cylinders representing the same points in the scene. Epipolar lines become curves in this example due to the cylindrical projection used – the epipolar lines in Figure 1 relied on a projection onto an image plane.

Using this technique, a point from one prestored image is located in the second image, and this correspondence gives an image flow field which is then used to compute correspondences with a third new position, where an image can be generated. Interpolation over unexposed areas is used, as in [6].

Plenoptic Modeling has many similarities to View Interpolation. It is more suited to producing walkthroughs of real scenes, since no depth information is required and the images may be captured from video without excessive care in calibrating the camera. Both systems have problems with occlusions and work purely in the image domain, with no reference to any geometrical scene representation. Where View Interpolation used the depth information to compute pixel offsets, Plenoptic Modeling requires point correspondences to be found between images by computer vision techniques.

2.4 Image layer techniques

An improvement on the image interpolation techniques above is achieved by breaking the image up into layers for each object in the scene. These layers are then manipulated independently and composited together to form the final image. While this means the animation is no longer independent of scene complexity, it allows better control over different elements of the image which may be evolving independently. Multiple layers also solve problems such as holes due to occlusion.

There are many advantages to dividing the scene into layers and then compositing these layers to the screen. The two dimensional operation of compositing is simpler than three dimensional rendering of polygons using a Z-buffer. Rendering objects into separate layers exploits coherence and allows less important layers to be updated less frequently, thus saving resources for more important layers [28].

2.4.1 Scene-shifting

G.R. Hofmann's 1989 work on 'The Calculus of the Non-Exact Perspective Projection: Scene-Shifting for Computer Animation' in some ways foresaw the growth of image based rendering, yet received little follow-up in the literature. As we will see in Chapter 3, scene-shifting is a special case of optic flow based animation, and is particularly well suited to the simplest types of optic flow: translation, scaling and rotation.

In his work, Hofmann sets up a mathematical basis for an animation system based on parallel planar 'scenes' which are translated or scaled to simulate the changing

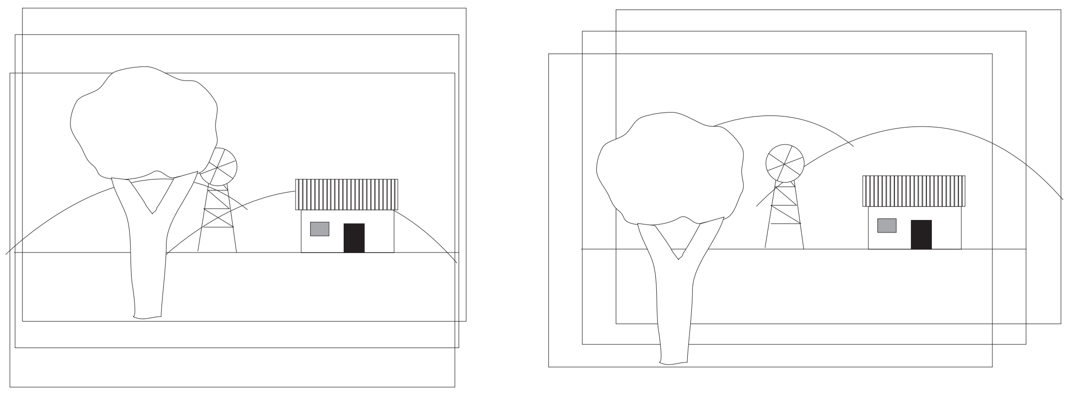


Figure 3: Translation of various scenes (layers) leads to a new view. Here the layers, in front to back order, contain the tree, the house and windmill, and the hills. By translating these layers, the new view appears to come from a higher position to the right of the first view.

image of the object represented on each scene (Figure 3). The calculus developed in this paper is used to ensure that only necessary updates are effected for each scene: a faraway or slow-moving scene does not change significantly from frame to frame, so only needs to be updated occasionally; for an object in the intermediate distance it may be sufficient simply to translate the image from one frame to the next; and a fast moving foreground object may require a new image to be rendered at each frame. Hofmann's calculus provides a mathematical framework for this.

Changes in the image are induced by relative changes in the position and orientation of the viewer. The orientation is specified by a viewing direction vector D and an up direction U, and the viewer's position is specified by a vector $E = (X, Y, Z)$.

The calculus of the non-exact perspective projection shows which points in the image will be changed by less than an amount ε given a change ΔV in the viewer. Points in space for which the image points change by amounts differing by less than ε are called ε-invariant subspaces. The differences may be between absolute translations in the image or an overall scaling factor of the image.

Changes in the up-direction, ΔU, cause the image to be rotated, while changes in viewing direction, ΔD, are implemented purely by changes in the image with no reference to the underlying structure. We will see later that this is in line with the Optic Flow Equation 22, where the optic flow field does not reveal underlying structure unless there is a change in relative position of object and viewer.

Changes in the viewer's position are the key to scene shifting. The method is motivated by a technique used in film making, where, in Hofmann's example, the motion from a train window is simulated by several parallel scenes, which are translated at different speeds depending on their depth away from the train.

For a translation $(\Delta x, \Delta y, 0)$ orthogonal to the viewing direction, the epsilon-invariant subspaces are parallel slices of space. Within each slice, a simple translation of the image would suffice. The magnitude of the translation depends inversely on the depth of the slice.

Similarly, for a translation $(0, 0, \Delta z)$ in the viewing direction, the epsilon-invariant subspaces are again parallel slices of space, and within each slice, a scaling or zoom of the image which is inversely proportionate to the depth is sufficient.

Experiments using scene-shifting in [18] achieve savings of between 36% and 84% of the rendering time, depending on positioning and motion of the objects and viewers. None of these experiments included a change in the viewing direction.

There are close similarities between scene-shifting and the image layers used in the more powerful Talisman architecture [38] discussed in Section 2.4.6. A more recent paper on cel animation [44] also uses many of the techniques Hofmann discussed (see Section 2.4.9).

2.4.2 Optic Flow

Optic flow analysis is one of the chief mathematical tools we will be using to simplify the animation of a moving three dimensional scene. The optic flow field was first proposed by Gibson [11] and since then has been used mainly in the field of computer vision. The definition below is due to Lee [26].

An environment consists of surfaces which we may consider as being covered by texture elements, each of which reflects light toward the observer. The *optic array*

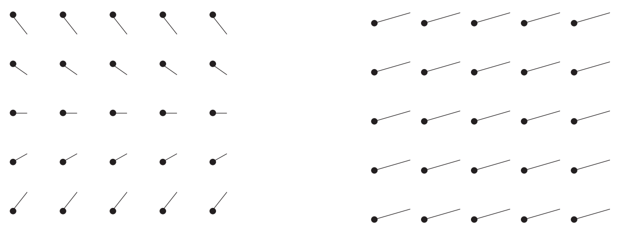


Figure 4: Two simple optic flow fields. The lines represent the instantaneous velocities of points in the image. The first represents a scaling and translation combined, and the second represents a simple translation.

is the bundle of cones, each with apex at the observer and base spanning a distinct environment texture element.

Since the observer is moving, the optic array is never static, and its changes give rise to the *optic flow field*.

The optic flow field is usually described by its projection onto a surface. The surface chosen is usually a planar one perpendicular to the viewing direction in front of the observer, although sometimes the pinhole camera model is used in which the projected image is behind the viewer. These models and other projection surfaces are all equivalent, in the sense that each representation can be uniquely transformed into any other form.

The optic flow field is thus made up of the velocity vectors of all image points due to the relative motion of scene objects and the viewer. Two simple optic flow fields are shown in Figure 4.

2.4.3 Optic flow for computer vision

Much of the work on optic flow in vision and computer vision has focused on how information on structure and motion can be extracted from the optic flow field. This helps determine how an animal or human makes sense of its environment visually, and how a robot could do the same.

Typically the flow field is fairly complicated due to occlusions and sharp edges, which divide it up into smoothly varying segments separated by discontinuities.

In the field of computer vision, optic flow analysis has been used to extract, from a video sequence of a moving three dimensional scene, the underlying structure and motion of that scene. Wohn, Waxman and Ullman[43, 40] analyse the flow field in terms of various *observables* in the moving image. These observables relate to the underlying three dimensional structure and motion of the scene, and it is possible to extract some of this information from the optic flow field.

Ambiguities occur in extracting structure and motion from the optic flow within a region of a uniform surface in motion, since little or no variation occurs in the image of that region, and hence the underlying three dimensional motion is not detectable. This is known as the *aperture problem.* It can be addressed by identifying features which can be tracked, such as the contours corresponding to boundaries of regions of smooth flow.

In computer vision two related flow fields are sometimes distinguished: the optic flow field and the *motion field.* The motion field assigns a velocity vector to each image point, and is determined purely geometrically. By contrast, the optic flow field is the apparent motion of the image taking into account lighting effects; it is considered to be zero where no changes can be tracked in the image, for instance due to the aperture problem or shadows [20]. This distinction is not widely used in computer graphics, and we will use the term *optic flow field* to refer throughout to the purely geometric effects of motion, ignoring lighting effects.

2.4.4 Optic flow for computer animation

In animation, we are concerned with a problem which is the inverse of computer vision. Whereas computer vision concerns itself with extracting the structure and motion of a scene from a moving image sequence, in animation we have all the information about three dimensional structure and motion and would like to produce the image sequence from it. Because of the symmetry between these two problems, we can use a tool from computer vision – namely optic flow analysis – in animation, even though the mathematical tool was designed for a different purpose.

Our main use for optic flow analysis is a description of the difference between frames in an animation sequence. The flow field gives the instantaneous velocities of every image point, so by translating the original points by their optic flow vectors, we get a good approximation of the subsequent image. It is at best only an approximation, since it assumes that the instantaneous optic flow vectors remain constant in the small time interval between consecutive frames.

Because motion tends to be smooth, and because neighbouring image points often come from nearby points on the same surface, an animation sequence exhibits a high degree of *frame to frame coherence.*

In [5], Blake distinguishes various orders of frame to frame coherence: , corresponding to an unchanging image, translation from one frame to the next, affine transformation between frames, and perspective and higher order transformations

This hierarchy is closely mirrored in the optic flow field, which is broken up into smoothly varying regions, which exhibit spatial coherence at various levels: a uniform field is a translation, while other transformations such as rotation and shear correspond to partial derivatives of the optic flow field. This will be discussed in detail in Section 4.5 of Chapter 4, where we will look at the optic flow fields of a moving planar object.

The terms in our expansion of the optic flow field will determine exactly which order of frame to frame coherence is present, and based on this we determine which transformation will suffice to derive the subsequent frame from the current one. The algorithm was originally suggested by Blake in [5] but is extended here.

The correspondence between the optic flow effects and the image transformations is given below for each of the various orders of optic flow effects:

- *zero order frame coherence*, or frame to frame coherence for optic flow effects of zero order and above, means that the previous frame can be reused. This means that all orders of optic flow fall below some threshold.
- *first order frame coherence* means that the previous frame can be translated to produce the current frame. Only the zero order optic flow is significant.

- *second order frame coherence* means that the current frame can be produced using an affine transformation of the previous frame
- *third order frame coherence* means that a perspective transformation is required to produce the current frame. Animations of planar objects exhibit this kind of coherence.

2.4.5 Optic flow for image compression

Although our work falls into the category of image based rendering, related work has been done in the field of image sequence compression. Here optic flow information is extracted from the consecutive frames of an animation sequence, and this information is used to encode the transformations between the frames.

In image sequence compression, information from frame N and frame $N+1$ is used to extract and encode the transformation between frames. In animation, we are interested in generating the transformation from frame N to frame $N+1$ and using it to produce frame $N+1$ using image transformations.

The MPEG compression scheme is a motion-estimation scheme used for image sequence compression, which extracts the information purely from consecutive frames of the image sequence. It is suitable for both synthetic animations and natural video sequences. However, for synthetic animations, MPEG and similar schemes do not take advantage of model information which can be used to guide the motion estimation algorithm.

To take advantage of this extra information, Agrawala, Beers and Chaddha [2] developed an algorithm based on optic flow which exploits the model information to find a transformation from frame N to frame $N+1$. Their algorithm is based on finding the preimage of each pixel in frame $N+1$, finding an approximation to the optic flow field between frames, and using this to encode the transformation from frame N to frame $N+1$.

The preimage is found by first backprojecting the pixel into world space by inverting the projection matrix, P_{N+1}^{-1} (the z coordinate of each pixel is required for this stage).

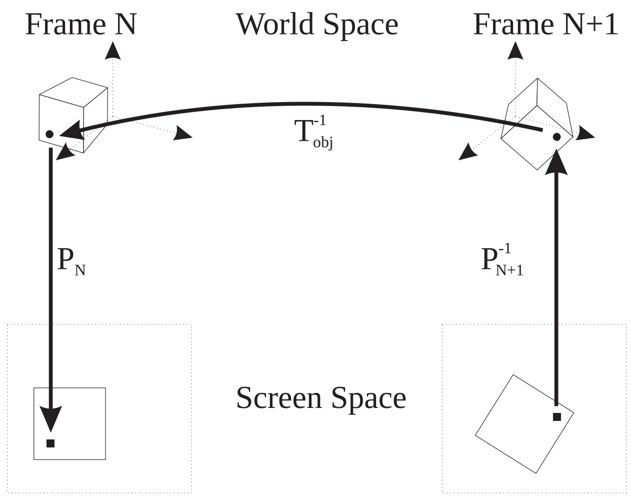


Figure 5: Backprojecting a pixel from frame $N + 1$ to frame N by inverting the projection matrix P_{N+1}, inverting the world space transformation between the frames T_{obj} and then projecting into frame N using P_N. Illustration due to Agrawala, Beers and Chaddha [2].

The inverse of the world-space transformation matrix from frame N to frame $N + 1$, T^{-1}_{obj}, gives the world-space preimage of this point, and the projection from frame N, P_N, gives the pixel's location in the previous frame. This is illustrated in Figure 5, and gives rise to the following equation relating an image point q_N with its location in the subsequent frame, q_{N+1}:

$$q_N = (P_N T^{-1}_{obj} P^{-1}_{N+1}) q_{N+1} \tag{1}$$

The matrix $B_{obj} = P_N T^{-1}_{obj} P^{-1}_{N+1}$ is called the *backtransform matrix.*

Instead of the huge overhead of storing this matrix for every pixel, the algorithm computes a block motion matrix which best preserves the pixel correspondences in a 16×16 block. This two dimensional transformation matrix is found by one of two least-square schemes: LSQ6 gives a six-parameter affine transformation, and LSQ9 gives nine-parameter perspective transformation. This matrix is an approximation of the optic flow within that block.

This algorithm can be adapted to animation by finding the pixel motions for frame N, and computing a block motion matrix to produce frame $N + 1$. The block motion matrix means that the algorithm is independent of the scene complexity since optic flow transformations are performed on a constant number of blocks rather than a variable number of polygons.

This least squares method is similar to that used by Talisman (Section 2.4.6), although Talisman uses a least squares approximation based on selected points on an object, rather than on a block.

There are many similarities between animation and this type of image sequence compression using the optic flow derived from model information, In animation the approach is more flexible, however. Both Talisman and the algorithm we develop later use a similar underlying mathematics, but extended to multiple layers, which is an approach not used in image sequence compression. These image based rendering methods apply the optic flow transformations to regions with arbitrary shapes corresponding to images of planar surfaces, and not a simple division into regular blocks. By definition these regions exhibit coherent optic flow, and the transformation can be computed directly from the geometry. Blocks may contain edges, and hence discontinuous flow, and the transformation must be found by an averaging method.

2.4.6 Talisman

A new approach to reducing the cost of 3D animation was taken by Torborg and Kajiya [38] in *Talisman: Commodity Realtime 3D Graphics for the PC* (1996), which describes an architecture for exploiting temporal and spatial coherence in animation. This paper was followed in 1997 by further papers describing how errors in the approximations are measured, and how the tradeoff between rendering cost of approximations and perceptual quality is decided [28, 21].

The Talisman architecture is aimed at making real-time three dimensional graphics accessible to the PC market. The requirements are smooth motion and low-latency interaction, rather than high spatial accuracy, in order to make real-time interaction possible at video frame rates.

Talisman is a layer based approach to animation, with independent objects being rendered into separate layers which are composited together for the final display. Layers can be reused from frame to frame after transforming them by an appropriate affine transformation instead of rendering the object in that layer at every frame. Lengyel and Snyder described how a scene may be divided into such layers, how the affine transformation is derived, and introduced *fiducials* to measure the fidelity of approximations. Fiducials are discussed further later in this section.

A further aim of Talisman is reducing memory bandwidth requirements. This is done by chunking the image into 32×32 pixel regions to allow on-chip depth buffering and anti-aliasing, and using run-time compression of texture and image data when it is transferred to and from chip.

Bandwidth requirements

The main limitation which Talisman seeks to overcome is that of memory bandwidth. The requirements for 3D animation at high resolution (1024×768, 24 bits) and 75 Hz refresh rates, with features such as trilinear texture filtering and Z-buffering, are more than 20 times that obtainable with existing 3D accelerators for the PC (under 500 MB per second), and improvements in hardware are not likely to reach these levels for a long time to come.

Exploiting coherence

The authors of Talisman point out that the powerful rendering hardware used in a graphics pipeline is guilty of overkill. Most importantly, the pipeline is capable of displaying a completely different model for each frame. In other words, the frame to frame coherence (temporal coherence) of an animation is not exploited.

Spatial coherence is also not traditionally exploited: both textures and images have a high degree of spatial coherence which means that much of the bandwidth used in transmitting uncompressed textures or images can be saved.

Spatial coherence is exploited in the Talisman architecture by means of composited *image layers*. Objects or groups of objects are rendered into separate image layers, and these layers are composited together to create the final animation. Each layer can thus be manipulated independently and at different update rates, based on priority.

The manual factoring into layers is described in [28], depending on relative velocity of parts, separation of background and foreground based on perceptual distinctness, and the fraction of the layer's area that the object takes up. Lighting effects are also described by means of storing diffuse and specular terms, shadow, lens flare and so on in separate layers.

The use of image layers in Talisman is thus a much more powerful approach than that of *scene shifting* as proposed by Hofmann in [17] and discussed here in section 2.4.1.

Frame to frame coherence is exploited by means of affine transformations of image layers (encompassing scaling, rotation and shears), to approximate three dimensional transformations on a layer by means of two dimensional image operations. In [28], the derivation of this affine warp is described. The vertices of the bounding polygon in the initial and final projections are compared, and a least squares approach used to find the warp matrix which most closely transforms the initial points to the final points. Minimizing the sum of squares of the differences is easier than minimizing the maximum error over all points. Various other warps are investigated, from pure translation up to general perspective, which is solved using a gradient-descent method to minimise error. Lengyel and Snyder conclude that affine warping is sufficiently flexible while also computationally simple, and that there is not a noticable improvement from using perspective transformations.

Errors in the approximations are monitored using four fiducials: the geometric fiducial is the maximum displacement between an image pixel and it's correct location; the photometric fiducial is the maximum colour error for the shading of an image point; sampling fiducials measures sampling distortion in the image; and visibility fiducials keep track of the change in visibility of surfaces. Geometric and sampling fiducials are discussed in our algorithm in Sections 5.4.5 and 5.4.1.

Horvitz and Lengyel in [21] addressed the question of how limited rendering resources could be allocated among various objects with various levels of detail in the Talisman system. They performed a cost-benefit analysis based on the perceptual benefit of the object versus its expected rendering cost. Their benefit measures were based on the fiducials of [28] as well as further discussions on the focus of the viewer's attention, which is estimated statistically based on a set of perceptual rules. Solving the cost-benefit optimisation problem is a type of ***knapsack problem*** which is known to be NP-complete, although Blake and Mason in [30] discussed approximate solutions which are $O(n \log n)$.

Texture compression and chunking

To exploit spatial coherence of textures and images, Talisman utilises compression when transferring this data on or off chip. The algorithm, called TREC, or *Texture and Rendering Engine Compression* is a lossy JPEG-like algorithm which compresses 8×8 pixel blocks. Compression ratios averaging 10:1 are achievable: typically 16:1 for textures and 5:1 for rendered images.

Talisman does away with the traditional frame buffer from which pixels may be accessed randomly. To allow for efficient compression as well as reducing memory requirements in antialiasing and Z-buffering, Talisman divides each image layer into 32×32 pixel regions called *chunks*.

The scene is then sorted according to which chunk each part will be rendered into, and each chunk is processed independently. This dramatically reduces the size of the Z-buffer down to the size of a single chunk, which can be implemented directly on the chip, reducing bandwidth further and allowing the use of extremely fast memory architecture.

A further advantage of chunking is that it allows advanced anti-aliasing with reduced memory, since each chunk is handled independently. Chunking also fits in well with the compression of rendered images using the block-oriented TREC algorithm.

Talisman's approach of dividing the scene into independent layers and dividing the image into independent chunks would be ideally suited to parallelisation, but this is not exploited.

Significance of Talisman to our approach

It is important to relate the approach of Talisman to our approach of animation using optic flow. The designers of Talisman have used a *model-independent* method of deriving the transformation from frame to frame, using a least squares approximation to find the warp which most closely matches key points between consecutive frames. Finding an affine warp to satisfy this requirement is significantly faster than finding the corresponding perspective warp. They conclude that using warps of higher order than affine is of little benefit.

By comparison, our algorithm in Chapters 4 and 5 uses optic flow analysis, *based on the underlying model*, to compute various possible frame to frame transformations. These transformation could be of various types, including an affine or perspective warp. The choice of transformation is then based on how much error would be incurred in the output image. Although perspective warps are more expensive to implement, they are no more complicated to compute from the optic flow field than the affine warp. This gives us more flexibility than Talisman: we can choose from of several frame to frame transformations, depending on their benefit, whereas Talisman is constrained to using an affine transformation.

2.4.7 Hierarchical Image Caching

Another approach to image based rendering using textures was published by Shade et al [36]. Their system combines hierarchical methods and image based rendering to facilitate a landscape flythrough using a hierarchy of cached images of portions of the scene.

The full three dimensional scene is partitioned into regions and the objects in each region are rendered as a texture onto a quadrilateral within the region. These texture-mapped surfaces are displayed instead of the full geometry which they represent. The spatial partitioning into nodes is by means of a BSP tree with splitting planes placed between objects. Many nodes may be also grouped together into a cluster at any level of the hierarchy. This is unlike Talisman's object based hierarchy.

The geometry inside a given node is represented by means of a textured plane within the node, normal to the viewer. This image is cached for reuse in subsequent frames, and perspective warping between frames keeps it valid for as long as possible. But as occlusions change, the image will have to be rerendered. An error metric keeps track of this, based on an angular disparity between the cached position of a point in the texture, and the point's actual position. This is approximated by computing the discrepancies for the eight corners of the node's bounding box.

The error metric is based on the offset between the viewer's new position and the original position on the normal to the centre of the quadrilateral. As long as this angle does not change radically, object points projected onto the quadrilateral will remain close to their actual locations in the image. Around the current viewpoint there is a 'safety zone' in which the cached image remains valid: a conservative approach is used in which the safety zone is a sphere centred at the viewpoint.

Barring the precomputation times required to create the initial image caches, the speedup factor achieved is between 4.1 and 25.2, with an average of 11.9. The speedup factor is higher for higher frame rates but falls off for very complex scenes because of the increased cost of creating each image cache.

This approach is similar to Talisman and optic flow for animation, allowing reuse of parts of previous frames through warping. The underlying model allowing grouping of objects is more general than our approach, while the method of keeping track on errors is simpler and looser.

2.4.8 Impostors in urban scenery

A paper by Sillion et al [37] has extended the planar textures of the previous section to a textured triangular mesh which acts as an impostor for distant parts of a densely occluded urban scene. This brings image based rendering out of the two dimensional domain it is usually restricted been restricted to, by creating a texture augmented with three dimensional information.

The urban landscape is divided into a local neighbourhood (defined by street boundaries) which is rendered using full three dimensional information, and a distant landscape containing the remainder of the scene, which is represented by an approximate view or impostor.

Planar textures as used in Section 2.4.7 need to be frequently updated because of parallax errors, so instead the impostors are given added three dimensional structure by means of depth information which allows the impostor to be divided into a triangular mesh in three dimensions. This depth information is extracted from the depth map when constructing the initial image, and sharp contours or disparities in the depth map are used to find a good triangulation. The impostor we are left with, although still a three dimensional textured object, is much simpler than the original landscape it represents.

The streets of an urban landscape naturally subdivides the scene into local neighbourhood as well as providing natural directions along which the distant landscape can be seen. As a preprocessing step, an impostor is created for the landscape visible in each direction along each street.

The use of impostors involves a trade-off between the cost of different impostors, and their perceptual benefits. This is the so-called *Multiple Choice Knapsack Problem* (MCKP), in which a single item must be selected from each of a collection of sets of candidate items. In this case the candidate items in a set correspond to the different representations of the same object. The objective is to maximize the total profit (perceptual benefit) of the selected representations while limiting the total rendering cost.

The solution to MCKP is NP-complete. This problem was discussed in [21] in connection with the Talisman architecture, and in [30, 10] which discussed approximate solutions which are $O(n \log n)$.

The use of non-planar impostors in [37] is the most complex image based rendering approach discussed here. An important distinction is that, in contrast to other methods, this approach does not use approximate frame to frame transformations. The contribution of this paper is in simplifying the scene by creating impostors; further work could include fast approximate transformations of the image of the impostor between frames.

2.4.9 Multiperspective panoramas

Hofmann's work on scene-shifting [17] (Section 2.4.1) is strongly related to a more recent work on *multiperspective panoramas* for cel animation [44].

Traditional two dimensional cel animation usually includes a static background image on a separate cel from the moving characters and foreground. This single backdrop is reused for all frames in a sequence, with different parts being displayed through a moving window. Because the backdrop is large in comparison to the window, the backdrop does not have to represent an perspective-correct view of the overall scene, since it is never seen in its entirety. In fact the overall backdrop appears strangely warped, not unlike an Escher print. All that is required is *local* correctness, i.e. for the view through a given window to be realistic.

This technique is most effective for a predetermined path through the scene. For a given flythrough, the backdrop or *multiperspective panorama* can be precomputed and then revealed frame by frame.

Camera motions are described as pan (changing the camera direction in a horizontal direction), tilt-pan (in an arbitrary direction), zoom (change of focal length) and truck (moving perpendicularly to the gaze direction, like Hofmann's view from a train window). An additional case, moving the camera in the direction of viewing, is not discussed.

In the case of pan, a single multiperspective panorama can be generated using a cylindrical projection. Tilt-pan requires a conical projection, while zoom is simply a scaling of a section of the image, possibly requiring multiresolution versions of the same image. Trucking, however, involves changes in visibility, so a single panorama will never be sufficient. Instead, the technique of multiplaning is used, where objects are grouped onto different multiperspective panoramas, each with it's own moving window. The partitioning is done by hand.

Multiperspective panoramas are a useful extension to image based animation, representing a further step away from realistic simulation towards convincing approximation or 'faking'. Their usefulness is limited because of the need for a predetermined path, and an incomplete range of camera motions. Because of this, multiperspective panoramas are the least general approach discussed in this chapter, unable to handle much of what is achieved in more general approaches like Talisman and animation using optic flow.

2.4.10 Depth maps and layered depth images

Depth maps augment two dimensional images with additional three dimensional information in the form of a depth value for each pixel. In graphics, this depth is the contents of the Z-buffer, computed during rendering, and in computer vision it could be obtained from a laser camera, or from an algorithm which extracts depth information.

In [27], Lemordant uses *fibre bundles* (a topological structure) to describe depth maps. A fibre bundle is a product between a base space (here, the two dimensional image) and a fibre (here, a depth value for each pixel of the image). When a new view of a depth map is required, it can be generated using a multiple-pass algorithm which decomposes the transformation into some passes which leave the depth unchanged, and others which change the depth but require no spatial displacement.

New views of depth maps entail a possible visibility change due to foldover, such as when part of a valley disappears behind a crest. Mcmillan and Bishop developed an entirely image based algorithm for resolving this in [31]. Their algorithm uses a particular drawing order which automatically determines visibility independent of the underlying geometry. The ordering guarantees that points arrive in the output image in back to front order, so that the painters algorithm can be used. Mcmillan and Bishop's algorithm is illustrated in Figure 2.4.10.

Gortler, He and Cohen extended depth maps to multiple layers in [13]. Their *layered depth images* include multiple pixels along each line of sight sorted in front to back

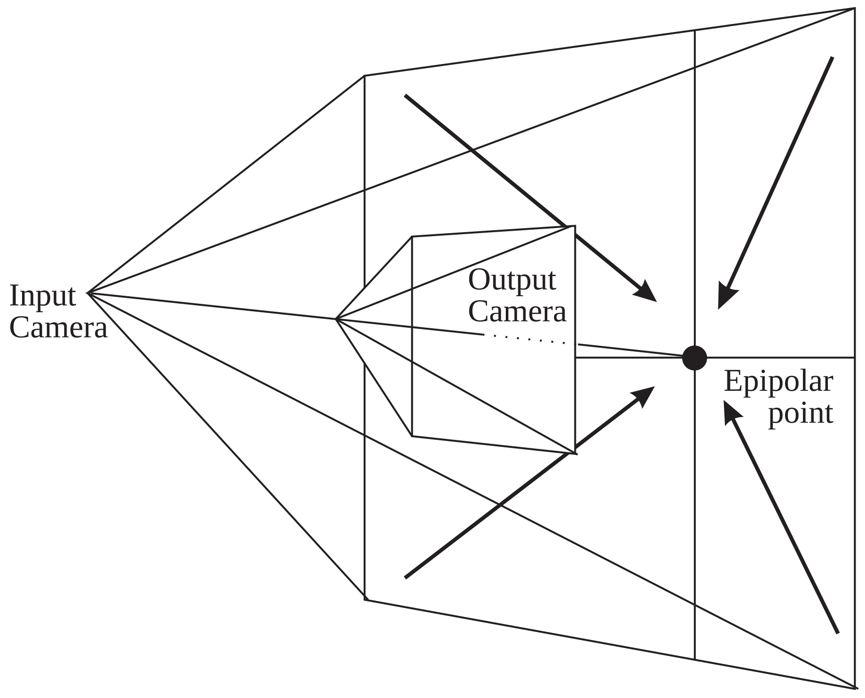


Figure 6: Mcmillan's ordering algorithm for computing visibility without depth. The epipolar point is the projection of the output camera in the input image. This point splits the input image into four quadrants. Depending on whether the input camera is in front of or behind the output camera, input pixels are processed inward towards the epipolar point or outward away from it. This guarantees that output pixels are produced in back to front order.

order. The front pixel value is the surface visible at that point, and further depth pixels are the hidden surfaces at that point. Unlike most other image based algorithms, this allows hidden surface to be included directly in an algorithm to generate new views.

