UNIVERSITY
of
GLASGOW

Computing Science

Stylised Procedural Animation

Jinhui Yu

A thesis submitted for the degree of Doctor of Philosophy

Stylised Procedural Animation

Jinhui Yu
Department of Computing Science
University of Glasgow

Abstract

This thesis develops a stylised procedural paradigm for computer graphics animation. Cartoon effects animations – stylised representations of natural phenomena – have presented a long-standing, difficult challenge to computer animators. We propose a framework for achieving the intricacy of effects motion with minimal animator intervention.

Our approach is to construct cartoon effects by simulating the hand-drawing process through synthetic, computational means. We create a system which emulates the stylish appearance, movements of cartoon effects in both 2D and 3D environments. Our computational models achieve this by capturing the essential characteristics common to all cartoon effects: structure modelling, dynamic controlling and stylised rendering.

To validate our framework, we have implemented a cartoon effects system for a range of effects including water effects, fire, smoke, rain and snow. Each effect model has its own static structure such as how the different parts are related spatially, and dynamic structure such as how the different parts are related temporally. The flexibility of our approach is suggested most evidently by the high-level controls on shape, colour, timing and rendering on the effects. Like their hand-drawn counterparts, they move consistently while retaining the hand-crafted look.

Since the movements of cartoon effects are animated procedurally, their detailed motions need not be keyframed. This thesis therefore demonstrates a powerful approach to computer animation in which the animator plays the role of a high level controller, rather than the more conventional hand-drawing slave. Our work not only achieves cartoon effects animation of unprecedented complexity, but it also provides an interesting experimental do-

main for related research disciplines toward more creative and expressive image synthesis in animation.

Acknowledgements

I would like to thank my supervisors Dr. John W. Patterson and Mr. Bill W.Findlay. This thesis would not have been possible without their encouragement and support.

My special thanks go to Professor Frank Van Reeth from the Expertisecentrum Digitale Media (EDM), Limburgs Universitair Centrum (LUC), Belgium, for generously agreeing to serve as external examiner of my thesis. Special thanks also go to Dr. Ron Poet for serving as internal examiner and Professor Chris Johnson for serving as the chairman on my thesis committee.

I thank the wonderful people of the GIST. My years at University of Glasgow would have been dull without them. I would like to express my sincere gratitude to Fionnuala Johnson, Neble Jean Christophe.

I would especially like to thank Stephane Etienne and Intaek Kim for their generous help and interesting talks on a variety of topics during my PhD program.

I particularly want to thank my friend Vivien Samuels for English checking.

Contents

I Introduction **1**

1 Traditional Animation 1
2 Computer Animation 2
2.1 Inbetweening Systems 3
2.2 Procedural Approaches 4
3 Stylised Effects Animation: Motivation and Challenges . . . 5
4 Methodology: Stylised Procedural Animation 7
4.1 Criteria and goals 7
4.2 Cartoon Effects 7
4.3 From Internal Models to Stylised Drawings 8
4.4 From Stylised Drawings to Computer Models 8
4.5 Dynamic Control 9
4.6 Stylised Rendering 9
4.7 From 2D to 3D 9
4.8 Fidelity 10
5 Contributions and Results 10
6 Thesis Overview 11

II Backgrounds of CAA and Procedural Animation **13**

1 Shape Transformation and Motion Control 13
2 Shape Approaches 15
2.1 Cartesian Coordinate Linear Interpolation 15
2.2 Moving Point Constraints 15
2.3 Polar Coordinate Linear Interpolation 17
2.4 A Variant of PCLI 19
2.5 A Physically Based Approach 20
2.6 Multi-resolution 21
2.7 Circle Union 22
3 Skeleton Approaches 23
3.1 Burtnyk and Wein's Skeleton Approach 24
3.2 Star-skeleton Approach 26
3.3 Localised Rotation and Scaling using Quaternions . . 29

4 Procedural Approaches in CAA 32
4.1 A Walk Model 32
4.1.1 Angle Constraints 33
4.1.2 The Speed of Walks 34
4.1.3 The Types of Walks 34
4.2 A Head Turning Model 35
4.3 Specification of the rotation axis 37
4.4 Edge detection 37
4.5 Hidden line removal 37
5 Assessment Criteria for 2D Shape Deformation 38
5.1 Area Preservation 39
5.1.1 Local Validity 41
5.2 Dynamics 42
5.3 Timing 44
5.4 Conclusion 45
6 Procedural Animation 46
7 Particle Set Animation 47
8 Animating Water Effects 49
9 Animating Gaseous Effects 50
10 Summary 50

III Cartoon Effects System **52**
1 Modelling 53
2 Control 54
2.1 Shape Control 54
2.2 Skeleton Control 54
2.3 Distribution Control 55
3 Rendering 55

IV Water Effects **56**
1 Flowing Water 56
1.1 Boundary Curves 59
1.2 Horizontal Waves 60
1.3 Vertical Waves 60
1.4 Caps 61
1.5 Foams 62
1.6 Structure of the Model 62
1.7 Colouring 63
1.8 Working in 3D 63
1.9 Strobing 64
1.10 Summary 65
2 Water Jet 65

2.1 Skeleton 66
2.2 Boundary Curves 67
2.3 Waves 67
2.4 Drops 68
2.5 Structure of the model 69
2.6 Working in 3D 70
2.7 Summary 71
3 Water Ripples 71
3.1 Ripple Skeleton 72
3.2 Ripple Shape Rendering 73
3.3 Splitting Up 74
3.4 Structure of the Model 75
3.5 Result 76
3.6 Summary 76
4 Shimmering 77
4.1 Basic Waves 78
4.1.1 Stylish Wave 78
4.1.2 Realistic Wave 79
4.2 Distribution 79
4.3 Model Control 79
4.4 Results 80
5 Reflections 80
5.1 Base Skeleton 81
5.2 Dynamic Skeleton 82
5.3 Shaping 83
5.4 Colouring 84
5.5 Structure of the Model 84
5.6 Result 85
6 Summary 86

V Fire and Smoke **87**
1 Fire 87
1.1 Fire flame model 89
1.2 Flame Position Skeleton 90
1.3 Flame Skeleton 91
1.3.1 Skeleton Types 92
1.3.2 Skeleton Symmetry 92
1.4 Flame Shape 93
1.5 Connection of Flames 94
1.6 Construction of the Matrix 95
1.7 Top Flames 96
1.8 Structure of the Model 96

1.9 Analysis of Parameters 97
1.10 Colouring 99
1.11 Simulation of Wind Effect 99
1.12 Summary 101
2 Smoke 101
2.1 Skeleton 102
2.2 Shape Rendering 103
2.2.1 Smoke Skein 104
2.2.2 Smoke Puffs 105
2.3 Model Structure 106
3 Conclusion 107

VI Rain and Snow **109**
1 Rain 109
2 Light Rain 111
3 Heavy Rain 112
3.1 Drops Moving in the Air 112
3.2 Drops Hitting the Ground 114
3.2.1 Modelling of Individual Drops 114
3.2.2 Distribution of Hitting Drops 115
3.2.3 Life Controller 115
3.2.4 Structure of the Model 115
3.3 Result 116
3.4 Summary 117
4 Snow 117
4.1 Snow Track Skeleton 118
4.2 Snow Flakes 119
4.3 Structure of the Snow Model 119
4.4 Summary 120

VII Conclusion and Future Work **122**
1 Additional Impact in Animation 123
2 Impact on Computer Art 125
3 Potential Applications in Art Education 125
4 Other Stylised Animations 126
4.1 Animation of Plants 126
4.2 Character Animation 127
5 Future Research Directions 128
5.1 A Variety of Cartoon Effects 128
5.2 Interaction between Objects and Effects 128
5.3 Hyperprocedure 129
5.4 From Realism to Stylisation 129

5.5 Stylised Animation 130

A Brush and Painterly Rendering **132**

1 Survey of the Existing Brush Models 133

2 A Skeletal Spray Brush Model 134

2.1 Skeleton 134

2.2 Footprint 135

2.3 Skeleton Attributes 136

2.4 Structure of the Model 137

2.5 Applications of the Brush 137

2.5.1 Computer Art 137

2.5.2 Computer Chinese Calligraphy 138

3 Rendering for Animation Using Brushes 139

4 Rendering Using Particles 141

5 Painterly Line and Surface Rendering 142

5.1 Line Rendering 143

5.2 Surface Rendering 144

6 Summary 147

List of Figures

II.1 Distortion of CCLI due to rotation 15

II.2 Completed patch net work 16

II.3 Illustration of PCLI 17

II.4 Hand moving about elbow 18

II.5 Definition of the variables 19

II.6 Distortion of the polar technique 19

II.7 Dancer 20

II.8 Geometric evolution of a closed simple curve 21

II.9 Union of circles representation and segementation 22

II.10 The mesh 24

II.11 The overall distortion exibited by the coordinate space 25

II.12 The "bones" and the surrounding image distribution 26

II.13 Star skeleton representation 27

II.14 Compatible star skeleton 28

II.15 Star-skeleton blend 29

II.16 Orientation control 30

II.17 Non-linear rotation and scaling 30

II.18 Spring skeleton controlled by non-linear rotation and scaling 31

II.19 Handrawn walk 32

II.20 Angle variations of upper and lower legs 33

II.21 A walk using angle constraints 35

II.22 Hand drawn head turn 35

II.23 Construct head skeleton 36

II.24 A monkey head turn 38

II.25 Area preservation criterion 40

II.26 Comparative results of morphing alogrithms applied to a run cycle 41

II.27 A practical cartoon character 42

II.28 A correct run cycle in hand drawn animation 42

II.29 Angle variation of two legs for a run 43

II.30 A cannonball and a balloon 44

II.31 Comparative results of morphing techniques applied to a fancy shape 46

III.1 System overview of the cartoon effects 53

IV.1 Hand drawn water wave 57

IV.2 Flowing water drawing process 58

IV.3 Boundary curves and horizontal waves 59

IV.4 Sample 1 of 3D flowing water model 63

IV.5 Sample 1 of the 3D flwoing water model 64

IV.6 Water jet 66

IV.7 Water jet skeleton 67

IV.8 Water jet wave 68

IV.9 Water jet drop 68

IV.10 Water jet generated by the model 70

IV.11 Ripple from a partially submerged object 71

IV.12 Ripple skeleton 72

IV.13 Ripple width function 74

IV.14 Switch and shape weight function 75

IV.15 Ripples generated by the model 76

IV.16 Shimering of sunlight 77

IV.17 Moving surface 77

IV.18 Stylish basic waves 78

IV.19 Realistic moving surface generated by the model 80

IV.20 Reflection of a walking character 81

IV.21 The reflection of a rougher water surface 82

IV.22 A cycle of a bright light 83

IV.23 Base skeleton and dynamic skeleton 84

IV.24 Reflections generated by the model 86

V.1 Candle flame 88

V.2 Dynamic candle flame 89

V.3 A hand drawn fire cycle 90

V.4 Flame skeleton 91

V.5 Skeleton types 92

V.6 Flame shape 93

V.7 Connection types 94

V.8 Parameters associated with the model 97

V.9 A fire series generated ny the model 99

V.10 A coloured fire frame 100

V.11 Effect of wind on fire 101

V.12 Hand-drawn smoke 102

V.13 Smoke skeleton 103

V.14 Smoke shape rendering functions 104

V.15 Smoke puff skeleton 106

V.16 Smoke skein 107

V.17 Smoke puffs 108

V.18 Billowing smoke 108

VI.1 Rain 110

VI.2 Light Rain 111

VI.3 Light rain generated by the model 113

VI.4 Skeleton and shape of the drop hitting the ground 114

VI.5 Heavy rain generated by the model 117

VI.6 Snow 118

VI.7 Snow track skeleton 119

VI.8 A snow frame generated by the model 120

A.1 Footprint, skeleton and shape of the brush 135

A.2 Brush width attributes 136

A.3 An image generated by skeletal spary brush 138

A.4 Different styles of Chinese calligraphy 138

A.5 A Chinese character generated by skeletal spary brush 139

A.6 A stylised keyframe drawing 140

A.7 A frame rendered by particles 142

A.8 A pavement with a water color look 146

Chapter I

Introduction

1 Traditional Animation

Literally to "animate" is "to give life to", so, animating is moving something which cannot move itself. Traditional animation is oriented mainly towards the production of 2D cartoons. Every frame is a flat picture and is purely hand-drawn. This makes traditional animation a costly industry, in both time and money. To make an animation, the overall work can be thought of as falling into two phases, that of design which might be 15% of the total effort and implementation which might be about 85%.

Animation is not just a series of funny drawings strung together in movements. At its most creative, it is a truly beautiful art form. The tradition of drawn animation is a relatively short one compared with other visual arts. It has only been in this century that the technology to produce any film – let alone an animated film – has been available. Animated film-making, in its widest expression, is not, however, traditionally an art form of individuals genius. A large team of dedicated, talented and cooperative artists is required to complete a high quality animation film. On a large-scale production, it is important that the team functions efficiently. A typical team for the production of a large-scale animated film includes a lot of people: a director; a producer; a number of animators and assistant animators; possibly a team of in-betweeners; a whole assortment of cleanup artists, tracers and painters, and special-effects artists; plus checkers, editors, and the rostrum cameraman. In addition, there are the production and administrative staff. Whether the film being produced is a 30-second television commercial or a full-length animated feature, the process of animation should follow

certain structured procedures. First, a preliminary story is decided on and a story board is developed which lays out the action scenes by sketching representative frames and writing down an explanation of what is taking place. This is used to review and critique the action. The detailed story is worked out which identifies the actions involved in more detail. Key frames are then identified and produced by master animators to aid in confirmation of character development and image quality. A test shot is a short sequence rendered in full colour to further test the rendering and motion techniques. To completely check the motion, a pencil test may be shot which is a shot of full-motion but low quality images such as pencil sketches. Associate and assistant animators are responsible for producing the frames between the keys; this is called in-betweening. Inking refers to the process of transferring the pencilled frames to cels, and then colouring is finally applied to these cels.

These days, the main market for animation is television and the budgets are usually small, therefore the cost-effectiveness dominates the thinking of the purchaser. Hand drawn animation relies on a large number of people thus the high-wages result in the high cost of the production. If something like making and painting the in-between drawings and even the background can be done by computer, then, hopefully, the cost of the production can be reduced. Furthermore, computer also enables things that can not otherwise be done by hand in making animation.

2 Computer Animation

The term "computer animation" today has come to mean almost anything involving a computer and a moving picture [TT85a] [TT85b] [PT88]. Computer animation, CA, is thus the core technology for such areas as multimedia, scientific visualisation and virtual reality.

There are two main categories of computer animation: *computer-assisted animation* (CAA) and *computer generated animation.* CAA usually refers to 2D and 2.5D systems that computerise the traditional animation process. Interpolation between key shapes is typically the algorithmic use of the computer in the production of this type of animation, in addition to the more "non-animation" uses of the computer such as inking, implementing a virtual camera stand, shuffling paper, and managing data. Computer generated animation usually refers to 3D computer animation which involves three main activities: object modelling, motion specification and image rendering. An advantage of computer generated animation is that the camera is virtual,

which means that there is absolutely no restriction on its movement.

2.1 Inbetweening Systems

Since drawing is a main element in producing animation, it attracts many researchers to work on automatic in-betweening which replicates the in-betweening process of the hand drawn animation, i.e. when given two key shapes, they transform the shape continuously from one to the other (the process is also called *morphing*, *shape blending* and *shape deformation* in literature). The main difficulty in automating CAA is that the 2D picture does not have the 3D information which the animator holds mentally. Drawings are just stylised 2D representations of 3D images and we expect them to behave the way our 3D mental models do. The key problem for automatic in-betweening is how to generate successive drawings of a figure which changes consistently with our 3D intuition of how the drawing should change. Essentially this breaks down into two sub-problems: how silhouettes change (for example a character runs into a wall and is flattened by the impact), and how the various parts of the figure occlude themselves (for example by head, body, or view point rotations).

The changing silhouette problem is one addressed directly by so called in-betweening systems, in which the problem can be thought of in two parts: the *spatial correspondence* problem and *interpolation* problem. The spatial correspondence problem is concerned with matching features on one object with those on another. Corresponding elements may include edges, local convexity, or vertices. Generally speaking this is a hard problem when dealing with two independently constructed objects, even if both objects are intended to be different poses of the same character. So the two poses provided for most in-betweening techniques are usually derived from the same model thus the correspondence problem is avoided rather than solved [PW94].

There are essentially two types of systems, *shape-based* [BW71] [Ree81] [Yu90a] [GG95] [SGWM93] [SG92b] [RF96] or *skeleton-based* [BW76] [SR95]. A shape approach manipulates shape directly, but does not take structure of the objects into account, therefore structural information should be provided by key-frames. In a skeleton approach shape is separated from interpolation. The idea is to have a skeleton of lines which, in a human figure, would be like bones (which can bend and stretch) and are normally jointed at the places one would expect. Interpolation is performed on the skeleton, from which intermediate polygons (shapes) are reconstructed by some models.

Existing in-betweening techniques pay much attention to shape information, interactive control and mathematical simplicity. Of these only a few take structural information (skeleton) into account. Since traditional animation does not pay a lot attention to realistic dynamics and timing but tends to exaggerate the reality both in features and timing, our research [YP97] showed that, to assess in-betweening techniques for the purpose of animation, it is not enough to consider shape information only, we need take into account other factors like structure, dynamics, timing etc as well. A fully objective assessment of the quality of an in-betweening algorithm is quite impossible, because it depends on multiple factors like the way timing is related to subjective judgement, but some objective assessments are locally possible such as the "area" preservation criterion proposed in [YP97].

2.2 Procedural Approaches

In-betweening systems aim at replicating the in-betweening process of hand drawn animation, the dynamics of the in-betweens is constrained by the two key-frames. To some common actions like walking, the assistant animator has the knowledge of how the two legs move and draws the correct in-betweens which are put into a walk cycle. If we take two extremes, which look the same in a walk cycle, as key frames, then we cannot control the correct dynamics of the movements of the two legs at all accurately with the existing in-betweening methods. It is possible to employ the moving point constraints technique [Ree81] to control the dynamics of the interpolating sequences, but a large number of moving reference curves are needed for effective control. This means a lot of manual work is involved so it is hard to apply the method in this case. The dynamics of the two legs, however, is more or less fixed for the walk, which we can employ to generate the walk process procedurally [Yu90b].

The self-occluding problem is the most serious obstacle to any purely 2D, or hand drawn based, automatic in-betweening strategy. To provide a completely general in-betweening capability it relies on a hierarchy of drawing elements, or a Hierarchical Display Model, HDM [PG92]. In many cases it can be dealt with simply by overlaying elements of the drawings and progressively covering or uncovering the elements as part of the in-betweening process. These elements form the leaves of an HDM and the order of placement of the elements (or cutting operations involving them) in effect re-introducing 3D information which is not readily apparent in the composite drawing. Here the HDM is acting as a model for the figure although it is only valid for a limited range of angles through which the figure might be perceived to be turning.

A collection of HDMs for the same character forms the main entries for an electronic model sheet for that character and, in effect, substitutes for a true 3D model for which HDMs could otherwise be deduced. Using the model like this is usually referred to as 2.5D modelling although all an animator needs to do is to select an HDM from a well-enough populated model sheet to start building a sequence.

An alternative approach to deal with this problem such as head turning was proposed in [Yu94a]. Provided with two extreme key frames, one front head image, the other profile head image, the model uses cut planes to cut these two key drawings to get control points, then interpolate those points to get cross sections of the head which are then piled up to construct a 3D head model. By specifying the rotation axis, edge detection and hidden line removal, the model can generate a correct head turning process in a wide range of angles.

Automatic Lipsynch is another paradigm of procedural approaches for CAA [Hun94]. Lipsynch is the synchronisation of the sound-track and the timing of the character. In hand drawn animation, the dialogue is invariably recorded before production and the timing of it is passed to the animator as a phonetic breakdown. The first step is to make the character's actions fit his words. The second step consists of moving the character's lips and perhaps the lower part of the face, to fit the frame by frame phonetic breakdown of the speech on an exposure sheet. In automatic Lipsynch the sound is processed to get the phonemes which are then used to select from a set of mouth keys.

3 Stylised Effects Animation: Motivation and Challenges

Characters are only parts of cartoon animation. They must be animated in an environment. Animated effects like water, fire, rain etc simulate our environment in a stylised manner to add realism, drama and atmosphere to the animation, and thus are important elements for an animator to master. For a large-scale and high quality production, it requires highly skilled professional effects animators to accomplish the task, which means a high cost in a high-wage economy. With the usual studio limitations on time and budget, effects drawings are often made as cycles and this means repeated cycles are required for the required length of time. Because of this, the effects usually look mechanical.

In recent years there have been some models for animating water surfaces, ocean waves and waterfalls [Whi80] [Per85] [Max81] [FR86] [Pea86] [GMM87] [Sim90], fire and smoke [Per85] [TT87] [OI88] [Ina90] [AKN91] [Gar92] [Gar92] [Sak93] [CMTM94] [SF95], as well as a particle system [Ree83] for generating fuzzy objects such as fire and clouds. However, all of them aimed at realistic representations of the process and there is no evidence in the literature that models had been devised for cartoon effects until a procedural approach dealing with cartoon fire, in terms of a hierarchical model, has been proposed by us [YP96a].

In this dissertation, we will investigate the problems of producing animation which captures the intricacy of motion evident in certain cartoon effects. These animations are intrinsically complex presenting a challenge to the computer graphics practitioner:

1. The actions of the effects involve random components thus causing discontinuity of features across more than a few frames, while automatic methods for in-betweening rely on some degree of continuity in both geometry and time, therefore, these automatic methods will fail utterly in such cases.

2. Cartoon effects are 2D stylised drawings by the animator, therefore they are difficult to be presented by particles and there is no physical knowledge available to help us to build the model.

Our goal will be to create stylised effects animation through procedural modelling of hand drawings and their movements. To this end, we expand our prior work [Yu93b] on Hand-drawing Synthesis (HdS) in hand-drawing sequences. In HdS we analyse how a painter draws an object and then simulate the hand-drawing process through synthetic, computational means. Therefore HdS is a procedural modelling paradigm.

Procedural texturing, modelling, and animation is an exciting, active, rapid growing area of research in computer graphics. One of the most popular early uses of procedural techniques was creating textures in the 1970's. The use of procedural techniques has increased since the mid 1980's to now include modelling techniques (fractals, hyper-textures, iterated function systems, L-systems, implicit surfaces, etc.) and even animation techniques. Since we will view the animation of cartoon effects as the process of visualising computer simulations of stylised drawings in movements, our work straddles the boundary between the fields of CAA and computer modelled (procedural) animation.

4 Methodology: Stylised Procedural Animation

4.1 Criteria and goals

In light of the preceding discussion, we seek an approach to animating cartoon effects that is capable of achieving stylised visual effects through a procedural process. The desired properties of such an approach are as follows:

1. The appearance and movement of the animated effects should be consistent with the style of cartoon.
2. The effects should permit and support the necessary degree of high-level animator control. For example, the animator should be able to alter initial conditions of the animation, such as position, timing of effects, etc.
3. The effects should be recreated in 2D and, where possible, in 3D as well.

The research reported in this thesis develops a procedural scheme to creating cartoon-like animations and validates it through a number of effects models. Stylised appearance is achieved through painterly rendering of the effects and background.

4.2 Cartoon Effects

There are several properties common to all cartoon effects. The most salient one is that all cartoon effects are stylised representations of natural phenomena, thus their movements are consistent and predictable. It is possible for us therefore to find out the underlying model to create the effects procedurally. However, cartoon representation of the effects, as some sort of abstraction of the reality, is diverse even in the same effect. For example, flowing water in a cartoon can be created by animating a series of shapes across a plain background colour for water and the shape can be abstract, stylised, or realistic [1].

[1] In contrast with photo-realistic representation, all cartoon representations can be regarded as the stylised which, however, can be divided further into the abstract, stylised, or realistic by the animator.

Whatever shapes used, they must move consistently along the path chosen. Thus movement of different shapes falls into a dynamic control problem and diversity of representation falls into a rendering problem.

In the following sections, we will identify the essential properties and mechanisms that allow cartoon effects to move effectively. From this we derive design methodologies for achieving stylised movements procedurally.

4.3 From Internal Models to Stylised Drawings

The animator, before aiming to represent effects, has some internal model about object actions and then transforms them into 2D drawings. The internal model is the 3D information about structure and dynamics of the object which the animator holds mentally. The transformation is the process of drawing successive frames of the effect on 2D which change consistently with our 3D intuition of how the drawings should change. The drawing process is sequential and so can be divided into what we call *drawing steps* and the animator usually tends to draw the parts of the same characteristics of the effect in each step, as a result the model structures are reflected in the final drawings: the *static structure* (the spatial relationship of different parts of the object) is in the drawing of one frame and the *dynamic structure* (the temporal relationship of different parts of the object) is in the drawings of multiple frames.

4.4 From Stylised Drawings to Computer Models

Our modelling approach consists of two phases: the first is an inverse process to the animator's drawing. We decompose hand drawings into different parts according to our knowledge about the drawing process (the author happens to be a researcher with a painting background) to extract the model structure (both static and dynamic) of the effect. The second is to implement the model through HdS method.

HdS not only ensures the stylised appearance of cartoon effects, it also allows the effects to be animated procedurally. Each effect has its particular dynamic structure which in turn dictates its particular movement. For example, the flowing water model is expressed in a hierarchical structure while the shimmering is expressed with a distributive model. As a result, the flowing water shows a running stream and the shimmering exhibits a random effect.

4.5 Dynamic Control

Upon the static and dynamic structures, dynamic control is achieved through deterministic and stochastic procedure. Deterministic controls such as mathematical models or in-betweening techniques are often used at the high-level to control the main motion of the effects. While stochastic controls are often used at the low-level to control local movements or variations in shape. More details about in-betweening and motion control can be found in Chapter II. Corresponding stochastic controls are described in the relevant individual effects models in Chapter IV, V and VI.

4.6 Stylised Rendering

The appearance of the effects depends not only on the model, but also heavily on rendering. The focus of most rendering research in the last two decades has been on the creation of photo-realism imagery. Recently a painterly rendering for animation was proposed in [Mei96], but it tended to create painterly images with the characteristics of an oil painting. Since our concern is cartoon animation, we need to develop the painterly line and surface rendering to keep the cartoon look in generating the background. More detail about our rendering technique will be found in Appendix. As for individual effects, we render them individually according to how the animator draws and paints to achieve the desired effects.

4.7 From 2D to 3D

Since our models are derived from 2D drawings, they are naturally able to recreate 2D cartoon effects. Considering the animator's 2D drawings are stylised transformations of 3D images, if we employ the 3D information that the animator holds mentally, it is possible for us to create cartoon effects in 3D as well. Since the static and dynamic structures are invariant in both 2D and 3D, therefore, we can use the structural information revealed in 2D drawings to reconstruct those structures in 3D, and then apply our cartoon rendering to generate 3D cartoon effects.

4.8 Fidelity

Our methodology aims at producing cartoon effects that not only look like, but also move like, their hand-drawn counterparts. An important question is: How closely should our models attempt to emulate hand-drawn effects? Clearly, a certain level of modelling fidelity is required in order to generate convincing results. For the purpose of animation, we can use some criteria such as structure, dynamics, timing etc, as proposed in [YP97], to achieve the intended purpose – in our case, cartoon appearance, timing and movements. As timing is sometimes related to a subjective judgement, in that case we need to play the animation back to see if everything goes right. If some part goes wrong, we modify the model by leveraging the synergy between parameters at the different levels of the model to ensure the correctness of the animation.

5 Contributions and Results

We have successfully applied the basic methodology outlined in the previous section to develop an animation framework of creating cartoon effects procedurally. When the animation program is initiated, the user specifies only initial conditions for the effects to be generated. Initial conditions include position, size, timing, colour, specification of control points etc which may vary from effect to effect and from application to application.

The visual results of this work are illustrated by a video "Stylised Effects Animation" [YP98] containing animations of twelve sessions. Effects are animated in the 2D or 3D painterly rendered scene. Colour plates of each animation session will be shown in the corresponding chapters of this thesis.

This thesis contributes to computer graphics, in particular to the fields of CAA, procedural animation as well as stylised rendering. Our contributions have been published in the computer graphics literature [YP97] [YP96a] [YP96b] [Yu94a] [Yu94b] [Yu93c] [Yu90b] [JJY96].

Our work is the first to combine within a unified framework extensive hierarchical models, procedural control and stylised rendering. The main contributions of this thesis in more detail are as follows:

1. We propose an approach to model objects based on a hand-drawing process. Our work shows that this approach is a powerful paradigm for

hand-painted image synthesis, because we capture the essential qualities of the object or motion constrained by hand-drawings along, say, how different parts of the drawing are distributed in time and space.

2. We use procedural models in dealing with cartoon animation which allow parametric controls at high-level over colour, timing etc in a sequence. We can also include as much artistic expression into the procedure as we choose.

3. We extend 2D cartoon effects to 3D. This novel result enables us to animate cartoon effects with the arbitrary specification on the virtual camera movement, the situation being hard to image in the traditional hand-drawn animation.

4. We propose assessment criteria of shape deformations in animation.

5. We develop painterly line and surface rendering for our stylised animation.

In addition, this thesis irons out the difference between CAA and computer modelled animation.

6 Thesis Overview

The thesis is organised as follows:

In Chapter II we review previous work upon which our research draws. At its lowest level of abstraction, our work is an instance of CAA. Therefore, we first survey shape and skeleton deformation techniques used in CAA as well as assessment criteria of shape transformations in animation. At a higher level of abstraction, our research is an instance of advanced procedural animation. We survey prior procedural animation work in CAA such as walk, head turning and realistic effects animations.

In subsequent chapters, we describe in detail the stylised procedural animation system that we have developed. In Chapter III we begin by presenting an overview of the cartoon effects system.

Chapter IV proceeds to describe models of water effects including flowing water, water jet, water ripples, shimmering and water reflections.

In Chapter V we present models depicting gaseous effects like fire, smoke skein and smoke puffs.

Chapter VI describes models of rain and snow. We first present the light rain model, then proceed to the heavy rain model composed of two sub-models of rain drops moving in the air and rain drops hitting the ground, and finally to the snow model.

In Chapter VII we review the contributions of the thesis, and list possible directions of future work.

Chapter II

Backgrounds of CAA and Procedural Animation

In this chapter, we review prior work in the fields of CAA and procedural animation upon which our research draws. Firstly, we progressively survey related research on shape transformation and motion control techniques such as shape-based and skeleton-based computerised inbetweening systems. In the shape-based approaches we start from the direct category such as simple linear interpolation, inbetweening using moving points constraints, polar co-ordinates based linear interpolation techniques, a physical based approach to 2D shape blending, to the indirect category such as polygon morphing using a multiresolution representation, match and interpolation of shapes using unions of circles. In the skeleton-based approaches we proceed to review the interactive skeleton techniques, star-skeleton representation, as well as localised rotation and scaling using Quaternions. Secondly, we survey some of our early procedural approaches in CAA such as human walk and head turning. Thirdly, we present assessment criteria for 2D shape transformations. Finally we review briefly prior research on procedural realistic effects animation such as particle system, water and gaseous effects animations.

1 Shape Transformation and Motion Control

In order to animate something the animator has to be able to specify, either directly or indirectly, how the "thing" is to move through time and space. In a computerised approach for computer animation, the basic problem is to select or design animation tools which are expressive enough for the animator

to specify what he wants to specify while at the same time are powerful or automatic enough that the animator doesn't have to specify the details that he is not interested in. Obviously, there is no one tool that is going to be right for every animator, for every animation, or even for every scene in a single animation. The appropriateness of a particular animation tool depends on the effect desired by the animator. An artistic piece of animation will probably require different tools that an animation intended to simulate reality.

Many shape transformation techniques for computer animation have been proposed in the past two decades [BW71] [Ree81] [Yu90a] [GG95] [SGWM93] [SG92b] [RF96] [BW76] [SR95], which tend to replicate the in-betweening process of hand drawn animation and work in the following way: when given two key shapes, they transform the shape continuously from one to the other. Those techniques also form so called key-frame systems which take their name from traditional animation.

There are essentially two types of in-betweening systems: *shape-based* and *skeleton-based.* With the shape-based approach there are two categories: *direct* and *indirect.* In the first category (direct) the shape is transformed directly, while in the second category (indirect) the shape is transformed into some representation first, then interpolation is performed on certain parameters in the representation and finally the shape is reconstructed.

Shape-based techniques associated with their types and categories are listed in the following table and their more detailed description will be given in the subsequent sections.

Technique	Type	Category
Simple linear interpolation	shape-based	direct
Moving points constraints	shape-based	direct
Polar coordinates linear interpolation	shape-based	direct
Physical based approach	shape-based	direct
Multiresolution representation	shape-based	indirect
Unions of circles	shape-based	indirect

Figure II.1: Distortion of CCLI due to rotation

2 Shape Approaches

2.1 Cartesian Coordinate Linear Interpolation

The simplest shape interpolation is linear interpolation proposed by Burtnyk and Wein [BW71], which is carried out on vertices in Cartesian coordinates and place all the intermediate positions of the reference points at equal intervals along a straight line joining the initial and final positions of the point. We refer to this as Cartesian Coordinate Linear Interpolation, or CCLI. This method executes quickly but can produce unpleasant results when dealing with rotations, see Figure II.1. Another drawback of CCLI is that it can sometimes generate motion discontinuities which we call *clicks*. Clicks often appear as the motion passes through intermediate keyframes because the eye receives a change in trajectory and dynamics.

2.2 Moving Point Constraints

Reeves presented a method of in-betweening using moving point constraints [Ree81]. In keyframing, to animate an object the animator specifies an ordered set of keyframes $KF_1, KF_2, ..., KF_K$ which define the form of the object at the animator specified times $t_1, t_2, ..., t_K$. Keyframes are usually sketched by the animator on a digitising tablet. Each keyframe can be thought of as a *static shape* positioned at a fixed point in time which acts as a *constraint* on the motion sequence. It is possible for the animator to specify an additional set of constraints to control the in-betweening process. These constraints, $[MP_1, MP_2, ..., MP_q]$, are called *moving points*. A moving point is a curve in space and time which constrains both the *trajectory* and *dynamics* (i.e. path and speed) of a point on the animated object. Moving points are normally

sketched by the animator and can have any shape and dynamics desired. At any time in the motion sequence, a moving point is located at a particular $X - Y$ position. Points on the keyframes not directly constrained by a moving point are constrained by a smooth blending of their neighbouring moving points (Reeves gave two algorithms to perform this blending in [Ree81]). The animator can specify as many moving points as necessary to control the dynamics. The set of keyframes and the set of moving points specify a *patch network* of the motion sequence.

In-betweening algorithms are designed to operate on complete patch networks as shown in Figure II.2.

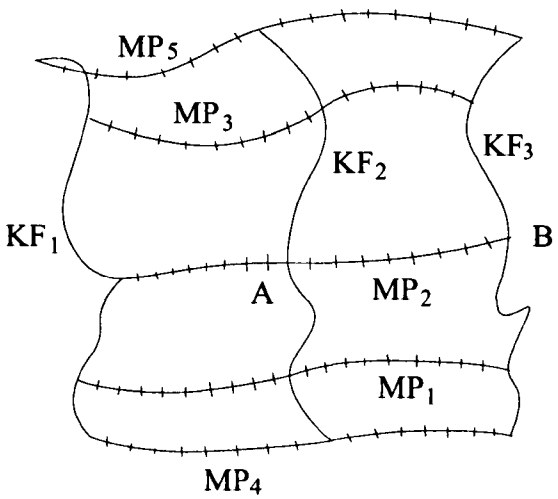


Figure II.2: Completed patch net work

A completed patch network is subdivided into *patches*. Each patch is defined by four boundary curves - two are static boundaries derived from the two bounding keyframes, and two are dynamic boundaries derived from the two bounding moving points. Then the inbetweening problem on a patch network can thus be reduced to many inbetweening problems on its patches.

There are different methods for implementing the idea represented here. Reeves introduced two algorithms:*Miura in-betweening* [MIT67] and *Coons' Patch in-betweening*. Details about them can be found in [Ree81].

The disadvantages of a computer implementation of Reeves' method are that a large number of curves are needed to control a complex image effec-

tively, because this is the only way in which 3D information can be encoded into the problem specification.

2.3 Polar Coordinate Linear Interpolation

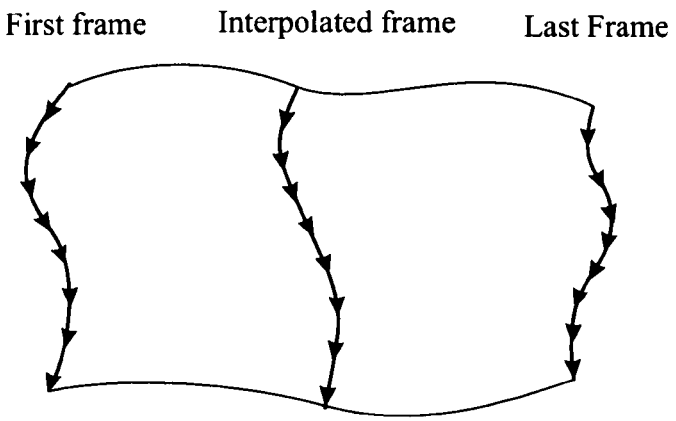


Figure II.3: Illustration of PCLI

In 1990 we suggested the linear interpolation of local polar coordinates [Yu90a] [Yu93c], or the Polar Coordinate Linear Interpolation, PCLI. Here we model a free-form curve as a polyline, that is a series of straight-line segments between points chosen to approximate the original curve. Each segment of the polyline is used to define a sliding vector which we model by treating one end of the segment as being at the origin and give the coordinates of the other end only as specifying the vector. Here we use polar coordinates r for the length of the segment and θ for the angle between the segment and a fixed ("horizontal") line, as the coordinate specifying the vector in 2D space. We use the polar coordinates r, θ in a linear interpolation to get the corresponding vector in an intermediate curve. The technique falls into four steps:

1. Break down a curve into N-1 vectors (N is the number of the points contained in a curve).
2. Calculate the polar coordinates r and θ for each sliding vector.

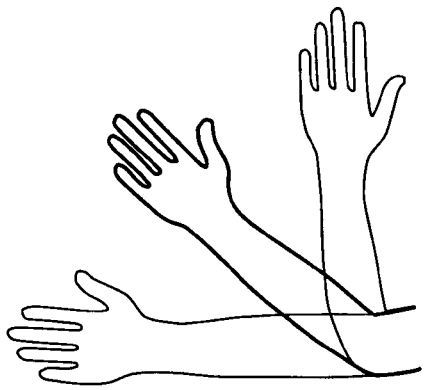


Figure II.4: Hand moving about elbow

3. Interpolate r and θ linearly, which can be expressed as:

$$\begin{aligned} r_{t,i} &= t.r_{1,i} - (t-1).r_{0,i} \\ \theta_{t,i} &= t.\theta_{1,i} - (t-1).\theta_{0,i} \qquad (i = 1, ...N-1) \end{aligned} \tag{II.1}$$

   where $r_{0,i}, \theta_{0,i}$ and $r_{1,i}, \theta_{1,i}$ are the polar coordinates of the ith vector in the first and last frame respectively, $t \in [0,1]$ is a time in between the times [0,1] for the two extreme vectors, and $r_{t,i}$ and $\theta_{t,i}$ are the polar coordinates of the resultant vector.