Gortler et al make the important observation that Mcmillan's ordering algorithm determines visibility for layered depth images as well as depth maps. They have implemented a system for rendering layered depth images, producing five frames per second on a 200 MHz Pentium Pro, with a relatively unoccluded scene.

2.4.11 Layer based computer vision techniques

A layer based approach is also possible in computer vision. Image sequences are decomposed into overlapping layers with transparency information which allows them to be transformed independently using optic flow information but with the correct occlusion effects. The layering approach allows for image sequence compression with better compression ratios than the MPEG algorithm, and also for efficient image interpolation, since the foreground and the background can be separated into layers and interpolated independently.

Adelson [1] has also proposed a layered system which separates intensity, alpha, depth and so on into additional layers of an image. An additional layer includes the optic flow information, in the form of a velocity map. The technique identifies areas of the image sequence which are moving coherently, by using a least squares approach to identify regions with the same affine flow. This segmentation provides both the layering and the transformation between frames for each layer.

Adelson illustrated his work with an experiment which is a stronger version of Hofmann's scene-shifting. Three layers, a background building, a flower garden, and a foreground tree trunk, are extracted from an image sequence, and a moving figure from a separate sequence is added to the scene. The layers are used like Hofmann's scenes, but with an arbitrary affine flow applied between frames instead of Hofmann's approach which only allows for rotations and translations. The approach also allows real scenes to be animated, by extracting layers from a real image sequence, whereas Hofmann's scenes were rendered. It is fully image based because no model information is used to extract the layers from the image sequence.

As in our approach to animation, where objects or surfaces are rendered onto independent layers, computer vision becomes less complex when multiple layers are allowed. In both areas, this is because occlusion becomes simpler to handle. Unless we allow for separate layers, the discontinuities in optic flow at occlusion boundaries will always cause any approach based purely on optic flow to fail.

2.5 Light field rendering

The last approach to image based rendering which we will discuss is *light field* rendering, a somewhat different approach to the other image based techniques discussed here.

Two simultaneous 1996 papers [29, 12] developed the *light field* as a means of rendering different views of an object without having to store an explicit three dimensional model.

The *plenoptic function* represents the complete optic information available to a viewer at a given point. In general it is a five dimensional function giving the light information as a function of position and direction: what light reaches point (x, y, z) from direction (θ, ϕ).

An important simplification is that the plenoptic function reduces to four dimensions since light does not change along a given direction in open space. This simplification applies when the light field surrounds a convex object where visibility does not change along a given line.

Both Light Field Rendering [29] and the Lumigraph [12] store this four dimensional data set from a set of preacquired images (real or synthetic) which sample the light field. Very high compression rates may be obtained on the data set, due to coherence in each of the four dimensions.

A view of the object from any position corresponds to a slice of the four dimensional light field, and may be generated by sampling.

Light field rendering is included here for the sake of completeness in the course of our survey of image based rendering techniques, although it does not have direct relevance to the layered approach we will take to animation using optic flow analysis. An alternative approach to optic flow analysis could be formulated in terms of moving 'slices' of the four dimensional light field. Such an extension would be limited to static scenes, because of the nature of the light field.

2.6 Summary of image based rendering techniques

The various image based systems described in this chapter have a number of important differences and are suited to different situations. Below we have identified various criteria on which to assess the systems, which are then summarised in a table.

- **Independence of scene complexity**
  An ideal image based system would be independent of the scene complexity and the only dependency would be on the image. This is not always possible in practice: a typical problem is that holes appear when occlusion occurs (such as Chen and Williams' View Interpolation and McMillan and Bishop's Plenoptic Modeling.) Other techniques such as Talisman and Scene-shifting rely on multiple image layers or scenes, and are therefore dependent to an extent on the scene complexity.

- **Synthetic or natural scenes**
  Several of the methods presented are suitable for both synthetic and natural scenes, while others are only suitable for synthetic scenes. Some issues with natural scenes, for instance finding correspondences between points in different frames, fall into the computer vision field and do not concern us.

- **Interactivity and degrees of freedom**
  This looks at any restrictions on the viewer. Some systems allow the viewer

to move only within specific bounds, while others do not restrict the viewer. Scene-shifting allows the viewer to translate but only to look in one specific direction.

- **Method of handling occlusion**
  Occlusion presents an important problem to most image based rendering techniques. Some systems, including light field rendering and view morphing, rely on no occlusion occurring. View interpolation and plenoptic modelling use interpolation across the gaps caused by occlusion. Most other systems handle polygons or objects as multiple layers.

- **Moving or static scenes**
  An important distinction is whether the system can handle scenes with moving objects. Many of the systems discussed cannot; those that can include Talisman, scene-shifting and optic flow for animation.

- **Forward estimation**
  It may prove useful to be able to predict the user's motion for the next few frames, to allow pipelining.

- **Precomputation and storage requirements**
  Image based systems usually require some type of precomputation or storage, for instance to produce the key views in the View Interpolation system, or the light field data set.

- **Orders of optic flow considered**
  We have also looked at the accuracy of the image transformations used by each system, breaking them down into translation or scaling, affine or higher order. Interestingly, Talisman, which is the most general system we have considered, considers affine to be sufficient for its purposes. This is because of the expense of computing the required perspective transformation. Our approach to animation using optic flow analysis allows this transformation to be calculated for little more expense than the affine transformation.

The approaches we have examined in this chapter are summarised according to these criteria on page 40.

Summary of image based rendering methods discussed in this chapter

	Independence of scene complexity	**Synthetic or natural scenes**	**Interactivity and degrees of freedom**	**Moving or static scenes**	**Method of handling occlusion**	**Forward estimation**	**Precomputation and storage requirements**	**Orders of optic flow considered**
View morphing (Section 2.3.1)	Yes	Natural	In between two existing views	Either	No occlusion	No	Two key images per morph	N/A (direct interpolation between views)
View interpolation (2.3.2)	Partial (increased refinement of quadtree in complex scenes)	Synthetic	Any motion	Static	Interpolation across gaps	Yes (but holes appear)	Large set of precomputed views of the scene from key points	Translation of regions
Plenoptic Modeling (2.3.3)	Yes	Natural	Any motion	Static	Interpolation across gaps	No	Two cylindrical views of scene	N/A (interpolation using point correspondence)
Scene shifting (2.4.1)	Partial (affects number of layers)	Synthetic	Partially restricted to one viewing direction	Either	Multiple layers	Yes	One image of each object	First order (scaling, rotation)
Optic flow for animation (2.4.2)	No (one layer per polygon)	Synthetic	Any motion	Either	Multiple layers	Yes	One image of each polygon	Second order
Talisman architecture (2.4.6)	No (one layer per group of polygons)	Synthetic	Any motion	Either	Multiple layers	No (possible but not included)	One image of each object	First order (affine)
Light field rendering (2.5)	Yes	Synthetic or natural	Outside convex hull of object	Static	Convex objects (no occlusion)	N/A (static scene)	Large 4D data set	N/A (not based on optic flow)

2.7 Chapter Summary

In this chapter we have motivated the need for an alternative to physical simulation in rendering, and summarised the existing types of image based rendering. These are categorised into three areas.

Light field rendering uses a four dimensional representation of the complete optic information surrounding an object. Views are produced by taking two dimensional 'slices' of this field. We have not made any use of this type of image based rendering in our further work in this dissertation.

Image interpolation techniques, which are used to produce intermediate views from a set of precomputed or prestored images. The set of these images restricts which intermediate views can be produced, but animation is independent of scene complexity. Warping and morphing are used to produce intermediate views, as in image layer systems, and there is some overlap between image interpolation and layered systems, for instance in View Interpolation [6].

Image layer techniques subdivide the scene into independent layers, using two dimensional transformations on these layers and then compositing them to create the final image. This is the most natural approach to image based rendering, and it is the one we have chosen in later chapters. Frame to frame transformations are based on two dimensional image manipulation of the layers instead of three dimensional rendering, thereby exploiting frame to frame coherence. Our approach will be similar to the Talisman approach, but using optic flow analysis to derive the transformation between frames.

The Talisman architecture uses affine warps and concludes that perspective warps are both too expensive to compute, and also of little benefit. Because the optic flow derivatives give us the entries in either an affine or a perspective matrix (Section 5.2), we are able to compute perspective transformations at little extra cost over affine transformations.

Chapter 3

Scene shifting by optic flow analysis

3.1 Introduction

In this chapter we will investigate two different mathematical approaches to animation, one discrete and one continuous. In comparing the two approaches we will discuss a fundamental difference between them: that of finite versus infinitesimal changes. However we will also show that this difference can be avoided and that the approach we have used in Chapter 5, using infinitesimal changes, is at least as general as the approach using finite changes.

The case study we will use in this chapter is a comparison between optic flow for animation, and Hofmann's approach using 'scene shifting' [17] for image based animation using image layers. In this chapter we will demonstrate that optic flow for animation is a general enough technique to handle all the cases handled by scene shifting.

Scene shifting was discussed in section 2.4.1 of Chapter 2. In summary, it partitions the three dimensional scene into parallel slices which are rendered separately into image layers or *scenes*. Depending on then viewer's motion, each layer is then translated, scaled or warped independently to create the animation. Hofmann developed a calculus to determine how the partition into layers is determined, and how each layer is transformed. These transformations are discrete, based on the finite changes ($\Delta \mathbf{x}$) of the image points between frames.

Optic flow analysis, discussed in Section 2.4.2 of Chapter 2 looks at instantaneous changes in the moving image. In image based animation it can be used to construct subsequent frames from the current frame by computing the changes in the image. The calculations are based on instantaneous velocities of the image points ($d\mathbf{x}/dt$), extrapolated over the time interval Δt between frames.

In the first section of this chapter, we describe mathematically the process whereby a stationary three dimensional scene seen by a viewer is rendered as a raster picture. The second section deals with changes in the image due to changes in the position or orientation of the viewer. Each alteration described by Hofmann is described here in terms of optic flow. We then use these formulae in the next section to derive the technique of scene shifting, and finally show how the differences between the two approaches can be overcome.

3.2 Scene shifting: a layered approach to animation

Scene shifting is an image based rendering technique based on partitioning the three dimensional scene into parallel slices which are rendered separately into image layers or *scenes*. Depending on then viewer's motion, each layer is then translated, scaled or warped independently to create the animation.

In [17], G.R. Hofmann presents a calculus which determines the partition of the scene into slices, and what transformation should be applied to each image layer.

Hofmann's calculus also permits a hierarchical approach to animation: depending on their motion, different layers may be updated using different transformations, and at different frequencies, thus allowing computational resources to be allocated according to the importance of that part of the scene to the animation. For instance a faraway or slow-moving object may not change at all from frame to frame, while a closer one may need to be translated or scaled at every frame.

The key to this is to find an image transformation which which will be correct for a portion of the scene within a tolerance ε. The collection of scene points to which this applies is called an ε-invariant subspace. For instance, if the transformation is a translation, then all image points within the subspace are translated by the same amount within a variance of at most ε pixels.

We can now proceed to the mathematical details of scene shifting which were not covered in section 2.4.1 of Chapter 2.

3.3 Calculus of the Exact Perspective Projection

3.3.1 Definitions

Hofmann provides us with a formulation for the perspective projection of a three dimensional scene which we will follow closely.

The three dimensional *scene* is represented by $\Sigma = (V, O, L)$, consisting of a viewer V, a set O of three dimensional objects and a description L of the lighting of the scene. We will not be looking any further at either O or L.

The *viewer* is a 4-tuple

$$V = (E, D, \omega, U)$$

where $E \in \mathbf{R}^3$ gives the position of the eye or camera, $D \in \mathbf{R}^3$ is the viewing direction, and the viewer's up-direction is $U \in \mathbf{R}^3$.

The vectors D and U are assumed to be orthonormal, so we can specify U as an angle in radians anticlockwise from some fixed direction. Alternately, we could specify both U and D simultaneously using a unit quaternion.

A *normalised viewer* V_n has position at the origin $E_n = (0, 0, 0)$, viewing direction along the Z axis $D_n = (0, 0, 1)$, and up-direction $U_n = (0, 0, 1)$ in the Y axis.

The quantity $\omega \in \mathbf{R}$ defines the angular width of the viewing frustum, like the effect of a camera's zoom lens. The linear scaling on the image plane is then given by the factor $k = \tan^{-1}(\omega)$. We will ignore this factor in our calculations, assuming $k = 1$, i.e. a viewing angle of $\omega = 45°$.

As before the scene Σ normalised viewer is projected onto the image plane $Z = 1$ by the function τ $\{(x, y, 1)|x, y \in \mathbf{R}\}$:

$$\tau : (x, y, z > 1) \longmapsto (k\frac{x}{z}, k\frac{y}{z}, 1) \qquad (2)$$

3.3.2 From world to image

The mapping from the world to the display image can be broken down into two steps: firstly, the projection onto the plane $Z = 1$, which is effectively an image of infinite resolution, and secondly a sampling step which produces the discrete image.

The continuous image $I(\Sigma)$ is a mapping from the continuous image plane to the colour space. The map I includes projection onto the image plane (using the map τ above), and takes into account visibility and lighting. $I(\Sigma)$ is an image of infinite resolution since it operates on the continuous plane $\mathbf{R}^2$ and not the discrete one $\mathbf{Z}^2$. [1]

The sampling step (Figure 7) produces a discrete image, or raster picture, which Hofmann calls a *scenery*. The discrete image is a map $\sigma(\Sigma) : \mathbf{Z}^2 \to C$ which assigns to each discrete point or pixel (x, y) a colour in the display colour space C.

3.4 Calculus of the Non-exact Perspective Projection

In the previous section we looked at a static situation where the characteristics of the viewer did not change. In order to extend the calculus above to develop new tools for

[1] In the formulation by Hofmann, I is first defined as the plane $Z = 1$, but later $I(\Sigma)$ is the image of a particular scene. Our formulation clears up this inconsistency.

$$\begin{tikzcd}
\boxed{\text{Scene } \Sigma} \arrow[d, "\text{projection, lighting, visibility}"] \\
\boxed{\text{Continuous image } I(\Sigma)} \arrow[d, "\text{sampling}"] \\
\boxed{\text{Discrete image } \sigma(\Sigma)}
\end{tikzcd}$$

Figure 7: The mathematical steps from world to image coordinates

animation, Hofmann developed the Calculus of the Non-exact Perspective Projection. Below we will explore each type of alteration to the viewer and explain the change induced on the image in terms of optic flow.

Suppose we have a normalised viewer V_n which undergoes a general alteration

$$\Delta V = (\Delta E, \Delta D, \Delta\omega, \Delta U),$$

consisting of a translation in space, a rotation in three dimensions, and a zoom. The translation is represented by $\Delta E = (\Delta X, \Delta Y, \Delta Z)$. The rotation is defined by the change in the viewer's up-direction ΔU, an angle measured anticlockwise in radians, and by $\Delta D = (\Delta\Phi, \Delta\Psi)$, the change in viewing direction given in spherical polar coordinates. The zoom is represented by $\Delta\omega$.

Under the change in viewing direction, the coordinates of the point (X, Y, Z) are transformed,

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \overset{\Delta D}{\longmapsto} \begin{pmatrix} X - Z\Delta\Phi \\ Y - Z\Delta\Psi \\ Z + X\Delta\Phi + Y\Delta\Psi \end{pmatrix} = \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix}$$

where we have approximated $\cos(\Delta\Phi)$ by $\Delta\Phi$, $\sin(\Delta\Phi)$ by 0, and made the same approximations for $\Delta\Psi$.

We denote by $\mathbf{Rot}_\theta$ the matrix $\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$.

The original projection of the scene point $(X, Y, Z > 1)$ is given by

$$(X, Y, Z) \longmapsto tan^{-1}(\omega)(\frac{X}{Z}, \frac{Y}{Z}) = (x, y) \tag{3}$$

The new projection is given by

$$(X, Y, Z) \longmapsto \mathbf{Rot}_{\Delta U} \tan^{-1}(\omega + \Delta\omega)(\frac{X'}{Z'}, \frac{Y'}{Z'}) = (x', y') \tag{4}$$

We can now look at some special cases and formulate them in terms of optic flow equations, that is to say partial derivatives with respect to time and image coordinates, in place of the finite changes used by Hofmann.

3.4.1 Changes ΔE in the viewer: translation and scaling

Firstly suppose the viewer is translated by an amount $\Delta E = (\Delta X, \Delta Y, \Delta Z)$. Then the transformation from frame to frame is

$$k(\frac{X}{Z}, \frac{Y}{Z}) \overset{\Delta E}{\longmapsto} k(\frac{X - \Delta X}{Z - \Delta Z}, \frac{Y - \Delta Y}{Z - \Delta Z})$$

So

$$\Delta \mathbf{x} = k(\frac{X - \Delta X}{Z - \Delta Z} - \frac{X}{Z}, \frac{Y - \Delta Y}{Z - \Delta Z} - \frac{Y}{Z})$$

Now there are two special cases, corresponding to translation in the image plane XY or along the viewing direction Z. For translation in X and Y we have $\Delta Z = 0$, and the velocity of an image point can be found in terms of the velocity of the moving viewer:

$$\frac{\partial \mathbf{x}}{\partial t} = -\frac{k}{Z}(\frac{\partial X}{\partial t}, \frac{\partial Y}{\partial t}) \tag{5}$$

In the case of translation in Z, we have $\Delta X = \Delta Y = 0$, and the velocity of an image point is then

$$\frac{\partial \mathbf{x}}{\partial t} = \frac{k}{Z}\frac{\partial Z}{\partial t}\mathbf{x} \tag{6}$$

These two types of motion (translation in Z and in the XY-plane) make up the foundation of scene shifting, and in Section 3.4.4 we will explain scene shifting in terms of the above two equations.

3.4.2 Changes ΔU in the viewer: rotation

Secondly, assume that there is a change ΔU in the viewer's up-direction, i.e. a rotation about the viewing axis. Then, from (4), the new projection after rotation is

$$(X, Y, Z) \overset{\Delta U}{\longmapsto} \mathbf{Rot}_{\Delta U} \tan^{-1}(\omega)(\frac{X}{Z}, \frac{Y}{Z})$$

So

$$(x'y') = \mathbf{Rot}_{\Delta U}(x, y)$$

This is just a rotation of the original image, as expected.

We can formulate this simple transformation in terms of partial derivatives with respect to the image coordinates. The change in the image point is

$$\Delta \mathbf{x} = \mathbf{Rot}_{\Delta U}(x, y) - (x, y)$$

To find the instantaneous velocity of the image point, we take $\frac{\Delta \mathbf{x}}{\Delta U}$ and let $\Delta U \to 0$, and then multiply by $\frac{\partial U}{\partial t}$ (the angular velocity of the viewer). The velocity of an image point $\mathbf{x} = (x, y)$ is then

$$\frac{\partial \mathbf{x}}{\partial t} = (-y, x)\frac{\partial U}{\partial t} \tag{7}$$

As expected the instantaneous velocity is equivalent to an image rotation about the origin. However, if we take this instantaneous velocity and extrapolate it over a finite time interval Δt, we find that there is a difference between this and the finite rotation that we expect. This is because the instantaneous velocity is extrapolated linearly, tangentially to the circle of rotation, rather than along the curved circumference. The net result is that the image rotates by the correct amount, but also expands slightly. The result is a good approximation to the correct rotation, but the difference is cumulative, and can become significant after several frames.

Although this is an inherent shortcoming to using instantaneous velocities rather than finite changes, it can be avoided. As we will see in Chapter 4, by keeping track of higher order derivatives (not just the image velocities), we can keep track of these errors.

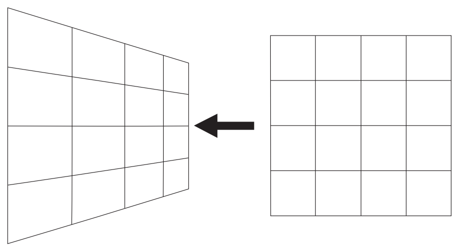


Figure 8: The distortion of the image required to implement a change in the viewing direction

3.4.3 Changes ΔD in the Viewer

Suppose the viewing direction changes by an amount $\Delta\Phi$, in other words the viewer pans horizontally to the left. The original point $(x, y) = k(\frac{X}{Z}, \frac{Y}{Z})$ goes under the transformation to the point

$$\begin{aligned} & k(\frac{X - Z\Delta\Phi}{Z + X\Delta\Phi}, \frac{Y}{Z + X\Delta\Phi}) \\ = \; & k(\frac{x - \Delta\Phi}{1 + x\Delta\Phi}, \frac{y}{1 + x\Delta\Phi}) \end{aligned}$$

Now we calculate $\frac{\Delta \mathbf{x}}{\Delta\Phi}$ to find the optic flow:

$$\begin{aligned} \frac{\Delta \mathbf{x}}{\Delta\Phi} &= \frac{1}{1 + x\Delta\Phi}(-1 - x^2, -xy) \\ \Rightarrow \frac{\partial \mathbf{x}}{\partial t} &= (-1 - x^2, -xy)\frac{\partial\Phi}{\partial t} \end{aligned} \tag{8}$$

The motion of image points here depends only on their image coordinates, and not on the three dimensional depth information Z. This means that for a rotating viewer, the image may be reused provided it is distorted suitably. But the transformation derived from the optic flow in Equation 8 is only an approximation to the correct transformation, which is illustrated in Figure 8. The optic flow approximation results in a slightly curved trapezium-like shape, whereas the correct transformation, using a finite change, is a trapezium. We will see exactly this problem in Section 4.7 of Chapter 4. Again it is solved by keeping track of higher order derivatives to ensure that the error remains within certain bounds.

3.4.4 Scene Shifting

Hofmann introduces scene shifting by referring to a filmed simulation of the view from the window of a moving train. The simulation relied on three moving canvases, one for the bushes beside the train, one for the middle distance landscape, and one for the far landscape, horizon and sky. Each belt painted canvas moved past at a different speed: the nearby canvas moving the fastest and the distant one moving the slowest. This background was seen through a rain-blurred window, giving a strong illusion of motion.

We can use the formulae in the previous sections to determine how to divide the scene into slices for scene shifting.

In the case of the view from a train window, the motion of the viewer is at right angles to the viewing direction D (Equation 5), at some velocity $\mathbf{V}$. The velocity of an image point then depends only on its depth Z. If we want to know which points fall within the same 'slice' as a point with depth Z, we must make sure that the velocities of their image points differ by less than some amount ε.

So, we want to maximize W (the width of our slice) such that

$$|\frac{k}{Z}\mathbf{V} - \frac{k}{Z+W}\mathbf{V}| < \varepsilon$$

That is,

$$W = \frac{\varepsilon Z^2}{\max\{\delta, k|\mathbf{V}| - \varepsilon Z\}} \tag{9}$$

where $\delta > 0$ is a very small real number. W as a function of Z has an asymptote at $Z_\infty = \frac{\varepsilon}{k|\mathbf{V}|}$, which means that all points with depth greater than Z_∞ fall into one infinitely wide slice (the far background slice) where all image points have velocity less than ε. [2]

Something similar happens for a viewer moving in the direction of viewing (the viewer here could be the train driver rather than a passenger). As derived in Equation 6, for points on a plane at fixed depth Z, the resulting image motion is just an expansion

[2]The quantity δ prevents the mathematics from producing anything absurd from happening beyond the asymptote at Z_∞.

about the center by a factor $-\frac{k}{Z}\mathbf{V}$. This expansion factor is greatest for the nearest slices, and zero for the slice beyond Z_∞. Grouping points from planes at different depths such that their scaling factors differ by at most ε gives us the same width W for the slice at depth Z which we derived for the previous case in Equation 9.

3.5 Optic Flow for Computer Animation

Optic flow has been used for computer animation by Blake [5], using the mathematics developed by, among others, Waxman and Wohn [41] for use in the field of computer vision. The velocity of each image point is computed in terms of the translational and rotational motion of the object relative to the viewer, to derive the optic flow field $\mathbf{v}(\mathbf{x}, t)$. The subsequent image after time Δt is extrapolated from the current image using this field, assuming that the velocity of each image point remains constant during this time.

In Blake's work, the field $\mathbf{v}$ is then decomposed by means of a Taylor series in x and y, where the zero order terms represent translation, first order represent affine transformations, and higher orders represent perspective effects, the effect of object curvature and so on. For a planar object, the terms beyond second order are zero.

Blake then used this decomposition to compute a warp of the original image which would produce the subsequent frame in an animation sequence. To speed this process, if all orders above a certain order, N, are insignificant, the animation is said to have Nth order frame to frame coherence, and only effects up to Nth order are used to produce the subsequent frame.

3.6 Discussion and comparison

3.6.1 Limitations of scene shifting

Scene shifting relies on a partition of the world into parallel slices and computing an image of each slice. In the case of a viewer translating perpendicular to the viewing direction, this is ideal, since the subdivision into slices is unchanged by the translation. Each scene remains at a constant depth away from the viewer. This subdivision is also unaffected by changes to the viewer's up direction.

In the case of motion in the viewing direction, however, the depth of the scenes is changing as the viewer moves towards or away from them. This means that points initially in the same scene slice may end up in another. Points near the edge of the original slice may be transferred to the adjacent slice. This means that the image of the slice may change at each frame. However, this movement of points between slices has less impact on more distant slices, which means updates on more distant scenes could happen less frequently.

In addition to implementing translation of the viewer ΔE using scene shifting, we noted earlier that for a change in the viewing direction ΔD may be implemented by a warp of the original image. Although this can be simply implemented, it causes an unwanted complication. Note that in previous cases, the slices keep their orientation before and after the translation. The sequences examined in Hofmann's Thesis [18] cover viewers whose viewing direction remains constant.

In contrast, if the viewing direction changes by ΔD, all the scene slices have to be recalculated. This affects all slices, independent of depth, and is an important shortcoming to the technique.

3.6.2 Limitations of optic flow for animation

The significance of the optic flow approach over scene shifting is that it can be used for any translational or rotational motion, whereas scene shifting relied on constant viewing direction.

In our attempt to use optic flow to reformulate the transformations derived by scene shifting, we have noted two cases where the instantaneous velocities produce transformations which are only approximations to the correct transformations. This difference between finite and infinitesimal changes applies whenever a rotation of the viewer occurs.

The problem arises because instantaneous velocities must be extrapolated linearly over a finite time interval Δt, which introduces errors since the real path of the image points may be curved rather than linear. This accumulated error can be detected and taken into account by looking at higher order derivatives of the optic flow field, which is discussed in some detail in Chapter 5.

3.7 Chapter Summary

In Section 3.4.1, we used optic flow to derive the various alterations of the viewer described by Hofmann. We were able to derive the same division of the scene into slices, and the same results in the case of translations of the viewer.

In the case of a rotating viewer, we found that extrapolating from the instantaneous velocity only gives an approximation to the correct transformation; however, in Chapter 4 we will show how higher order derivatives of the optic flow field will allow us to keep these errors within bounds.

The result is that we have successfully derived scene shifting as an application of optic flow analysis. Functionally, we have not yet produced any results which could not be derived using Hofmann's formulation, but in the Chapter 4 we will develop optic flow into a more general tool for animation, deriving the image motion corresponding to a general world motion.

There are several advantages to using optic flow analysis. As an analytic tool, optic flow gives us access to a more powerful toolbox including such techniques as Taylor series. Since it is already in use in the closely related field of computer vision, the mathematics of optic flow can easily be adapted to animation, allowing us to exploit results from computer vision. In addition, optic flow analysis is a general approach which can be used for any translation or rotation of the viewer.

Hofmann's initial paper on scene shifting [17] has not been developed further in the literature, although it has been referred to in several papers, notably by Chen and Williams in [6], and similar work has been done on multiperspective panoramas [44] (see Section 2.4.9). However, since scene shifting achieves a significant time saving in animation through a particularly simple model, it is important to know that our choice of optic flow analysis will be capable of capturing all the aspects of scene shifting.

Noting the limitations of scene shifting in dealing with changes in the viewing direction, and also having shown that optic flow is a more general technique than scene shifting, we have not proceeded with scene shifting but rather developed an animation algorithm using optic flow. This algorithm will be developed in the subsequent two chapters.

Chapter 4

Optic Flow Analysis

4.1 Introduction

In this chapter we present a theory of optic flow analysis, including both previous work and our own contribution to this theory for animation using optic flow. We introduce the optic flow field as described in previous work, and describe how a Taylor analysis of the field can be used to derive frame to frame image transformations in animation. We concentrate on animating planar surfaces because the Taylor series simplifies in this case.

Our contribution is to extend the analysis up to second order Taylor terms, and categorise the frame to frame transformations for a moving planar object. To do this we have introduced a new mathematical basis for these second order terms. This basis is much simpler than the second order Taylor terms, and in Chapter 5 we will relate the terms of the basis to a perspective warp between frames. Previous animation algorithms using optic flow have found calculating frame to frame perspective warps complicated to derive; however our method makes them only marginally more difficult than affine transformations.