4. Connect these interpolated vectors one after another to form the interpolated curve. Figure II.3 shows this technique.

The second formula in step 3 can be rewritten as

$$\theta = t.\theta_{1,i} - (t-1).\theta_{0,i} = \theta_{0,i} + t.(\theta_{1,i} - \theta_{0,i}) = \theta_{0,i} + t.\delta_i \qquad (i = 1, ...N-1)$$

where $\delta_i = \theta_{1,i} - \theta_{0,i}$ is the angle between their corresponding ith vectors in the first and last frame. In those cases where the value of δ_i is greater than π, in which case it becomes an exterior angle, experience shows we usually get correct results by taking the interior angle $2\pi - \delta_i$ to be interpolated. Accordingly, we confine δ_i to be less than π in the algorithm as this can meet the requirements of interpolation in most cases. With this restriction if the case of $\delta_i > \pi$ arises, an extra keyframe is needed. It should be pointed out that in our experience this case arises rarely in practice.

Compared with CCLI, PCLI is a more suitable shape transformation technique because it naturally tends to respect shapes and produce circular motions, as does the hand in Figure II.4.

2.4 A Variant of PCLI

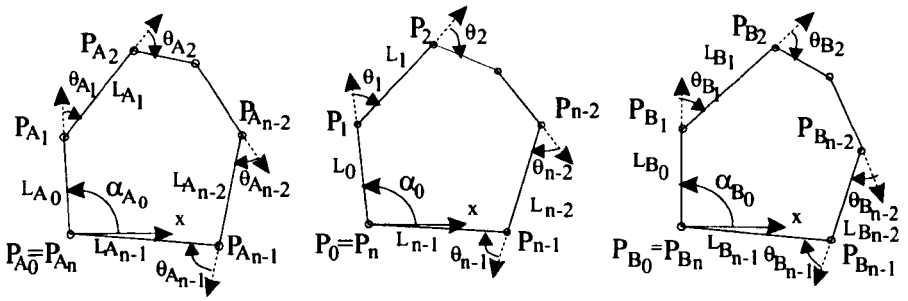


Figure II.5: Definition of the variables

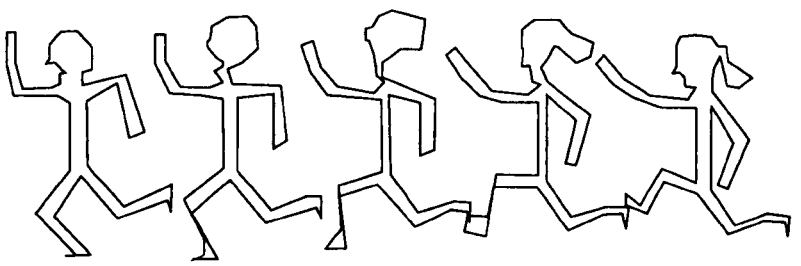


Figure II.6: Distortion of the polar techniques

In 1993 Sederberg *et al* proposed a technique similar to PCLI using angle and length interpolations [SGWM93]. The difference between PCLI and Sederberg's polar technique lies in the definition of angular parameter. They chose the respective vertices angle shown as Figure II.5 to interpolate, i.e.

$$\begin{aligned} \alpha_0 &= (1-t)\alpha_{A_0} + t\alpha_{B_0}, && (i=0) \\ \theta_i &= (1-t)\theta_{A_i} + t\theta_{B_i}, && (i=1,2,...,n) \end{aligned} \tag{II.2}$$

An optimisation algorithm is performed, for the purpose of ensuring that the intermediate polygons define closed shapes. Both PCLI and Sederberg's technique handed many cases successfully, including cases where shapes are affine transformations of each other or where parts of the shapes are transformed affinely. However, in some cases they produce self-intersections of the boundary and tend to distort the polygon area in intermediate shapes. Figure II.6 gives an example.

2.5 A Physically Based Approach

Sederberg and Greenwood proposed an algorithm for smoothly blending between two 2D polygonal shapes [SG92b]. Given two polygons P^0 and P^1 with the same number of vertices, i.e.

$$\begin{aligned} P^0 &= [P_0^0, P_1^0, ..., P_n^0] \\ P^1 &= [P_0^1, P_1^1, ..., P_n^1] \end{aligned} \tag{II.3}$$

To model polygon P^0 as a piece of wire made of some idealised metal and the shape blend is the one which requires the least work to deform P^0 into P^1 through bending and stretching. Stretching work is computed for each line segment (i.e. each adjacent pair of vertices) whereas bending work is computed for each adjacent pair of line segments (i.e. for each set of three adjacent vertices). Detailed formulation to calculate the stretching work and bending work can be found in [SG92b].

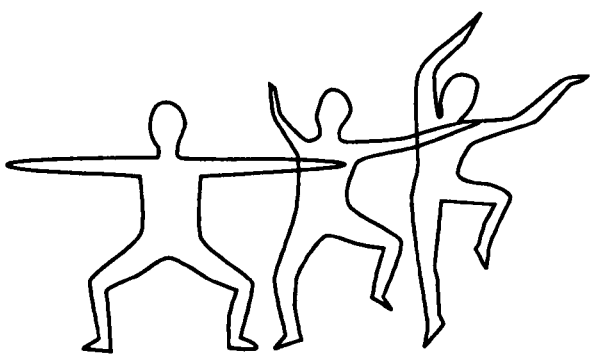


Figure II.7: Dancer

The algorithm can avoid self-intersecting which may arise from CCLI, but it sometimes may also produce local self intersect and distortion when dealing with rotation movements, as shown in Figure II.7.

2.6 Multi-resolution

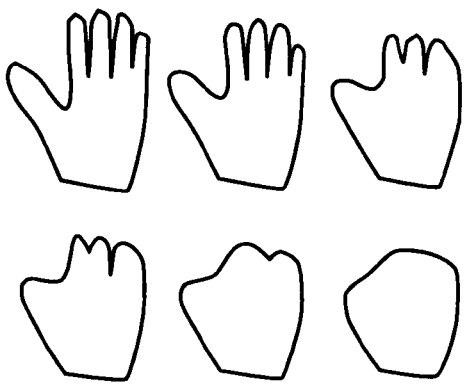


Figure II.8: Geometric evolution of a closed simple curve

Eli Goldstein and Craig Gostman presented a polygon morphing approach using a multi-resolution representation [GG95]. The multi-resolution they used is based on polygon evolution schemes. This representation contains information on the polygon at many levels of detail. Morphing is then performed in the more natural space of the representation, from which the intermediate polygons are reconstructed. A curve evolution method that takes into account directly the intrinsic curve geometry was given in [EG94] [KTZ94]. The evolution is described by the motion of the points on the curve in time. Loosely speaking, during evolution, a point on the curve advances in the direction of the vector normal at that point, by a distance proportional to a function of curvature at that point (see Figure II.8).

An example of multi-resolution method is given in section 5.1.1 of this chapter from which we can see the distortion caused on the leg shape of a running figure.

2.7 Circle Union

Recently, Ranjan and Fournier presented their work which uses a representation of objects as a union of circles, or UoC, to base a method to interpolate between the two [RF96]. This method can be used in a totally automatic fashion (i.e. without any user intervention), or users can guide a pre-registration phase as well as a segmentation phase, after which the matched segments are interpolated pair-wise. The UoC representation of the two objects is obtained from the Delaunay triangulation of their boundary points. The circles can be simplified to obtain smaller data sets. The circles are then optimally matched according to a distance metric between circles which is a function of their position, size, and feature, i.e. a local configuration of circles. The interpolation between the two objects is then obtained by interpolating between the matched pairs of circles (the interpolation can be affine or non-affine).

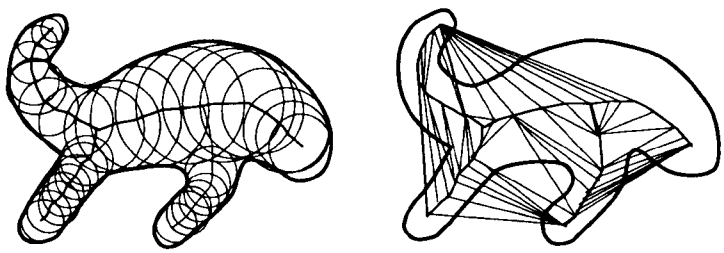


Figure II.9: Union of circles representation and segementation

The approach can be summarised by the following steps:

1. Compute the union of circles representation of the two objects.
2. Simplify the representation to stabilise it and remove redundancies.
3. Align the objects globally if needed (optional).
4. Decompose the objects into parts, if necessary (i.e. segment objects) (optional).
5. Match parts between objects if they have been segmented (optional).
6. Match circles in matched parts.

7. Use the matches to compute transformations between parts or to interpolate between them.

8. Finally, integrate the information about matches from all the parts.

Figure II.9 shows UoC representation and segmentation of a 2D fish shape.

For interpolation, the goal is to determine a path between the matched circles. Again in general the match implies a translation, a scaling and a rotation. The easiest solution is to interpolate all these linearly, i.e. the centres of the circles move in a straight line, the size varies linearly as a function of time, and so do the angles of rotation. However many cases have been observed where such a linear interpolation is not appropriate. Two other possible strategies can be used and detailed description of them can be found in [RF96].

3 Skeleton Approaches

The greatest shortcomings in shape approaches result from incomplete control of motion dynamics, both in complexity associated with the structure of the object like human body and in smoothness or continuity.

There is a dilemma in that smoothness is achieved by having as few key images as possible (and therefore widely spaced in time), while close control requires many closely spaced keys. In addition, a large number of closely spaced drawings negates much of the economic advantage of using computers.

An examination of the methods used in conventional animation has led to a solution to this problem. To visualise a complex movement, the animator often sketches stick figure representations at equal-time intervals between key positions. He may use smooth curves through related skeletal points as a further guide. This set of stick figures achieves both objectives: the frame to frame spacing conveys the rate of movement and the shape of each skeleton represents the shape of the object at that instant. Thus the problem reduces to animating a stick figure representation of the image which will in turn impart the movement to the actual images sequence. For the computer implemented skeleton approaches, the appearance of the object is determined not only by the skeleton but also by a "flesh" or "skin" made up of segments

whose geometric handles are geometrically related to the skeleton. The existing skeleton techniques usually consist of three parts:

1. Compose the skeleton.
2. Interpolate the skeleton.
3. Skinning (how to put "flesh" or "skin" to the skeleton).

A significant aspect of the skeleton technique is that the skeletons are simple images composed of only a few points, so that it is possible to provide a high level of interaction.

In the 2D case only a few skeleton techniques have been proposed which are listed in the following table and we will give their more detailed description in the following sections.

Technique	Type	Dimension
Interactive skeleton	skeleton-based	2D
Star-skeleton	skeleton-based	2D
Using Quaternions	skeleton-based	2D/3D

3.1 Burtnyk and Wein's Skeleton Approach

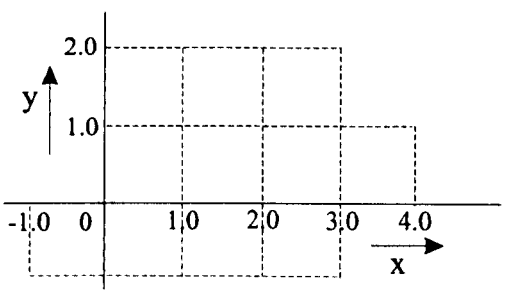


Figure II.10: The mesh

Burtnyk and Wein described a system that incorporates the use of skeletons into CCLI [BW76]. In their skeleton representation of an image, they

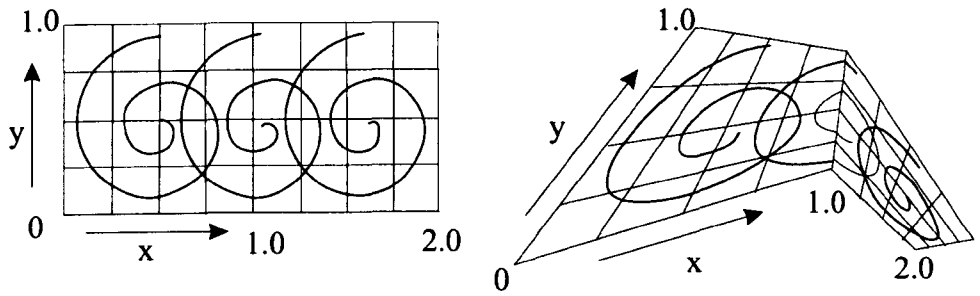


Figure II.11: The overall distortion exibited by the coordinate space

define some coordinate space within which the image, described in relative coordinates, is distributed. The nature of the coordinate space that is used to define relative skeleton coordinates may be thought of as a network of polygons that form a mesh as shown in Figure II.10.

Each polygon has a relative coordinate range of 0 to 1.0 along each axis. Now the nodes in this mesh may be displayed relative to one another to change its geometry. However, because the relative coordinates system within each polygon is based on its geometry, coordinate values remain continuous across common edges between adjacent polygons. Thus any image whose coordinates are defined within this system will take on the overall distortion exhibited by the coordinate space. (Figure II.11).

The notion of skeleton control implies a central core of connected "bones" with a surrounding image distribution. In order to restrict the transverse distance away from the core over which skeleton control will be active, delimiting boundaries must be specified. Consequently, the practical form of skeleton coordinate space spans two units in width, but may extend in length as desired, see Figure II.12.

For convenience, the central core always represents the L axis, which is also the $W = 0$ coordinate reference; the delimiting boundary which is specified first is the positive or $W = 1.0$ boundary, the other is the $W = -1.0$ boundary. The L coordinate range starts at $L = 0$ and is incremented by one for each node on the central core. If desired, the L coordinate space may be separated at any coordinate boundary by providing a redefinition of that coordinate boundary before continuing the coordinate space.

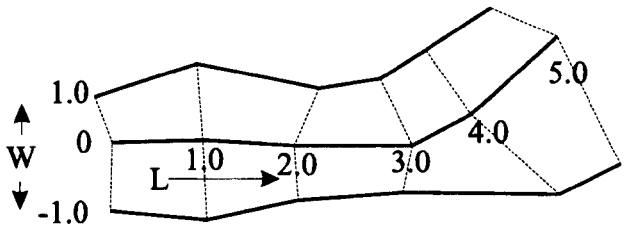


Figure II.12: The "bones" and the surrounding image distribution

The effect of skeleton control is to take any specified area of the display plane and distort it into another area of the display plane as if it were made up of rubber sheet patches. The simple linear interpolation is still used to the skeletons. A problem associated with this model is that it cannot simulate the wrinkling of skin or clothes, caused, for example, when an arm is bent at the elbow.

3.2 Star-skeleton Approach

In 1995, Shapira and Rappoport proposed a star-skeleton approach for the purpose of polygon shape blending [SR95]. A star polygon is a polygon for which there exists a point – the star point – from which all other points are visible. The star skeleton consists of two parts:

1. A decomposition of the polygon into star-shaped pieces, each represented by its vertices and a special star point called the star origin shown as in Figure II.13(a).

2. A planar graph, the skeleton, that joins the star origins shown as in Figure II.13(b).

In Burtnyk and Wein's skeleton approach the skeleton is defined first, while the star-skeleton is calculated from the given polygon automatically, and for this purpose some notations and basic definitions are introduced. A polygon is defined by a sequence of points (vertices) such that the first vertex is identical to the last one. A polygon edge is the line segment connecting two consecutive vertices. A simple polygon is a polygon whose edges do not

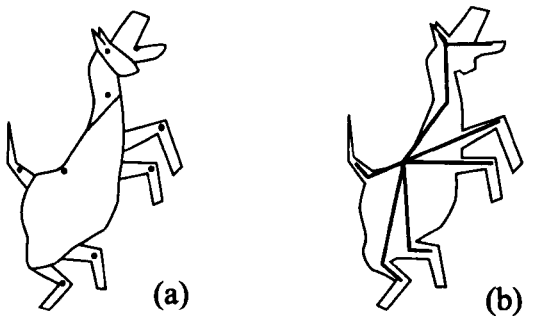


Figure II.13: Star skeleton representation

intersect each other except when they topologically share a vertex. Let P denotes a polygon, including its interior and boundary.

A simple polygon is convex if every pair of points in it sees the others. For every simple polygon a graph called the visibility graph is defined. Its nodes are the polygon vertices, and an arc exists between two nodes if and only if the corresponding vertices see each other. A complete correspondence between two polygons is a one-to-one mapping between their vertices that preserves vertex adjacencies and orientation.

The star skeleton of a simple polygon P comprises two components: the *star set* and the *skeleton*. The star set is a set of star-shaped polygons called star pieces. Each star piece possesses a special star point called the star origin and a direction called the reference direction. If there is more than one star piece in the star set, each star piece possesses at least one star piece neighbour, in the sense that they share an edge. Such an edge is called shared edge, and its vertices are called shared vertices.

The vertices of each star piece are represented in polar coordinates with respect to the star origin of the piece and its reference direction. That is, each vertex is defined by the distance from the star origin and by the angle formed between the vector from the star origin in the reference direction and the vector from the star origin to the vertex.

The skeleton is a planar tree composed of two alternating kinds of points: the star origins and the midpoints of shared edges, referred to as *midpoints*. Each skeleton vertex has an associated direction called the reference direction. The root's reference direction is the x axis. The reference direction of

any other vertex is the vector from the vertex to its parent. The reference direction of a star piece is the reference direction of its star origin.

The root is represented in Cartesian coordinates. Every other vertex is represented in polar coordinates with respect to the parent and its reference direction. That is, a vertex stores the distance to the parent and the angle between the reference directions of the parent and itself. Figure II.14 shows two star skeletons. For the polygon on the left, the star set consists of three star pieces. For example, the vertices 2 through 7 define the bottom star piece. The three star origins are shown as black squares. The star origin of the middle star piece coincides with vertex 8, which is also the root of the skeleton. There are two shared edges, [0, 11] and [2, 7], and two midpoints, drawn as white squares.

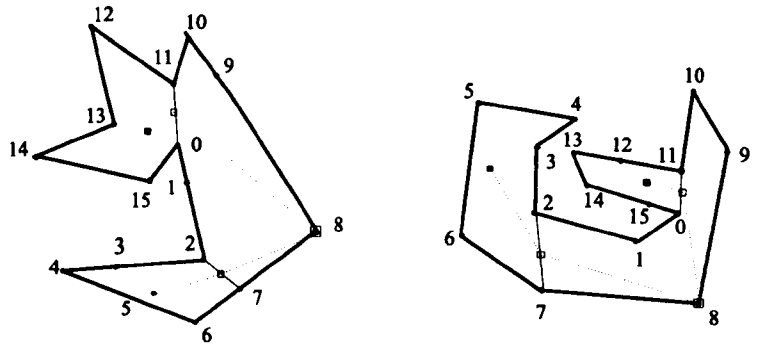


Figure II.14: Compatible star skeleton

The star skeleton is defined by a collection of different point types: polygon vertices, star origins, and midpoints. An intermediate star skeleton is generated by interpolating the coordinates of these points at the desired time instance. Each point, except the skeleton root, is represented in polar coordinates. Polar coordinates interpolation address both distances and angles, and the simple linear interpolation is performed. The root vertex is interpolated in Cartesian coordinates.