In Chapter 5 the theory derived in this chapter is used in an image based rendering algorithm based on optic flow analysis.

4.1.1 Background

Optic flow describes the way points in a moving image sequence evolve over time. In Section 2.4.2 of Chapter 2, its relevance to computer vision, image sequence compression and animation was discussed.

In animation, the key to using optic flow is as a means of describing the differences (changes) between consecutive frames. This allows the previous frame to be reused providing the correct changes are applied to the image. This allows us to move our animation calculations from the three dimensional world to the two dimensional image, thus exploiting the coherence between consecutive frames and consequently saving calculation time.

Section 4.2 of this chapter expresses the flow between consecutive frames using standard graphics and texture mapping methods, also introducing the tool of homogeneous matrices which we use throughout this and subsequent chapters.

The mathematical theory underlying optic flow was originally derived in the context of computer vision (Section 4.3) by Koenderink [23], whose theory was based on vector field quantities such as grad, curl, div and def. Waxman et al [40, 43, 42] used the notation we have used and also made the simplifications for a planar object (Section 4.6) and derived various sets of basis deformations for use in detecting three dimensional structure and motion from the two dimensional moving image. Their work expresses the optic flow field as an instantaneous field of velocities, and breaks it down using a Taylor series approximation.

This work was used as a tool in computer animation by Blake [5], who used the work of Koenderink, and Waxman et al to produce an image based animation of a moving planar object using derivatives of the optic flow field. Perspective effects were not accounted for, and only affine transformations used.

4.1.2 Our contribution to this field

Our contribution to this field is an extension of Blake's algorithm which includes perspective effects. To do this we have extended the approximations of the optic flow field up to second order derivatives (Section 4.4), which gives us a more accurate approximation than in earlier work.

The work by Waxman et al [40, 43, 42] using partial derivatives in the x and y directions, is better suited to image synthesis than the earlier work by Koenderink [23] using div, grad, etc. As far as we know, this is the first time work such as Waxman's from the computer vision field has been used in computer animation.

We have also simplified the very complex set of transformations represented by the second order Taylor series terms. The simplifications apply in the case of the optic flow of a moving planar object, and are obtained by taking linear combinations of the Taylor terms.

Unlike previous bases, each term in our basis relates strongly to a particular motion of a planar object in three dimensions when projected onto the image plane: translation in X and Y are just image translations, translation in Z is image scaling, and rotation about the viewing axis is just rotation of the image. We introduce two more terms which are approximations to perspective warps, and which relate closely to rotation about the other two axes.

Finally, we have related our new basis terms to the entries in a 3×3 homogeneous matrix, which gives us the required texture mapping parameters.

In Chapter 5, we will use this theory to develop a new image based animation algorithm based on optic flow. It will be based on a hierarchical description of the transitions between consecutive frames, the different levels of the hierarchy corresponding to terms in the Taylor series decomposition of the optic flow field.

4.2 Optic flow between consecutive frames

This section serves as an introduction to the graphics and texture mapping techniques we will use throughout this and subsequent chapters. We will derive the optic flow between two images of a planar surface in terms of the relative changes between the

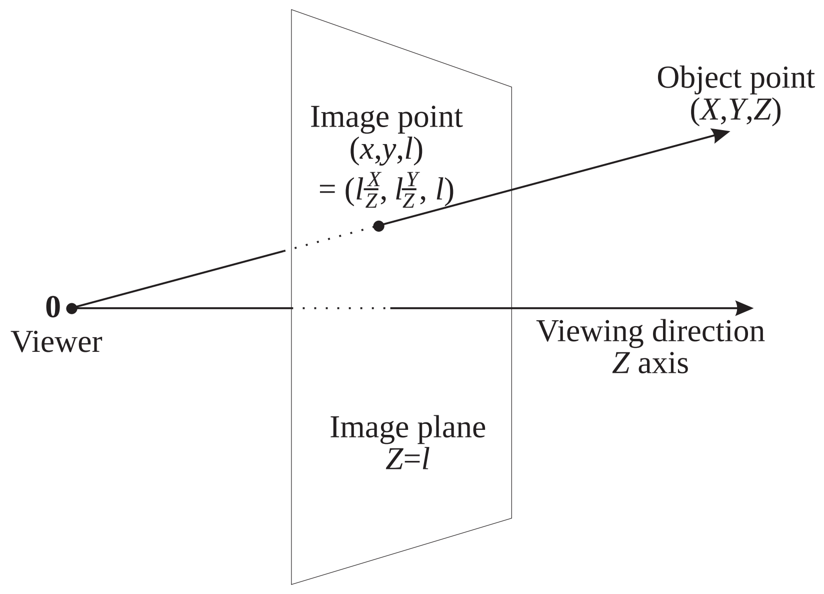


Figure 9: Projection of a point (X, Y, Z) onto the image plane $Z = l$

viewer and object. Texture mapping using homogeneous matrices is the main tool used here.

4.2.1 Projection

The projection equation which we use throughout this work is illustrated in Figure 9. Using a pinhole camera model, points (X, Y, Z) in the three dimensional scene are projected onto the plane $Z = l$ by the equation

$$\rho : (X, Y, Z) \longmapsto (x, y, l) = l(X/Z, Y/Z, 1) \tag{10}$$

As a convention, use upper case refer to scene points and lower case refer to image points. In most cases the image plane is at a depth of $l = 1$, and (x, y) is often written instead of $(x, y, 1)$ where the distinction is not necessary.

4.2.2 Transformations on an image

Image transformations can be grouped into different classes of increasing levels of complexity.

- The simplest transformation is a *translation* in x or y direction. This can be trivially implemented, either by simply relocating the bitmapped image on the display, or, when subpixel accuracy is required, by a resampling step. Translation of the image corresponds to zero order optic flow, which was discussed in Section 2.4.2.

- *Affine transformations* are linear transformations of the image, corresponding to second order optic flow. They are represented in homogeneous matrix notation as

$$\begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} ax+by+c \\ dx+ey+f \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{11}$$

  Affine transformations include elementary operations such as scaling, rotation and shearing. Linear operations such as these are also relatively easy to implement.

- An important class is that of *perspective transformations.* Mathematically these are known as *Möbius transformations*, a ratio between two linear expressions. They can be expressed in homogeneous matrix notation:

$$\begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \frac{1}{gx+hy+i} \begin{pmatrix} ax+by+c \\ dx+ey+f \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{12}$$

  Perspective transformations are the most general two dimensional transformations which preserve straight lines. Unlike linear transformations, which of course also preserve straight lines, uniform spacing on a straight line is not preserved under a perspective transformation.

These transformations are exactly what is required to perform perspective texture mapping, where a planar texture is mapped to the display using the Projection Equation 10. This is because the map from texture to world coordinates is affine, and the transformation from world to image coordinates is just homogenising (division) by the Z-coordinate.

An important mathematical feature of perspective transformations is that they are not preserved under linear combination. This is because in general their sum is a ratio of quadratics and not linear expressions. This has important implications: we cannot do a linear interpolation on two perspective views to obtain an intermediate view, as we saw in Section 2.3.1. However perspective transformations are preserved under inverses, matrix multiplication and multiplication by a scalar.

Perspective transformations are inefficient to implement due to the 'division by Z' requirement. Often a quadratic approximation is used instead.

4.2.3 The transformations from object to world to image coordinates

With the basics of projection and image transformations explained in the previous two subsections, we now derive the mapping used to produce an image of a planar surface.

The surface is represented as a planar texture, and a point in this texture has coordinates (u, v). In order to work with 4×4 homogeneous matrices, we need two extra coordinates: $(u, v, 0, 1)$. This point is transformed into world coordinates by the homogeneous matrix $M_{texture}$:

$$M_{texture} \begin{pmatrix} u \\ v \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & t_1 \\ m_{21} & m_{22} & m_{23} & t_2 \\ m_{31} & m_{32} & m_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{13}$$

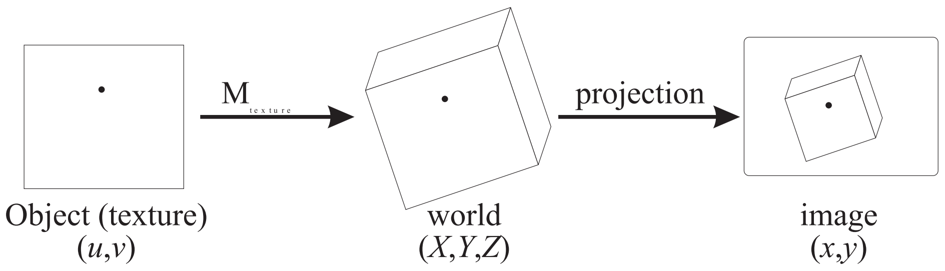


Figure 10: The texture coordinates (u, v) are first mapped to world coordinates using the mapping $M_{texture}$ and then projected to the image by a homogenising step (See Equation 14)

Here (t_1, t_2, t_3) is the origin of the texture coordinate system in world space, and 3×3 matrix (m_{ij}) is the matrix expressing the orientation of the texture coordinate system, i.e. the orientation of the planar surface, in world coordinates. ($M_{texture}$ is a unitary matrix multiplied by a scaling factor). The vector m_{i1} is the texture's x-direction, m_{i2} is the y-direction, and m_{i3} is the normal to the plane onto which the texture is mapped.

The projection from world to image coordinates is by means of Equation 10, which is just homogenising by the third coordinate.

So the transformation from texture to world to image coordinates, shown in Figure 10, is given by the matrix $M_{texture}$ and a homogenising step. (We discard the last coordinate.)

$$M_{texture} \begin{pmatrix} u \\ v \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \equiv \begin{pmatrix} X/Z \\ Y/Z \\ 1 \end{pmatrix} \tag{14}$$

4.2.4 Optic flow expressed in terms of object motion

For our synthetic animation we have full information about the object's motion and can therefore compute the optic flow exactly by projecting this motion onto the image plane. This means calculating the image-space transformation for a given world-space transformation.

Because we know the underlying model which produced the current image, we can backproject the image to the model, apply the world space transformation, and then project to image space for the subsequent frame.

First we have to define the world-space transformation. Any motion of our rigid planar surface can be expressed by means of a rotation about the object's origin and a translation in world space. The rotation is expressed by means of a 3×3 unitary matrix (n_{ij}) and the translation by a vector (T_1, T_2, T_3). If the texture coordinate system origin is (t_1, t_2, t_3), then we first have to translate back to the world origin, perform the rotation and translation there, and then translate back. The whole transformation in world space is given as follows:

$$\begin{aligned}\begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} &= T_{world} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} n_{11} & n_{12} & n_{13} & T_1 + t_1 \\ n_{21} & n_{22} & n_{23} & T_2 + t_2 \\ n_{31} & n_{32} & n_{33} & T_3 + t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -t_1 \\ 0 & 1 & 0 & -t_2 \\ 0 & 0 & 1 & -t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \end{aligned} \tag{15}$$

To backproject from image to world space, we have to use the plane equation of the surface we are projecting. The plane equation is of the form

$$Xm_{13} + Ym_{23} + Zm_{33} = k = t_1 m_{13} + t_2 m_{23} + t_3 m_{33} \tag{16}$$

where (m_{i3}) is the normal to the planar surface as before. We also know, from the projection equation, that $x = X/Z$ and $y = Y/Z$.

It is then possible to find the backprojection matrix P_{obj}^{-1} specific to this planar surface:

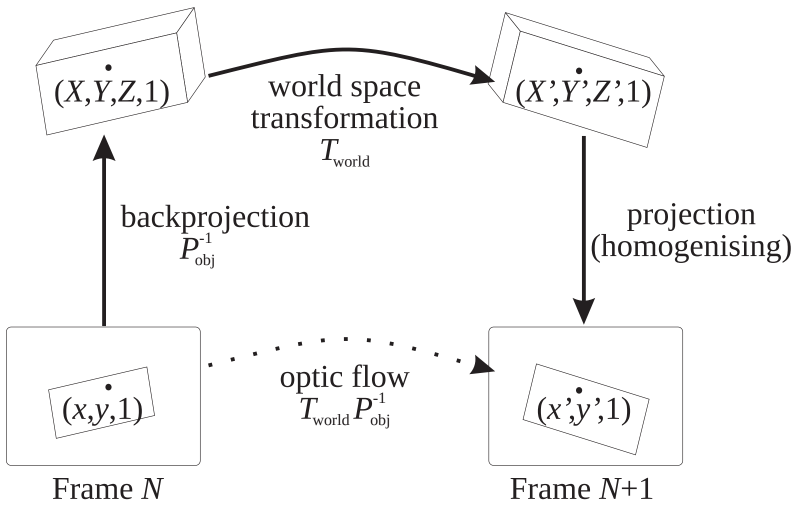

(X,Y,Z,1)
world space transformation T_{world}
(X',Y',Z',1)
backprojection P^{-1}_{obj}
projection (homogenising)
(x,y,1)
optic flow $T_{world}P^{-1}_{obj}$
(x',y',1)
Frame N
Frame N+1

Figure 11: The optic flow between two frames N and $N+1$ is found by first back-projecting frame N to world space using the matrix P_{obj}^{-1} (which relies on knowing the surface which is shown in the frame), and performing the transformation T_{world} in world space before projecting back to produce frame $N+1$.

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ m_{13} & m_{23} & m_{33} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \\ w \end{pmatrix} = P_{obj}^{-1} \begin{pmatrix} x \\ y \\ 1 \\ w \end{pmatrix} \tag{17}$$

Finally, the image transformation is $T_{world}P_{obj}^{-1}$, which is of this form:

$$T_{world}P_{obj}^{-1} = \begin{pmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & 1 & 0 \\ m_{13} & m_{23} & m_{33} & 0 \end{pmatrix} \tag{18}$$

Although $T_{world}P_{obj}^{-1}$ is a 4×4 matrix, we are interested in its action on homogeneous image coordinates which are 3-vectors, so we can drop the last column and row, and further simplify by homogenising by the $(3,3)$ element of the matrix.

This leaves us with the optic flow expressed in matrix form in terms of image coordinates:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} : \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{19}$$

It will also be useful to decompose this into a product of translation, affine and perspective transformations:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & c \\ 0 & 1 & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a-cg & b-ch & 0 \\ d-fg & e-fh & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ g & h & 1 \end{pmatrix} \tag{20}$$

We have succeeded in expressing the optic flow for a moving planar surface in terms of a 3×3 homogeneous matrix. This gives an image-space transformation between frames. This was done by means of backprojecting from the image to world space, which relied on knowledge of the surface being projected, using the world space transformation, and projecting back to the next image.

In the next section we will look at a means of decomposing the optic flow field in terms of instantaneous velocities, rather than the absolute transformation between frames.

4.3 The instantaneous optic flow field

In this section we will look at the instantaneous flow field and express it in terms of the world space motion of the object. This field can be broken down using a Taylor series, which gives us a hierarchical means of rendering motion. Only those types of motion which represent significant terms in the Taylor series need to be implemented.

In order to use optic flow information for an animation of a moving object, we need the object's position $\mathbf{R} = (X, Y, Z)$, instantaneous translational velocity $\mathbf{V} = (V_1, V_2, V_3)$ and angular velocity $\boldsymbol{\Omega} = (\Omega_1, \Omega_2, \Omega_3)$. These motions are illustrated in Figure 12. (It is assumed that all motions are expressed in the world coordinate system of a

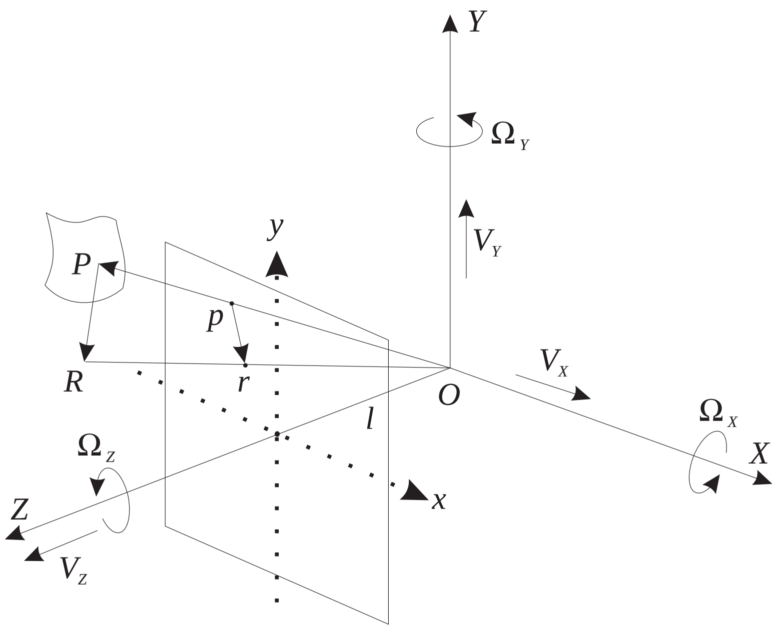


Figure 12: The rotational and translational velocities of a rigid object and their relationship with the world and image coordinate systems. This illustration is due to Wohn and Waxman [42].

stationary viewer.) We will assume uniform motion, in other words $d\mathbf{V}/dt = d\mathbf{\Omega}/dt = \mathbf{0}$.

The optic flow field is the instantaneous velocities of all image points induced by the motion of the object. It is derived by projecting the velocities of all world points onto the image plane.

The velocity of object points in terms of $\mathbf{V}$ and $\mathbf{\Omega}$ is

$$\dot{\mathbf{R}} = \mathbf{V} + \mathbf{\Omega} \times \mathbf{R} \tag{21}$$

The projection Equation 10 gives the relation $\mathbf{r}/l = \mathbf{R}/Z$ between a world point $\mathbf{R}$ and the corresponding image point $\mathbf{r}$. The projection of the object velocities $\dot{\mathbf{R}}$ produces the *optic flow field* $\dot{\mathbf{r}}$, also written $\mathbf{v}$ or $\mathbf{v}(\mathbf{r}, t)$: the velocity of the image point at position $\mathbf{r}$ at time t. We will denote the second derivative, $\ddot{\mathbf{r}}$, by $\mathbf{a}$.

Differentiating this with respect to t, we then substitute for $\dot{\mathbf{R}}$, $\mathbf{R}$ and $\dot{Z} = \mathbf{k} \cdot \dot{\mathbf{R}}$ to get an expression for the optic flow field in terms of the object motions $\mathbf{\Omega}$ and $\mathbf{V}$, and the image coordinates $\mathbf{r}$. This is the Optic Flow Equation:

$$\begin{aligned} \mathbf{v} &= \dot{\mathbf{r}} = \frac{l\dot{\mathbf{R}}}{Z} - \frac{l\mathbf{R}\dot{Z}}{Z^2} \\ &= \frac{l\mathbf{V} - (\mathbf{k}\cdot\mathbf{V})\mathbf{r}}{Z} + \boldsymbol{\Omega}\times\mathbf{r} - [\mathbf{k}\cdot\boldsymbol{\Omega}\times\mathbf{r}]\frac{\mathbf{r}}{l} \end{aligned} \tag{22}$$

The expression for $\mathbf{v}$ is broken into a part which is independent of the depth Z but dependent on the rotation $\boldsymbol{\Omega}$, and a part which is dependent on translation $\mathbf{V}$ and also on the depth. The depth Z of course depends on the scene being animated, and in particular the optic flow field in a region depends on the surface structure of the object visible in that region. Later in Section 4.6 we will look at planar surfaces which give us a simple way of determining Z from $\mathbf{r}$.

We would now like to look at various approximations to the optic flow field.

4.4 Approximating the field in the time and space domains

By itself, the optic flow field indicates the current velocity of each moving image point. But to build an image based animation system we need to know what happens to whole regions of points: it would be very inefficient to handle each point individually when whole neighbourhoods are transforming in coherent ways.

The information about how regions move comes from the way the optic flow field is changing over that region; in other words, the partial derivatives of the field. Previous work [5] has concentrated on the first order derivatives: we have extended this to include second order derivatives, giving us a more accurate approximation to the changing image, and a wider set of warps we can use to approximate the frame to frame deformations.

We can track the motion of an image point $\mathbf{r}$ over time t using a Taylor Series in the time domain. Up to the second order derivatives, the expression is as follows:

$$\mathbf{r} \longmapsto \mathbf{r} + \mathbf{v}(\mathbf{r},0)t + \frac{1}{2}\mathbf{a}(\mathbf{r},0)t^2 \tag{23}$$

We can also approximate $\mathbf{v}$ and $\mathbf{a}$ in the space domain. This is a two-variable Taylor series, since $\mathbf{v}(\mathbf{r}) = (v_1(x,y), v_2(x,y))$ and $\mathbf{a}(\mathbf{r}) = (a_1(x,y), a_2(x,y))$. In this and the next few paragraphs we have dropped the dependence of $\mathbf{v}$ and $\mathbf{a}$ on t from our notation for clarity.

Up to second order terms, we can approximate $\dot{\mathbf{r}}(\mathbf{r})$ as follows, where all derivatives are evaluated at $\mathbf{0}$:

$$\begin{aligned}\mathbf{v}(\mathbf{r}) &\doteq \mathbf{v}(\mathbf{0}) + \begin{pmatrix} \frac{\partial v_1}{\partial x} & \frac{\partial v_1}{\partial y} \\ \frac{\partial v_2}{\partial x} & \frac{\partial v_2}{\partial y} \end{pmatrix} \mathbf{r} + \begin{pmatrix} x^2 \frac{\partial^2 v_1}{\partial x^2} + 2xy \frac{\partial^2 v_1}{\partial x \partial y} + y^2 \frac{\partial^2 v_1}{\partial y^2} \\ x^2 \frac{\partial^2 v_2}{\partial x^2} + 2xy \frac{\partial^2 v_2}{\partial x \partial y} + y^2 \frac{\partial^2 v_2}{\partial y^2} \end{pmatrix} \\ &= \mathbf{v}(\mathbf{0}) + J\mathbf{r} + \mathbf{f}(\mathbf{r}) \end{aligned} \tag{24}$$

The matrix J is the Jacobian of the field $\mathbf{v}$, and to a first approximation $\mathbf{v}(\mathbf{r}) = \mathbf{v}(\mathbf{0}) + J\mathbf{r}$. Now we expand $\mathbf{a}(\mathbf{r})$ similarly, up to the first order derivatives:

$$\begin{aligned}\mathbf{a}(\mathbf{r}) &\doteq \mathbf{a}(\mathbf{0}) + \begin{pmatrix} \frac{\partial a_1}{\partial x} & \frac{\partial a_1}{\partial y} \\ \frac{\partial a_2}{\partial x} & \frac{\partial a_2}{\partial y} \end{pmatrix} \mathbf{r} \\ &= \mathbf{a}(\mathbf{0}) + N\mathbf{r} \end{aligned} \tag{25}$$

Combining the previous three approximations gives us the approximation we have been looking for.

$$\begin{aligned}\mathbf{r} &\longmapsto \mathbf{r} + [\mathbf{v}(\mathbf{0}) + J\mathbf{r} + \mathbf{f}(\mathbf{r})]t + [\frac{1}{2}\mathbf{a}(\mathbf{0}) + N\mathbf{r}]t^2 \\ &= \mathbf{v}_0 t + \frac{1}{2}\mathbf{a}_0 t^2 \\ &\quad + [\mathbf{I} + Jt + Nt^2]\mathbf{r} \\ &\quad + \mathbf{f}(\mathbf{r})t \end{aligned} \tag{26}$$

We have decomposed the optic flow field into three components: a translation approximated by $\mathbf{v}_0 t + \frac{1}{2}\mathbf{a}_0 t^2$, an affine transformation approximated by $(\mathbf{I} + Jt + Nt^2)\mathbf{r}$, and a non-linear component approximated by $\mathbf{f}(\mathbf{r})t$. These correspond to the various orders of optic flow which we are going to use to perform animation using image transformations.

4.5 Decomposing the optic flow field by means of basis deformations

We have already pointed out in the previous section that partial derivatives of the field describe how a region is deforming. In this section we illustrate our contribution on how different linear combinations of these partial derivatives give different basis sets for the possible deformations of the image.

The inspiration for this section comes from work in computer vision by Waxman, Wohn and Ullman [40, 43].

Our contribution here is in relating the work in computer vision to animation, and in developing it further by introducing our simplified basis for the optic flow field of a moving planar object. As far as we know this is also the first time the idea of basis deformations has been applied to animation.

Taylor series decomposition of the optic flow field gives us a natural way of decomposing it into deformations of increasing complexity. The successive approximations correspond to translation, affine transformation, and higher order effects. However we can develop 'better' basis sets by taking different linear combinations of the Taylor terms.

The original work in computer vision on these basis sets relates the three dimensional orientation and motion of objects in the scene to the various partial derivatives of the optic flow field. Waxman, Wohn and Ullman [40, 43] extracted the three dimensional structure and motion information from a set of *observables* or *deformation parameters* which are linear combinations of various partial derivatives of the flow field. We use the term basis deformations in place of observables or deformation parameters.

4.5.1 Taylor series decomposition

Here we will show the effect on a region of the image for each of the partial derivatives of the optic flow field.

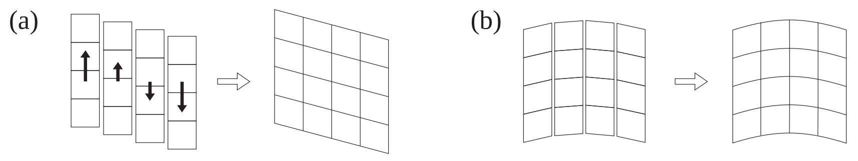


Figure 13: (a) The change in y-velocity over x results in a shear in y. (b) A change in y-shear over x results in an arching effect.

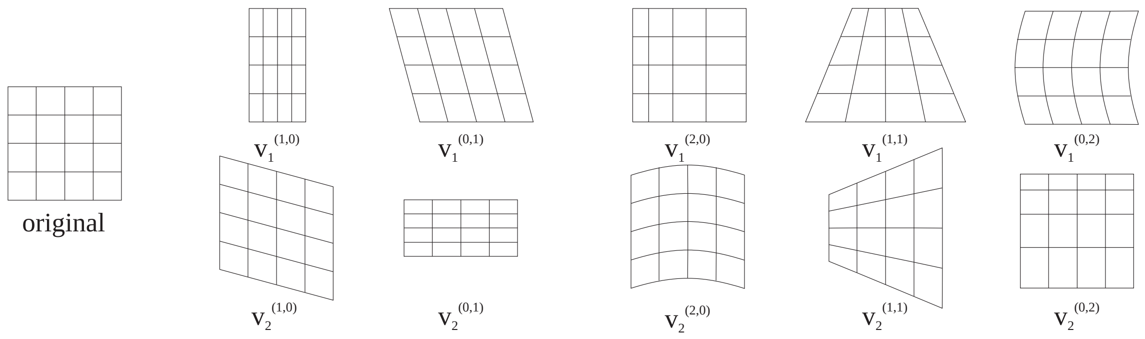


Figure 14: The full set of four first order and six second order partial derivatives of the optic flow field. Each term is represented here by the basic deformation which it induces. Illustration due to Wohn and Waxman [42].

As an example, Figure 13 (a) represents a 'shearing' in the y-direction. This means that as we move from negative to positive x, the y-velocity v_2 is changing from negative to positive. This is an example of the partial derivative $\partial v_2/\partial x$.

Now we can look at the second order terms. If we take as our example $\partial^2 v_2/\partial x^2$, this is $\frac{\partial}{\partial x}(\partial v_2/\partial x)$, in other words the change in y-shear in the x-direction. Figure 13 (b) shows how this results in an 'arching' effect.

Together, the four first order and six second order Taylor terms make up a basis in terms of which we can express the first order (affine) and second order parts of the optic flow field:

$$\begin{gathered} v_1^{(1,0)}, v_1^{(0,1)}, v_2^{(1,0)} v_2^{(0,1)} \\ v_1^{(2,0)}, v_1^{(1,1)}, v_1^{(0,2)} \\ v_2^{(2,0)}, v_2^{(1,1)}, v_2^{(0,2)} \end{gathered} \tag{27}$$

The full set of basic transformations is shown in Figure 14. Each illustration depicts the deformation given that only that particular basis term is non-zero. The illustrations are those given by Wohn and Waxman in [43] and [42].

In the illustration, we see how the first order terms describe scaling or shearing in x and y, while the second order effects are harder to describe. None of these effects seem to relate relate closely to 'physical' effects such as zooming or rotating which occur in a real moving image sequence. We will remedy this in the next two sections by taking linear combinations of the Taylor terms to build a different basis for the optic flow effects.

4.5.2 Waxman, Ullman and Wohn's decompositions

Based on the Cauchy-Stokes decomposition theorem, Waxman and Ullman [43, 42] grouped the first order Taylor terms ($\frac{\partial v_i}{\partial x_j}$) into a symmetric spin tensor and an anti-symmetric rate-of-strain tensor, using the decomposition

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2a & b+c \\ b+c & 2d \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & b-c \\ c-b & 0 \end{pmatrix} \tag{28}$$

The elements of these tensors are linear combinations of the original Taylor terms:

$$\begin{array}{rclcl} e_{xx} & = & v_1^{(1,0)} & = & \frac{\partial v_1}{\partial x} \\ e_{yy} & = & v_2^{(0,0)} & = & \frac{\partial v_2}{\partial y} \\ e_{xy} & = & \frac{1}{2}(v_1^{(0,1)} + v_2^{(1,0)}) & = & \frac{1}{2}(\frac{\partial v_1}{\partial y} + \frac{\partial v_2}{\partial x}) \\ \omega & = & \frac{1}{2}(-v_1^{(0,1)} + v_2^{(1,0)}) & = & \frac{1}{2}(-\frac{\partial v_1}{\partial y} + \frac{\partial v_2}{\partial x}) \end{array} \tag{29}$$

The six second order terms are then the gradients of these quantities:

$$e_{xx,x}, e_{xx,y}, e_{yy,x}, e_{yy,y}, \omega_{,x}, \omega_{,y} \tag{30}$$

The deformations represented by this basis are shown in Figure 15.