Skinning is the process of computing the Cartesian coordinates of each vertex on the boundary of the polygon so that it can be displayed directly. It composes two stages: computing the Cartesian coordinates of each skeleton vertex and computing the Cartesian coordinates of each vertex of the resulting polygon. The skeleton vertices adjacent to the root are computed first,

then their neighbours, in a recursive manner. Once all star origins on the skeleton have been computed, the vertices of star piece can be computed.

The star-skeleton method yields good results because it explicitly considers the polygon interiors and explicitly models an interdependence between the polygon vertices. To produce the simplest star skeleton possible, a *minimal* star decomposition is chosen. However, the decomposition algorithms [SR94] are computationally expensive, taking a few seconds for the simple example in Figure II.15, and it might still create local self intersections.

Figure II.15: Star-skeleton blend

3.3 Localised Rotation and Scaling using Quaternions

In 1996, we proposed a technique for object deformation using Quaternions [YP96b]. The technique may be regarded as a 3D extension of PCLI, but scaling and rotation operations are implemented using Quaternions. The enhanced control introduced over the scaling and rotation applied to the skeleton is a significant improvement of the technique over PCLI. This is based on the fact that two formulae in equation II.1 are independent of each other, which leaves room for introducing non-linear control over the rotation and scaling. Together with former linear control, they are listed as follows:

1. Linear rotation and linear scaling. These operations are used in the polar techniques such as PCLI and Sederberg's one, as well as star-skeleton technique.

2. Non-linear rotation and linear scaling. Such cases often arise in the movements of an articulated structure, such as arms and legs on the human body, where a series of rigid links connected at joints. The length of each link does not stretch or shorten during the movements, but their orientation may vary in a flexible manner.

3. Linear rotation and non-linear scaling. This case may happen when some objects are acted by multiple forces. For instance, a weight suspended from a fixed position by means of a spring when it is pulled aside and set in a circular motion. The tension in the spring will cause the weight to bounce up and down as it follows the circular path. As the energy becomes exhausted the weight's path spirally reduced in diameter until eventually to rest in the centre.

4. Non-linear rotation and non-linear scaling. An example that is similar to the previous one in which the weight does not move along a circular path but a plane, again with bouncing movement added.

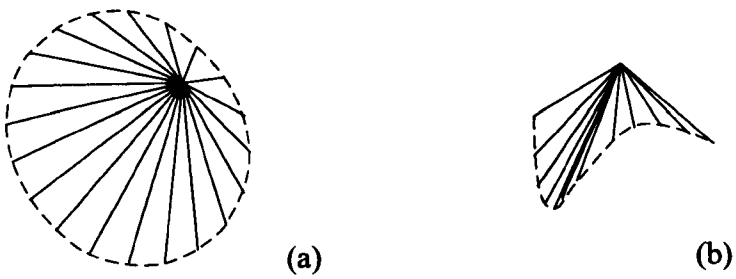


Figure II.16: (a) Orientation controlled by a circular function (b) by a spline

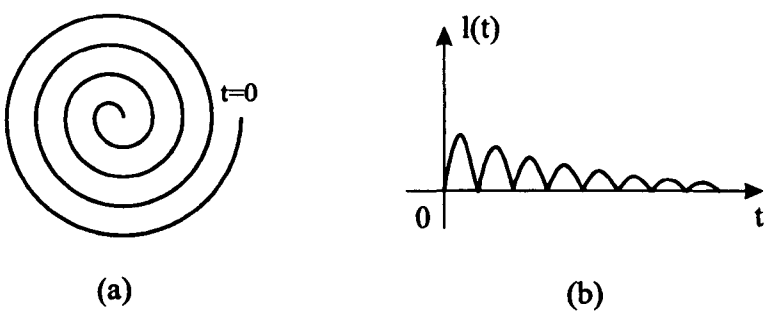


Figure II.17: (a) Spiral curve (b) Non-linear scaling function

Figure II.16 shows examples of enhanced control over the rotation with functions and splines. Suppose the segment represents the arm skeleton of an

articulated character moving in front of the body, which corresponds to case 2 – non-linear rotation and linear scaling. In Figure II.16(a) the orientation is controlled by a circular function and in Figure II.16(b) it is controlled by a spline which can handle free rotation movements.

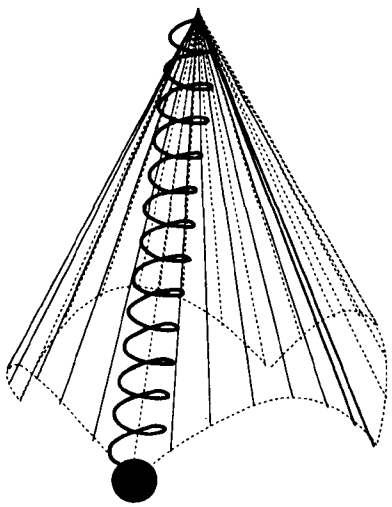


Figure II.18: Spring skeleton controlled by non-linear rotation and scaling

Figure II.17 shows case 3 – non-linear control over the rotation and scaling. Here the skeleton of the spring is just a line running down the centre of the spring's local symmetry, the "flesh" of the skeleton is the spring itself which is related to the skeleton by a simple rule (points on the spring are a fixed distance from the skeleton). We use a spiral curve shown in Figure II.17(a) to control the orientation of the skeleton, and a non-linear function $l(t) = l_0[1+a(t)|sin(t.n.\pi)|]$ shown in Figure II.17(b) to control scaling, where l_0 is the length of the skeleton corresponding to the rest shape of the spring, $a(t)$ controls the magnitude of the weight bouncing up and down which decreases exponentially as the weight goes toward its rest position, and n controls how fast the bouncing is. Figure II.18 shows the deformed skeletons of the spring during the movement and we draw a spring related to one skeleton in the figure. We should point out that our control over the rotation and scaling here is only an approximation of the movement for the purpose of animation rather than accurate physical simulation, the path of the weight traced out is determined by the combined effect of rotation and scaling which is difficult to specify directly by other means.

Enhanced control can be straightforwardly applied in the 2D case and a walk mode described in the following section can be regarded as a 2D implementation of case 2 – non-linear rotation and linear scaling applied to two legs.

4 Procedural Approaches in CAA

4.1 A Walk Model

Character animation is about the promotion of the illusion of movement for human-like figures. In animation one of the commonest things a human figure will do is to walk about and this movement is often seen from the side. In the studio, the animator breaks down the action into sections. The action is in fact a walk cycle which is repeated more or less exactly to give the appearance of continuous walking. Usually the drawings which make up the cycle are drawn separately and used repeatedly in the same walking sequence. The end positions of the cycle, that is the first and last drawings, consist of the leg positions around the point at which one foot (the "free" foot) is about to be put on the ground at the end of a stride. The first drawing in the cycle consists of this drawing while the last drawing may most easily be described as the drawing for the frame immediately *before* the figure's position in the first drawing.

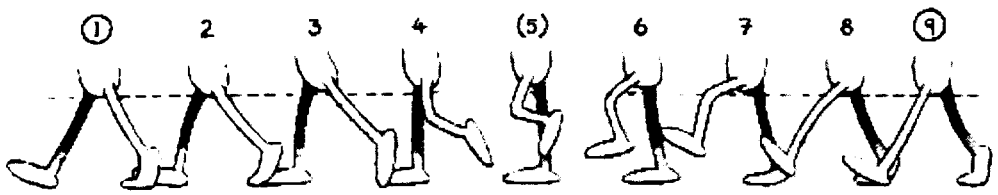


Figure II.19: Hand drawn walk

The in-betweening problem is tacked in hand animation by first drawing the mid way position, called the passing position, which here is drawing 5 in Figure II.19 [Whi86]. Once the passing position has been drawn the

remaining in-betweens are added, starting with the drawings which halve the remaining time interval. In this case this means drawing 3 and 7, then 2, 4, 6, 8 in any order. We note that the height of the two in-between body positions fall natural in-between. In drawing 3, the toe is still in contact with the ground, otherwise, the body weight would not be in balance with the legs and the figure would seem to fall backward. In drawing 7 the heel of the foot is also touching (or in near contact with) the ground – again, to aid with balance.

In order to cope with leg dynamics during walking, we devised a walk model [Yu90b] [Yu92] to generate correct in-betweens procedurally for this case. The model is based on angle constraints and uses two parameters to control the speed and type of the walk.

4.1.1 Angle Constraints

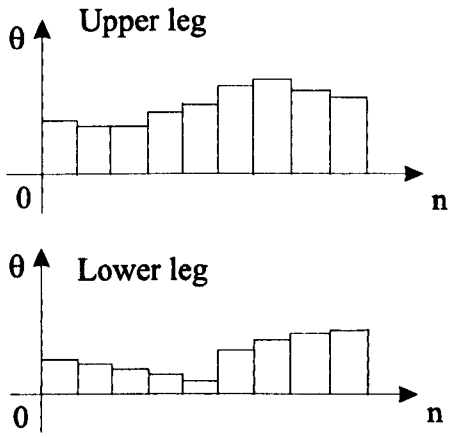


Figure II.20: Angle variations of upper and lower legs

The method used for reverse-engineering hand-drawn walks is as follows. We put the hand-drawn in-betweens of the walk into computer, and model the leg silhouette as a skeleton, working away from the joint in assigning key points or knots. Typically the upper and lower parts of the leg silhouette can be described along straight lines, with a concentration of short line-segments about the joint. The length and the angle of the upper and lower leg are calculated for each frame respectively. Since each section of the leg does not

stretch or shorten during the walk, their length remain the same in every in-between as in Figure II.19. We identify an angle θ between the upper and lower parts of the front leg silhouette skeleton and determine how this angle varied with the frame number n. The graph of the $\theta - n$ relationship is shown for successive frames in Figure II.20. We see that each angle changes significantly from one frame to the next in a non-linear way. By working with several examples of hand-drawn walks we get similar results. This means that this $\theta - n$ relation could be used as the angle constraints for walks.

4.1.2 The Speed of Walks

Above mentioned angle constraints are discrete ones of 9 frames and are only valid for a normal walking speed. In order to control walking speed, we convert the discrete angle constraints into continuous ones by interpolating them with splines, and then sample it with different intervals to generate inbetweens with different numbers of frames for each step, thus giving control over the different walking speeds.

4.1.3 The Types of Walks

There are variations on walks as described in [Har81]. Usually the orientation of the body, head and arms are established by the two extremes, so the problem here is reduced to how to present the leg movement data for different types of walks. In our model, the type of walks is derived by modifying the amplitude of the angle constraints. Here we give an example to show the basic idea.

Example: We multiply the angle constraint for the upper leg by a factor $fa = 1 + a.sin(n.\pi/N)$, where $a \in [0, 0.3]$ is a parameter derived from the experiment to control the variety of the walks, n denotes the ith frame and N is the number of frames contained in one step.

Details about the in-betweeners used in the model for the controls of other parts of the body are given in [Yu90b]. Figure II.21 gives a walk cycle generated by our model.

Figure II.21: A walk using angle constraints

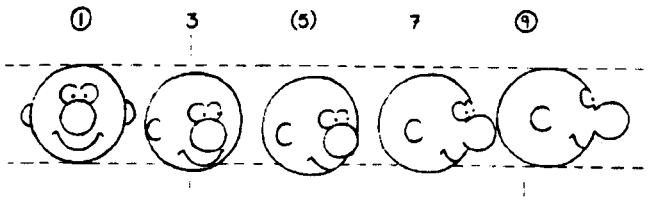


Figure II.22: Hand drawn head turn

4.2 A Head Turning Model

Head turning is another one of the commonest things a human figure will do. In the studio animating head turning, the animator draws two key frames, one seen in front, another in profile, see Figure II.22 [Whi86] 1 and 9, then a junior animator adds in-between drawings 2 to 8. Since the head moves in an arc (everything in life does so), so it dips a little down during the turning process, as 5, then 3 and 7 are added, so 2, 6, 8, to get a series drawing of the head turning process.

To deal with head turning with a computer in-betweening method, the computer is given 1 and 9 of Figure II.22, then it calculates additional drawings 2 to 8 with interpolation algorithm. Unfortunately interpolation algorithms fail to generate correct in-betweens for complex shapes. This is because, in essence, all drawings in Figure II.22 are 2D representations of the 3D head, and the information provided with two extreme drawings on

2D is not enough for the task of automatic in-betweening. In the studio, the junior animator relies on the knowledge of the way in which the head depicted by the drawing is supposed to turn to fill in the rest of each in-between drawings. The computer, however, has no such "knowledge" and thus cannot specify the head movement in a way which preserves 3D properties of the head. Another problem is even worse, that is presented here, because some lines in drawing 1 have no correspondence to lines in drawing 9, which happened in the right part of 1 and we cannot find out their corresponding lines in 9, so the computer cannot determine the corresponding lines in some of the as-yet undrawn in-betweens.

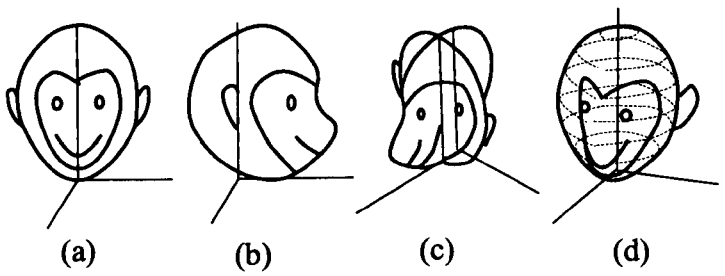


Figure II.23: Construct head skeleton

In 1994, we proposed a head turning model to solve those problems [Yu94a]. In the model we put drawing 1 and 9 in the Cartesian coordinate system as shown in Figure II.23 (a) and (b), then rotate drawing 1 clockwise about Y axis by 90^0 to construct a basic skeleton of the head as shown in Figure II.23(c), then cut them with a plane parallel to the $X - Z$ plane to derive four intersection points, interpolating those points with a spline we can get a closed curve which approximates the cross section of the head. Increasing the height of the cut-plane along the Y axis we can get a series of the head's cross section, then pile them together to construct a 3D head image, as shown in Figure II.23(d).

We can get the coordinates of eyes, nose, mouth and ears in a similar manner.

4.3 Specification of the rotation axis

The rotation axis corresponds to the central axis of the neck. Considering the neck center remains fixed and the top of the head moves along a rose curve approximately during the head turning process, we specify a rotation axis passing through the origin and then control a point on its upper part corresponding to the top of the head using the following equation written in Polar coordinate:

$$P = A \cdot sin(t \cdot \pi) \tag{II.4}$$

where t is time, $t = 0$ corresponds to the drawing 1 in Figure II.22 and $t = 1$ corresponds to 9, parameter A and the height of the eyes H_e control the degree of head dipping. Experience shows that we could get a good result if the ratio A/H_e is confined in the range between $0.4 \sim 0.5$.

4.4 Edge detection

We use a simple method to detect the edge of the 3D head image by first projecting all level cross sections on the $X - Y$ plane one by one and then calculating the maximum and minimum values of the corresponding projections along X axis, those values are finally connected in a certain order to draw the contour of the head.

4.5 Hidden line removal

The hidden line removal is also carried out in a simple manner. In the model we use sliding vectors to approximate curves as done in PCLI and then calculate the angle θ_v between X axis and the projection of each sliding vector on the $X - Z$ plane, if $\theta_v \leq \pi/2$, the vector is visible, othewise not. For some curves of complex form, this method could not remove hidden lines completely in which case an interactive graphics editor is needed.

Figure II.24 shows the result of the approach.

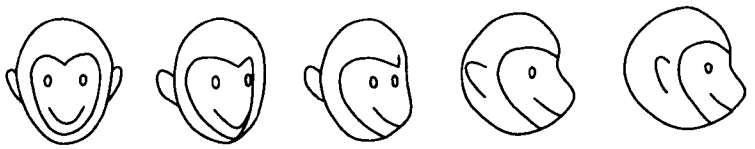


Figure II.24: A monkey head turn

5 Assessment Criteria for 2D Shape Deformation

At present judgement of the correctness of an animation sequence generated by a morphing technique is entirely subjective: play it back and judge it in our minds. If the result looks wrong, we need to modify the transformation according to the information revealed in the playback phase, then generate the sequence and check it again. From this process we see that correct morphing is a vital step in automatic in-betweening.

Most objects we might meet in animation are living objects, natural objects, man-made objects, effects etc. Despite their different forms they all have structures. When they move, their dynamics are constrained by these structures. In animation, reality is exaggerated, not only in terms of features, but also in terms of timing. Thus for the correct morphing of a moving object, we should pay attention not only to *shape*, but also to *structure*, *dynamics* and *timing* and such other factors as required by the principles used in traditional animation.

Existing morphing algorithms pay much attention to shape information, interactive control and mathematical simplicity. Of these only a few take structure information (the skeleton) into account. Since traditional animation does not pay a lot of attention to realistic dynamics and timing, correct results can only be achieved when the key shapes represent the structure, dynamics and timing of the movement at the same time.

Since morphing is a multi-dimensional problem, the assessment of a morphing technique should be made in multiple dimensions, as pointed out by us in [YP97]. Here we propose an "area preservation" principle as an acceptable approximation to "volume preservation" as our shape criterion. A similar approach offers the possibility of objective assessment of dynamics

and timing under certain conditions.

In this section, we first describe how to express our shape criterion in algebraic form, then we discuss other factors such as dynamics, timing.

5.1 Area Preservation

In 2D computer animation, the object is represented by its silhouette and may be regarded as being approximated by a polyline or a polygon. (If, for example a B-spline curve is used it is rendered as line segments which approximate the curve to vanishing accuracy, so in the end everything is represented as polygons or polylines, however small). In traditional animation, usually the object shape deforms but its features retain their character during the movement. A typical technique used in traditional animation is *squash and stretch* in which the object is stretched out in the air and squashed when it hits the ground. In squash and stretch animators are taught to think in terms of preserving the volume of the shape, i.e. no matter how squashed or stretched out a particular object gets, its volume remains constant. This principle is taught to animators in terms of *volume* and in the 2D case experience has shown that this can be approximated by *area*, whence *area preservation*, and from this we can form our shape criterion to judge morphing results.

It is well known that, given the vertices of a polygon, the polygon's area is determined by

$$S = \left(\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \ldots + \begin{vmatrix} x_{N-1} & y_{N-1} \\ x_N & y_N \end{vmatrix} + \begin{vmatrix} x_N & y_N \\ x_1 & y_1 \end{vmatrix} \right) /2 \tag{II.5}$$

where S refers to the polygon area, x_i, y_i $(i = 1, \ldots N)$ are the coordinates of the ith vertex of the polygon, and N is the number of vertices contained in the polygon.

If we are given two key polygons P_1 and P_2, we can calculate their corresponding areas S_1 and S_2. S_1 and S_2 can either be the same or different, for example when zooming into an object. In most practical character and object animation the intermediate area should change progressively according to the foregoing principle. Geometrically this can be described in terms

of an intermediate area S_t $(0 < t < 1)$ varying linearly from S_1 to S_2 as t goes from 0 to 1. Thus the criterion is that if S_t meets this condition then the result is correct, otherwise not.