The motivation for this decomposition was to identify the type of optic flow by recognising which members of these basis flow types are present. Again, we would like

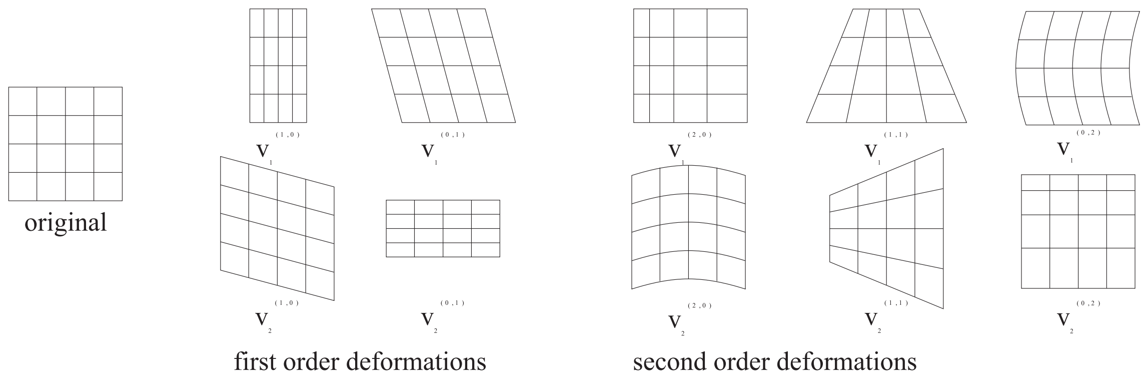


Figure 15: The neighbourhood deformations used by Waxman and Ullman after taking linear combinations of the Taylor terms [42].

to adapt this computer vision technique for animation, and warp the image using just those basis flow fields which are represented. A particular case we would like to consider is the image of a planar object.

4.6 Simplifications for a planar object

The optic flow equation 22 gives the optic flow in terms of the object motions $\mathbf{\Omega}$ and $\mathbf{V}$, the depth of the object Z and the image coordinates $\mathbf{r}$. In particular the optic flow field in a region depends on the surface structure of the object visible in that region.

In the case of a planar object, some important simplifications can be included in our calculations.

Suppose the plane has normal $\mathbf{N} = (N_1, N_2, N_3)$ and intersects the Z axis at Z_0. Then the plane equation is

$$\begin{gathered} \mathbf{R} \cdot \mathbf{N} = Z_0 N_3 \\ X N_1 + Y N_2 + Z N_3 = Z_0 N_3 \quad \text{(equivalently)} \end{gathered} \tag{31}$$

and by substituting the projection equation 10, we have an expression for the depth Z at an image point (x, y):

$$Z = \frac{l Z_0 N_3}{x N_1 + y N_2 + z N_3} = \frac{l Z_0 N_3}{\mathbf{r} \cdot \mathbf{N}} \tag{32}$$

Now this expression for Z can be substituted into the Optic Flow equation, and we can derive expressions for the partial derivatives of $\mathbf{v}$. The derivation is tedious, and is given in Appendix 4.10 at the end of this chapter.

The resulting expressions for $\frac{\partial v_1}{\partial x}$, $\frac{\partial v_1}{\partial y}$, $\frac{\partial v_2}{\partial x}$ and $\frac{\partial v_2}{\partial y}$ are given below:

$$\begin{aligned}
\frac{\partial v_1}{\partial x} &= pV_1 - \frac{p}{l}V_3 x - \frac{s}{l}V_3 + \frac{2x}{l}\Omega_2 - \frac{y}{l}\Omega_1 \\
\frac{\partial v_1}{\partial y} &= qV_1 - \frac{q}{l}V_3 x - \Omega_3 - \frac{x}{l}\Omega_1 \\
\frac{\partial v_2}{\partial x} &= pV_2 - \frac{p}{l}V_3 y + \Omega_3 + \frac{y}{l}\Omega_2 \\
\frac{\partial v_2}{\partial y} &= qV_2 - \frac{q}{l}V_3 y - \frac{s}{l}V_3 + \frac{x}{l}\Omega_2 - \frac{2y}{l}\Omega_1
\end{aligned} \tag{33}$$

Where we have substituted $p = \frac{N_1}{Z_0 N_3}$, $q = \frac{N_2}{Z_0 N_3}$, and $s = \frac{\mathbf{r}\cdot\mathbf{N}}{Z_0 N_3}$.

In addition we can calculate the six second order partial derivatives of $\mathbf{v}$, and here the situation is considerably simplified due to the planar surface being animated. We find that two of the six terms are zero, and that the remaining four are related, as below:

$$\begin{aligned}
v_1^{(0,2)} &= v_2^{(2,0)} &&= 0 \\
\tfrac{1}{2}v_1^{(2,0)} &= v_2^{(1,1)} &&= (\Omega_2 - pV_3)/l \\
\tfrac{1}{2}v_2^{(0,2)} &= v_1^{(1,1)} &&= (-\Omega_1 - qV_3)/l
\end{aligned} \tag{34}$$

This is interesting as it shows that the second order effects for a planar object are independent of translation in X and Y and also of rotation about the Z axis.

In Section 2.4.1 we saw a similar situation in Hofmann's scene-shifting method, where changes in the up-direction and translations in X and Y could be represented by means of affine transformations. Note that the second order derivatives above would also be zero in the case of a planar scene facing the viewer ($N_1 = N_2 = 0$) and translating in Z, and in this case scene-shifting is implemented by scaling the image, also an affine transformation.

4.7 A new basis for the image deformations of a moving planar object

The result of Equation 34 also motivates a new basis for the second order image deformations into which the optic flow field can be decomposed. This new basis is better suited to image synthesis of planar textures. Instead of the second order bases chosen by Wohn, Waxman and Ullman, we use the following set:

$$\begin{array}{rl} & v_1^{(0,2)} \\ & v_2^{(2,0)} \\ & \frac{1}{3}v_1^{(2,0)} - \frac{2}{3}v_2^{(1,1)} \\ & \frac{1}{3}v_2^{(0,2)} - \frac{2}{3}v_1^{(1,1)} \\ p_x = & \frac{1}{2}v_2^{(0,2)} + v_1^{(1,1)} \\ p_y = & \frac{1}{2}v_1^{(2,0)} + v_2^{(1,1)} \end{array} \tag{35}$$

So for the optic flow induced by a planar object, only the last two terms are non-zero, which means only two independent basis deformations, p_x and p_y, are sufficient to describe all possible second order flows induced by a planar object. The effect of these two terms is shown in Figure 16, and further discussed in Chapter 4. The two second order deformations take straight lines to parabolas, but for a small region about the origin they are good approximations of a perspective transformation which takes a square region to a trapezium.

4.8 Another first order basis

We have also created a new basis for the first order optic flow terms, aiming for flows which more closely represent real optic flow fields induced by a moving object. Our new set is the following:

$$\begin{array}{ll} \text{scale} & \frac{1}{2}(v_1^{(1,0)} + v_2^{(0,1)}) \\ \text{squash} & \frac{1}{2}(-v_1^{(1,0)} + v_2^{(0,1)}) \\ e_{xy} & \frac{1}{2}(v_1^{(0,1)} + v_2^{(1,0)}) \\ \omega & \frac{1}{2}(-v_1^{(1,0)} + v_2^{(0,1)}) \end{array} \tag{36}$$

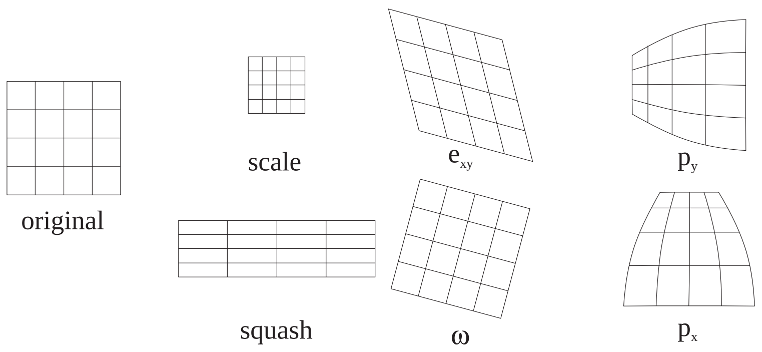


Figure 16: The neighbourhood deformations associated with the optic flow of a planar object. The first order deformations include a pure scaling and a rotation ω, while the set of six second order terms has been reduced to two due to the planar object. Note that the two second order terms are similar to perspective transformations which would take the square neighbourhood to a trapezium.

These transformations are illustrated in Figure 16. The first represents a pure scaling of the image, and the last represents a pure rotation (represented earlier by ω). All except the scaling are area-preserving transformations.

4.8.1 Relating computer vision and animation once more

Culling our basis set down to two translation terms, four first order terms inducing affine transformations, and two second order terms is a necessary step. In the work of Waxman and Ullman [40], various parameters describe the motion and structure of the object. The six parameters of motion are

$$\begin{array}{ccc} V_1/Z_0, & V_2/Z_0, & V_3/Z_0, \\ \Omega_1, & \Omega_2, & \Omega_3 \end{array} \tag{37}$$

There are also two parameters of structure (for a non-planar surface, there are five):

$$\begin{aligned} \frac{\partial Z}{\partial X} &= -\frac{N_1}{N_3} \\ \frac{\partial Z}{\partial Y} &= -\frac{N_2}{N_3} \end{aligned} \tag{38}$$

The problem in computer vision which Waxman, Wohn and Ullman were addressing is to derive the parameters of structure and motion from the optic flow, and to do this they decompose the optic flow field as we have done into a set of 'Observables' or deformation parameters, which are the basis deformations illustrated in Figure 15.

Our approach is the inverse of this: knowing all the parameters of structure and motion, we want to derive the deformation parameters in order to use the optic flow field to produce the animation, i.e. the deformations between frames. As we have seen above and in Figure 16, there are eight deformation parameters for a planar object, which is the correct number of degrees of freedom given the six parameters of motion and two of structure.

From the six parameters of motion $(V_1, V_2, V_3), (\Omega_1, \Omega_2, \Omega_3)$ and two of structure, we have now derived eight basis deformations for a moving planar object. There is a one to one relation between the parameters of motion and six of our basis entries:

Parameter	Description	Deformation	Value	Description
V_1/Z_0	Translation in X	v_1	See Equation 44	Translation in x
V_2/Z_0	Translation in Y	v_2	See Equation 44	Translation in y
V_3/Z_0	Translation in Z	scale		Image scale
Ω_1	Rotation about X	p_x	$(\Omega_2 - pV_3)/l$	Perspective effect
Ω_2	Rotation about Y	p_y	$(-\Omega_1 - qV_3)/l$	Perspective effect
Ω_3	Rotation about Z	ω		Image rotation

In the next chapter we will show how our eight basis entries can be used as the entries in a homogeneous matrix for an approximate deformation between consecutive frames.

4.9 Chapter Summary

We have analysed the optic flow field and how it can be used to produce an image-space transformation between frames of an animation.

The optic flow field describes the motion of image points, and we are developing an algorithm in which the instantaneous velocities of the image points are used to approximate the next frame in an animation sequence. In this chapter we have analysed the optic flow field in terms of a Taylor series (partial derivatives). These partial derivatives were expressed in terms of the underlying three dimensional motion of the object.

These partial derivatives were broken up into a hierarchy of three levels: the flow field itself (image point velocities) approximate the translational motion of a neighbourhood, the first order partial derivatives (change in velocity over x and y) determine an affine warp which applies to a neighbourhood, and the second partial derivatives determine more complex deformations of the image.

Our extension to previous work is to include the second order optic flow terms and to make some important simplifications for the second order optic flow in the case of a moving planar object.

For a planar object, the number of independent second order terms in the Taylor series decomposition reduces from six to two, and higher order terms disappear altogether. By taking certain linear combinations of the Taylor terms, we have derived a set of of eight basis deformations of which two describe translation, four describe an affine warp, and remaining two relate very closely to the perspective terms in a 3×3 homogeneous matrix.

In the next chapter these results will be used to develop an algorithm to perform image based animation using optic flow.

4.10 Appendix: derivation of the partial derivatives of the optic flow field for a moving planar object

In order to derive the partial derivatives of the optic flow field we have made use of some basic tensor calculus. The main results are in Equations 44, 46 and 47. This work comes largely from [5].

Suppose the planar surface has a normal $\mathbf{N}$ and passes through the line of sight (the Z axis) at Z_0. Then its equation is

$$\mathbf{R} \cdot \mathbf{N} = Z_0 N_3 \tag{39}$$

or in tensor notation, where summing over $i = 1, 2, 3$ is implicit:

$$X_i N_i = Z_0 N_3 \tag{40}$$

Substituting into the Projection Equation 10 gives:

$$X_3 = \frac{l Z_0 N_3}{x_i N_i} \tag{41}$$

Tensor notation gives a general-purpose method of writing a vector quantities in terms of coordinates. For instance the ith coordinate of a cross product, $(\mathbf{a} \times \mathbf{b})_i$ is written $\varepsilon_{imn} a_m b_n$. The convention is that we sum over all possible values of the indices m and n. The permutation symbol ε_{ijk} is 1 for even permutations, -1 for odd permutations, and 0 if any two indices are equal, and the Kronecker delta δ_{ij} is 1 if $i = j$ and 0 if $i \neq j$.

The Optic Flow equation (22) can be written in tensor notation as:

$$\dot{r} = v_i = \frac{l V_i}{X_3} - \frac{V_3 x_i}{X_3} + \varepsilon_{imn} \Omega_m x_n - \frac{x_i}{l} \varepsilon_{3mn} \Omega_m x_n \tag{42}$$

Now for a planar surface, we can substitute in Equation 41:

$$\dot{r} = v_i = \frac{x_m N_m V_i}{Z_0 N_3} - \frac{V_3 x_m N_m x_i}{Z_0 N_3} + \varepsilon_{imn} \Omega_m x_n - \frac{x_i}{l} \varepsilon_{3mn} \Omega_m x_n \tag{43}$$

We can convert this back into straightforward vector notation by setting $i = 1$ or 2 to find v_1 and v_2:

$$\begin{aligned} v_1 &= s V_1 - s V_3 \frac{x}{l} - \frac{xy}{l} \Omega_1 + \left(l + \frac{x^2}{l} \right) \Omega_2 - y \Omega_3 \\ v_2 &= s V_2 - s V_3 \frac{y}{l} - \left(l + \frac{y^2}{l} \right) \Omega_1 + \frac{xy}{l} \Omega_2 - x \Omega_3 \end{aligned} \tag{44}$$

where $s = \frac{\mathbf{r}\cdot\mathbf{N}}{Z_0 N_3} = \frac{x_1N_1 + x_2N_2 + x_3N_3}{Z_0 N_3}$.

4.10.1 Derivatives of the optic flow field

As we have seen earlier, it is the spatial derivatives of the optic flow field which determine which type of warp the image is undergoing.

To differentiate the tensor-form optic flow equation (42) we need two results:

$$\begin{aligned}
\frac{\partial}{\partial x_j}(\varepsilon_{imn}\Omega_m x_n) &= -\varepsilon_{imn}\Omega_m \\
\frac{\partial}{\partial x_j}(x_i\varepsilon_{3mn}\Omega_m x_n) &= -x_i\varepsilon_{imn}\Omega_m + \delta_{ij}\varepsilon_{3jm}\Omega_m x_n
\end{aligned}$$

This gives us an expression for the first order derivatives of the optic flow field, i.e. the entries of the Jacobian matrix:

$$\begin{aligned}
\frac{\partial v_i}{\partial x_j} &= (-lV_i + V_3x_i)\frac{N_j}{lZ_0N_3} - \varepsilon_{imn}\Omega_m + \frac{x_i}{l}\varepsilon_{3jm}\Omega_m \\
&\quad -\frac{\delta_{ij}}{l}\left(\frac{V_3x_mN_m}{Z_0N_3} + \varepsilon_{3mn}\Omega_m x_n\right) \qquad (45)
\end{aligned}$$

We can expand these first order derivatives if we like:

$$\begin{aligned}
\frac{\partial v_1}{\partial x} &= pV_1 - \frac{p}{l}V_3x - \frac{s}{l}V_3 + \frac{2x}{l}\Omega_2 - \frac{y}{l}\Omega_1 \\
\frac{\partial v_1}{\partial y} &= qV_1 - \frac{q}{l}V_3x - \Omega_3 - \frac{x}{l}\Omega_1 \\
\frac{\partial v_2}{\partial x} &= pV_2 - \frac{p}{l}V_3y + \Omega_3 + \frac{y}{l}\Omega_2 \\
\frac{\partial v_2}{\partial y} &= qV_2 - \frac{q}{l}V_3y - \frac{s}{l}V_3 + \frac{x}{l}\Omega_2 - \frac{2y}{l}\Omega_1 \qquad (46)
\end{aligned}$$

Here we have substituted $p = \frac{N_1}{Z_0N_3}$, $q = \frac{N_2}{Z_0N_3}$, and $s = \frac{\mathbf{r}\cdot\mathbf{N}}{Z_0N_3}$.

In addition we can calculate the six second order partial derivatives of $\mathbf{v}$, and here the planar surface simplifies things considerably. Out of six possible terms, we find that two are zero, and the remaining four are related, as below:

$$\begin{array}{rcccl} v_1^{(0,2)} & = & v_2^{(2,0)} & = & 0 \\ \frac{1}{2}v_1^{(2,0)} & = & v_2^{(1,1)} & = & (\Omega_2 - pV_3)/l \\ \frac{1}{2}v_2^{(0,2)} & = & v_1^{(1,1)} & = & (-\Omega_1 - qV_3)/l \end{array} \tag{47}$$

We have also calculated the entries from the Jacobian $\left(\frac{\partial a_i}{\partial x_j}\right)$ (recall $\mathbf{a} = \dot{\mathbf{v}}$). A typical entry is shown below. The other entries $\frac{\partial a_1}{\partial y}$, $\frac{\partial a_2}{\partial x}$ and $\frac{\partial a_2}{\partial y}$ are similarly complicated. However we note that they are all $O(|\mathbf{\Omega}|^2\mathbf{x})$. We will use this as an approximation to these terms.

$$\begin{aligned} \frac{\partial a_1}{\partial x} = & -\Omega_2^2 - \Omega_3^2 + \frac{2\Omega_2}{l}\left(\Omega_2 - \Omega_3 y + \frac{lV_1}{Z}\right)^2 - \left(\frac{\Omega_1\Omega_3}{l}\right)x \\ & + \left(\frac{\Omega_1 y - \Omega_2 x}{l} - \frac{V_3}{Z}\right)^2 - 2\Omega_2\left(\frac{\Omega_1 y - \Omega_2 x}{l} - \frac{V_3}{Z}\right)x \\ & -\Omega_1\left(\frac{\Omega_3 x - \Omega_1}{l} + \frac{V_2}{Z}\right) + \Omega_2\left(\frac{\Omega_3 x - \Omega_1}{l} + \frac{V_1}{Z}\right) \end{aligned} \tag{48}$$

Chapter 5

Animated planes using Optic Flow

5.1 Introduction

In this chapter we develop an image based animation algorithm which will animate a moving planar object using the decomposition of the optic flow field given in the previous chapter. Two versions of the algorithm are presented: firstly, a single moving plane animated using optic flow, and secondly an extension to this algorithm to handle multiple planes.

Firstly the analysis of the optic flow field in the previous chapter has to be related to frame to frame transformations in terms of homogeneous matrices which can be used for warping. This is made possible by the fact that we have already reduced the six second order transformations to a basis of just two independent terms. Hence there are eight independent partial derivatives, and eight terms in the homogeneous matrix (Equations 19 and 35).

The transformations fall into a hierarchy of rerendering, perspective warps, affine warps, translation, and no update. The algorithm must decide which level of this hierarchy to implement, and accumulate the errors due to each approximation.

The problem of how to accumulate a long chain of frame to frame warp matrices is addressed, and we show how a prewarped perspective image may be cached to speed up the warping step.

In a multiple-polygon environment, the problem of occlusion must be addressed, and we do this by means of BSP trees, using a layered approach similar to many of the image based rendering algorithms in Section 2.4.

Finally we present the algorithm for animating a single planar objects using optic flow, and describe the extension to handle multiple objects.

5.2 Relating deformation parameters to homogeneous matrices

The image point at $\mathbf{x}$ is moving with velocity $\mathbf{v}(\mathbf{x})$, so the transformation during a time interval of t is:

$$\mathbf{x} \longmapsto \mathbf{x} + \mathbf{v}(\mathbf{x})t \tag{49}$$

The optic flow field in a region can be expressed in terms of all the partial derivatives we have derived above. This means finding the optic flow at point (x, y) based on the optic flow at point $(0, 0)$. To do this we use a Taylor series in two variables:

$$\begin{aligned} v_1(\mathbf{x}) &= v_1(\mathbf{0}) + \frac{\partial v_1}{\partial x}x + \frac{\partial v_1}{\partial y}y + \frac{\partial^2 v_1}{\partial x^2}x^2 + \frac{\partial^2 v_1}{\partial x \partial y}2xy + \frac{\partial^2 v_1}{\partial y^2}y^2 \\ v_2(\mathbf{x}) &= v_2(\mathbf{0}) + \frac{\partial v_2}{\partial x}x + \frac{\partial v_2}{\partial y}y + \frac{\partial^2 v_2}{\partial x^2}x^2 + \frac{\partial^2 v_2}{\partial x \partial y}2xy + \frac{\partial^2 v_2}{\partial y^2}y^2 \end{aligned} \tag{50}$$

Using this and our simplifications for a planar surface (Equation 35) Equation 49 reduces to an expression for the transformation between consecutive frames. Recall that p_x and p_y are the two terms in our simplified set of basis deformations for a moving planar object. All other second order terms fall away except p_x and p_y, which Figure 16 indicates are closely related to perspective warps.

$$\begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} v_1(\mathbf{0})t \\ v_2(\mathbf{0})t \end{pmatrix} + \begin{pmatrix} 1 + \frac{\partial v_1}{\partial x}t & \frac{\partial v_1}{\partial y}t \\ \frac{\partial v_2}{\partial x}t & 1 + \frac{\partial v_2}{\partial y}t \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} p_x x^2 t + p_y xyt \\ p_x xyt + p_y y^2 t \end{pmatrix} \tag{51}$$

Although this transformation can't be expressed in homogeneous notation, since it involves terms in x^2, xy and y^2, we can make some important approximations.

Firstly, suppose p_x and p_y are zero, in other words the second order optic flow is zero. In that case, Equation 51 is already affine, since the last term falls away. An example of this case is when an object undergoes a pure translation, in which case only translation, scaling or other affine warps are required.

Secondly, suppose that p_x and p_y are non-zero, but that the first order derivatives are all zero. This would be the case if the object was stationary and undergoing a rotation not in the image plane. In this case we have a good homogeneous approximation shown below:

$$\begin{aligned} \begin{pmatrix} x \\ y \end{pmatrix} &\longmapsto \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} v_1(\mathbf{0})t \\ v_2(\mathbf{0})t \end{pmatrix} + \begin{pmatrix} p_x x^2 t + p_y xyt \\ p_x xyt + p_y y^2 t \end{pmatrix} \\ &\doteq \begin{pmatrix} 1 & 0 & v_1(\mathbf{0})t \\ 0 & 1 & v_2(\mathbf{0})t \\ p_x t & p_y t & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \end{aligned} \tag{52}$$

For a reasonably small timestep t (such as a typical time delay between consecutive frames), the 2×2 affine matrix in Equation 51 is close to the identity matrix. So, combining the two approximations for $p_x = p_y = 0$ and the first order partial derivatives being zero, we have the following:

$$\begin{pmatrix} x \\ y \end{pmatrix} \longmapsto \begin{pmatrix} 1 + \frac{\partial v_1}{\partial x}t & \frac{\partial v_1}{\partial y}t & v_1(\mathbf{0})t \\ \frac{\partial v_2}{\partial x}t & 1 + \frac{\partial v_2}{\partial y}t & v_2(\mathbf{0})t \\ p_x & p_y & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & t_x \\ a_3 & a_4 & t_y \\ p_x t & p_y t & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{53}$$

Finally, since we have made a number of approximations, it is necessary to put a bound on the error so that we know when the approximations have become inaccurate and the next frame should be calculated exactly.

If we implement just an affine warp, the approximation is that $p_x = p_y = 0$. Ignoring this term in Equation 51 means the maximum error in any output pixel is $O(p_x t\ x_{max}^2)$ or $O(p_y t\ y_{max}^2)$, where x_{max} and y_{max} are the maximum x and y value of the pixels in the input image. We denote by *size* the maximum of x_{max} and y_{max}.

If we implement a perspective warp, we have considered all the non-zero partial derivatives of the flow field $\mathbf{v}(\mathbf{x})$, so there is no error introduced here. But here the approximation is in the assumption $\mathbf{x} \longmapsto \mathbf{x} + \mathbf{v}(\mathbf{x})t$, which assumes that the optic flow field $\mathbf{v}$ is constant, thus ignoring the second derivative, $\mathbf{a} = \dot{\mathbf{v}}$.

Equation 26 indicates that the error in this approximation comes from discarding the terms in $\mathbf{a}(\mathbf{r})$. In Equation 48 we expanded one of these terms, and indicated that the terms $\frac{\partial a_i}{\partial x_j}$ are $O(|\Omega|^2 size\ t)$. We use this as an estimate of the error that accumulates through this approximation.

5.2.1 Optic Flow algorithm: a hierarchy of image transformations

In Chapter 6, we will present the results of our test bed for an optic flow based animation system, consisting of a single planar object (a butterfly cut-out) which moves around on the screen. An outline of the experiment will be given at this early stage to clarify the discussions in the next few sections.[1]

The butterfly is represented by a bitmap, and the image at each frame is produced by affine or perspective warping algorithms (texture mapping.) The butterfly's motion follows a predetermined script which indicates, at each timestep, the object's position (X, Y, Z), the orientation of the plane specified by a normal (N_1, N_2, N_3) and an up-direction (U_1, U_2, U_3) and the object's velocities (V_1, V_2, V_3) and $(\Omega_1, \Omega_2, \Omega_3)$.

The frame rate is fixed and at each frame the eight basis deformation parameters for the optic flow field are calculated. From these parameters, a decision procedure determines which of a hierarchy of different frame to frame deformations will suffice:

[1]We will also perform an experiment using multiple planes, but for the moment the single planar case will be sufficient clarification.

1. no update
2. translation of the previous frame only
3. affine transformation
4. perspective transformation[2]
5. rerendering the object

Each deformation is associated with a certain accumulated error, which is then stored and passed to the decision procedure for the next frame. The remainder of this chapter will discuss the outline and details of this algorithm.

[2]The use of the term *perspective* needs some clarification here. Equation 53 gives us a perspective warp which is an approximation to the frame to frame transformation. The frame to frame transformation determined by the deformation parameters is actually a combination of the two warps represented by p_x and p_y in Figure 16. However the algorithm will use the perspective warp as an approximation instead.

5.3 Handling sequences of image warps

The algorithm we are developing relies on frame to frame transformations in the form of image warps. In this section we will examine how to handle the composition of a long sequence of image warps efficiently. We discuss the source of various errors which accumulate, and describe how to split the product up and produce an intermediate warped image, which is cached for reuse on several frames.

5.4 Successive frames as a product of image warps

In the implementation of our animation algorithm, the underlying representation of the planar object is a bitmapped texture, denoted by T in this section. (It could just as well be some type of model or a procedural texture.)

The initial image I_0 is given by a warp of this texture, $W_{init}T$. Thereafter, the optic flow algorithm outputs a warp matrix W_i which produces each image I_i based on the preceding image I_{i-1}:

$$\begin{array}{rcl} I_0 & = & W_{init}T \\ I_1 & = & W_1 I_0 = W_1 W_{init}T \\ I_2 & = & W_2 I_1 = W_2 W_1 W_{init}T \\ I_3 & = & W_3 I_2 = W_3 W_2 W_1 W_{init}T \\ & \vdots & \\ I_n & = & W'_{init}T \end{array} \tag{54}$$

5.4.1 Accumulated errors due to sampling

Unfortunately this chain of warps cannot be used indefinitely, since errors will accumulate which are not due to the optic flow approximations. These errors come from two sources: firstly due to errors inherent in the warping algorithms, and secondly from the discrete representation of each image. In the context of the Talisman architecture (Section 2.4.6), these errors are measured by *sampling fiducials*.

Such sampling errors are inherent to warping algorithms. For example, if a fast point-sampling warp is used, output pixels are displaced with a maximum error of half a pixel. If a further warp is applied, the errors accumulate to a maximum of one pixel, and so on. More accurate sampling does a lot to reduce these errors, but can never eliminate them.

The discrete nature of the image is also a cause of accumulated error due to the limit that is placed on information content. Even with a perfect warping algorithm, a small image which is gradually increasing in size could not increase its information content. In fact in the real world, warping leads to a loss of information. This is especially problematic when a planar goes through a position oriented edge-on to the viewer, in which case the output image has zero information content. Of course no subsequent frames can be produced by warping at all.

This problem of accumulated errors is not due to the optic flow. The optic flow errors are due to the decision to implement only an Order-i flow instead of a complete rerendering of the object. This error is accumulated in an error term E_i to which we also add a measure of the sampling error which has accumulated.

5.4.2 Affine animation: accumulating a single matrix

In Section 6.5.1 we will compare our results with a simplified algorithm which uses only affine warping, ignoring perspective warping altogether.