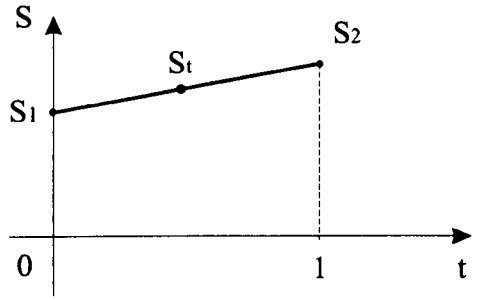


Figure II.25: Area preservation criterion

The foregoing criterion is an ideal one, and can be described using a straight line drawn from S_1 to S_2 as shown in Figure II.25. In practice, however, we should recall the "area preservation" is only an approximation we can get away with. We do not require that the area should be strictly preserved in the object deformed, so it is sensible for us to allow S_t to vary in a tolerant range along the ideal straight line. If S_t falls in the range we can still get an acceptable result. To obtain this tolerant range we first calculate the error between achieved intermediate areas and the ideal one as expressed by the following formula:

$$Error(t) = (S_{ai} - S_t)/S_t \tag{II.6}$$

where S_{ai} refers to the achieved intermediate area and S_t refers to the ideal area which is derived by the linear interpolation between S_1 and S_2. Next we explore the numerical conditions for the tolerant range. We take the area of an object in a correct animation sequence as a reference, then we distort the object thus causing a change in its area, and then play this back to see if it is acceptable. The test was done with different values of $Error(t)$ and we found that if $|Error(t)|$ was less than 10% then the result was acceptable. From this we have a numerical condition to apply.

For an object which is represented by a single curve (polyline) rather than a polygon, we calculate the length of the curve instead and thus the criterion

turns to be *length preservation.* Here area parameters in equation II.6 are replaced by length parameters, and we use the same acceptance criteria.

Usually area is preserved in skeleton techniques and, as long as the skeleton is correctly transformed, we can apply our length preservation criterion to the skeleton as before.

5.1.1 Local Validity

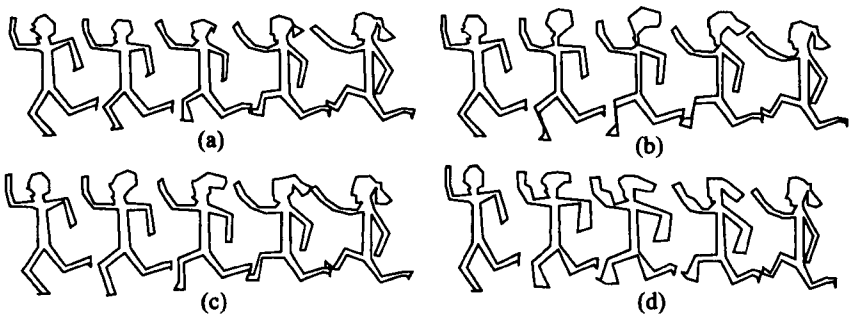


Figure II.26: Comparative results of morphing alogrithms applied to a run cycle

When certain interpolation algorithms are applied to a polygon representing a structural object like a human figure, the outcomes include varying local distortions in intermediate shapes as shown in Figure II.26. As a result, only example (c) with the star-skeleton technique looks right throughout the sequence. In other examples, the lower leg becomes shorter in (a) with CCLI, thinner in (b) with PCLI, or thicker in (d) with the multiresolution method, none of which are acceptable in character animation. Those examples actually pass our numerical condition because the resulting distortions are local and the error in the entire intermediate area is less than 10%. Therefore our numeric condition fails in such cases.

In practice, however, this problem is not serious. Figure II.27 [Har81] shows a typical human figure used in traditional animation. Here different (local) parts of the body are drawn with individual free form lines which in turn are represented by polygons or polylines in computer and it is rare for an animator to draw the whole body with a single polygon as shown in

Figure II.27: A practical cartoon character

Figure II.26. Therefore we can apply our criterion locally to each individual polygon which tiles the figure and make separate assessments.

The foregoing criterion can be used as a general one to accept or reject the intermediate shapes generated by different morphing techniques. As a bonus of applying this criterion to existing morphing techniques, we can derive applicability conditions for CCLI and PCLI [YP97].

5.2 Dynamics

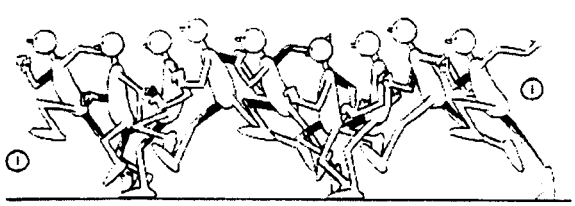


Figure II.28: A correct run cycle in hand drawn animation

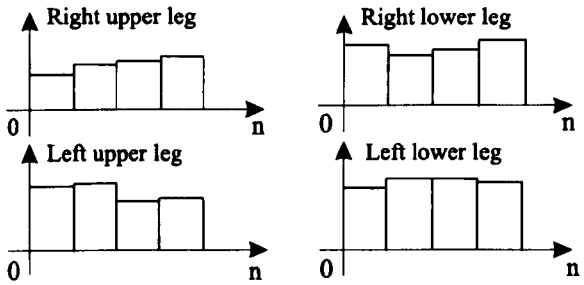


Figure II.29: Angle variation of two legs for a run

At present our numerical criterion concerns the static shape area of deformed object only. It still cannot tell for sure whether these intermediate shapes are dynamically correct, but dynamic correctness of the movement is also a key factor to the success of animation.

From the static aspect, many people claim example (c) of star-skeleton [SR95] in Figure II.26 is correct because it uses a natural representation of a polygon, accounting for its important geometric features. But from the dynamic aspect, it is unclear whether linear interpolation performed on the star-skeleton will produce correct results. The human figure in Figure II.26 looks like it is running, and Figure II.28 [Whi86] shows the correct dynamics for a run used in hand drawn animation. In fact the dynamics of the two legs cannot be controlled correctly by linear angle interpolation on leg skeletons, because the angles of two leg skeletons vary non-linearly within one run step. Figure II.29 shows the graph of angle against frame n which is drawn by measuring angles in the skeletons of the two legs in Figure II.28. A possible solution to this problem is controlling them procedurally as proposed in [Yu90b] and the corresponding angle values of Figure II.29 can serve as its angle constraints.

Since actions like walking and running for animate objects are represented in a fixed manner in hand drawn animation, those angle constraints characterise the dynamics of legs in such motions. Therefore they can be used for an objective assessment of morphing techniques when dealing with these cases. If we take the running figure (c) in Figure II.26 as an example, where their angle interpolation is performed on the star-skeleton (especially those segments corresponding to legs) and is then in accordance with those angle constraints, then the result is correct, otherwise not. In a similar way that

the area preserving criterion is a guiding principle, so these angle constraints serve to aid assessment of dynamical behaviour.

5.3 Timing

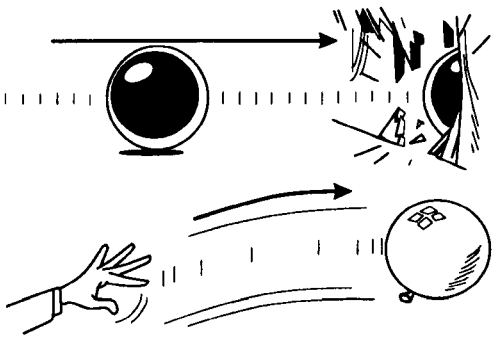


Figure II.30: A cannonball and a balloon

Timing is another key factor in animation. Since timing is reflected by positions of a moving object in space, it is also related to the shape interpolation process. Like the dynamic control problem, we cannot guarantee correct timing even if we get the correct shape. Figure II.30 [Har81] shows a cannonball and a balloon both represented by circles. In both examples a circle is being animated, but the timing of the movement can make it look heavy or light on the screen. The way an object behaves on the screen, and the effect of weight that this gives, depends entirely on the spacing of the animation drawings and not on the drawing itself. It does not matter how beautifully drawn the cannonball is in the static sense, it does not look like a cannonball if it does not behave like one. The same applies to the balloon and indeed to any other object or character.

Timing in animation is an elusive subject. It only exists while the film is being projected. We need good timing in animation, so that enough time is spent preparing the audience for something to happen (anticipation), then on the action itself, and then on the reaction to the action (follow-through). To judge this correctly depends upon an awareness of how the minds of the audience work. This requires a good knowledge of how the human mind reacts when being told a story. In animation, timing becomes a dangerous

factor to try to formulate – something which works in one situation or in one mood may not work at all in another situation or mood. The only real criterion for timing is: if it works effectively on the screen it is good, if it does not, it is not.

Nevertheless, the experience of animators may provide a basic understanding of how timing in animation is ultimately based on timing in nature and how, from this starting point, it is possible to apply such a difficult and invisible concept to the maximum advantage in film animation. Despite these warnings we still think it is possible to make objective timing assessments in certain conditions. If we return to the cannonball and balloon. Suppose their trajectories are the same, so can be represented by $x(t)$, $y(t)$ in the $X - Y$ plane (screen). The differential timing for the cannonball and balloon can be obtained by different sampling on $x(t)$, $y(t)$, i.e. $CTx(t_i)$, $CTy(t_i)$ for the cannonball timing and $BTx(t_i)$, $BTy(t_i)$ for the balloon timing. With such timing functions the distance between each pair of adjacent timing sampling points is measurable and this can be used for objective timing assessment. At least we can use these functions to distinguish them. But it is hard to make a general timing assessment because timing is object dependent like the case of the cannonball and balloon, or application dependent like the different timing of rain can suggest different moods.

5.4 Conclusion

It is not enough to consider shape information only to assess morphing techniques for the purposes of animation. We need to take into account other factors like structure, dynamics, timing etc. A fully objective assessment of the quality of a morphing algorithm is quite impossible, because it depends on multiple factors like the way timing is related to subjective judgement, but some objective assessments are locally possible such as those we have presented in this section. We should point out that our criteria are heuristics rather than formally based rules, yet are the sorts of rules that animators are taught.

Since our numerical criterion was derived from the principle of traditional animation, it is naturally applicable to the problems we might encounter in practical hand drawn animation. Some papers on 2D morphing techniques show examples of morphing from one abstract object to another. The results from these examples do not have any correspondence to ideas of what should happen, so are indistinguishable from magic, where the same phenomenon applies. In these cases any result could be regarded as correct, or for that

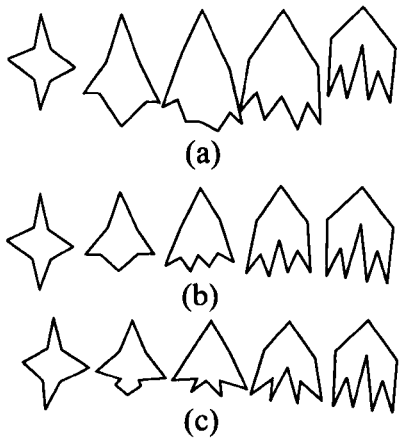


Figure II.31: Comparative results of morphing techniques applied to a fancy shape

matter incorrect, on arbitrary criteria. For the fancy shape shown in Figure II.31 given in [SR95] we cannot find its counterpart in our experience even for its static shape, to say nothing of its dynamic behaviour. Though the results in the figure are used for comparing different morphing techniques, we still cannot tell right from wrong so using any, let alone our, criterion to assess them is meaningless.

6 Procedural Animation

Procedural animation is an area where computer graphics comes into its own. In an essential sense, anything done with a computer can be thought of as being "procedural", but we in computer graphics have a somewhat more specific idea of what constitutes "proceduralism", though the term denies exact definition. At it most basic level, procedural animation means building an object and then using a procedure to control or animate some attribute of the object. Proceduralism is a powerful paradigm for image synthesis. In a procedural approach, rather then explicitly specifying (and storing) all the complex details of a scene or sequence, we abstract them into an algorithm (i.e. a procedure) and evaluate that procedure when and where needed. We gain a saving in storage, as the details are no longer

explicitly specified but rather implicit in the procedure, and shift the time requirements for specification of details from the animator to the computer. We also gain the power of parametric control with its conceptual abstraction and the serendipity inherent in an at least semi-autonomous process.

Another major benefit of procedural techniques is flexibility. As the designer of the procedure, we can capture the essential qualities of the object or motion being modelled without being constrained by physical laws. We can include any amount of physical accuracy we desire into the procedure. We can also include as much artistic expression into the procedure as we choose. The effects achievable are constrained by our procedural design abilities.

The process of developing a procedural model embodied the basic scientific discovery: a formal model is posited, observation and comparisons of the model and nature are made, the model is refined accordingly, and more observations are made. The process of observation and refinement proceeds in a iterative loop.

Procedural techniques have been used in computer graphics since the 1970's. One of the most popular early uses of procedural techniques was creating textures. Procedural techniques have been used for many years to create texture for objects. The use of procedural techniques was explored with the introduction of 3D texturing (solid texturing) by Gardner, Peachey and Perlin in 1985. Realistic wood, stone, marble, water, and clouds could now be included in computer generated images. This inspired many researchers to develop their own solid texture procedures for simulating natural materials. The use of procedural techniques has increased since the mid 1980's to now include modelling techniques (fractals, hypertextures, iterated function systems, L-systems, implicit surfaces, etc) and even animation techniques.

In the following sections, we will review briefly procedural approaches dealing with natural effects phenomena.

7 Particle Set Animation

Reeves [Ree83] is the pioneer in the field of particle systems. This technique uses particle sets to model fuzzy objects such as fire and clouds. In this work an object is represented by a set of particles, each of which are born, evolve in space and die or extinguish, all at different times depending on their individual animation. In this method scripts can be written that control not only the position and velocity of the particles, but also their final appearance

parameters – attributes such as colour, transparency and size. Thus the dynamic behaviour of the particles and their appearance, as a function of time, can be used to control both these aspects of particle behaviour.

Reeves describes the generation of a frame in an animation sequence as a process of five steps:

1. New particles are generated and injected into the current system.
2. Each new particle is assigned its individual attributes.
3. Any particle that has exceeded its lifetime is extinguished.
4. The current particles are moved according to their scripts.
5. The current particles are rendered.

From this it can be seen that the overall shape of the particle cloud, as a function of time, is controlled by any or all of the first four processes.

The instantaneous population of a particle cloud is controlled or scripted by an application-dependent stochastic process. For example, the number of particles generated at a particular time t can be derived from:

$$N(t) = M(t) + rnd(r)V(t)$$

where $M(t)$ is the mean number of particles, $rnd(r)$ a procedure returning a uniformly distributed random number between -1.0 and +1.0, $V(t)$ its variance. The time dependency of this equation can be used to control the overall growth (or contraction) in fire size.

The number of particles can also be related to the screen size of the object – thereby allowing the amount of computation undertaken to be controlled efficiently.

Although this mechanism will clearly contribute something to the shape evolution of the cloud, this is also determined by individual particle scripts. The combination of these two scripting mechanisms was used to animate phenomena such as an expanding wall of fire and multicoloured fireworks. Individual particle scripting is based on the following attributes:

1. initial position,
2. initial velocity and direction,
3. initial size,
4. initial transparency,
5. shape,
6. lifetime.

Velocity and lifetime scripts can be used on dynamic constraints. An explosion, for example, may cause a particle to be ejected upwards and then pulled down under the influence of gravity.

8 Animating Water Effects

Many methods for representing water surfaces, ocean waves, waterfalls have been proposed in the past two decades. However, most of them focused on realistic representation of the phenomena and relatively little time has been spent in modelling the appearance of cartoon water effects. Whitted animated realistic reflections from ripples in a small pool by using ray tracing [Whi80]. The ripples were created by bump mapping the flat pool surface, perturbing the surface normal according to a single sinusoidal function. Perlin has used bump mapping with a richer texture map to convincingly simulate the appearance of the ocean surface as one might see it from an aircraft well out to sea [Per85]. Max used a "height field" algorithm to render explicitly modelled wave surfaces for his film "Carla's Island" [Max81]. His wave model consisted of several superimposed linear sinusoidal waves simulating ocean waves of low amplitude. Peachey presented a model of ocean waves which is capable of simulating the appearance and behaviour of waves as they approach a sloping beach, steepening, breaking, and producing a spray of water droplets from the crests of the waves [Pea86], based on the Gerstner [Ger09], or Rankine, model where particles of water describe circular or elliptical stationary orbits. Fournier used a parametric surface to model the ocean surface [FR86] including the effects of depth such as refraction and surf, and some of the effects of wind. In a paper entitled "Fourier Synthesis of Ocean Scenes" [GMM87] Mastin *et al* used a model based on the work of Pierson and Moskowitz who used wind-driven sea spectra, derived from observed data, to describe the motion of deep ocean waves in fully developed

windy seas. Wave animation is invoked by manipulating the phase of the Fourier transforms. Sims developed some general tools for animating and rendering particle systems that permit both kinematic and dynamic control of particles. They are used to create effects such as wind, snow, waterfall and fire [Sim90]. Mallinder presented a method of implicitly storing particles, and illustrates its use with the modelling of larger waterfalls [Mal95].

9 Animating Gaseous Effects

In recent years methods of depicting gaseous phenomena, such as haze, fog, clouds, dust, smoke and flames, have been studied by many researchers. Again, most of them have aimed at realistic representation of the phenomena, and there are have been few results relevant to the line drawing representation of fire and smoke in the conventional styles of 2D animation. Reeves and Sims simulated firework effects by using particle systems [Ree83]. Perlin generated a solar corona using a turbulence function [Per85]. MIRALab implemented more general fire functions using the same approach [TT87]. Inakage presented a technique based on a physical model of combustion, and succeeded in the photo-realistic representation of the flames of a candle and a Bunsen burner [Ina90]. Ohshima and Itahashi [OI88] presented a simulation method employing 2D fractal texture and other processing techniques for generating animations of flames such as in a bonfire and in candle flames. Augui, Kohno and Nakajima [AKN91] proposed cellular automata with simple state transition rules for simulating flames like those of an alcohol lamp. Gardner [Gar92] modelled fire with fractal ellipsoids. Sakas [SG92a] [Sak93] proposed simulation methods based on the spectral theory of turbulence. Nishita *et al* presented a display method for producing a still image of smoke [NN87] [NMN87]. Chiba *et al* simulated 2D flames and smoke by visualising turbulence [CMTM94]. Stam and Fiume used diffusion processes to animate fire and other gas phenomena [SF95]. In 1996, we proposed a method for dealing with a cartoon fire [YP96a] which will be described in more detail in Chapter V.

10 Summary

Most of the traditional computer animation techniques, such as keyframing, have been used to create many great animations. However, they have several limitations as listed below.

1. Significant animator intervention
2. Low degree of automation
3. Low-level motion specification

Our approach to developing a cartoon effects system creates a new category in procedural animation – stylised effects animation. Their functional design, including modelling, control and rendering, presents challenge to those traditional techniques in CAA. Our work tackles effects movements much more complex than those modelled in existing CAA work as the above. To deal with the broad appearance and movements of effects, we exploit ideas from HdS modelling, from brush models, from painterly rendering, from CAA and from procedural animation.

Chapter III

Cartoon Effects System

In cartoon animation, there are usually two reasons for the use of animated effects. First, we need effects for visual realism. Effects such as water, fire, smoke, rain and snow are a part of our every day environment. In order to create cartoon images of our environment, these effects must be included. Both indoor and outdoor scenes benefit from the addition of effects. The realism and mood of outdoor scenes, such as a dark, dreary forest can be increased greatly by the addition of elements such as fog. Realism of indoor scenes can also be enhanced by the inclusion of steam rising from a cup of coffee or smoke from a fireplace. Second, effects can be used for artistic purpose. For example, rain with different timing can be used to suggest different moods for dramatic effects: a miserable mood can be expressed by rain falling vertically at perhaps half of its normal speed (6 frames crossing the screen for foreground rain), while the speed can increase with a greater tilt from the horizontal for more violent moods.

Automatic methods for in-betweening rely on some degree of continuity in both geometry and time. However, many cartoon effects are implemented either by blending between discontinuous frames or by putting up successive frames where there is no planned continuity between elements, and correspondences are difficult to establish or are even non-existent. In such cases these automatic methods will fail utterly. As for previous procedural methods for dealing with effects, unfortunately, too much effort has been focused on realistic representation of the process, and very few researchers have addressed the issue of modelling stylised effects.

In some cases the effects defeat conventional in-betweening because of lack of continuity of features across more than a few frames, there is nevertheless an underlying structure model which may be amenable to the in-betweening

processes aforementioned in the previous chapter.

There are diverse aspects to the stylistic modelling of cartoon effects, from superficial appearance to procedural dynamic control. Figure III.1 presents an overview of the cartoon effects system. As the figure illustrates, the system consists of three subsystems: *Water effects*, *Fire and Smoke* and *Rain and Snow*. Computer implementation of each effect in the system comprises three parts: *modelling*, *control* and *rendering*.

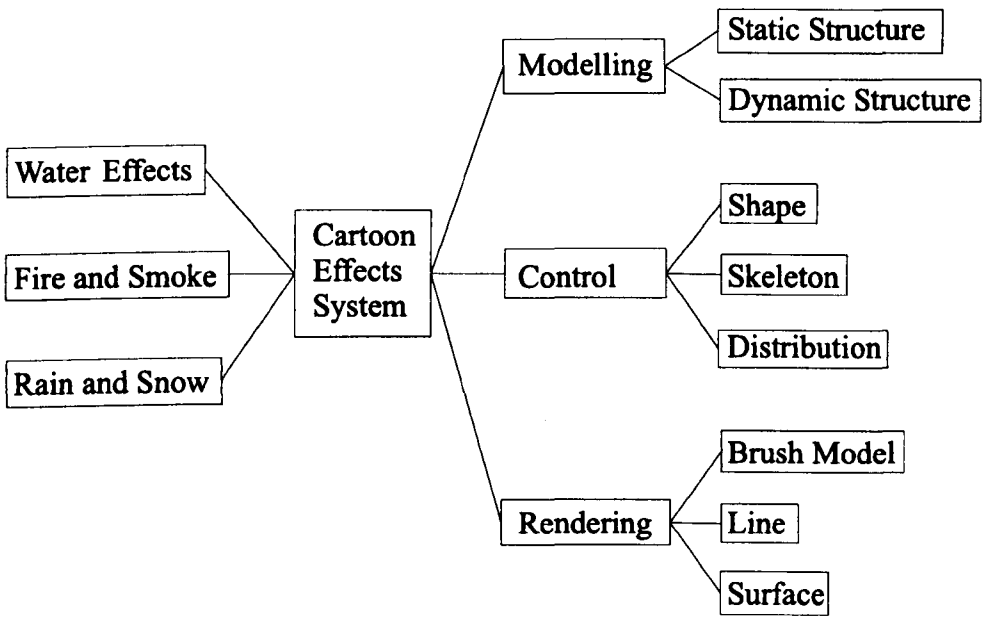


Figure III.1: System overview of the cartoon effects

1 Modelling

Modelling is about establishing the structure of the effects on the static and dynamic bases. The *static structure* captures the spatial relationship of the different parts of the effect. The *dynamic structure* captures the temporal relationship of the different parts of the effect, i.e. which parts remain stationary, which parts are movable and how they move. Since we use HdS modelling approach in the system, model structure of each effect is hierar-

chic (of course detail of hierarchy is effect dependent) which allows us to control different aspects at different levels.

2 Control

Control deals with dynamics both at high level, such as main movements, and at low level, such as local movements or deformations, to ensure the movement consistency. To date, we use three kinds of methods – *shape*, *skeleton* and *distribution* – individually or combined together according to different situations, to achieve the desired controlling goals.

2.1 Shape Control

Shape control aims at direct manipulating on object shape, which can be further divided into two categories: the *time-invariant* and *time-variant*. Simple translation and rotation on the object fall into the first category and most shape blending techniques fall into the second category. Since it is unable to deal with dynamic movement, shape control is usually limited to some local movements at low level in our system.

2.2 Skeleton Control

The skeleton control works indirectly to control the object shape. Technically it comprises *skeletonising* and *skinning*.

Skeletonising, in our approach, relies on the drawing of the object, which can be worked out by our knowledge about the object or just by intuition. Skeletonising involves a *static designing* and *dynamic control*, i.e. the skeleton's shape and how the skeleton moves in space. The skeleton is virtual and can represent the underlying model of the object, or a path of the motion.

Skinning or shape rendering is a process of adding shape to the skeleton which also can be divided into time-invariant and time-variant approaches. The first means the model used to add shape to the skeleton is independent of time, and the second means the model is dependent on time. In the time-variant approach we use deterministic and stochastic models.

2.3 Distribution Control

The distribution approach aims at dealing with some objects which move in masses like shimmering, rain, snow etc. Here each individual element moves randomly but in masses they follow a distributive function. The distribution can be applied to different aspects like position distribution for shimmering and trajectory distribution for rain drops and snow flakes.

3 Rendering

Rendering of shape and colour ensures the appearance of computer generated effects in consistency with that of hand drawn cartoon effects. We use our skeletal spray brush model as well as painterly line and surface rendering models described in Appendix to fulfil the task to render both effects and background.

In the subsequent chapters, we detail the modelling of the aforementioned subsystems of the cartoon effects system. We begin first by presenting water effects.

Chapter IV

Water Effects

In this and the following two chapters we proceed to describe our approach to model stylised (cartoon) effects ranging over water effects, fire and smoke, rain and snow. The hierarchic models we develop are simple, but are nonetheless effective for faithfully animating cartoon effects. We will begin in this chapter by presenting a number of water effects models including flowing water, water jet, water ripples, shimmering, and reflections. In each model, we first describe how an individual effect is created in hand drawn animation and how to extract the structure of the object from hand drawings to construct the model. Subsequently we describe a procedural control scheme based on the static and dynamic structural information abstracted above. Simultaneously examples are given for the relevant models.

1 Flowing Water

In the studio, on a simple level, the effect of flowing water can be created by first establishing a plain background colour for the water and then animating a series of shapes across it to simulate the feeling of movement we get when we watch a running stream. The shape can be abstract, stylised, or realistic. Whatever shapes are selected, they should flow in a consistent way, following the chosen path of action. Because water action is never mechanical, the animator has to draw many random frames which is quite time consuming and expensive. If restricted in time and budget, the animator usually produces a number of cycles and repeat them at random so that there is no observable repetition. If the water action is more violent than a simple flowing movement, the animator can add another level of animation, depicting white caps

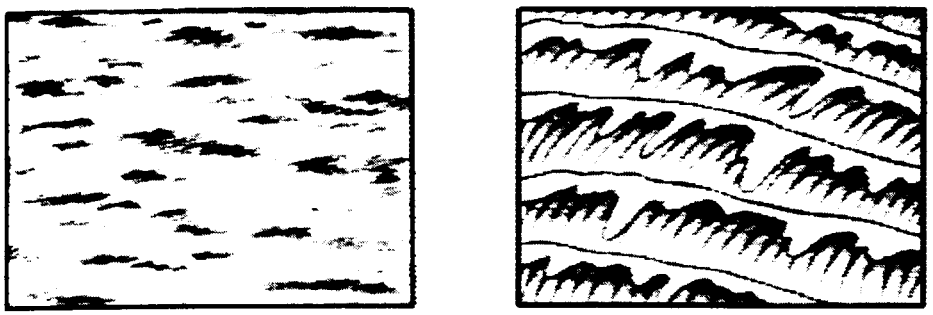


Figure IV.1: Hand drawn water wave

at the top of the waves. Figure IV.1 [Whi86] shows two frames of different flowing water shapes drawn by the animator, the first can be regarded as realistic one (note it is not photo-realistic) and the second stylised one.

In this section we would like to describe the flowing water model synthesising the second drawing in Figure IV.1 based on our early work [Yu94b]. A critical step towards higher level, procedural modelling is to abstract the structure and the dynamic behaviour of cartoon flowing water. To this end, we begin with tracing the animator's drawing process and build our flowing water model using the information abstracted from the decomposition of the hand drawing process.

An animator, in drawing a frame of flowing water representing a river, say WF_1, is liable to start with two *boundary curves*, then proceeds to a series of curves spanning the boundary curves (which we refer to as *horizontal waves*), and then to short curves attached to each horizontal wave (which we refer to as *vertical waves*) as shown in Figure IV.2. Finally, he adds white *caps* at the top of the horizontal waves and *foams* surrounding the caps.

In the succeeding frame, say WF_2, he follows the same procedure by first drawing identical boundary curves with those in FW_1, then by drawing horizontal waves in different positions with respect to where their counterparts are in WF_1 (to show the movement of waves along the river) as depicted in dash lines in Figure IV.2(b), and finally by adding vertical waves, caps and foams in corresponding new positions.

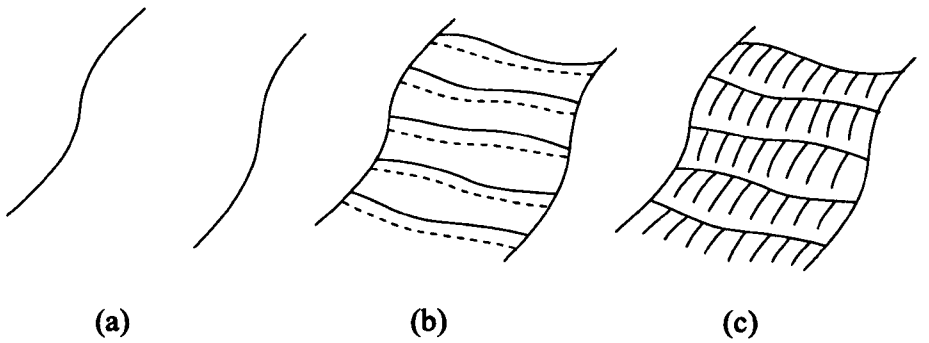


Figure IV.2: Flowing water drawing process

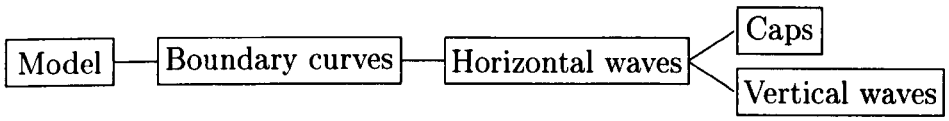
(a) Drawing two boundary curves (b) Adding horizonyal waves (c) Adding vertical waves

In a similar fashion, the animator completes the animation series of required length by drawing $WF_3, WF_4, ...WF_{WN}$, where WN is the number of frames contained in the animation.

It is important to note that animator's drawing procedure is sequential hence we are able to decompose the procedure along the time axis in terms of *drawing-frame* and *drawing-step*. A drawing-frame corresponds explicitly to drawing a frame and a drawing-step is concerned with drawing some parts of similar characteristics within one drawing-frame such as boundary curves, horizontal waves, vertical waves, caps and foams. Different drawing-steps involved in one drawing-frame reveal the spatial relationship of different parts of the water drawing from which we can extract the static structure for our model. Any variation in position, shape etc involved in those drawing-steps between different drawing-frames reveals the temporal relationship of different parts of the water drawing from which we can extract the dynamic structure for our model.

It should be pointed out that decomposing a drawing-frame into drawing-steps may be person dependent and our decomposition is adequate for the modelling task at hand.

The static structure of the flowing water model may be thought of as a ladder with two supports representing two boundary curves and rungs representing horizontal waves which can be expressed by the following hierarchic model:



To obtain the dynamic structure of the flowing water our first task is to analyse which parts remain *stationary*, which parts *move* and *how* they move depicted in the water drawing series. Clearly, the stationary parts are two boundary curves and moving parts are horizontal waves, vertical waves, caps and foams which flow consistently with variations in shape along the boundary curves.

We construct our flowing water model using the spatial information provided by the static structure and gain a procedural control over the moving parts using the dynamic information provided by the dynamic structure. Implementation details will be given in the remainder of this section.

1.1 Boundary Curves

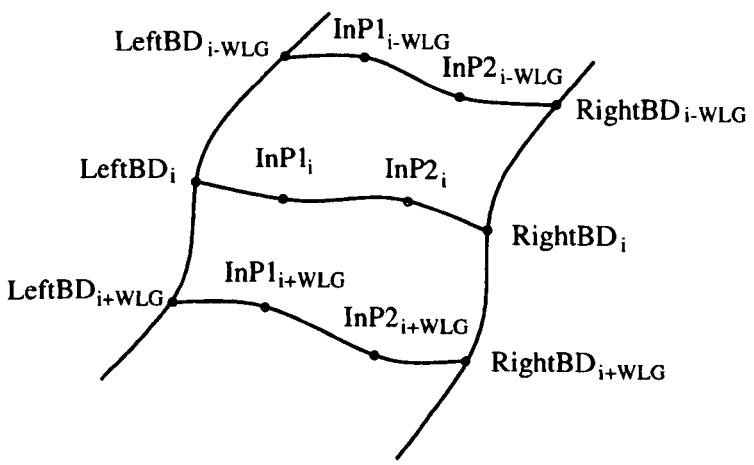


Figure IV.3: Boundary curves and horizontal waves

Boundary curves are drawn to confine the river path according to the scene, thus in our model we require the user to specify some control points defining the main shape of the path which are then interpolated by splines

to get $LeftBD_i$ and $RightBD_i$ $(i = 1, ...BdN)$ to draw the curves as shown in Figure IV.3, where BdN is the number of points contained in two curves. With the index assigned in the two arrays we can define the reference direction of the flowing water. The number of interpolated points between the two successive control points should be sufficient to ensure the motion continuity of horizontal waves, because the movements of the horizontal waves are represented by the displacement between two successive interpolated points on the boundary curves.

1.2 Horizontal Waves

Horizontal waves span in series the river path. The distance between two successive horizontal waves corresponds to the *wave length*, namely WLG, which is a number to count certain points contained, say, between $LeftBD_i$ and $LeftBD_{i+WLG}$. Provided with BdN and WLG, the number of horizontal waves contained along the river can be simply obtained by $HrzN = BdN/WLG$.

In designing individual horizontal waves, we first take $LeftBD_i$ and $RightBD_i$ $(i = 1, ...HrzN)$ as extremes, then introduce two intermediate points InP_1 and InP_2 as shown in Figure IV.3 by the following formulae:

$$\begin{aligned} InP_{1,i} &= LeftBD_i + u_1(RightBD_i - LeftBD_i) + rnd(r)V(InP) \\ InP_{2,i} &= LeftBD_i + u_2(RightBD_i - LeftBD_i) + rnd(r)V(InP) \end{aligned} \quad \text{(IV.1)}$$

where u_1 and u_2 are positional parameters which, together with random variables of variance V(InP) would make a wary line. Finally, we interpolate those four points, $LeftBD_i$, InP_1, InP_2 and $RightBD_i$ with splines to get $HrzWV_{i,j}$, $(i = 1, ...HrzN, j = 1, ...HrzM)$ that draws horizontal wave curves, where $HrzM$ is the number of points contained in the interpolated curve.

1.3 Vertical Waves

Given $HrzWV_{i,j}, (i = 1, ...HrzN, j = 1, ...HrzM)$, vertical waves attached to the horizontal waves can be generated through the following steps:

For each horizontal wave i

1. Set j=1.
2. Calculate the length $LGHW_j$ between two points $HrzWV_{i,j}$ and $HrzWV_{i+1,j}$.
3. Interpolate $HrzWV_{i,j}$ and $HrzWV_{i+1,j}$ linearly with p varying randomly between [0.75, 0.9] to get terminal control points $CtrlP_2$ where the magnitude of p controls the length of the vertical wave which is determined through experiment.
4. Calculate the middle point between $HrzWV_{i,j}$ and $HrzWV_{i+1,j}$, then add $\beta LGHW_j$ to the Y coordinate of this point to get a control point $CtrlP_1$. Where an additional component is added to the Y coordinate for generating an arc vertical wave , we have found through experiment that $\beta \in [0.1, 0.13]$ would make a good looking curve.
5. Interpolate $HrzWV_{i,j}$, $CtrlP_1$ and $CtrlP_2$ with a spline to get a vertical wave curve.
6. Increase index j and repeat the above steps to generate the successive vertical waves until j reaches $NrzM$.

End (of each each horizontal wave).

1.4 Caps

The modelling of each white cap at the top of waves involves two parts. One part is concerned with the front shape of the cap correlated with vertical waves which can be generated as follows:

1. Set j=i.
2. Generate a positional parameter p varying randomly between [0.25, 0.55] (The magnitude of p is determined through experiment to avoid possible strobing which will be described shortly).
3. Interpolate $HrzWV_{i,j}$ and $HrzWV_{i+1,j}$ linearly with above p to get a control point $CapCP_j$.
4. Increase index j and repeat above two steps to generate the successive control point $FCapCP_{j+1}$ until j reaches $HrzM$.

5. Interpolate $FCapCP_{i,j}(j = 1, ...HrzM)$ with a spline to get $FCap_{i,k}$ $(k = 1, ...FCapN)$ (where $FCapN$ is the number of interpolated points) that draws the final front boundary line of the cap.

The other part is concerned with the back shape of the cap which is depicted readily by the horizontal wave $HrzWV_{i,j}, (j = 1, ...HrzM)$. The complete contour of the ith cap $Cap_{i,j}$ $(j = 1, ...CapN)$ can be derived by appending $FCap_{i,k}(k = 1, ...FCapN)$ to $HrzWV_{i,j}$ $(j = 1, ...HrzM)$ where $CapN = FCapN + HrzM$ and all caps can be generated by varying index i from 1 to $NrzN$.

1.5 Foams

Foams are represented by small circles with variations in size and distributed surrounding the contour of the white caps with a simple model.

1.6 Structure of the Model

The structure of the model can be expressed as follows:

Initialise the model by specifying the following parameters:

a. BdN, number of points contained in two boundary curves;
b. $HrzM$, number of points contained in horizontal waves;
c. WLG, wave length;
d. Spd, moving speed of horizontal waves;
e. $HrzN = BdN/WLG$, number of horizontal waves.

Specify two sets of control points.

Interpolate control points to generate two boundary curves $LeftBD_i$, $RightBD_i, (i = 1, ...BdN)$;

For each frame t:

1. Generate horizontal waves $HrzWV_{i,j}(t) = HrzWV_{i+t*Spd,j}$.
2. Generate vertical waves.
3. Generate caps $Cap_{i,j}(t)$.
4. Distribute foams.

End (of each frame).

1.7 Colouring

In comparison with shape, colour is less important for the water movement. This is true when we see early black and white animation. Even line drawn sequences by themselves can produce the illusion of movements. But colouring does improve the visual quality of the animation. A straightforward way to colour water is using a plain colour such as light blue to fill water area, dark blue to draw horizontal and vertical waves, white colour to draw capes and foams and the result seems satisfactory. An alternative way to colour water is using gradient colour varying from light to dark blue to fill the area surrounded by two adjacent horizontal and boundary curves in which case we discard drawing vertical waves and get a variant of flowing water derived from the same model. Furthermore, effects such as sun rise and sun set can be simulated by simply changing the colours of flowing water over time t.