In this simplified algorithm, all the warps W_i are affine, and since affine warps are closed under composition, they can be combined into a single affine warp by taking the matrix product $W_i W_{i-1} \ldots W_1 W_{init}$. By applying this matrix, each frame is produced directly from the texture representing the object. If a model or a procedural texture map is used in place of a bitmap, the initial image of the object should be cached and used instead. This is equivalent to using $W_{init}T$ instead of T.

At some frame n, the optic flow algorithm tells us to recalculate the image based on a new matrix W'_{init}. At this stage we scrap our accumulated matrix $W_n W_{n-1} \ldots W_1 W_{init}$ in favour of W'_{init}.

This approach assumes that applying an affine warp to the original texture is no more expensive than warping the preceding frame. This is true if the output image is similar in size to the original texture. If a point-sampling approach is used, the time complexity of the affine warp depends only on the size of the output image.

5.4.3 Perspective animation: Caching prewarped copies of the object

In the case where $W_i W_{i-1} \ldots W_1 W_{init}$ is a product of both perspective and affine warps, we can't accumulate all the matrices and use the product to refer back to the original texture: this may mean applying a non-affine matrix (from texture to frame i) even if the optic flow algorithm determines that an affine warp W_i is sufficient between frame $i-1$ and frame i.

We have developed a decomposition of the warp matrix into a product of translation, affine and perspective warps, as follows:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{55}$$

$$= \begin{pmatrix} \frac{1}{\Delta}(a' - c'g') & \frac{1}{\Delta}(b' - c'h') & 0 \\ \frac{1}{\Delta}(d' - f'g') & \frac{1}{\Delta}(e' - f'h') & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{\Delta}g' & \frac{1}{\Delta}h' & \frac{1}{\Delta} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} c' \\ f' \\ 0 \end{pmatrix}$$

where $a' = a/i$, $b' = b/i$ etc. The scaling factor Δ can be any non-zero value. Its meaning is to determine how the overall scaling effect of the product is split between the two matrices, W_{persp} an W_{aff}. The significance of this value is in determining the relative expense of applying the two warps, with a corresponding tradeoff against loss of image quality due to repeated sampling when the two warps are concatenated.

- A value of $\Delta = 1$ means that the perspective matrix W_{persp} has no overall scaling effect: the cached image is at the same average resolution as the original texture, and all scaling is done in the affine warping step. This corresponds to minimal loss of image quality in the first warp, but large expense.
- A value of $\Delta = \begin{vmatrix} a' - c'g' & b' - c'h' \\ d' - f'g' & e' - f'h' \end{vmatrix} = |\det(A)|$ means that W_{persp} does all the scaling in the initial step, and the cached image is at the same resolution as the output image. This minimises the expense of the perspective warp, but also corresponds to a potential loss of image quality. The loss of quality would occur when the first warp scales the image down significantly, and the second warp operates on a much smaller image. Since information loss occurs early in the process, subsequent warps are less accurate.
- An intermediate case is $\Delta = \sqrt{|\det(A)|}$, which means that the scaling effect is divided evenly between W_{persp} and W_{aff}.

We have found that $\sqrt{|\det(A)|}$ is the most suitable value of Δ when $|\det(A)| < 1$, and 1 when $|\det(A)| \geq 1$. A full discussion of the experimental results which led to this choice is given in Section 6.3.

Now that the product is broken down into the form $W_{aff} W_{persp} T$, the animation begins by caching the perspective-warped image $I_{cache} = W_{persp} T$, and then displaying the image $W_{aff} I_{cache}$.

At each frame the optic flow algorithm produces a matrix W_i which is either affine or perspective (ignoring for the moment the translation or rerendering cases). Then there are two cases:

- **If W_i is affine**
  let $W'_{aff} = W_i W_{aff}$
  apply the affine warp W'_{aff} to the cached image I_{cache}

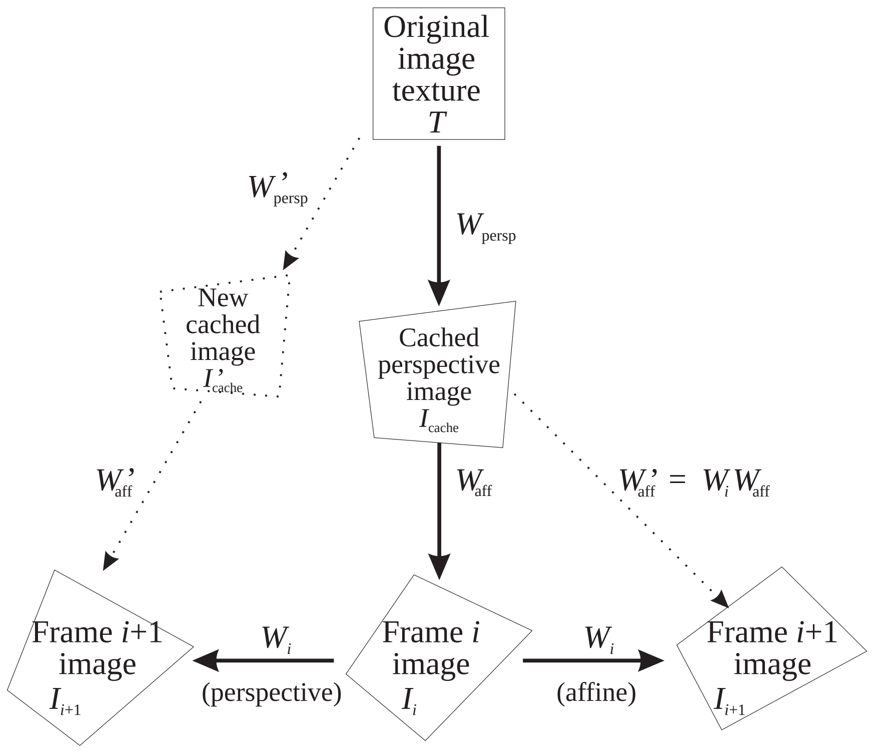

Original image texture T
W'_{persp}
W_{persp}
New cached image I'_{cache}
Cached perspective image I_{cache}
W'_{aff}
W_{aff}
$W'_{aff} = W_i W_{aff}$
Frame i+1 image I_{i+1}
W_i (perspective)
Frame i image I_i
W_i (affine)
Frame i+1 image I_{i+1}

Figure 17: Breaking our warp into a product of perspective (W_{persp}) and affine (W_{aff}) matrices allows us to create a cached perspective image by applying W_{persp} to the texture. If the algorithm requires an affine warp W_i between frames, the product $W'_{aff} = W_i W_{aff}$ is applied to the cached image. If a perspective warp W_i is required, the product $W_i W_{aff} W_{persp}$ is decomposed into a new product $W'_{aff} W'_{persp}$ and a new cached image is generated by applying W'_{persp}.

- **If W_i is a perspective warp**

  split the new product $W_i W_{aff} W_{persp}$ into $W'_{aff} W'_{persp}$ as above

  calculate a new cached image $I'_{cache} = W'_{persp} T$

  apply the affine warp W'_{aff} to I'_{cache}

The decomposition is illustrated in Figure 17.

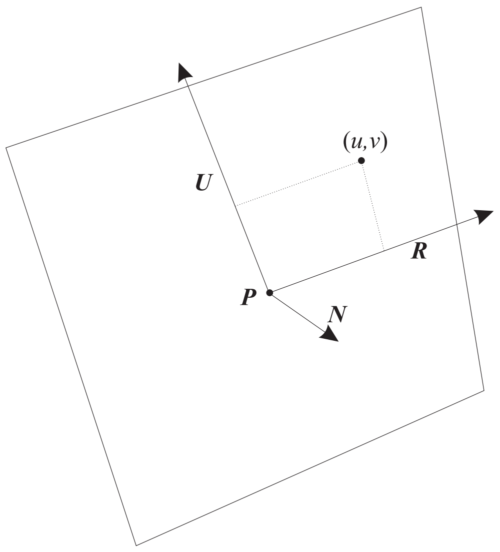


Figure 18: The position and orientation of the planar object are specified by a position vector **P** and two orthogonal unit vectors **N** (the normal) and **U** (the up-direction). The right direction **R** is related by the rule $\mathbf{N} = \mathbf{U} \times \mathbf{R}$. The point with object coordinates (u, v) is located at world coordinates $u\mathbf{R} + v\mathbf{U} + \mathbf{P}$.

5.4.4 The initial warp from texture to image

The initial image of the planar texture is determined by the position **R** of the object, and its orientation, which is specified by the normal **N** and up-direction **U** (see Figure 18). The world location of an object point with object coordinates (u, v) is $u\mathbf{R} + v\mathbf{U} + \mathbf{P} = (uR_x + vU_x + X, uR_y + vU_y + Y, uR_z + vU_z + Z)$. The projection of this point onto the image plane $Z = l$ is

$$\left(l\frac{uR_x + vU_x + X}{uR_z + vU_z + Z}, l\frac{uR_y + vU_y + Y}{uR_z + vU_z + Z} \right)$$

This projected point can be found from the coordinates (u, v) using an homogenous matrix, representing a perspective warp as follows:

$$\begin{pmatrix} lR_x & lU_x & lX \\ lR_y & lU_y & lY \\ R_z & U_z & Z \end{pmatrix} \tag{56}$$

Equation 56 give us the initial (accurate) warps W_{init} to use in our algorithm. Subsequent warps W_i are concatenated onto the initial warp.

5.4.5 Keeping track of errors

Since the algorithm can use an approximate transformation at each frame of the animation, it must have a means of keeping track of the small errors which accumulate. In Section 2.4.6 these errors were called *geometric fiducials* in the context of the Talisman architecture. The algorithm records four quantities, $err_0, err_1, err_2, err_3$ and err_4, corresponding to the four different levels of transformation.

These errors are all initialised to zero at the start and whenever the object is rerendered. When an approximation is used at a particular level, the error for that level is incremented, and when the error exceeds a predetermined threshold for that level, the algorithm knows it can no longer use an approximation at that level. The threshold ensures that the approximate image never differs too much from the 'exact' image.

We have used a pixel metric in recording the errors; the error is the maximum displacement error in any output pixel. The thresholds to one pixel throughout, on the basis that an error of less than one pixel is by definition below the display resolution, while an error of more than one pixel will be visible. In practice, however, smaller errors may be detectable due to rounding and aliasing effects.

At each frame of the animation, the algorithm calculates the optic flow terms t_x, t_y (translation), a_1, a_2, a_3, a_4 (affine) and p_x, p_y (perspective). These terms determine transformations which are applied to the image, and their effects are thus also dependent on the size of the image.

We have used a fairly crude metric for image size: we have taken the maximum of the width and the height of the bounding box, and computed the optic flow effects based on this. A tighter bound is possible, computing the optic flow based on the vertices of the bounding quadrilateral (Figure 27), which is tighter than the axis-aligned bounding box.

The maximum translational effects on the output image are the quantities $flow_0$, $flow_1$, $flow_2$ and $flow_3$, computed as follows:

- $flow_0$ = translation = $\max\{t_x, t_y\}$ (Translation is already computed in terms of pixels.)
- $flow_1$ = affine = $\max\{a_1, a_2, a_3, a_4\} \times size$ (The affine terms depend linearly on the size of the image; see Equation 51.)
- $flow_2$ = perspective = $\max\{p_x, p_y\} \times size^2 t$ (The perspective terms depend on the size of the image squared; see Equation 51.)
- $flow_3$ = higher orders = $|\Omega|^2 size^2 t$

For each level i, $flow_i$ is added to the accumulated error err_i, and if this error is less than the threshold $thresh_i$, the optic flow effects at this level may be ignored. The algorithm then selects the highest remaining transformation and applies the appropriate frame to frame transformation (rerendering, perspective warp, affine warp, translation or no change).

Supposing the transformation at level I was selected, then the errors are then updated as follows:

- $err_i = |err_i - flow_i|$ for $i \geq I$
- $err_i = err_i + |flow_i|$ for $i < I$

The only exception is when the image is rerendered, in which case all errors are reset to zero.

5.5 Animating multiple planes

The preceding algorithm involving only a single planar object has been extended into an experimental walkthrough system involving multiple planar objects. This algorithm was implemented by graduate students Karen Greaves, David Kossew and Oliver Saal, supervised by the author and Professor Edwin Blake.

This experiment aimed to investigate the use of optic flow in a walkthrough animation, the associated problems such as occlusion, compare optic flow with conventional methods by means of performance testing, and examine which types of scene are most suited to optic flow animation.

The algorithm is based on the previous animation algorithm using perspective optic flow. It is illustrated schematically in Figure 5.5. An additional constraint, however, is that multiple planes or polygons may cause occlusion, and a solution to the occlusion problem is dealt with in the next section.

5.5.1 Dealing with occlusion using BSP trees

Occlusion of one object by another is a problem in all image based rendering systems. Part of an object may be hidden in one frame and yet become visible in the next. Purely image based systems can't deal with this because the missing information has to come from a source other than the previous frame.

Many image based systems deal with occlusion using a layered approach, where separate objects are dealt with independently and then composited at the final stage. Such systems include Talisman [38], and are discussed in detail in Section 2.4.

We have used a layered approach by keeping a separate layer for each polygon. These layers are structured in a BSP tree [9]. Traversing the BSP tree for a given viewpoint produces a back to front ordering of the polygons in the scene, and the painters algorithm is then used to composit the objects into the final image without resorting to a Z-buffer.

Although the use of BSP trees solves the problem of compositing multiple layers into the final image, it is not an ideal solution to the occlusion problem. If a polygon is partly visible, the optic flow calculations and image warps are done for every part of that polygon regardless of whether that part is visible or not. By comparison, a standard rendering approach would do the texture calculations and only for visible

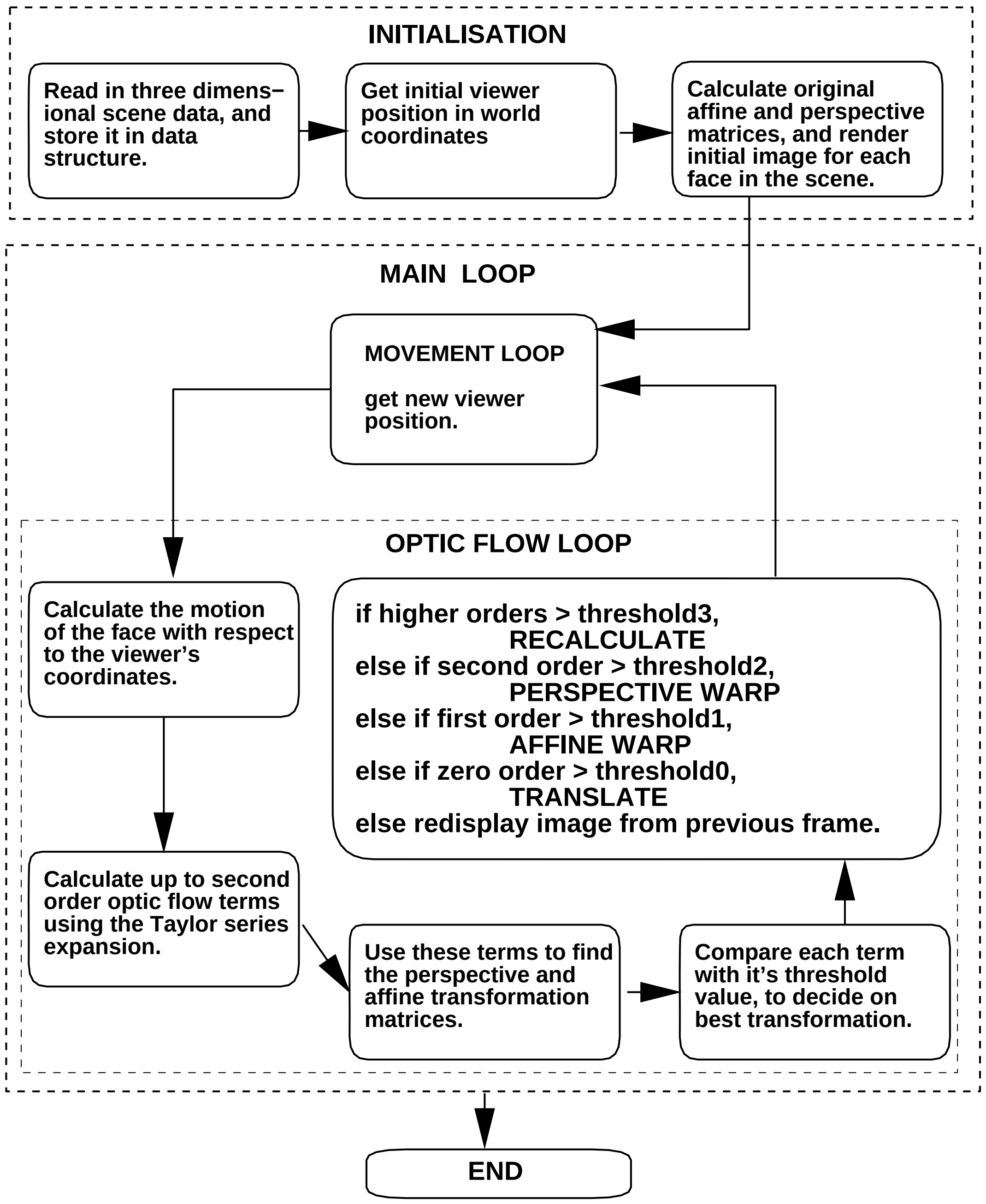

INITIALISATION
Read in three dimensional scene data, and store it in data structure.
Get initial viewer position in world coordinates
Calculate original affine and perspective matrices, and render initial image for each face in the scene.
MAIN LOOP
MOVEMENT LOOP
get new viewer position.
OPTIC FLOW LOOP
Calculate the motion of the face with respect to the viewer's coordinates.
if higher orders > threshold3, RECALCULATE
else if second order > threshold2, PERSPECTIVE WARP
else if first order > threshold1, AFFINE WARP
else if zero order > threshold0, TRANSLATE
else redisplay image from previous frame.
Calculate up to second order optic flow terms using the Taylor series expansion.
Use these terms to find the perspective and affine transformation matrices.
Compare each term with it's threshold value, to decide on best transformation.
END

Figure 19: Schematic diagram of the Perspective Optic Flow algorithm. The case with multiple polygons involves a BSP tree structure which is handled in an additional loop (not shown here.)

pixels. This is an important constraint on many image based systems, making them more suitable to lightly occluded scenes such as a landscapes.

5.6 Algorithms

Here we present our algorithm for animating a single moving planar object. We then describe how the algorithm can be extended to handle multiple planes and the consequent problems of occlusion, using BSP trees as discussed in Section 5.5.1.

5.6.1 Algorithm 1: Optic Flow for a single planar object

This algorithm takes as its input a sequence of positions and orientations for the moving planar object at each frame, and the four threshold values. It computes which transformation to use at each frame of the animation, applies the transformation, and outputs the resulting image as the subsequent frame.

```
begin
    INPUT error thresholds $thresh_0, thresh_1, thresh_2, thresh_3$
    Set $err_0 = err_1 = err_2 = err_3 = err_4 = 0$
    INPUT position $(X, Y, Z)$ and plane normal $(N_1, N_2, N_3)$ for first frame
    Compute $W_{aff}$, $W_{persp}$ and $R_{offset}$ using (56)
    Create cached image $I_{cache} = W_{persp}T$
    OUTPUT image $I_i = W_{aff}I_{cache}$ at position $R_{offset}$

    for each subsequent frame
    begin
        INPUT position $(X, Y, Z)$ and plane normal $(N_1, N_2, N_3)$ for this frame
        INPUT velocities $(V_1, V_2, V_3)$ and $(\Omega_1, \Omega_2, \Omega_3)$ between frames
        $size$ = maximum dimension of current image
        Compute $a_1$, $a_2$, $a_3$, $a_4$, $t_x$, $t_y$, $p_x$, $p_y$
```

```
$flow_0 = \max(t_x, t_y)$
$flow_1 = \max(a_1, a_2, a_3, a_4) \times size$
$flow_2 = \max(p_x, p_y) \times t\ size^2$
$flow_3 = |\Omega|^2 size^2 t$
**if** $flow_3 + err_3 > thresh_3$
    Rerender the planar surface from the model:
    recompute $W_{aff}$, $W_{persp}$ and $R_{offset}$ using (56) and (55)
    create cached image $I_{cache} = W_{persp}T$
    $err_0 = err_1 = err_2 = err_3 = err_4 = 0$
    OUTPUT image $I_i = W_{aff}I_{cache}$ at position $R_{offset}$
**else if** $flow_2 + err_2 > thresh_2$
    Perspective warp required:
    let $W_i = \begin{pmatrix} a_1 & a_2 & t_x \\ a_3 & a_4 & t_y \\ p_x t & p_y t & 1 \end{pmatrix}$
    split the product $W_i W_{aff} W_{persp}$ into $W'_{aff} W'_{persp}$ including new $R'_{offset}$
    create cached image $I_{cache} = W'_{persp}T$
    accumulate errors according to Section 5.4.5
    OUTPUT image $I_i = W'_{aff}I_{cache}$ at position $R'_{offset}$
**else if** $flow_1 + err_1 > thresh_1$
    Affine warp required:
    let $W_i = \begin{pmatrix} a_1 & a_2 & t_x \\ a_3 & a_4 & t_y \\ 0 & 0 & 1 \end{pmatrix}$
    let $R'_{offset} = R_{offset} + (t_x, t_y)$
    let $W'_{aff} = W_i W_{aff}$
    accumulate errors according to Section 5.4.5
    OUTPUT image $I_i = W'_{aff}I_{i-1}$ at position $R'_{offset}$
**else if** $flow_0 + err_0 > thresh_0$
    Translation required:
    let $R'_{offset} = R_{offset} + (t_x, t_y)$
```

```
            accumulate errors according to Section 5.4.5
            OUTPUT image $I_{i-1}$ at position $R'_{offset}$
        else
            Leave the image unchanged.
            accumulate errors according to Section 5.4.5
        end if
    end
end
```

5.6.2 Extending Algorithm 1 to handle multiple objects

Algorithm 1 can be extended using BSP trees to handle the occlusion which results from multiple planar objects.

This extension has no effect on the underlying optic flow algorithm, simply adding an additional outer loop (**for all planes do** ...). Each planar object is treated separately, and the image of each face is stored in memory. The BSP algorithm affects only the display routine, providing us with a back to front order in which to display the faces. This ordering is recomputed once per frame, outside the loop through all faces.

In our implementation of the algorithm for multiple planar objects, we have included features not included in our implementation of Algorithm 1 for a single object. These included backface and viewing frustum culling.

5.7 Chapter Summary

In Section 5.2 we showed how the optic flow analysis of Chapter 4 can be used to produce the frame to frame transformation in an animation, in the form of a homogeneous warp matrix. We showed the decision procedure which determines which transformation should be implemented: rendering, perspective warp, affine warp, translation or no update.

The chain of frame to frame warp matrices is split into a product of a perspective warp and an affine warp, and the perspective warp is used to produce a cached image from which affine warps may be performed quickly. We showed how the initial ('correct') warps are determined in terms of the relative position and orientation of the viewer and planar object.

We developed an algorithm for animation a planar object using optic flow.

Finally we discussed the implications of a scene involving multiple objects and the BSP-tree approach we have used to address the resulting occlusion problem.

The results of experiments involving these algorithms are presented in Chapter 6. Limitations of the algorithms are noted and suggestions for extensions to our algorithm are discussed in Section 7.4.2 of the Conclusion.

Chapter 6

Optic Flow experiment

6.1 Introduction

In this chapter we present and analyse the results of two experiments in animating of planar objects using optic flow analysis.

The intention of these experiments is to show the saving that is achieved by means of the optic flow algorithm as opposed to a conventional rendering system which does not use approximations based on the optic flow. Two indications of this saving are the total time spent in rendering all frames in the test run, compared to the time spent by a conventional renderer, and the number of frames (or polygons) for which an approximation is used instead of rerendering.

In presenting the results of the experiments, we will test the algorithm using scripts describing various types of motion of the object.

The first experiment involves a single planar object, animated by the optic flow algorithm (Section 5.6.1, Algorithm 1). A limited version of this experiment, using only affine warping, is also included for comparison.

The second experiment is based on the optic flow algorithm, extended for scenes involving multiple planar surfaces. The extension to Algorithm 1 is described in Section 5.5.1.

These experiments lead us to conclude that there is a significant saving of between 30% and 90% to be achieved through the use of the optic flow algorithm.

6.2 Outline of the experiments

The first experiment implements Algorithm 1, considering only a single planar object.

The second experiment (Section 6.6) is an extension of the optic flow algorithm to allow for multiple objects, using a BSP tree to handle occlusions. There are several differences between the implementations of the two experiments, including the programming language (C in the first and C++ in the second), and the graphics library (GL in the first and GLUT in the second).

The experiments are designed to test our algorithm against an algorithm not using the optic flow approximations, determining whether the algorithm gives a time saving and where this saving is achieved, and testing the effectiveness of other aspects of our implementation including the use of BSP trees to handle occlusion.

6.2.1 Outline of Experiment 1

The first experiment(Section 6.4) is an implementation of the algorithm in Section 5.6.1, using optic flow analysis to animate a single planar object. The algorithm is tested against six scripts representing different types of motion. Sample frames are given for all the scripts, and timing results for each script are graphed.

The experiment based on a single object uses scripts which describe a variety of motions. In this experiment, the object appears to be moving relative to a fixed viewer. In each case we will describe the motion specified by the script verbally and by means of a diagram. We have taken sample views of the object (a butterfly) at various frames during the script. We have also graphed the time taken to compute each frame, and indicated which level of frame to frame transformation the algorithm chose to implement.

The results of the first experiment is compared with an implementation using a naive algorithm, where every frame is rerendered making no use of optic flow approximations.

6.2.2 Outline of Experiment 2

In this experiment, we use a variety of different scenes consisting of up to 20 polygons. The motion scripts are similar to those in the previous experiment, but including a compound script made up of sections from all the others. Unlike the earlier experiment where the butterfly appeared to be moving in front of a static viewer, these scripts were designed considering the scene as fixed while the viewer moves through it. This does not alter the underlying algorithm, but it does affect the user's perception of the output.

Our output includes graphs of the timing information for various scripts, comparisons based on number of polygons and occlusion, and a comparison with output of a comparable algorithm not using any of the optic flow approximations.

6.2.3 Timing comparisons

It is important to note several issues in the comparison between animation with the optic flow algorithm and animation without the optic flow approximations.

Two different criteria have been used: the number of frames rendered with each order of optic flow, and the total time taken for the rendering. These two measures have different strengths and weaknesses.

The number of frames (or polygons) rendered using optic flow approximations gives us an idea of how often the algorithm represents a saving, but does not give an idea of the size of this saving. For instance, a large polygon for which we can do an affine warp instead of perspective may be a far more significant saving than a smaller polygon which requires no update at all.

The measure of total time saving is a better indicator. However it too requires some interpretation, particularly in analysing what we are using as the baseline in the comparison.

In the first experiment, we have compared against a sample run which uses rerendering at every frame instead of an optic flow approximation. This is a fair comparison since every other aspect of the code remains unchanged.

In the second experiment using multiple objects, the situation is more complicated. Here we have compared against an implementation using standard GLUT calls to render the textured polygons. This was done without hardware texture mapping, to keep the playing fields level. There is still a fundamental difference between this and our method using BSP trees: in the GLUT implementation, texture mapping calculations are done after the visibility calculations, and therefore only for those pixels which are visible, while in our BSP implementation, they are done for each pixel of each surface, before the visibility calculation is done. A more complicated version of our algorithm could perform the warping at the last stage after the BSP visibility calculations have occurred, but we have not implemented this. This optimisation would improve the performance advantage of our algorithm as compared to the GLUT implementation.

Cost comparison of different transformations

We have also looked at the relative costs of the different frame to frame transformations we have used. We have optimised all our warping implementations by using nearest neighbour point sampling.

A key difference is between affine and perspective warping. Affine warping, being linear, can be implemented extremely efficiently with only additions, for instance using a Bresenham-like algorithm. By comparison a perspective warp requires a division at every pixel. It is possible to approximate a perspective warp by doing one division for every 8 or 16 pixels, and this is commonly done in time critical texture mapping environments such as action games. This amounts to a piecewise linear approximation to the hyperbola of $1/Z$ values; other approximations such as parabolic are also possible. Further approximations which we have not implemented apply in special cases such such as an upright viewer looking at a vertical wall: in this case vertical scanlines are at constant depth, so only one division per vertical scanline is required.

In Appendix A we have found experimentally that our implementation of a perspective warp is slightly more than twice as expensive as our affine warp. Further optimisations are possible on both implementations, and are likely to favour the perspective warp.

The final discussion point is the cost of rerendering. In Experiment 1, this is the same as the cost of a perspective warp, the highest order of frame to frame transformation. At first glance this suggests that there is no benefit in choosing rerendering instead of warping; however, this weakness is entirely an implementation issue, depending on the representations used to render the planar objects:

Since we have chosen to use bitmapped textures on the polygons in our implementation, rendering the polygon is just a warping step. If the textures were stored procedurally, however, rendering would be more complex than warping, and warping instead of rendering would be a significant saving.

If our primitives were not polygons, but collections of polygons making up near-planar objects (for instance the facade of a building), the rendering step could be to create a planar impostor from a more complex underlying geometry, and then use warping on this impostor for subsequent frames. This is the approach of Talisman and other image based algorithms [38, 37]. Here warping is clearly far cheaper than rerendering.

The increasing expense of rendering using procedural texture maps or planar impostors means that in a more complex implementation, our algorithm could represent *more* significant savings than those shown experimentally below.