1.8 Working in 3D

Figure IV.4: Sample 1 of 3D flowing water model

As we have mentioned in Chapter I that static and dynamic structures are invariant in both 2D and 3D, which allow us to add the third dimension to parameters involved in above model. The left boundary curve, for instance, can be expressed by $LeftBD_i = LeftBD(x_i, y_i)$ in 2D and $LeftBD_i =$

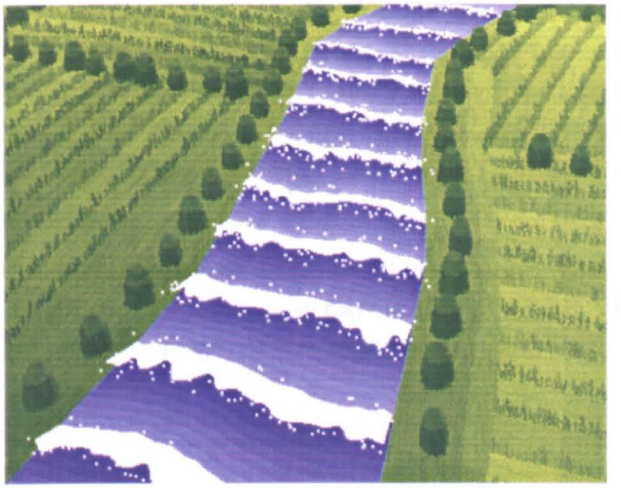


Figure IV.5: Sample 1 of the 3D flowing water model

$LeftBD(x_i, y_i, z_i)$ in 3D respectively. As a result we achieve 3D stylised water animation and Figure IV.4 as well as Figure IV.5 show two frames of our 3D cartoon flowing water animation (To keep a stylised look in this example we use our painterly rendering model for other objects in the background, see Appendix).

1.9 Strobing

Strobing, originally, is an effect which is an integral part of the mechanism of the cinema and is liable to occur in the movement of an object which has a number of equally spaced similar elements. A well known example is the illusion of stage-coach wheels appearing to turn backwards in films. At the early stage in developing our model, we experienced a different strobing problem occurring on caps.

In animation, any variation in position or shape in one object between two successive frames would be perceived as a movement. Among the moving parts involved in the flowing water model, white caps are big in size and liable to attract our attention more than vertical waves and foams do. In the model the caps move $|Spd|$ in every frame (where $|Spd|$ is the distance covered by Spd points on the boundary curve) with their front shapes $FCap_{i,k}(k =$

$1, ...FCapN$) varying stochastically. Let $\Delta FCAP_{i,k}(t) = |FCap_{i+(t+1)*Spd,k} - FCap_{i+t*Spd,k}|$ denote the distance of front shapes of the ith cap between frame $t+1$ and t, if situations such as $\Delta FCAP_{i,k}(t) > |Spd|$ and $\Delta FCAP_{i,k}(t+1) < |Spd|$ arise, then the local front shape of the ith cap would appear moving backwards in frame $t+1$ with respect to frame t thus causing a strobing effect.

The cure for this strobing effect is to avoid situations where it may occur. In the model we restrict the positional parameter p within the range [0.25, 0.55] in controlling the front shape of the caps to ensure $\Delta FCAP_{i,k}(t) < |Spd|$ in every frame.

1.10 Summary

In this section we described a cartoon flowing water model which offers several advantages over the hand-drawn water effects. First, the model is flexible for representing water effects such as rivers, waterfalls, taps, sea surface, etc provided with relevant initial control points for the effect of interest. If, furthermore, the control points are specified on a pole surface, it is possible to animate the flowing water around the pole–a fancy scene we may use in a fiction cartoon. Second, the model is flexible for controlling water actions. One example would be representing how violent the flowing water is by changing the sizes of the caps. Another example would be simulating the process of water flowing into some dry area or disappearing, which we might see when a sluice gate is opened or shut on a sluice-way.

As for synthesising the first drawing in Figure IV.1, we need to design different wave shapes visualising the moving parts in the model and this is a topic for a future work. We would like to point out that although wave shape visualisation may be user and application dependent, our dynamic control scheme remains appropriate.

2 Water Jet

Water under pressure can exert a considerable force. If a jet is directed upward at an angle it describes a parabola as each individual drop behaves, as shown in Figure IV.6 [Har81], partial gaps in the jet help to avoid strobing in animation.

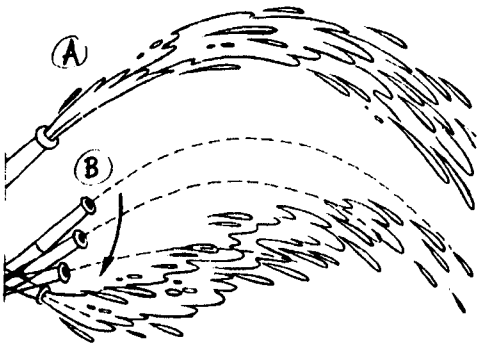
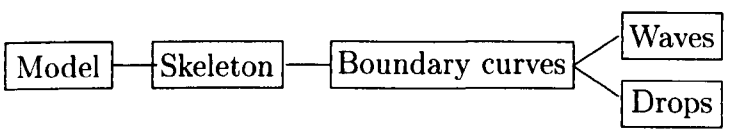


Figure IV.6: Water jet

As far as water jet drawing is concerned, it resembles very much the flowing water in structure composed of boundary curves, waves and drops. However, in addition to water moving along the parabola, the water jet may animate itself, such as the jet coming from a nozzle held by a fireman (In the remainder of this section, we will use the nozzle for describing our model). In order to gain an effective control over the dynamics of the parabola, we introduce a skeleton to govern the parabola in the model which, together with other parts of the jet, can be expressed by the following structure:



The following subsections will give a detailed description of the model.

2.1 Skeleton

The skeleton approximating the parabola or the trajectory of the water jet can be constructed by interpolating a few control points, either specified manually or computed from a simple model, as depicted in a dash line in

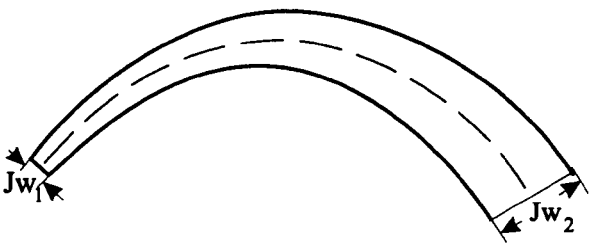


Figure IV.7: Water jet skeleton

Figure IV.7. Those control points determine the position and the main shape of the skeleton that, if given two sets of control points representing the initial and end positions of the skeleton respectively, we are able to interpolate the skeleton positions to animate the skeleton moving from its initial to end positions. An alternative way of animating the skeleton is by means of varying the parameters attributed to a parabolic model.

2.2 Boundary Curves

Boundary curves represent the jet outline in the air. The jet may be thought of as a cylinder with its diameter increasing as water moves outward from the nozzle. In the 2D case, we define two parameters, JW_1 corresponding to the diameter of the nozzle and JW_2 corresponding to the width of the other end of the jet determined according to the scene (see Figure IV.7), from which two boundary curves can be derived by a simple rule, as shown in solid line in Figure IV.7.

2.3 Waves

Waves feature in the water texture caused in the jet moving along the parabola and the illusion of the movement is created by drawing them in different positions in different frames. In the model we control the jet wave moves in a similar manner to controlling the horizontal waves in the flowing water model. However, the wave shape differs dramatically between the two and we model jet waves using a number of drop shape curves put on a skeletal curve on the jet as shown in Figure IV.8.

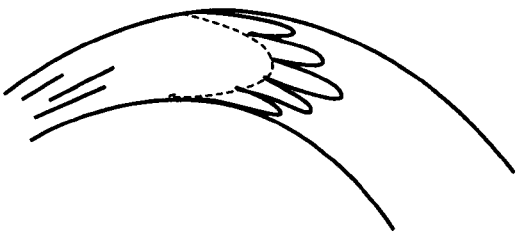


Figure IV.8: Water jet wave

The skeletal curve of the jet wave can be approximated by a sinusoid wave spanning the two boundary curves at a corresponding pair with its amplitude varying stochastically. Implementation details of modelling drop shapes put on the skeleton will be described shortly.

Short lines showing the speedy effect of the jet just coming out of the nozzle can be modelled simply by putting a few lines near the nozzle with variations in length and orientation.

2.4 Drops

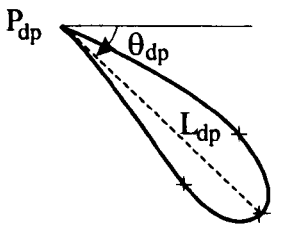


Figure IV.9: Water jet drop

As the jet moves outward from the nozzle, it gradually splits into drops. The farther the jet moves from the nozzle, the more the drops are split. To model an individual drop we define three parameters, a position P_{dp}, drop length L_{dp} and drop angle θ_{dp}. The drop width $JDpW$ corresponding to

the widest part of the drop is drop length dependent and, our experience shows that, if $JDpW$ is set to be about $0.1L_{dp}$ then we could get a good looking drop. Above parameters are sufficient to define a few control points as shown by crosses in Figure IV.9 and the final drop shape is derived by interpolating those points with a spline. All parameters attributed to the drop are controlled by stochastic models to avoid the mechanical look of drops resulted from using uniform parameters.

The foregoing drop model, together with an additional flag *open* associated with a simple mechanism to keep the drop open, can be used to generate drop shapes shown in Figure IV.8.

2.5 Structure of the model

The structure of the model can be expressed as:

Initialise the model by specifying the following parameters:

a. Specify a few control points for the skeleton according to the scene;
b. JN, number of points contained in the skeleton;
c. $JWLG$, jet wave length;
d. $JSpd$, speed of jet waves moving;
e. $JHrzN = JBdN/JSpd$, number of jet waves.

Generate a skeleton $JSkltP_i, (i = 1, ...JN)$ by interpolating the control points with a spline.

Calculate two boundary curves $JLeftBD_i, JRightBD_i, (i = 1, ...JN)$.

For each frame t:

1. Draw a few short lines near the nozzle.
2. Generate wave skeletons at corresponding pair $JLeftBD_{i+t*JSpd}$, $JRightBD_{i+t*JSpd}$.
3. Put a few drop shapes to the wave skeletons.
4. Generate individual drops inside and outside the jet with number increasing from the nozzle to the far end of the jet.

End (of each frame).

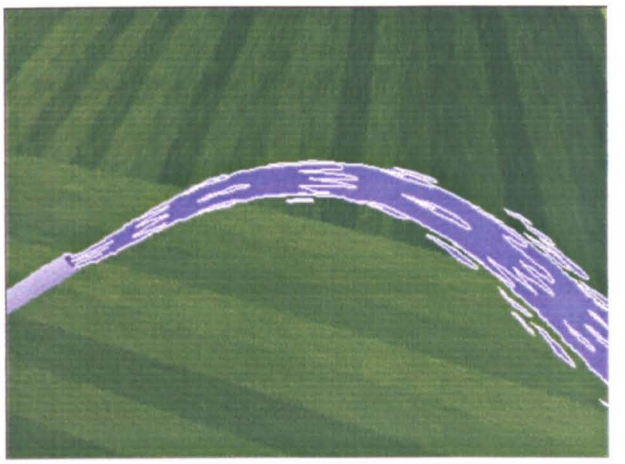


Figure IV.10: Water jet generated by the model

2.6 Working in 3D

The hierarchical structure used in the model offers a higher controllability which in turn allows us to construct the jet in 3D with relative ease. We begin with the driving element – parabola skeleton – at the top level of the model and build a 3D jet using a hybrid approach. Detailed implementations are as follows:

1. Generate a 3D skeleton by specifying control points and interpolating them in 3D.
2. Project the 3D skeleton onto 2D plane.
3. Based upon the 2D skeleton projected, generate the other parts of the jet according to the 2D water jet model.

The obvious advantage of this hybrid approach is that it is fast because it avoids operations such as hidden line/surface removal and projections from 3D onto 2D in the rendering phase. The limitation of this hybrid model is the lack of volume information that the 3D jet is supposed to have hence fails to produce the water jet at certain angles. For instance, the jet skeleton is perpendicular to the screen. Nevertheless, this is not a serious problem

because in animation the jet is often seen from the side or some point above the jet and our hybrid model is adequate for the task at hand. Figure IV.10 shows an example of the 3D jet animation.

2.7 Summary

This section described a skeleton-driven water jet model which can be used directly for representing water coming from a nozzle, jet for irrigating etc. In comparison with the flowing water model, the introduction of the skeleton offers a high level control over the dynamics related to the position of the water jet.

3 Water Ripples

Effects around objects partially immersed in water are usually shown as ripples (Figure IV.11 [Har81]), which radiate from around the object and gradually split up and disappear. If the object is not moving or if it is moving in a cyclical way, then the effects can be made into a cycle. But if the object itself is animating, the effect must be animated continuously, which can mean a great deal of work in hand drawn animation.

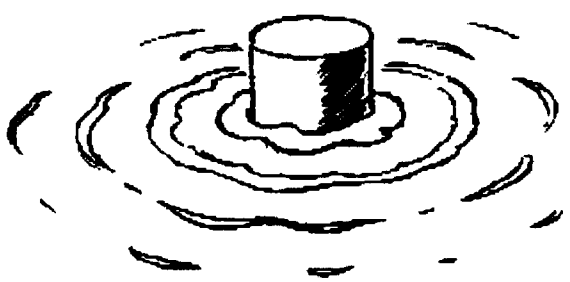
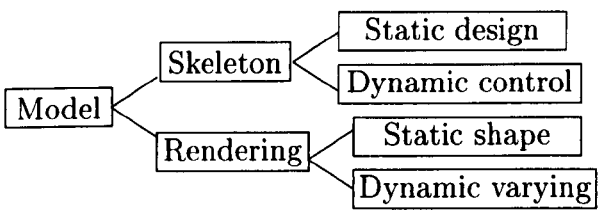


Figure IV.11: Ripple from a partially submerged object

In regard to Figure IV.11, a straightforward way to decompose the ripples drawing process is as follows:

1. Draw a partially submerged object.
2. Draw some concentric ellipse skeletons.
3. Draw ripple's shapes along the skeletons.

Apparently, the static structure of the ripple's model is composed of a number of concentric ellipses and the dynamic structure of the model is featured by the centre of the ellipses (the stationary part) and ellipses radiating at a constant speed (the moving parts). In conjunction with the shape rendering of the ripples which varies over time, our ripples model can be expressed with the following structure:



In the following sub-sections we proceed to describe how we implement the skeleton control and shape rendering in the model.

3.1 Ripple Skeleton

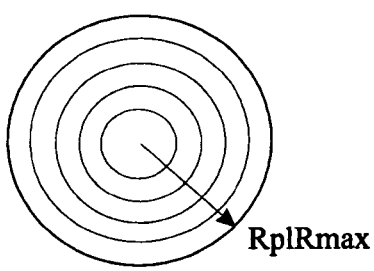


Figure IV.12: Ripple skeleton

At the skeleton level the model involves two parts: static shape design and dynamic control.

The shape of the ripple skeleton is determined by the boundary curve formed between the object and the water. Take Figure IV.11 for example: the object is a cylinder, so the radiating ripples display a round shape and Figure IV.12 shows their corresponding static skeletons viewed from the top. The parameter $RplRmax$ confining the dynamic range of skeletons and the number of skeletons falling into the range $RplN$ are specified by the user according to the scene.

To give a perspective view of the round ripples we choose to use ellipses as skeletons instead. The major and minor axes of the ellipses are defined by $RplRmax$ and $\beta RplRmax$ respectively, where $\beta \in [0, 1]$ is a factor representing the degree of the perspective. For some objects of interest with different shapes, a ship, for instance, we can specify some control points and then interpolate them with a spline to draw the skeleton shape desired.

Dynamic control of the skeletons is achieved by simply scaling the major and minor axes of the ellipses with time. In order to avoid strobing effect, we design five intermediate positions between two successive skeletons that, for a given frame t, the major axes of radiating ellipses are determined by $RplR_{i,t} = i \cdot DR + t \cdot 0.2RD, (i = 1, ...RplN)$, where $RD = RplRmax/RplN$ is the distance between two adjacent ripple skeletons.

3.2 Ripple Shape Rendering

The ripple shapes shown in Figure IV.11 suggest that each boundary curve can be approximated by a serial of sinusoidal curves drawn along two sides of the skeleton. If provided with relevant attributes, we can accomplish the task of ripple shape rendering using our brush model described in Appendix with relative ease.

In the model, we define the inner and outer width attributes InW and $OutW$ associated with each skeleton, as shown in Figure IV.13 where the horizontal axis corresponds to the skeleton. Those attributes are characterised by connecting a series of sinusoidal units with variations in amplitude $RplAm$ and radian frequency ω. The ripples moving outward shown in Figure IV.11 appear progressively thiner and this is controlled by a parameter $RplW_i$ which value decreases with index $i(i = 1, ...RplN)$ assigned from inner-to-outer in the ripples.

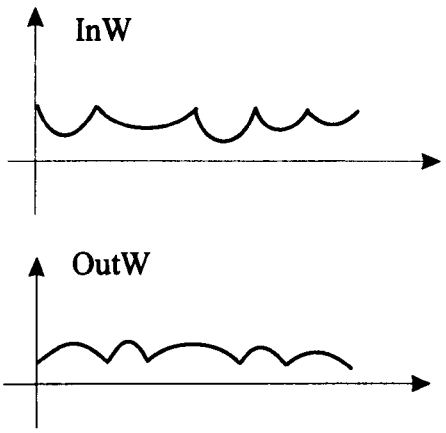


Figure IV.13: Ripple width function

3.3 Splitting Up

In Figure IV.11 splitting up may appear in the outer ripples. The farther the ripple is from the object, the smaller the ripple becomes and, correspondingly, the distance between broken ripples becomes longer. We model splitting up at two levels: skeleton controlling and shape rendering. At the skeleton level splitting up is controlled in two phases: in the first phase we are concerned with the over all skeletons and set a threshold $RplT = 0.5RplRmax$, if the major axis of the skeleton is less than the threshold, the ripple is closed, otherwise it splits up. In the second phase we determine the positions of splitting up in the individual split up ripples by setting a boolean function shown in the upper part of Figure IV.14 where the horizontal axis corresponds again to the ripple skeleton: the ripple appears where the function is 1 and disappears where the function is 0. The duration of positive value in the boolean function varies stochastically with its mean value decreasing as the ripple moves outward and, inversely at the same time, the duration of 0 in the boolean function varies stochastically with its mean value increasing. At the shape level, the appearing ripple pieces are rendered further by the shape weight of a sinusoidal form shown in the lower part of Figure IV.14 to get a more natural look of the broken pieces.

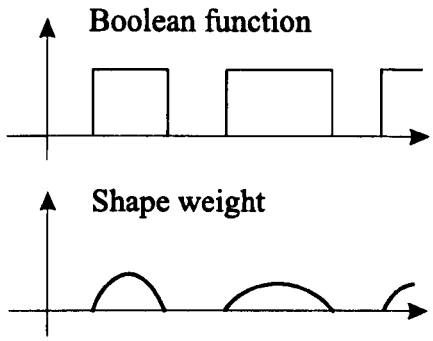


Figure IV.14: Switch and shape weight function

3.4 Structure of the Model

The structure of the ripple model can be expressed as:

For each frame t:

For each skeleton i, $(i = 1, ...RplN)$.

1. Generate inner width attribute $InW(t)$.
2. Generate outer width attribute $OutW(t)$.
3. Generate boolean function $Bl(t)$.
4. Generate shape weight $W(t)$.
5. Calculate the final width attributes: $RplInW_i(t) = RplW_i \cdot InW(t) \cdot Bl(t) \cdot W(t)$ and $RplOutW_i(t) = RplW_i \cdot OutW(t) \cdot Bl(t) \cdot W(t)$.
6. Calculate the radius of the skeleton. Take a round skeleton for example: $RplR_i(t) = RplR_0 + t \cdot RplD + i \cdot RplRmax/RplN$, where $RplR_0$ is the radius of the object.
7. Call brush model to draw the final shape of the ith ripple.