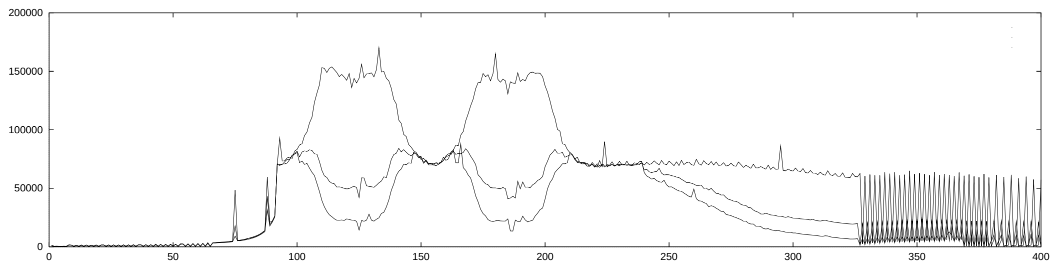


Figure 20: The effect of the scaling factor Δ in the decomposition of the warp matrix into a product of perspective and affine warps. The top graph corresponds to $\Delta = 1$, the bottom graph to $\Delta = |\det(A)|$ and the intermediate graph to $\Delta = \sqrt{|\det(A)|}$. The results are discussed in Section 6.3

6.3 Comparison of different Δ values in matrix decomposition

Before we proceed to our experiment using the optic flow algorithm for animation, we present a result that was used in developing this algorithm: the choice of a scaling factor Δ used in decomposing the warp matrix into a product of two matrices.

In Section 4.5 the warp matrix from texture to image is decomposed into a product of an affine and a perspective warp, $W_{aff}W_{persp}$. The perspective warp is applied first, and its output cached for use in subsequent frames, with a different affine warp being applied at each frame until the error becomes significant.

As discussed in Section 4.5, the size of the intermediate image can be varied depending on a factor Δ which we introduce to determine how the overall scaling effect is distributed between the two matrices. This factor has a critical effect on the speed of the two warping steps, and we have tried to maximize speed without the trade-off of reduced image quality. Three possible values were investigated:

(A) A value of $\Delta = 1$ means all the scaling is done in the second, affine warp. This means that the intermediate image is at the same resolution as the original texture. This has the advantage of minimising loss of resolution due to sampling, but it has the serious disadvantage that the expense of the perspective warp is constant, regardless of the size of the output image: a heavy overhead, given that the perspective warp is the more expensive of the two warps.

Figure 20 shows significant peaks around frames 120 and 190 when the butterfly is almost edge-on, while these peaks are not present for either of the other two values of Δ. From frames 220 to 325, the frame time was almost constant, even though the output image is significantly smaller due to increasing distance and a more acute angle to the viewer. The spikes from frame 325 onwards correspond to rerendering, and again almost constant expense is involved because the perspective warping dominates for the smaller output images. In the other two cases, the expense reduces proportional to the size of the output image.

(B) A value of $\Delta = |\det(A)|$ means that all the scaling is done in the first step, the perspective warp. This means the cached image is at the same resolution as the final image. Although this minimizes the time taken for the warps, it also results in lower quality output because of the lower resolution of the cached image, which may be used in several subsequent frames with a variety of affine warps.

(C) We have used a compromise of $\Delta = \sqrt{|\det(A)|}$. This means that the scaling factor is evenly divided between the perspective and affine warp.

Examining the graph of frame times using the three decompositions in a sample run show that (B) and (C) are always better than (A) in terms of frame time, sometimes hugely so.

Of the two, (B) is always better than the compromise value (C) we have decided on, but at the cost of image quality due to sampling. But (C) is only marginally worse than (B) in the more expensive frames, while (B) scores most highly on the frames that are relatively inexpensive on either (B) or (C). So we don't mind using (C).

6.4 Experiment 1: optic flow animation of a single planar object

In this experiment, a single planar object is animated using the optic flow algorithm from Section 5.6.1.

System components:

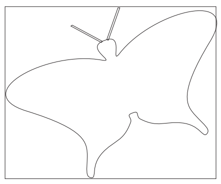
The components of the system are as follows:

- A bitmapped image of a planar object (a butterfly image, 243×198 pixels)
- A script, giving, for each timestep:
  - the object's position (a 3D vector),
  - its orientation (surface normal and up-direction vectors),
  - the translational velocity of the object, and its rotational velocity.
- Four thresholds giving the level at which each order of motion is considered significant (All these thresholds were set at 1 pixel)
- An image warping routine, which produced a fast affine warp of an image using point sampling, and a fast perspective warping algorithm, also using point sampling.

6.4.1 Stationary object

Description of script

The object is at depth 1, in the image plane, where it remains stationary.



Sample output frames

Summary of results

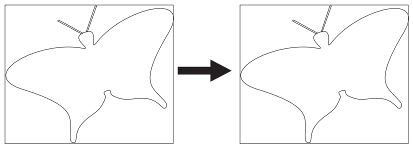
- Time saving: 100%

As expected no updates are required in this trivial case.

6.4.2 Translation 1: object parallel to image plane

Description of script

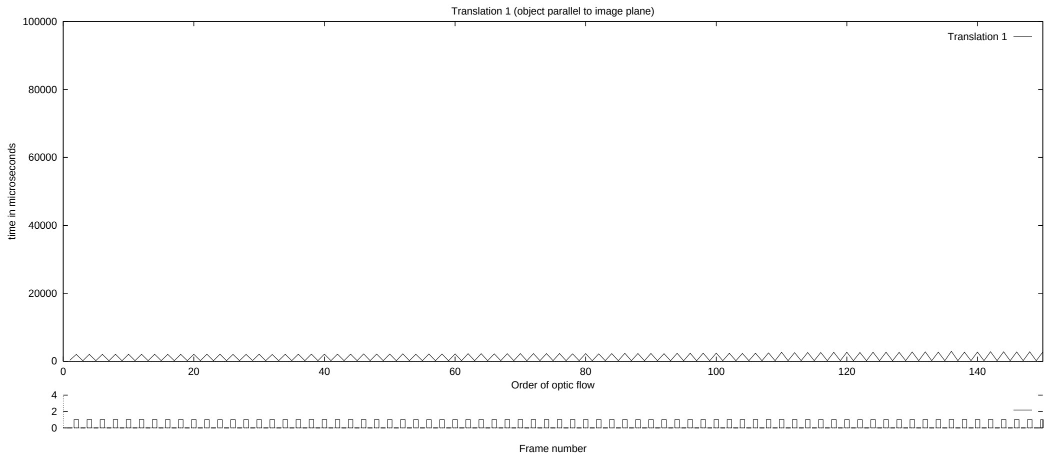
The object is in the image plane and translates along the X axis at a rate of 0.003 units per frame.



Sample output frames

The sample output frames are the same as in the previous sample run, Section 6.4.1.

Timing of sample run



The topmost of the above graphs shows the time taken in rendering each frame of this sample run. The lower graph shows the order of transformation, from 0 up to 4, which was used in each frame. (The same information is graphed for each of the subsequent sample runs.)

Summary of results

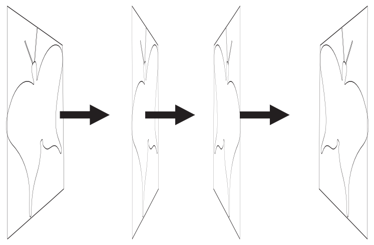
- Time saving: 98.5%

Every second frame requires an update, and this is always a translation. No rerendering or warping steps are required. The saving of 98.5% is based on every second frame requiring no work, and for the remaining frames, rendering would be about 35 times the cost of translation.

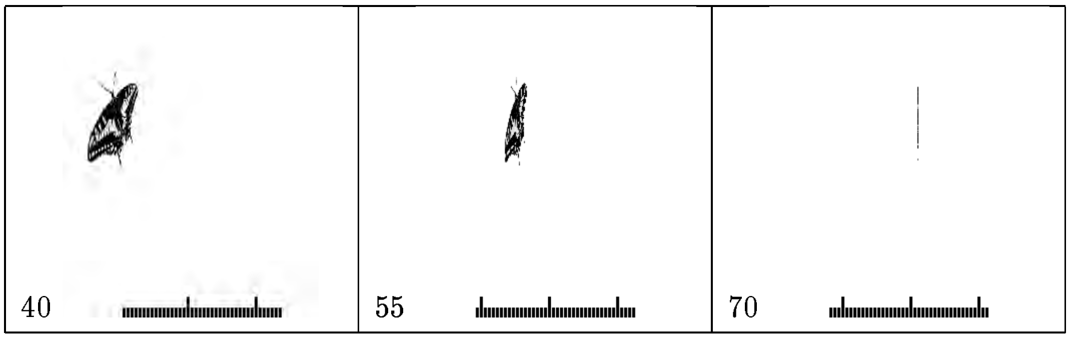
6.4.3 Translation 2: object perpendicular to image plane

Description of script

The object is parallel to the YZ plane, and translates along the X axis past the viewer with a velocity of 0.03 units per frame at depth $Z = 2$.

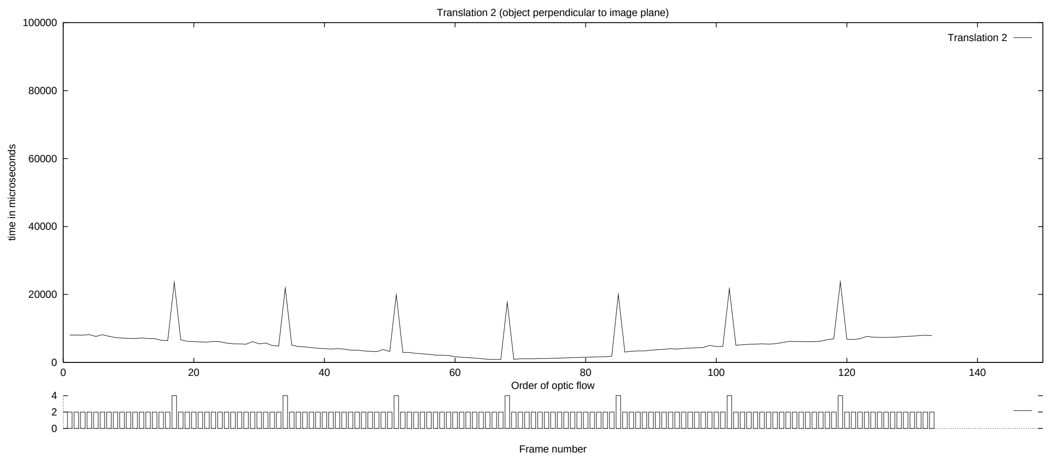


Sample output frames



The illustrations show snapshots of the object at frames 40, 55 and 70, corresponding to X positions of -0.80, -0.40 and 0.00. The bar charts on each snapshot correspond to the order of optic flow used in the last 40 frames of the animation.

Timing of sample run



Summary of results

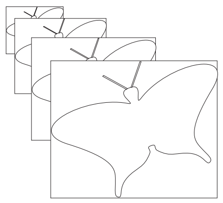
- Time saving: 76.1%

The optic flow algorithm represents a saving in 16 out of every 17 frames, with rerendering used just once for every 16 frames produced by affine warping. The graph shows that as the object approaches the edge-on position (which occurs as it passes the origin at frame 68), the expense of the affine warp decreases linearly.

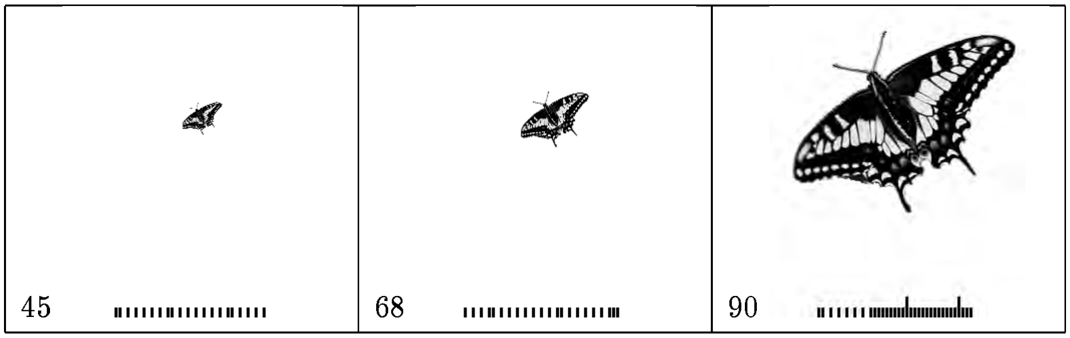
6.4.4 Zoom

Description of script

The object moves from a depth of $Z = 10$ to $Z = 1$ along the Z axis, facing the viewer.

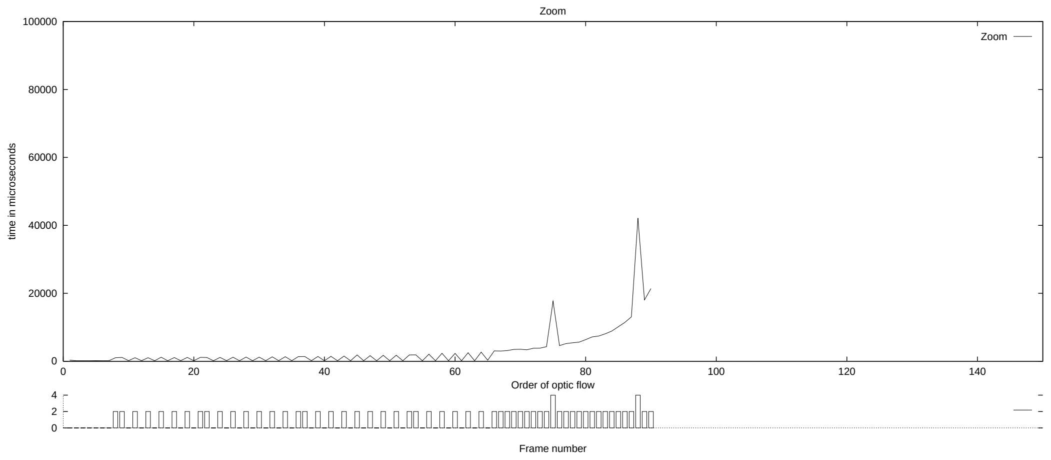


Sample output frames



Frames 45, 68 and 90 above show snapshots of the butterfly at depths of 5.6, 3.3 and 1.0 units as it moves towards the camera at a rate of 0.1 units per frame.

Timing of sample run



(The graph stops short because there were only 90 frames in this sequence.)

Summary of results

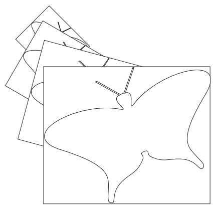
- Time saving: 84.0%

Our algorithm represents a saving in 88 of the 90 frames, as only two require rerendering. The rerendering is needed to correct errors due to sampling. 34 frames are not updated at all, and the remaining 54 require affine warps. It is clear that more updates are required as the object gets closer to the viewer, and that these updates become more expensive. The large expense of rerendering using perspective warping is clear: it is approximately three times the expense of affine warping.

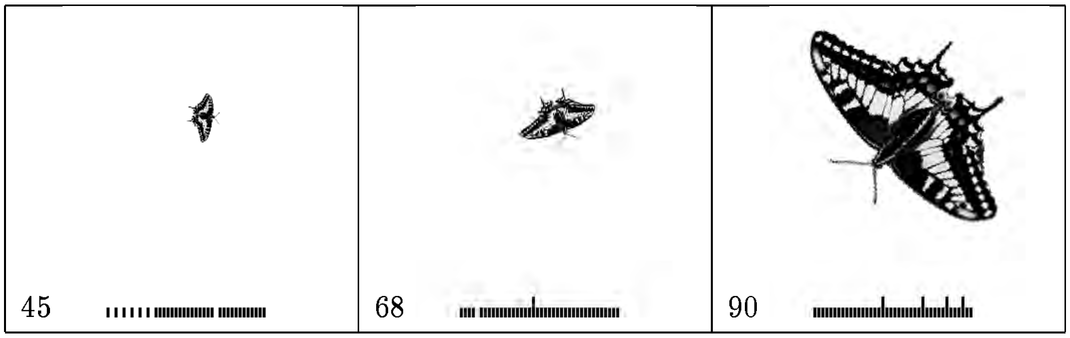
6.4.5 Zoom with rotation

Description of script

The object faces the viewer and moves from a depth of $Z = 10$ to $Z = 1$ along the Z axis at a rate of 0.1 units per frame. At the same time it rotates about the Z axis at 0.05 radians per frame.

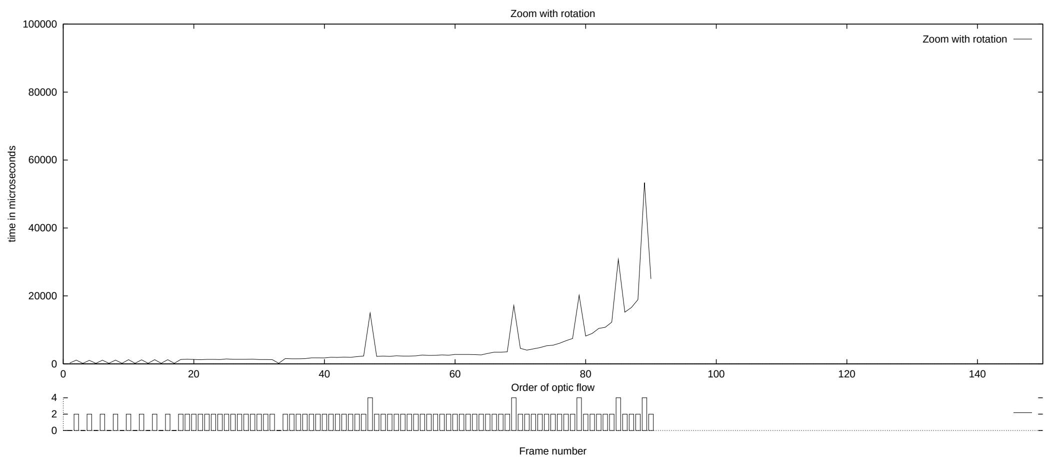


Sample output frames



Frames 45, 68 and 90 are snapshots of the butterfly at depths of 1.0, 3.3 and 5.6 units as it moves towards the viewer and rotates.

Timing of sample run



(The graph stops short because there were only 90 frames in this sequence.)

Summary of results

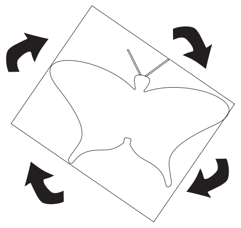
- Time saving: 76.3%

Our algorithm represents a saving in 85 of the 90 frames. Rerendering is needed in 5 frames to correct errors corresponding to the rotation $\mathbf{\Omega}$, as described in Section 5.2. Ten of the 90 frames require no update at all, and the remaining 75 require affine warping. The rotation while zooming means that far more frames need to be updated than in the previous script.

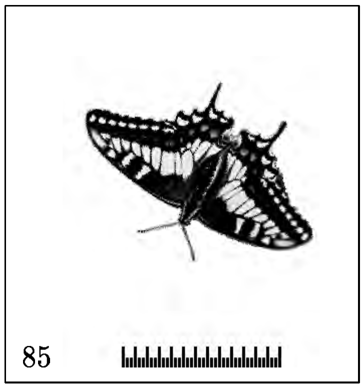
6.4.6 Rotation

Description of script

The object is in the image plane and rotates in this plane at a rate of 0.05 radians per frame, or one revolution every 314 frames.

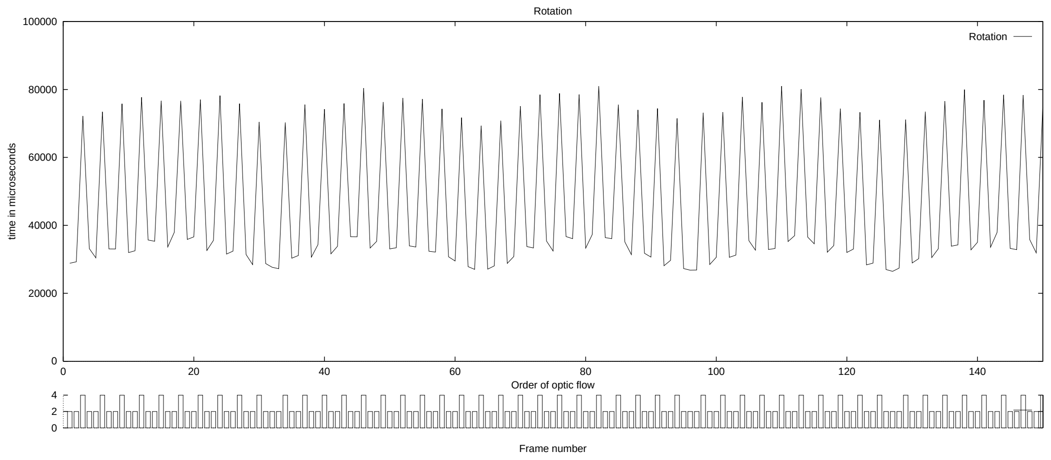


Sample output frame



This illustration shows the object at frame 85 of the rotation. Other snapshots at different angles are similar.

Timing of sample run



Summary of results

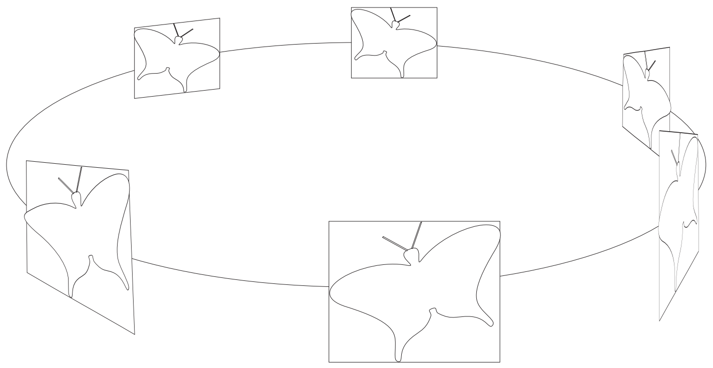
- Time saving: 42.7%

Of the 150 frames shown, almost exactly every third frame requires rerendering, or 48 frames in total. The spikes at these frames show that this is approximately 2.1 to 2.5 times the cost of the remaining 102 frames, which are produced by affine warping. The sinusoidal envelope to the graph is because the object at 45° has a larger bounding rectangle, and hence covers more pixels, than the object when aligned to the axes.

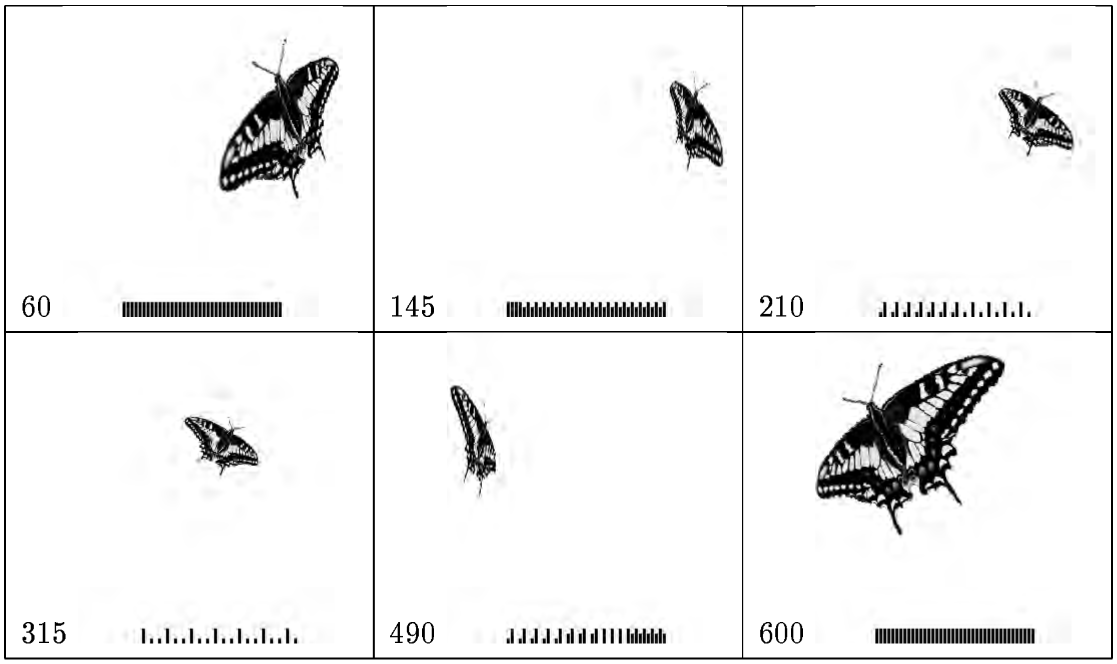
6.4.7 Loop

Description of script

The object loops along a circular path, always facing the center of rotation at $(0, 0, 2)$. It begins at position $(0, 0, 1)$ and takes 629 frames to complete the loop.

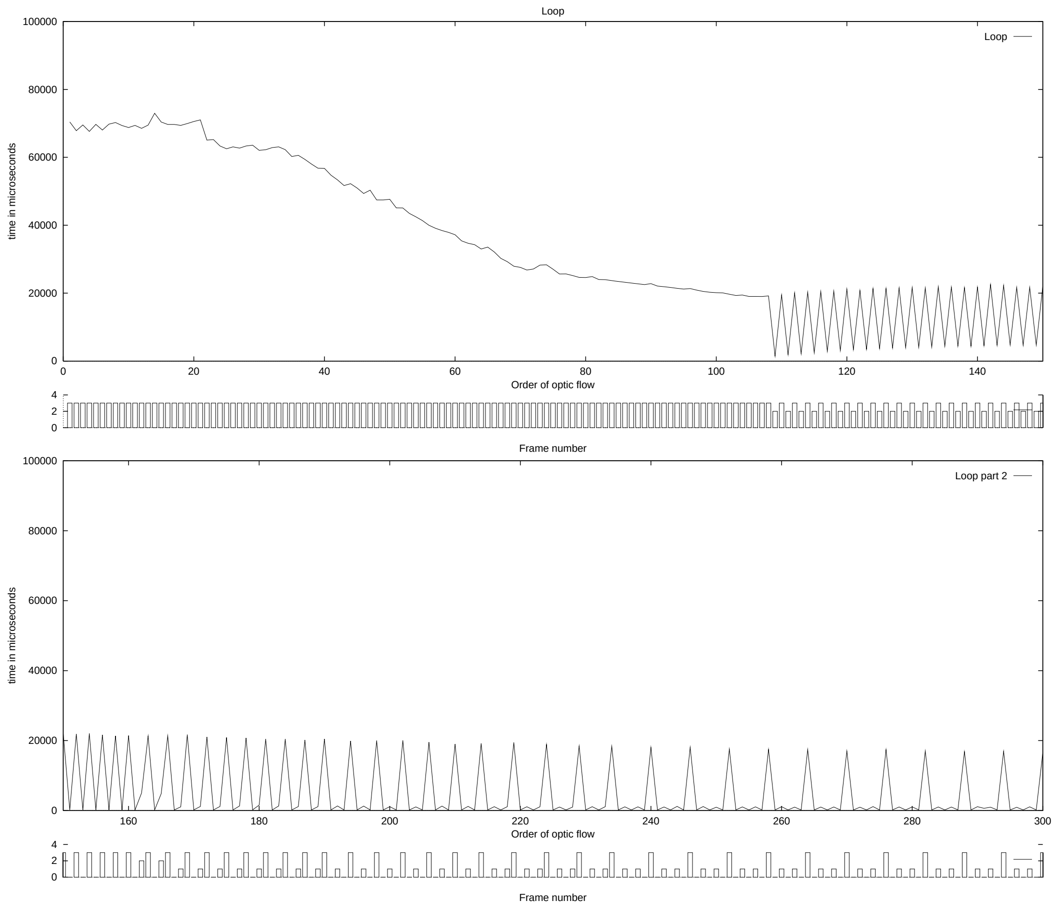


Sample output frames



The snapshots show frames 60, 145, 210, 420, 490 and 600 as the butterfly completes its loop.

Timing of sample run



The above graphs show frames 0 to 300 of the 629 frame loop. The remainder of the loop repeats the same cycle in reverse.

Summary of results

- Time saving: 33.0%

This is the first script where the motion is complex enough for the algorithm to choose a perspective warp, which is chosen frequently in many parts of the more complex script used in Experiment 2. This script shows a large variation in the cost per frame, largely due to the size of the image, which varies by a factor of 9 between the nearest and the furthest point of the loop, and vanishes at the edge-on positions. For the first

110 frames, a perspective warp is needed at every frame, but the cost of these warps reduces as the image gets smaller. As the object moves further away and into an edge-on position (frame 105), perspective warps become less frequent, with every second frame produced by affine warping or translation, and some not requiring an update at all. On the more distant part of the loop, translation or no update are sufficient approximately four frames out of five, the remaining frame requiring a perspective warp. The expense of both perspective warps (the peaks in the graph) and affine warps (troughs) is less for the distant object.

6.4.8 Summary of performance of the perspective algorithm

The results of the six sample runs are itemised in the table below, showing how many frames were produced using each level of frame to frame transformation, and the time savings in comparison to an equivalent run not making use of any of the approximate transformations.

Percentage of frames rendered with each order of optic flow						
	Order of optic flow					
Script	Do nothing	Translate	Affine	Perspective	Rerender	Time saved
1: Stationary	100	0	0	0	0	100%
2: Translation 1	50	50	0	0	0	98.5%
3: Translation 2	0	0	95	0	5	76.1%
4: Zoom	38	0	60	0	2	84.0%
5: Zoom/rotation	11	0	83	0	6	76.3%
6: Rotation	0	0	68	0	32	42.7%
7: Loop	26	17	7	50	0	33.0%

The results above show that we have achieved significant savings in every case. This is largely because of the number of frames which can be produced using an affine warp instead of a perspective warp or rerendering. In all cases the perspective algorithm was able to choose an affine transformation, translation, or no update on more than half the frames, thereby showing a large advantage over rendering every frame.

Only the last of these scripts, involving a three dimensional loop, was complex enough for the algorithm to require perspective warping. The other scripts involved idealised cases such as motion parallel or orthogonal to the viewing direction or to the plane of the object: In Experiment 2 the motion is more complex and the perspective warp is chosen more frequently.

The biggest savings are in the fairly trivial cases of the stationary object and the first translation, where all frames require just a translation or no update.

Substantial savings of between 76% and 85% are achieved for the second translation case and the two zooms. In both of these the majority of frames require an affine warp, with only 2 to 6% requiring rerendering.

Smaller savings of 33% and 43% are achieved in the rotation and loop cases. Here a larger number of frames require a perspective warp or need to be rerendered. The loop case is the most interesting, showing a large variation in performance over the different positions and orientations of the object. When the object is nearest, every frame requires a perspective warp. As it moves away, every second frame requires just an affine warp. At the furthest point of the loop, when the image is smallest, only every second frame requires any update, and these are either translations or affine warps.

The timing data in Appendix A.2 are confirmed in the above experiment, showing that a perspective warp is approximately twice as expensive as an affine warp, and approximately 35 times as expensive as translating the image.

Most of these sample runs reveal specific cases where savings are achieved: faraway objects or (equivalently) objects with small images, and objects undergoing simple translations. Although the time saving with faraway objects is relatively small because of their smaller cost of rendering, it is significant as a proportion for that specific object. Since the number of faraway polygons may be large compared to nearer ones, the saving by using this algorithm may turn out to be substantial.

6.5 A simplified implementation using only affine warping

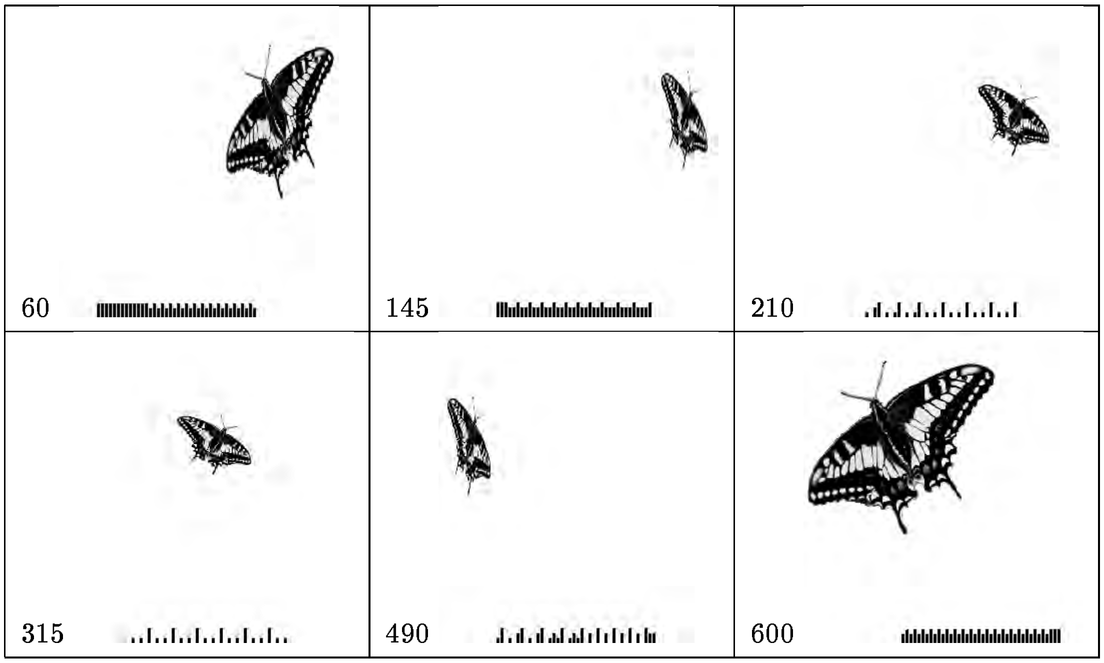
A simplified version of Algorithm 1 was also developed in which no perspective warping was used, either as a frame to frame transformation or in the initial view of the planar object.

The initial rendering is done by an affine warp which is an approximation to the 'correct' perspective view. Although this approximation is inaccurate, the results are surprisingly realistic and the speed increase is considerable: by a factor of up to three. We have also suggested certain cases where these time savings may make this algorithm appropriate.

6.5.1 The Loop sample run using only affine warping

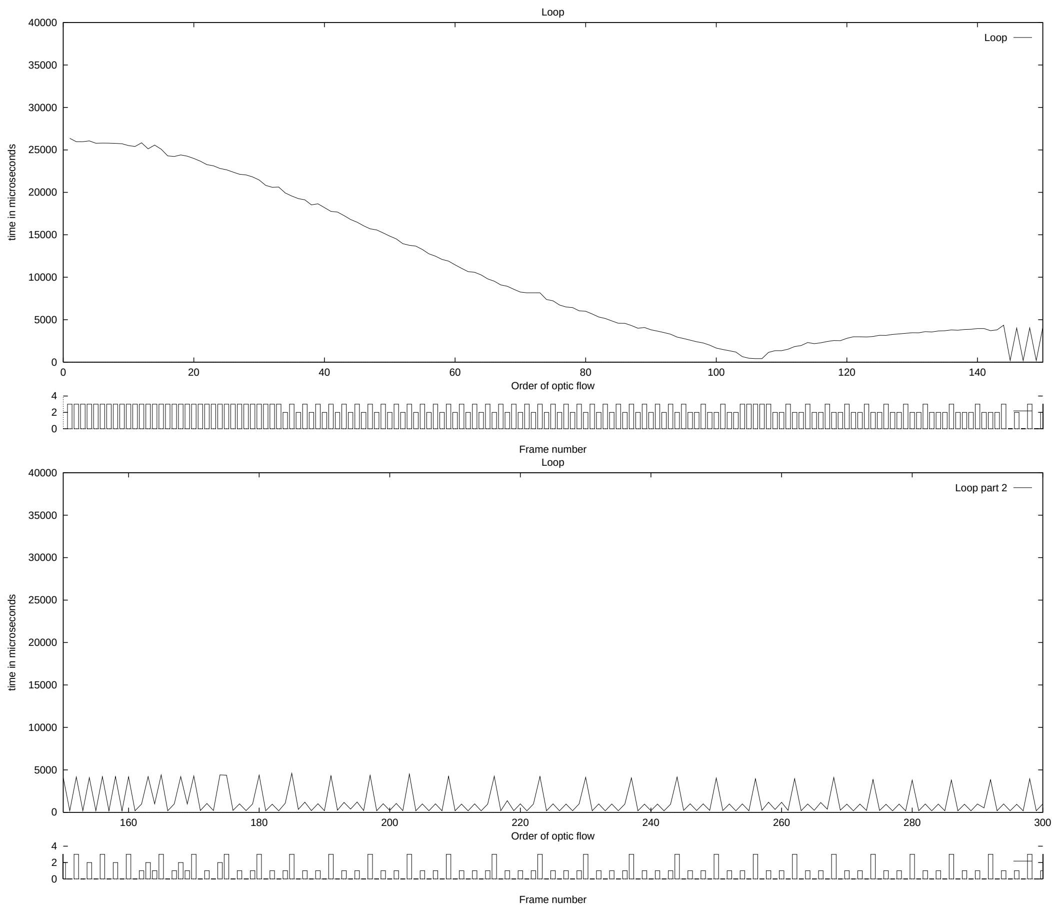
Below we have repeated one of the sample runs from Experiment 1, the loop script (Section 6.4.7). Recall that this script used the widest variety of frame to frame transformations.

Sample output frames



The snapshots show frames 60, 145, 210, 420, 490 and 600 as the butterfly completes its loop, using the affine optic flow algorithm.

Timing of sample run



The above graphs show frames 0 to 300 of the 629 frame loop. The remainder of the loop repeats the same cycle in reverse.

Summary of results

For the first 144 frames an affine transformation or rerendering (also affine) is used every frame, although the cost varies with the size of the image as it first reduces to zero (edge on) and then increases again. However on the more distant part of the loop, an update is only needed roughly every second frame, and translation is sufficient for most updates. Rerendering or affine warps are needed about one frame in 6 for this part of the loop.

The differences between this and the 'complete' optic flow algorithm including perspective warping are in the overall speed, the distribution of the different orders of transformation

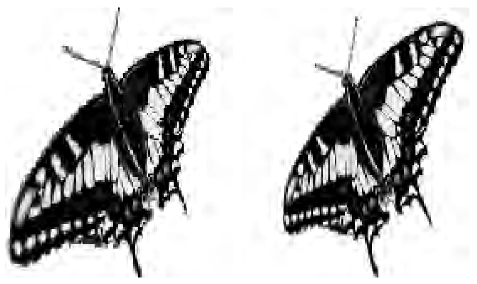


Figure 21: The difference between the 'correct' perspective view of the object (left) and an affine approximation (right).

chosen, and the nature of the output image.

The reduced sample run took approximately one third of the time of the run using the complete algorithm. Some of the other sample runs showed less of a difference, for instance where the complete algorithm was able to perform affine warping in any case. The timing graphs from the perspective algorithm are much more 'spiky' than those from the reduced affine algorithm, because of the large cost difference between affine and perspective warps. In the affine experiment, spikes occur less frequently because there is less freedom to choose between different levels of frame to frame transformation.

The distribution of the different orders of transformation is fairly similar between this and the complete algorithm. Here 26 frames require no update, 18 are translated, 21 require affine warping, and 34 rerendering. The only real difference is that the earlier sample run used affine warping 7 times and 50 perspective warps.

The output animation is as convincing as for the complete algorithm, although the lack of perspective effects is noticeable when the two are shown in succession. As we will see below, the affine algorithm is unsuitable for scenes with multiple polygons, but it would probably be sufficient for a small simple object, ideally a single polygon, moving quickly as a separate part of a larger scene. In this case the time saving may be useful, and the slight inaccuracy in the image would probably go unnoticed.

6.5.2 Drawbacks of the affine algorithm

The affine algorithm can represent a significant time saving over full rendering, as shown above. However it can produce less realistic results because of the drawback of only being an

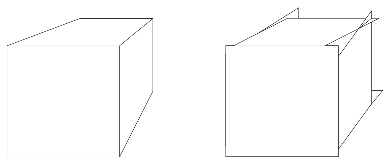


Figure 22: An affine approximation used in scenes with multiple planes results in gaps and overlaps. In this example there is no way the affine images of the sides of the second cube can fit together correctly to form the first image.

approximation. The output for some views can be unrealistic, as can be seen in Section 6.5.1, where there is no foreshortening when the object is seen from edge-on. Figure 21 illustrates the differences between an approximate affine view, and the 'correct' perspective view. In the same way, multiple planes can't be joined together into a shape such as a cube, because the affine approximations to each face would not meet correctly on the edges. (Figure 22). The affine algorithm would also be no use to render fairly large objects covering a wide depth range, such as a flat floor stretching away from the viewer. These drawbacks limit the cases where the affine optic flow algorithm can be used. However, the affine algorithm is surprisingly realistic in the case of a single planar object.

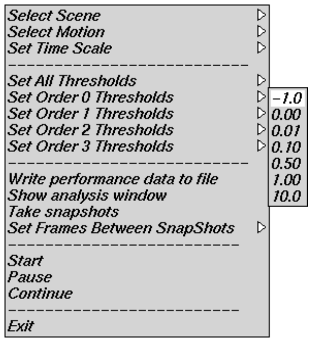


Figure 23: The main pulldown menu providing access to all the parameters and the choice of scene and script.

6.6 Experiment 2: Multiple planes

A second experiment was performed to develop a basic walkthrough animation involving of a static scene, involving multiple planar objects or polygons, animated using the perspective optic flow algorithm. The algorithm is discussed in Section 5.5.

The experiment aimed to investigate the use of optic flow in a walkthrough animation and the associated problems such as occlusion, to compare optic flow with conventional methods by means of performance testing, and to examine which types of scene are most suited to optic flow animation.

The extension to multiple objects introduces the problem of occlusion. The image of each polygon was calculated separately using the optic flow algorithm from Experiment 1, and the resulting images stored in a BSP tree. The BSP tree was then used to order these images for compositing into the final image.

Unlike the first experiment, where conceptually the viewpoint was fixed and the object was in motion, the third experiment regarded the viewer as moving around a static scene. The two situations are mathematically equivalent, however.

6.6.1 Implementation and Interface

The implementation of this algorithm was programmed in C++, with the GLUT graphics library (an Open GL library).

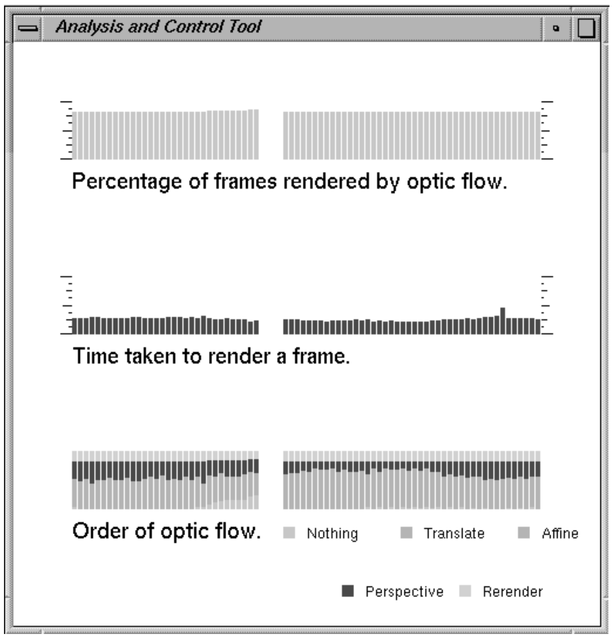

Analysis and Control Tool
Percentage of frames rendered by optic flow.
Time taken to render a frame.
Order of optic flow.
Nothing
Translate
Affine
Perspective
Rerender

Figure 24: The analysis tool displays information about the algorithm's performance, showing the history of the preceding 80 frames.

The GLUT library provided the possibility of pull-down menus, which were used to set all parameters such as thresholds and frame rate, and to select which scripts and scenes to use. The main pulldown menu is illustrated in Figure 23.

Analysis information was provided in a separate window (Figure 24). The key information is the percentage of polygons in the frame for which an approximation was used, how many polygons are being rendered with each order of optic flow, and the time taken to render the frame. The window displays a history of this information over the last 80 frames.

Sample scenes were constructed using DWB[1], and translated into OpenGL compliant scene code. The DWB structures also provide information for texture mapping. The sample scenes were as follows:

- a single polygon positioned at the origin, of size=0.1, 1, and 3 units.
- a scene of two polygons, one on the X axis and one at an angle (Figure 25(i))
- five colinear polygons along the X axis (Figure 25(ii))

[1]Designer WorkBench, ©Coryphaeus Software

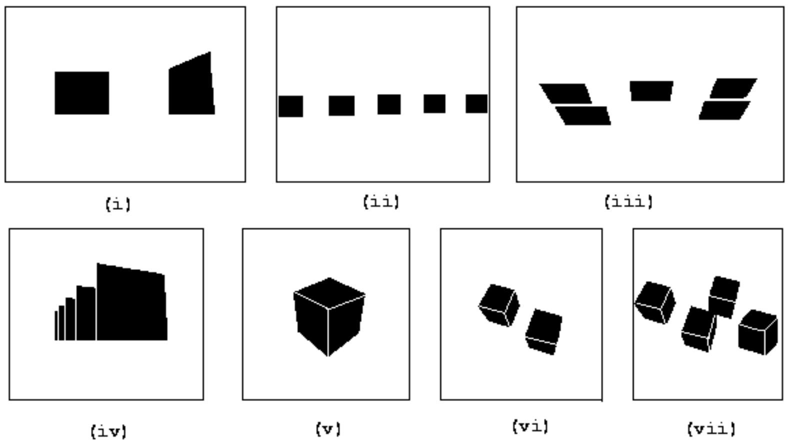


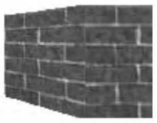
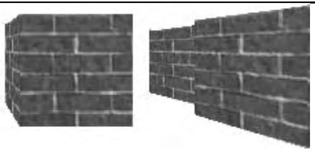
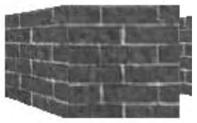
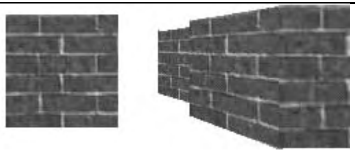
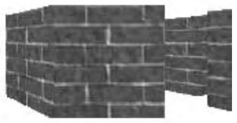
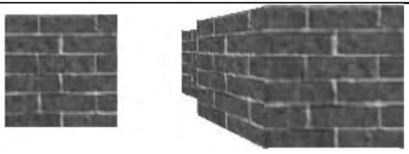
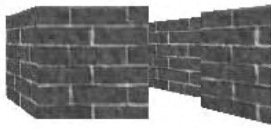
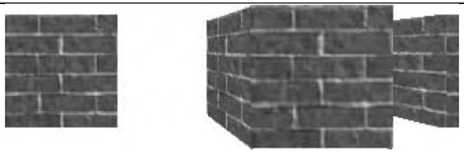
Figure 25: Images of the scenes used in performance testing. (i) Small scene of two polygons. (ii) Five colinear polygons with no occlusion. (iii) Five polygons, the two polygons in front occlude those behind. (iv) Five parallel polygons, the four polygons behind are occluded by the polygon in front. (v) Five polygons arranged as a cube, the bottom face is missing. (vi) Scene of ten polygons arranged as two cubes. (vii) Scene with twenty polygons arranged as four cubes

- five semi-occluded polygons, two on the X axis and three behind them. (Figure 25(iii))
- five parallel polygons all parallel to the X axis (Figure 25(iv))
- five polygons in the shape of a cube positioned at the origin (Figure 25(v))
- a scene with ten polygons arranged as two cubes (Figure 25(vi))
- a larger scene with twenty polygons arranged as four cubes (Figure 25(vii))

All sample runs were done on a 132 MHz R4600 Indy workstation. Although faster platforms were available, the Indy was preferred as it did not offer the advantage of hardware texture mapping which would disadvantage our implementation in the comparison with the conventional renderer. During the timed runs, no runtime information was displayed except the image itself, and no snapshots were taken.

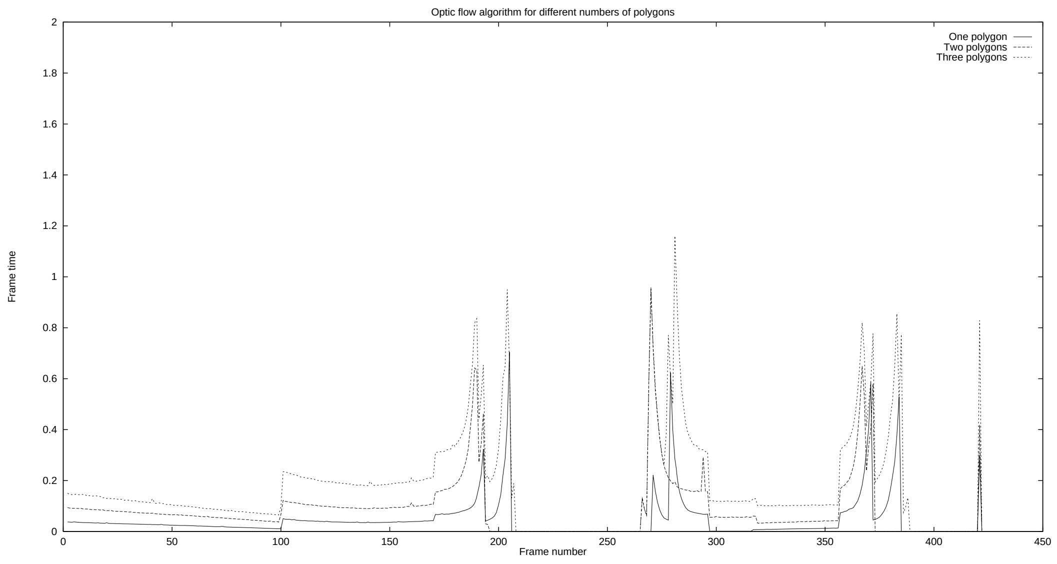
6.6.2 Sample frames

Sample frames from a scene involving 20 polygons arranged into four cubes is shown here. The snapshots are taken every ten frames, and show the scene as the viewer translates past it in the positive X direction. All polygons are mapped with the same texture.

 0	 40
 10	 50
 20	 60
 30	 70

6.7 Graphs and Comparisons

6.7.1 Optic flow algorithm with different numbers of polygons

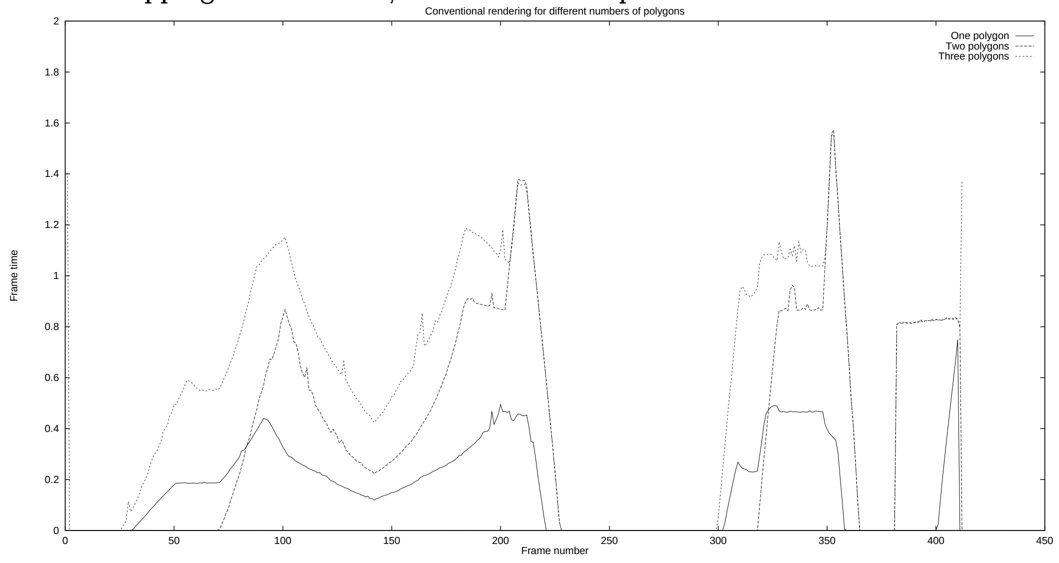


The graph shows the performance of the optic flow algorithm with simple scenes consisting of 1, 2 and 5 polygons. The script used consists of translation in X for frames 0 to 99, zoom out until frame 139 and then back in until frame 199, a rotation from frame 200 until frame 325, stationary until frame 346, and circling until frame 400. The spikes in the graph are caused when one or more polygons cross the image plane $Z = 1$ and are culled from the set of visible polygons.

The graph shows that frame time with the optic flow algorithm is directly proportional to the number of polygons.

6.7.2 Optic flow algorithm compared to conventional rendering

A conventional rendering algorithm was implemented using standard GLUT calls, executed on the same machine as the optic flow algorithm. Accelerated features such as hardware texture mapping were avoided, to make the comparison fair.



The graph shows the performance of the conventional rendering algorithm with small scenes consisting of 1, 2 and 5 polygons.

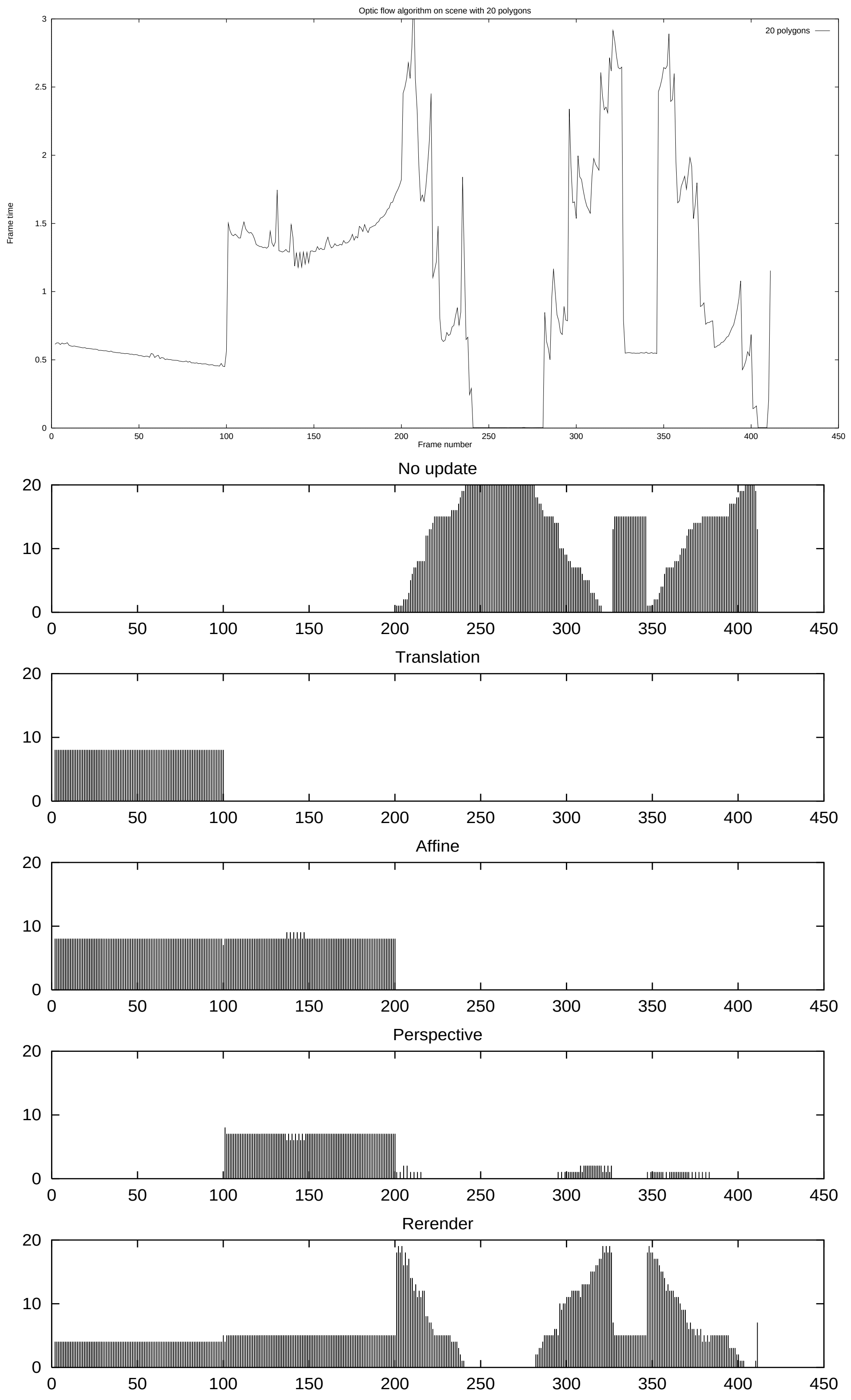
The most important observation is the significant saving achieved by the optic flow algorithm as compared to conventional rendering. This can be seen by comparing the preceding two graphs, which are drawn to the same scale.

In the case of conventional rendering, the dependence on polygon count is increasing less than linearly, probably because the GLUT implementation only does the texture calculations on visible portions rather than the whole of each polygon, while our optic flow implementation is guilty of overkill in these cases,. This is not a problem with our algorithm, but rather an indication that our implementation is more suited to scenes with relatively little occlusion.

Difficulties in heavily occluded scenes are common to most image based rendering algorithms; however, in this case the problem is more likely to be because our BSP tree-based approach to occlusions is too simplistic. The image of every polygon in the viewing frustum is calculated at each frame, even if that polygon turns out to be partially or totally occluded, resulting in a fair amount of wasted computation. Clearly a better method of visibility culling is needed. This is discussed further in Section 7.5.3

6.7.3 The orders of optic flow chosen

For this test we used the scene consisting of 20 polygons arranged into four cubes, and the compound script described earlier in Section 6.7.1. The time for rendering each frame is graphed on the next page, as well as the number of polygons in each frame which were produced using each of the five possible transformations: no update, translation, affine warp, perspective warp and rerendering.


Optic flow algorithm on scene with 20 polygons
20 polygons
Frame time
Frame number
No update
Translation
Affine
Perspective
Rerender

The graphs on the previous page show that no update was required for 33.0% of the polygons, translation for 9.7%, affine for 19.5%, perspective for 9.5% and rendering for 28.3%. Counting anything less than a perspective warp as a saving, this effectively means a saving has been achieved for 62.2% of the polygons.

6.8 Chapter Summary

This chapter analyses the results of experiments based on two versions of our algorithm for animating planar objects using optic flow analysis.

The first experiment used optic flow analysis to derive one of a hierarchy of frame to frame transformations to use at each frame. The visual results for this experiment are convincing, and the sample runs showed a time saving of 33% and more, with four of the six sample runs achieving more than 75% saving. This is because of the great advantage in choosing affine warping instead of rendering or perspective warping. We also noted that our measurement of the time savings are conservative, in that a more complex implementation may not use bitmapped textures as the underlying representation of planar objects, which would mean that rerendering would be even more expensive than in our implementation, making the potential savings more significant. Clearly there is significant benefit to be achieved by using this algorithm.

The second experiment was a more complex implementation addressing the issue of multiple polygons. The experiment used an extension to the second algorithm to address the issue of occlusion by means of a BSP tree. We observed that significant savings were achieved as compared to a conventional renderer. The algorithm performed well, using at most an affine warp in 62.2% of the polygons. However the results also show that the approach to occlusion is too simplistic, leading to wasted computations on polygons which are not visible.

Chapter 7

Conclusion

7.1 Outline

We began this dissertation with the aim of developing optic flow analysis as a mathematical tool for use in image based rendering, and to develop and test an algorithm for animating scenes consisting of planar surfaces.

This problem was first approached mathematically to develop the necessary theory, which was also compared to other approaches. The theory led to the design of an algorithm for image based rendering, and this algorithm was tested in a simple implementation of an animation system. Both our theoretical contribution and our experimental results have demonstrated the success of this approach: that the theory of optic flow analysis can be used successfully in an animation system.

In this conclusion we show how we have achieved our aims, first presenting our main contributions in summary, then in more detail in the order in which they were presented in this dissertation, and finally a discussion of our experimental results, highlighting the conclusions we have drawn from them. This dissertation then concludes with comments on the limitations we found, and our suggestions for further research.

7.2 The main contributions of this dissertation

The main contributions of this dissertation are contained in our survey of the literature of image based rendering, our contributions to the theory of optic flow analysis, the development of an algorithm based on this theory, and the advantages of this algorithm over other approaches.

7.2.1 Literature survey

In our survey of the image based rendering literature (Chapter 2) we developed a classification scheme dividing these approaches into three categories: image interpolation techniques, image layer techniques, and light field rendering. The results of this classification scheme are tabulated on Page 40. Our classification scheme showed that the the majority of recent image based animation systems have based their approach on multiple image layers. The results of this survey are discussed further in Section 7.3.1. Our work falls into this category, deriving its underlying theory both from animation systems based on layers, and from mathematical theory developed in vision research.

7.2.2 Theoretical contribution

The theoretical contribution of this work is an extension to the theory of optic flow analysis and its use in image based rendering (Chapter 4). Our work has added to the understanding of first and, more especially, second order derivatives of the optic flow field, and we have derived a new decomposition of these terms into bases. As far as we know this is the first time that the work of [40, 43, 42] on such decompositions of optic flow derivatives has been used for animation, and the first time the second order effects have been understood in this way.

Using the basis we derived, our work then related the derivatives of the optic flow field directly to the image transformations of a moving planar surfaces, broken down into a hierarchy of warps in terms of increasing implementation cost. We have extended the work of [5] to include perspective transformations into the hierarchy of measures of frame to frame coherence.

7.2.3 Algorithm for animation using optic flow

Based on the decomposition of the optic flow field using a Taylor series, we have developed an image based rendering algorithm for moving planar objects (Chapter 5). The frame to frame transformation is an approximation computed directly from the model and motion information, using the optic flow terms to compute which type of transformation to use within certain error thresholds. This algorithm was tested for a single object and also for a small scene with multiple polygons.

7.2.4 Comparison of our approach to Talisman

The most important recent work in the field of image based rendering is the Talisman architecture [38]. Like Talisman, our approach is based on deriving the image transformation (or warp) to apply to the image of a planar surface. The key difference is that our transformations are derived directly from the geometry, while Talisman's are derived statistically to match key points in one frame with their images in the next. Because our calculation is direct, there is very little increase in the cost of computing a perspective warp over an affine warp, whereas the Talisman authors conclude that calculating a best-fit perspective warp is infeasible. Because of this, we reach different conclusions to Talisman over the use of perspective transformations.

7.3 A more detailed look at the contributions of this dissertation

7.3.1 Survey of image based rendering techniques

The aim of this dissertation, as stated in the Abstract and in the opening paragraph of this chapter, is motivated by the trend towards image based rendering. In our survey of image based rendering, we found that image based rendering techniques [38, 19] are based on the high performance demands of realtime three dimensional animation, which are unlikely to be achieved by increasingly accurate simulation, mainly due to the power of human perception, the vast complexity of natural scenes, and the complexity at which light interacts instantaneously with a scene (2.1.1).

These demands and constraints necessitate approximate methods based on perceptual benefit rather than accuracy of simulation. Image based rendering substitutes image processing for aspects of physical simulation, making use of *coherence*, especially *frame to frame coherence* in the animation sequence. Parts of a frame in an animation may be reused provided the correct transformation is applied.

Out literature survey in Chapter 2 (tabulated in Section 2.6) discusses the characteristics of various image based rendering techniques, and develops a scheme to categorise these techniques into three groups, which are also analysed according to various criteria such as types of scene handled, ability to handle occlusion, types of frame to frame transformation accounted for, and so on. The three categories we used are as follows:

Interpolation techniques (2.3) compute additional views from a limited set of exiting views. Key issues are establishing point correspondences between the views, and the method of handling those new pixels which are not visible in the existing views.

Image layer techniques (2.4) break an image up into independent layers which are transformed separately and then composited to create the final image. We found that an approach based on layers has several advantages for an animation system, in terms of a natural partitioning of the scene, exploiting the coherence of discrete objects undergo different motions, a natural means of solving the problem of occlusion, and a means of allocating resources to updating the more important layers. Our approach later in this work was a layered algorithm using optic flow analysis to derive the transformations for each layer.

Light field rendering techniques (2.5) allow new views of a scene to be calculated by taking slices of a four dimensional light field, which captures all the optic information in a region surrounding the object.

Chapter 3 compared our work with *Scene shifting* [17], a layered technique which manipulates parallel slices of the scene to simulate three dimensional motion, based on a mathematical analysis. By reformulating the mathematics of scene shifting in optic flow notation, we showed that optic flow analysis is more general than scene shifting.

Our conclusion here is significant because optic flow is based on instantaneous image velocities, and hence on infinitesimal changes, whereas an animation system is always concerned with finite changes over a discreet time interval between frames. Scene shifting uses finite changes, but we showed how the error caused by extrapolating infinitesimal can be controlled using the second order Taylor derivatives of the optic flow field.

7.3.2 Optic Flow

In Chapters 4 and 5 we developed and extended the theory of optic flow analysis, which is used to calculate approximate frame to frame transformations for the image of a moving planar object. These transformations exploit various orders of frame to frame coherence, thus saving on expensive rendering steps at each frame.

We found the approximations using a Taylor analysis of the optic flow field, which gives a natural hierarchy of approximations and also keeps track of accumulated error due to the use of these approximations.

7.3.3 The significance of the Taylor Series decomposition

The main contribution of our analysis is in relating the Taylor series expansion of the optic flow field, up to second order terms, to a hierarchy of image transformations which can be practically implemented.

This relation is as follows: translation is computed directly from the optic flow vector itself (the zero order terms), the affine transformation from the first order derivatives of the optic flow field, perspective from the second order derivatives, and higher order terms determining when rerendering is necessary. We showed how the optic flow derivatives can be calculated directly from the underlying geometry and motion in the scene, in opposition to other systems such as Talisman [38], where the transformations are calculated indirectly by an approximate method based on matching projections of key points between the two images.

By using a Taylor we were able to determine the accuracy of each order of approximation by looking at the next order term in the series. This allowed us to set various thresholds where we considered each level of error to have become significant. This choice of several independent thresholds allows flexibility in future implementations of this or similar algorithms, which could exploit the different approximations differently depending on specific

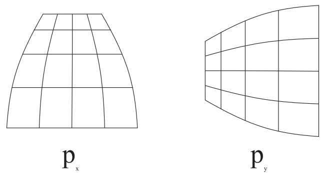


Figure 26: Our two new basis deformations p_x and p_y which span all possible second order optic flow effects for a moving planar image. They are related closely to perspective transformations of the image.

implementation or perceptual issues.

The transformations we have derived are the basic types of warp used in most texture mapping; furthermore the correspondence with the Taylor series orders them by increasing cost. This is significant as it suggests that our decomposition is the 'right' one, both in a mathematical sense, and for the purposes of implementation. We have thus provided a better understanding of the decomposition of the optic flow field itself, in terms of the images of real motions.

7.3.4 A new basis for second order optic flow derivatives

An important contribution of this dissertation is in understanding the second order Taylor derivatives of the optic flow field (4.8).

We have shown that the six second order terms can be reduced by linear combination to a new basis of just two independent terms. We found that these two terms, together with the zero and first order terms, relate closely and in a one-to-one fashion to the entries in a homogeneous matrix representing a perspective frame to frame transformation.

The relationship which we found was as follows. Given a planar object at position (X, Y, Z) with linear velocity (V_x, V_y, V_z) and rotational velocity $(\Omega_x, \Omega_y, \Omega_z)$, the six second order derivatives contain two which are zero, and the remaining four related as shown (Equation 34):

$$\begin{aligned} v_1^{(0,2)} &= v_2^{(2,0)} &= 0 \\ \tfrac{1}{2}v_1^{(2,0)} &= v_2^{(1,1)} &= (\Omega_2 - pV_3)/l \\ \tfrac{1}{2}v_2^{(0,2)} &= v_1^{(1,1)} &= (-\Omega_1 - qV_3)/l \end{aligned}$$

We then reduce this basis to just two non-zero terms as follows. The effects of these two terms are shown in Figure 26.

$$\begin{aligned} p_x &= \tfrac{1}{2}v_2^{(0,2)} + v_1^{(1,1)} \\ p_y &= \tfrac{1}{2}v_1^{(2,0)} + v_2^{(1,1)} \end{aligned}$$

This new basis gives us the means of calculating a perspective transformation between frames directly from motion and model information. The transformation between frames is then

$$\begin{pmatrix} a_1 & a_2 & t_x \\ a_3 & a_4 & t_y \\ p_x t & p_y t & 1 \end{pmatrix}$$

where a_i are the first order optic flow derivatives (related to affine transformations) and t_i is the translation.

All terms in this matrix are calculated directly, which gives us an advantage over other methods such as Talisman [38], which resorts instead to more time-consuming statistical methods such as finding a best-fit transformation using least squares between projections of key points.

This is particularly important for the perspective transformation. We showed how the matrix representing this transformation can be derived at very little extra cost over the affine transformation, which is a significant improvement over best-fit methods which are impractical for a 3×3 perspective matrix. The analysis thus allows us to consider implementing a further level of image transformation, which would otherwise be impractical due to difficulty in calculating the matrix representing the warp.

7.4 Experiment

In Chapter 6, we presented the results of two experiments using different variations of the optic flow animation algorithm.

Our first experiment used optic flow analysis to derive the frame to frame transformations to animate a single moving planar object. The inclusion of perspective transformations at this level of an image based rendering algorithm is a new contribution of this work. In our experiment, we found that substantial time savings, of amounts from 33% and up, are achieved. In most cases the advantage comes mainly from the ability to perform affine warping instead of rendering. Typically more than 50% of frames require an affine transformation or less. The biggest savings occur for polygons which are far away, small or slow moving, and objects undergoing simple translation. Our timing results for both experiments (6.4.8, 6.7) were conservative, because the cost of rerendering from a bitmapped texture in our implementation is lower than that in a more complex system using procedural texture maps or multiple objects making up a planar impostor.

We also presented a limited version of this algorithm which used only affine warping (6.5.1). We concluded this algorithm produced surprisingly convincing output, and was particularly fast, but was suitable only for single planar object. This algorithm may be used in certain limited cases, for instance a small planar object moving freely within a larger scene, for instance an avatar represented as a bitmap.

In the second experiment (6.6) we tested a different implementation of the perspective optic flow algorithm, which allows multiple polygons and uses a BSP Tree to handle the problem of occlusion. Again we conclude that substantial savings are possible, of the same order as the first experiment. But we also found that the BSP Tree approach was too simplistic to handle the complex occlusion issue adequately.

7.4.1 Why is affine warping such a realistic approximation?

Following the first experiment, we tested a restricted version of our algorithm using only affine warping to animate a single planar object. We noted that the results were surprisingly realistic even though we showed that in general affine warping is not particularly accurate (Figure 21). This is in contrast to the use of warping to produce a projected view of a scene: here affine warping is not sufficient (Figure 22), because of problems with alignment. In both experiments, affine warping was sufficient on more than half the frames in all sample runs - a significant observation.

There are several reasons why the affine transformation turns out to be so good. Firstly it is already a good approximation in geometric terms when the object is small or far away, where perspective effects on an individual polygon are less significant. The viewer is already used to interpreting an image with few perspective effects in this situation.

Secondly, in an animation system where the timestep is small related to the actual motion of the object, the two images are very similar. This means the transformation matrix will be close to an identity, and the affine and perspective matrices will be fairly similar. This is unlike the possibly large difference between a texture of a surface, and the image of that surface, where a perspective warp is clearly necessary.

7.4.2 The limitations of depth dependency

An important limitation is the dependence on depth complexity: image based rendering performs less well in scenes with large numbers of layers and multiple occlusions. This is a problem common to both image interpolation and image layer techniques, for different reasons: image interpolation copes poorly with the holes caused by occlusion, and image layer techniques require larger numbers of layers and hence more calculations. For this reason, different approaches to rendering and occlusion are usually used in relatively unoccluded scenes such as landscapes, and heavily occluded scenes such as interiors. Our implementation of BSP trees to handle occlusion proved to be too simplistic, and clearly more work is needed here.

7.4.3 Summary

We began this dissertation with the aim of developing the mathematical tools necessary to use optic flow analysis as a technique in image based animation. We have achieved this aim and contributed to this field through a further understanding of the second order derivatives of the optic flow field. In addition our test implementations of an optic flow animation system have successfully shown that significant savings can be achieved, and that there is scope for further development of these techniques.

7.5 Possible further work

7.5.1 Perceptual effects of the error thresholds

We have tried to build the model so that the 'correct' and rendered frames are perceptually the same, by ensuring that the difference is always less than one pixel. However, this requirement can be relaxed in several ways. These thresholds may be reduced or increased by the same amount, or they may be changed independently.

The guiding criteria of such a study would be the perceptual effects of these thresholds. In this work the thresholds were geometrically uniform (all set at one pixel) but possibly not perceptually uniform: for instance it is possible that a one pixel error in a translation is more noticeable than a one pixel error in a perspective warp. If this were the case it would be possible to relax one threshold and tighten another to increase the perceptual accuracy of the animation, and yet tune the algorithm in such a way that the cost remains constant. A study of these effects would thus allow a more flexible algorithm to be developed.

An experiment to measure these perceptual effects could be could be based on a number of criteria, including the following:

- The user's awareness of artifacts such as 'wobbling' as adjacent polygons appear to shift independently, the jump as a polygon moves from one level of transformation to another, loss of detail or creeping textures due to inaccuracies in warping.
- The viewer's feeling of comfort in navigating the scene.
- More specific measures of the viewer's ability to extract spatial information from the animation, such as judgment of heading or depth perception.

These criteria should be assessed with varying thresholds, and compared to a benchmark rendering algorithm at high accuracy.

7.5.2 Allocation of rendering resources

In any algorithm where a resource of set size must be distributed among tasks of different costs and benefits, the problem of optimising the allocation of resources for maximum benefit is a variety of Knapsack Problem. In this case the resource is the fixed rendering time available between frames, and each polygon may be rendered at a number of levels, from no update up to rerendering.

Our algorithm defines a minimum acceptable level and produces the frame at that level. However this may leave spare processing time available, or alternately require more than the available time. In these cases a more sophisticated decision procedure could be implemented to make better use of the resources.

While in general this variety of knapsack problem is NP Complete, a number of more efficient approximate solutions are known, and could be implemented.

7.5.3 Improved occlusion handling

Our BSP tree approach to handling occlusion in Experiment 3 is less than optimal, since the image of every polygon in the viewing frustum is calculated even if that polygon is partially or totally occluded. This could be addressed by postponing the actual warping or rendering of polygons until after the BSP tree has been used to determine visibility. This implementation would be fairly complex: the image would be subdivided into regions which depict different polygons, and each region would be filled by warping the image of that polygon from the preceding frame. However the image from the preceding frame may be incomplete due to occlusion, so the resulting holes would have to be filled by some method such as rerendering those pixels, or blending across the holes.

7.5.4 Predictive use of optic flow

In many types of motion, smoothness or coherence of the underlying motion leads to coherence in terms of constant or smoothly varying optic flow terms, and therefore of the frame to frame transformations. Our algorithm has not taken advantage of this, since the optic flow calculations are done independently at each frame.

One way to take advantage of this would be to calculate how long a given transformation would remain valid, assuming the motion does not change. Until that time, the accumulated errors could be incremented by the correct constant amount at each frame, and the correct transformation applied, without the optic flow calculations being redone. This would also allow for pipelining.

7.5.5 Optimisation of warping algorithms

Optimisations to warping algorithms would give an advantage. Optimisations could include multiresolution texture mapping techniques such as mipmaps, or they could be based on the special requirements of frame to frame transformations.

Currently there are many optimisations available for optimising warping algorithms such as perspective and affine warping, but these are fairly general, usually based on the requirements of texture mapping in rendering, with a wide range of possible warps from texture to image space.

Our requirements are somewhat different: since consecutive frames are very similar, we need a finer control over warps which are close to the identity matrix. Some further investigation on the specific mathematical issues this raises would be useful, providing a finer tool than existing texture mapping techniques, more suited to image based rendering. It may also be possible to optimise multiresolution textures such as mipmaps for the specific requirements of an optic flow based algorithm.

Further investigation into our method of decomposing a warp into a perspective component and an affine component (5.4.3) could provide better results than those in Section 6.3. Depending on the type of scene and the possible motions of the viewer, we could also consider breaking the transformations down into narrower classes: for instance, a vertical wall viewed by an upright viewer would result in vertical scanlines being preserved, within a certain scaling factor. It is possible to implement such specific types of warps more efficiently than a general perspective warp.

Appendix A

Warping

A.1 Affine and perspective warping algorithms

Image warping methods used in this dissertation are affine warping and perspective warping. The warping algorithms come from the field of texture mapping[15, 14, 16, 7, 8].

In all cases, a matrix M represents the transformation from input (texture) coordinates (u, v) to output (image) coordinates (x, y):

$$M : \begin{pmatrix} u \\ v \end{pmatrix} \longmapsto \begin{pmatrix} x \\ y \end{pmatrix}$$

The algorithms we use all step through output coordinates (x, y) and use the inverse matrix M^{-1} to find out which input point maps under M to this location. For simplicity, all our algorithms use point sampling, so the nearest pixel to this point is copied directly to the output image:

```
warp(in_image, M, out_image)
    calculate size of output image
    for $y$ runs through each scanline
        for $x$ runs from left to right of scanline
            $(u, v)$ = nearest integer value to $M^{-1}(x, y)$
            if $(u, v)$ is inside the input image
                copy input pixel at $(u, v)$ to output pixel location $(x, y)$
```

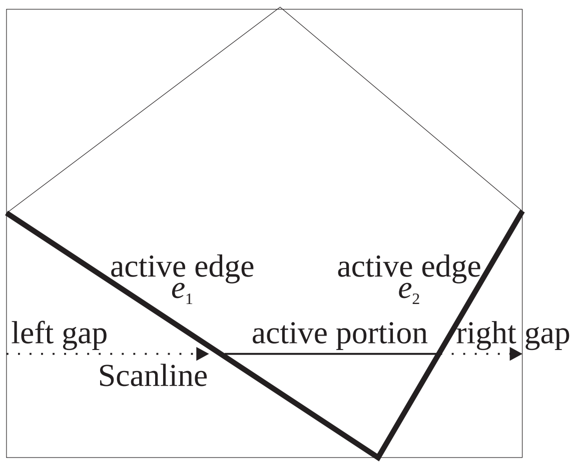


Figure 27: Scanline y in the output image runs through three coherent sections: a left and a right gap, and an active portion which contains the transformed input image. The boundaries of these sections are delimited by two active edges, e_1 and e_2, which intersect scanline y at points s_1 and s_2.

Check the validity of the corresponding input pixel at each output pixel is costly and unnecessary: as in Figure 27, each output scanline is broken into at most three coherent sections: a left and a right gap, which do not contain pixels from the input image, and an active portion which contains the transformed input image.

The endpoints of the active section, s_1 and s_2, are determined by the intersections of the two active edges, e_1 and e_2, with the scanline. Since the bounding quadrilateral is always convex, there are exactly two such intersections, disregarding horizontal edges which coincide with a scanline.

Since the endpoints s_1 and s_2 follow a bresenham line through the output image, they can be calculated incrementally using bresenham's algorithm.

If the matrix M is a 2×2 affine matrix, then the expression for (u, v) is

$$(u, v) = (ax + by, cx + dy)$$

where $M^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. The values of u and v also follow a bresenham line in (x, u) and (x, v) space, so they can be found efficiently.

If M is a 3×3 homogeneous matrix, then the expression for (u, v) is

$$(u, v) = \left(\frac{ax + by + c}{gx + hy + i}, \frac{dx + ey + f}{gx + hy + i}\right)$$

where $M^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. The multiplies and divides in the calculation of u and v can be reduced to two multiplies and one divide per pixel, but remains much more expensive than the affine warp above.

The final warping algorithm looks as follows:

```
warp(in_image, M, out_image)
    calculate size of output image
    create a list of non-horizontal edges ordered by lowest $y$-value
    set the first two edges to active
    for $y$ runs through each scanline
        calculate $s_1$ and $s_2$ incrementally
        for $x$ runs from left of scanline to $s_1$
            put blank pixel
        for $x$ runs from $s_1$ to $s_2$
            $(u, v)$ = nearest integer value to $M^{-1}(x, y)$
                copy input pixel at $(u, v)$ to output pixel location $(x, y)$
        for $x$ runs from left of scanline to $s_1$
            put blank pixel
        if $y$ is the end of one or both of the active edges
            replace with next active edge
```

By storing the edges of the polygon which contains the bitmap within our image data structure, we not only have an efficient method of culling the blank portions of the bounding rectangle, but we can also prevent multiple warps of the same image resulting in a buildup of these empty areas, as in Figure 28. The bounding quadrilateral can easily be extended to any convex bounding polygon: convexity is preserved by both affine and perspective warps, so we are guaranteed that each output scanline will cross exactly two active edges. If we allow for a scanline to be divided into more than three sections, we can extend the bounding quadrilateral to an arbitrary polygon.

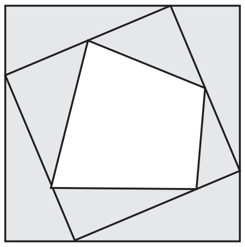


Figure 28: Buildup of empty space around a rectangular bitmap which results when warps are concatenated. This makes it essential to store more information than simply the bounding box of the bitmap.

A.2 Comparison of the time required for affine vs. perspective warping

A comparison between the affine and perspective warping algorithms was performed on a 132MHz R4600 Indy. Both warping algorithms used point sampling. The benchmark image was a 243×198 24-bit image of a butterfly, and a half-size version of the same image. The matrices used were the identity matrix, a 45° counterclockwise rotation, and a matrix

$$M = \begin{pmatrix} 0.8 & 0.6 & 0.0 \\ -0.4 & 0.8 & 0.0 \\ 0.1 & -0.3 & 1.0 \end{pmatrix}$$

Each warp was applied between 3000 and 5000 times and the resulting times averaged. The results of the comparison were as follows:

Time for warping algorithms		
243×198 image		
matrix	**affine**	**perspective**
I	28.93ms	63.36
Rot45	33.93	68.23
M	29.48	62.91
121×99 image		
M	6.92	14.60

This table shows us firstly that the perspective warping code is slightly more than twice as expensive as the affine warping. This is because of the divisions for each pixel in the perspective algorithm.

Secondly, for a given matrix the time is proportional to the size of the original image (or equivalently, to the size of the output image). This can be seen from the fact that the times for the smaller image (25% of the size of the large image) are a little less than 25% of the times for the larger image.

Thirdly, the overhead for non-image pixels within the bounding box of the output image is relatively small: in the Rot45 case, the output image is twice the size of the Identity matrix case, of which 50% of the pixels are non-image pixels. The overhead for these empty pixels is about 14% additional time spent in the affine algorithm and 7% in the perspective algorithm.

Bibliography

[1] E. H. Adelson. Layered representations for vision and video. In *IEEE Computer Society Workshop: Representation of Visual Scenes*, 1995.

[2] M. Agrawala, A. C. Beers, and N. Chaddha. Model-based motion estimation for synthetic images. In *ACM Multimedia 95*, 1995.

[3] S. Avidan and A. Shashua. Novel view synthesis in tensor space. In *Proc. Computer Vision and Pattern Recognition*, pages 1034–1040, 1997.

[4] D. C. Banks. The ImageSwitcher: A proposed system architecture to reduce VR lag. *Eurographics Hardware Workshop*, pages 71–77, 1996.

[5] E. H. Blake. *Complexity in Natural Scenes: A Viewer Centered Metric for Computing Adaptive Detail.* PhD thesis, Queen Mary College, London University, 1989.

[6] S. E. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics Proceedings, Annual Conference Series*, pages 279–288, 1993.

[7] D. Eberly. Mapping quadrilaterals to quadrilaterals. Technical report, University of North Carolina, 1995.

[8] M. Feldman. Texture mapping. In *The Win95 Game Programmer's Encyclopedia.* Preprint, 1997.

[9] Foley, van Dam, Feiner, and Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 1995.

[10] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics*

Proceedings Annual Conference Series, volume 27, pages 247–254. ACM SIGGRAPH, August 1993.

[11] J. Gibson. *The ecological approach to visual perception.* Houghton Mifflin, 1979.

[12] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *SIGGRAPH 96*, 1996.

[13] S. J. Gortler, L. wei He, and M. F. Cohen. Rendering layered depth images. *Microsoft Research technical report*, 1997.

[14] P. S. Heckbert. Texture mapping polygons in perspective. Technical Report 13, Computer Graphics Lab, New York Institute of Technology, 1983.

[15] P. S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, 1986.

[16] P. S. Heckbert and H. P. Moreton. Interpolation for polygon texture mapping and shading. In D. F. Rogers and R. E. Earnshaw, editors, *State of the art in Computer Graphics: Visualization and Modeling*, pages 101–111. Springer-Verlag, New York, 1991.

[17] G. R. Hofmann. The calculus of the non-exact perspective projection. In *Computer Graphics Forum (Proc. of Eurographics '88)*, pages 429–442, 1988.

[18] G. R. Hofmann. *Das Kalkül der nicht-exakten perspektivischen Projektion.* PhD thesis, Univ. Darmstadt, 1989.

[19] R. L. Holloway. *Registration Errors in Augmented Reality Systems.* PhD thesis, University of North Carolina at Chapel Hill, 1995.

[20] B. K. P. Horn. *Robot Vision.* The MIT Press, 1986.

[21] E. Horvitz and J. Lengyel. Perception, attention, and resources: A decision-theoretic approach to graphics rendering. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence*, pages 238–249, 1997.

[22] S. B. Kang. A survey of image-based rendering techniques. Technical Report 97-4, Cambridge Research Laboratory, 1997.

[23] J. J. Koenderink. Optic flow. *Vision Research*, 26(1):161–180, 1986.

[24] J. W. Krutch. The colloid and the crystal. In I. Gordon and S. Sorkin, editors, *The Armchair Science Reader*. New York: Simon and Schuster, 1959.

[25] S. Laveau and O. Faugeras. 3-D scene representation as a collection of images and fundamental matrices. In *Proc. Int. Conf. on Pattern Recognition*, pages 689–691, 1994.

[26] D. N. Lee. The optic flow field: the foundation of vision. *Phil. Trans. R. Roc. Lond.*, 290:169–179, 1980.

[27] J. Lemordant. Visualization of depth maps. In B. Facidieno, I. Herman, and C. Pienovi, editors, *Computer graphics and Mathematics*. Springer-Verlag, 1992.

[28] J. Lengyel and J. Snyder. Rendering with coherent layers. In *Computer Graphics Proceedings, Annual Conference Series*, 1997.

[29] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96*, 1996.

[30] A. Mason and E. Blake. Automatic hierarchical level of detail optimization in computer animation. In *Eurographics Workshop on Rendering*, 1997.

[31] L. McMillan. Computing visibility without depth. *Unpublished paper*, 1996.

[32] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics Proceedings, Annual Conference Series*, pages 39–46, 1995.

[33] S. M. Seitz and C. R. Dyer. Physically-valid view synthesis by image interpolation. In *Proc. Workshop on Representation of Visual Scenes*, pages 18–25, 1995.

[34] S. M. Seitz and C. R. Dyer. View morphing. In *SIGGRAPH 96*, 1996.

[35] J. Seymour. Virtually real, really sick. *New Scientist*, pages 34–37, January 1996.

[36] J. Shade, C. Lischinski, D. H. Salesin, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96*, 1996.

[37] F. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16, 1997.

[38] J. Torborg and J. T. Kajiya. Talisman: Commodity realtime 3D graphics for the PC. In *SIGGRAPH 96*, 1996.

[39] B. A. Wandell. *Foundations of vision.* Sinauer Associates, 1995. 612.84 WAND.

[40] A. M. Waxman and S. Ullman. Surface structure and three-dimensional motion from image flow kinematics. *The International Journal of Robotics Research*, 4(3):72–94, 1985.

[41] A. M. Waxman and K. Wohn. Contour evolution, neighbourhood deformation and global image flow: Planar surfaces in motion. *The International Journal of Robotics Research*, 4(3):95–108, 1985.

[42] A. M. Waxman and K. Wohn. Image flow theory: A framework for 3-D inference from time-varying imagery. In C. Brown, editor, *Advances in Computer Vision*, chapter 3, pages 165–224. Lawrence Erlbaum Ass. Hillsdale, New Jersey, 1988.

[43] K. Wohn and A. M. Waxman. The analytic structure of image flows: Deformation and segmentation. *Computer Vision, Graphics and Image Processing*, 49:127–151, 1990.

[44] D. N. Wood, A. Finkelstein, J. F. Hughes, C. E. Thayer, and D. H. Salesin. Multi-perspective panoramas for cel animation. In *Computer Graphics Proceedings, Annual Conference Series*, 1997.