End (for each skeleton).

End (for each frame).

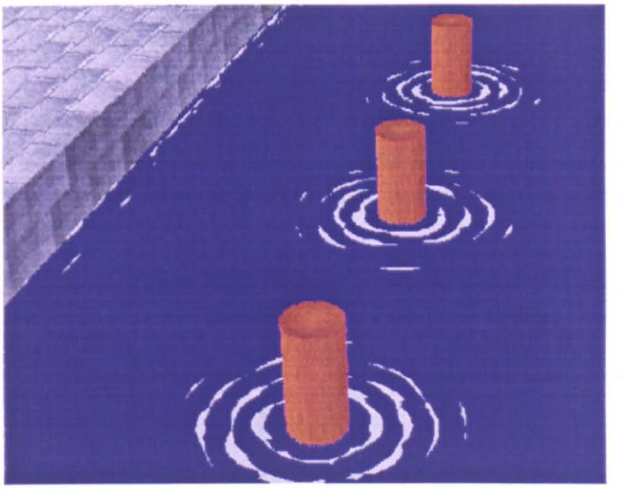


Figure IV.15: Ripples generated by the model

3.5 Result

Figure IV.15 shows a frame of ripples animation generated by the model described above. In this example the ripples skeleton are specified on $X - Z$ plane in 3D: circles for ripples surrounding the submerged poles and parallel lines for ripples near the embankment. The final ripples are generated by introducing the 2D counterparts of projected 3D skeletons into the model described above.

3.6 Summary

We presented a computer model currently capable of dealing with ripples near or around still objects. The model can further cope with ripples effects associated with a moving object. Consider a moving ship, for instance, as long as ripple skeletons are defined in front of the ship, we can animate them radiating while moving along with the ship with relative ease in comparison with hand-drawn animation achieving the same gaol.

4 Shimmering

In hand drawn animation, a different technique is used to show shimmering sunlight or moonlight on the surface of a lake, river, or sea. Basically, this is a series of cels with varied interpretations in white of the reflected light, see Figure IV.16 [Whi86].

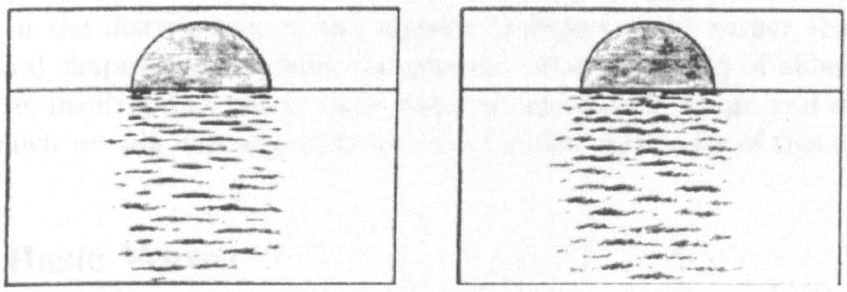


Figure IV.16: Shimering of sunlight

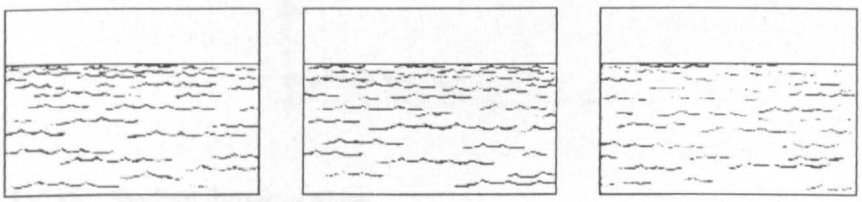


Figure IV.17: Moving surface

After the cels are completed, the animator dissolves from one to the other, at random, on simple mixes from two frames per mix to ten frames, depending on the speed of the effect required. This produces a beautiful changing pattern of reflected light on the surface of the water. The same technique can be used to indicate the moving surface of lake and sea, when a random effect of small waves shown in Figure IV.17 [Whi86] is desired. Several cels – the more, the better – are drawn in black or coloured line in

the required style, then they are mixed one to another, at random, at the required speed.

Shimmering differs dramatically from other water actions we have described in previous sections in that it has no deterministic structure associated with the movement but displays a random effect on the whole. The varied interpretations of the reflected light, realistic or stylish, are completely independent from one frame to the next that the frame coherence is maintained in the distribution of the massive interpretations rather than their individual shapes and dynamic movements. The modelling of shimmering, therefore, involves two parts: basic wave interpretation design and distribution, which we will describe in more detail in the remainder of this section.

4.1 Basic Waves

4.1.1 Stylish Wave

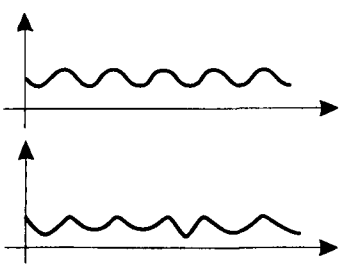


Figure IV.18: Stylish basic waves

The most commonly used stylish waves for the interpretation of reflected light are shown in Figure IV.18. The first is just a sinusoidal wave characterised by the wave cycle, wave phase and the number of cycles. The second is the same function as the inner brush width attribute used for ripples' shape rendering in section 3.2 of this chapter.

4.1.2 Realistic Wave

In cartoon animation, waves drawn in Figure IV.16 may be regarded as realistic ones. The method for generating ripple's shape can be used directly to generate realistic waves for shimmering hence the former inner and outer width attributes turn into up and down width attributes which are then assigned to the level skeletons with variations in length.

4.2 Distribution

Figure IV.17 suggests that it is reasonable to use two rules governing the horizontal and vertical distribution of individual waves. The first is uniform distribution and the second is a linear slop distribution with which waves are distributed progressively sparsely (i.e. the bottom part of the frame appears near to our eyes) in order to show the perspective effect.

4.3 Model Control

The procedure of generating a frame of shimmering can be outlined as follows:

1. Choose wave type: stylistic or realistic.
2. Specify the mean value and variance for the length of skeletons.
3. Specify the density of the waves.
4. Specify horizontal distribution.
5. Specify vertical distribution.
6. Generate a position $ShimP$ according to the above two distribution functions.
7. Generate a skeleton at $ShimP$.
8. Generate a wave at $ShimP$ with a method similar to that described in Section 3.4 according to the wave type chosen.
9. Repeat step 6, 7, 8 until the density of the wave distribution is met.

Figure IV.19: Realistic moving surface generated by the model

4.4 Results

Figure IV.19 shows a frame of shimmering animation generated by our computer model. Here the individual waves are animated in 3D by first distributing skeletons uniformly along both X and Z axis on the $X - Z$ plane and then equipping the model described above with the 2D skeletons projected.

5 Reflections

Reflection is achieved in hand drawn animation by reversing the animation to be reflected so it is upside down and shooting it at a percentage exposure in the required position. A character might, for example, be walking along a river bank on top pegs, then the animator traces, upside down, the identical action on a separate cel and then shoots them at less 100 percent exposure. See Figure IV.20 [Whi86].

Unfortunately, this method can be used only when the surface on the reflection is to appear perfectly smooth. If the water is moving or rippling, the reflected image should become distorted in proportion to the amount of disturbance present. The rougher the water, the greater the breakup of the image, as shown in Figure IV.21 [Whi86].

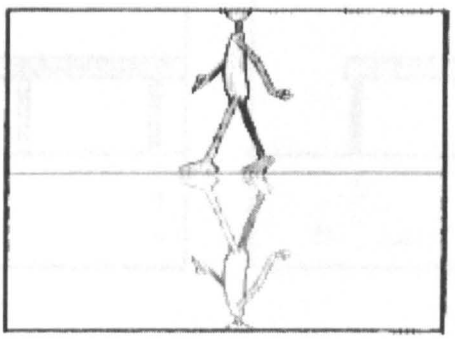


Figure IV.20: Reflection of a walking character

For the almost still water, the reflection is a distorted image and Figure IV.22 [Har81] shows part of a cycle of a bright light.

In this section we present a skeleton-driven model capable of reproducing reflections shown in Figure IV.22. In order to deal with the dynamics of the reflected light, we model its skeleton at two levels: a *base skeleton* framing the structure of the reflection and upon which the *dynamic skeleton* controls local movements of the reflection. The base skeleton may be either stationary or mobile: an example of the latter case is the skeleton used for representing the light of a car moving along the beach. Implementation details of the two skeletons as well as shaping of the reflected light associated with the skeleton will be described as follows.

5.1 Base Skeleton

The irregular appearance of the reflected light makes the structural extraction a difficult task. Dynamically the distorted image of every frame is proportional to the amount of disturbance in a random manner, while the "mean" reflection should remain an un-distorted image which we might get from a perfectly smooth surface. Clearly, a reflected light without distortion on the still water surface is straight so that we could draw a straight line in the hand-drawn reflected light as the *base skeleton* shown in the vertical dash line in Figure IV.23. Mathematically, the base skeleton can be defined by two points, a start point $RfStartP$ and an end point $RfEndP$, specified by

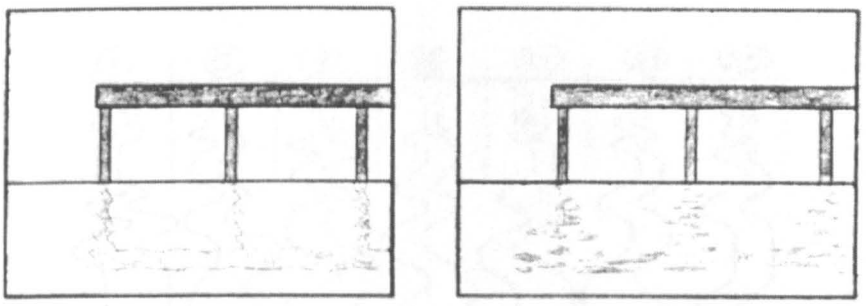


Figure IV.21: The reflection of a rougher water surface

the user according to the scene. In the case of a mobile base skeleton being required, the two points can be expressed as $RfStartP(t)$ and $RfEndP(t)$ where t is time.

5.2 Dynamic Skeleton

Taking the base skeleton as a reference, we mark the middle point between the peak of one side curve and the trough of the other side curve and draw short lines between those marks and the base skeleton to form the *dynamic skeleton* as shown in horizontal lines in Figure IV.23. Modelling of the dynamic skeleton is then concerned with positions and lengths of those horizontal lines.

Positions of these short lines are determined by the following formula:

$$\begin{aligned} StLineP_i(t) = RfStartP(t) + u(RfEndP(t) - RfStartP(t)) \\ + rnd(r)V(StLineP),\ (i = 0, 1...StN) \end{aligned} \tag{IV.2}$$

where StN is the number of short lines attached to the base skeleton. For a given length of the base skeleton, a bigger StN would simulate a more violent water surface and a smaller StN would emulate a more still water surface. $u \in [0, 1]$ is a linear interpolation parameter defined along the base skeleton when $u = 0$ corresponds to $RfStartP(t)$ and $u = 1$ corresponds to $RfEndP(t)$. $rnd(r)$ is a random variable with variance $V(StLineP)$ for positional perturbation.

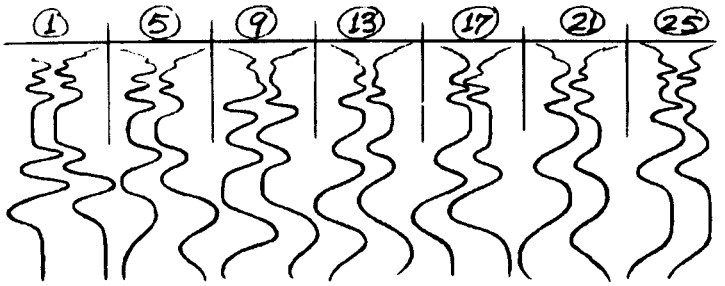


Figure IV.22: A cycle of a bright light

The lengths of short lines are determined by

$$StLG_i(t) = StLGM + u \cdot rnd(r)V(StLG),\ (i = 0, 1, ...StN). \qquad \text{(IV.3)}$$

where $u \in [1, 0]$ is the same interpolation parameter as used in IV.2, $StLGM$ is the minimum length of the short lines specified in advance, $rnd(r)$ is a random component with variance $V(StLG)$ added to make irregular variance in lengths of the horizontal skeletons. Both $StLGM$ and $V(StLG)$ are specified by the user according to the scene.

Figure IV.23 shows that the short lines are usually positioned on each side alternatively of the base skeleton but occasionally positioned on the same side. In the model we set a flag of orientation $StOrt$ using a stochastic model when a positive flag orients the short line to the left side while a negative flag orients the short line to the right side of the base skeleton.

5.3 Shaping

Figure IV.23 shows that the shape of a reflected light drawn by two side curves displays a reflected band with its width increasing from $RfStarP$ to $RfEndP$. Provided with $StLineP_i(t)$, $StLG_i(t)$ and the orientation information associated with short lines, we first calculate the coordinates corresponding to the terminal points of two side horizontal skeletons which are

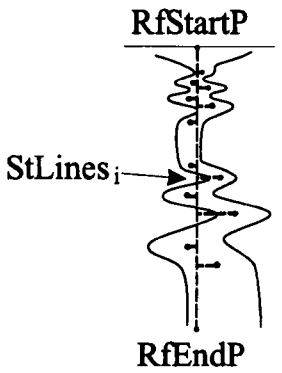


Figure IV.23: Base skeleton and dynamic skeleton

then interpolated by a spline to get a central skeleton. Next we calculate the two side curves $RfLEftBD(t)$ and $RfRightBD(t)$ according to a simple rule (the distance between the central skeleton and two side curves increases from $RfStart$ to $RfEndP$ that draw the reflection shape).

5.4 Colouring

Colour of the reflection is defined according to the colour of the reflected light $LtColor$. In order to achieve more realistic effects, we blend $LtColor$ with the colour of the water surface $WtColor$ along the skeleton by the following interpolation:

$$RfColor(u) = LitColor + cos(u\pi/2)(WtColor - LtColor)$$

where $u \in [0, 1]$ is the same interpolation parameter as used in IV.2 and IV.3.

5.5 Structure of the Model

The structure of the model can be expressed by the following:

Specify initial parameters:

1. $RfStartP(t)$.
2. $RfEndP(t)$.
3. $StLGM$.
4. StN.

For each frame t:

1. Use $RfStarP(t)$ and $RfEndP(t)$ to generate a base skeleton.
2. Set initial flag $StOrt$ randomly to be positive or negative.
3. Set i=0:
4. Calculate the interpolation parameter $u = i/StN$.
5. Calculate positions of short lines $StLineP_i(t)$.
6. Calculate lengths of short lines $StLG_i(t)$.
7. Position a short line according to the flag value.
8. Vary the flag $StOrt$ value stochastically.
9. Repeat steps 4, 5, 6, 7, 8 until i reaches StN.
10. Interpolate coordinates of terminal points of short lines to get a central skeleton.
11. Calculate two side curves $RfLeftBD(t)$ and $RfRightBD(t)$ using a simple rule.
12. Calculate rendering colour $RfColor(u)$ to colour the reflection surrounded by $RfLeftBD(t)$ and $RfRightBD(t)$.

End (of each frame)

5.6 Result

Figure IV.24 shows a frame of reflections animated in 3D. In this example, the base skeleton is specified in 3D by reversing the lights to be reflected with respect to the water surface while the dynamic skeletons associated with the 3D base skeleton are modelled parallel to both the vertical face of the embankment and water surface. They are then projected onto 2D and fed into the model described above to generate the reflections of the lights.

Figure IV.24: Reflections generated by the model

6 Summary

In this section we described a computer model of reflections on a relative still water surface. In regard to modelling reflections on a rippling water surface as shown in Figure IV.21, a possible approach would be distributing waves such as used in the shimmering model or some other stylised patterns around the base skeleton framing the structure of the objects to be reflected.

Chapter V

Fire and Smoke

This chapter presents models of fire and smoke. In the fire model we show how to devise a skeleton framework for generating animation sequences which match the hand-drawn series, and in particular how to match the flame orientations, shapes, and the connections curves, as required by the simulated style, between them. The parameters associated with these skeletons, flame types and connection curves are brought together into a matrix tableau for the particular representation of the fire base in the model. The model for the flames at the top of the fire are made up from three simple sub-models. By stochastically varying the parameters the model can generate plausible looking sequences of animated fire and include the effects of wind straightforwardly. In the smoke model, we show how to abstract a skeleton for generating smoke animation into a mathematical expression. Based upon the skeleton two sub-rendering models are used to produce smoke with looks of a smoke skein and smoke puffs. Relevant examples are given simultaneously in the models of both fire and smoke.

1 Fire

In cartoon animation, the simplest fire effect is that of a candle flame. A candle flame moves very little, unless it is fanned by a draft from an open door or window, so it can be quickly produced. Basically, there need only be three key positions of the flame, each moving only a little from the other, as shown in Figure V.1 [Whi86].

There are then be several in-betweens to produce a slow, almost imper-

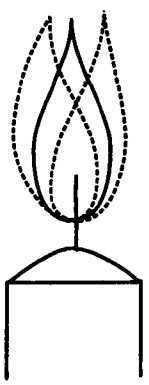


Figure V.1: Candle flame

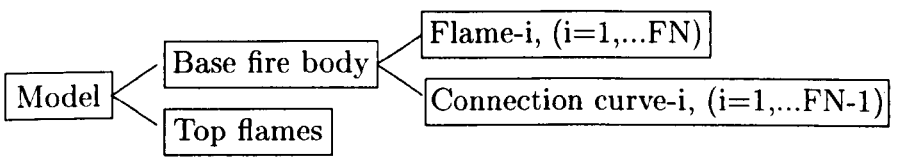
ceptible, movement. In cartoon animation, it is usually necessary to add a bright glow that back-lighting produces to make luminous fire, rather than the dead look of painted flames. The animation for a raging fire must be more dynamic, as shown in Figure V.2 [Whi86]: a fire flicks erratically, and the animation should reflect this.

For a large body of fire, the movements of flames are governed by the movements of air current above the fire. The hottest part of a fire is in the centre and above this the hot air rises. As it rises it is replaced by colder air rushing inward from the sides. This air in its turn is heated and rises and so the process is continuous. This flow of air usually gives a roughly conical shape to the flames, with a succession of indentations representing eddies of cold air, starting at the base of the fire and moving inwards and upwards. See Figure V.3 [Har81].

In this section we describe a fire model [YP96a] which aims at achieving a cartoon fire shown in Figure V.3. At present, the way of dealing with this problem in a computer-aided animation system like ANIMO is to scan the hand drawn fire cycles into the system, which at best only preserves the level of quality of hand drawn animation. With the computer model, however, we can introduce stochastic controls with which the model can be made to generate different frames all the time. Thus we can avoid using repeated cycles and as a result improve the quality of the effect over hand drawn animation. Additionally we can simulate the effects of wind on the model.

Figure V.2: Dynamic candle flame

Our model can be expressed as a hierarchical structure:



where FN is the number of flames arising out of the body of the fire. The flames from the fire body are controlled by position skeletons and shape skeletons. Indices here have been assigned from left-to-right in the fire body. We construct a matrix out of all the parameters we have associated with the skeletons to represent the model of the base fire body. Top flames are controlled by another model which is related to the base fire model by additional parameters.

In this section, we first describe how to construct the fire body model and the model for flames coming out of the top of the fire. Next, we analyse the parameters of the matrix in which we can control stochastically. Finally, we describe how to generate wind effect based on the foregoing model, and show examples of the relevant cases.

1.1 Fire flame model

In Figure V.3 we see that the fire body is composed out of several flames of different shapes which form a fan connected by curves of a specific nature.

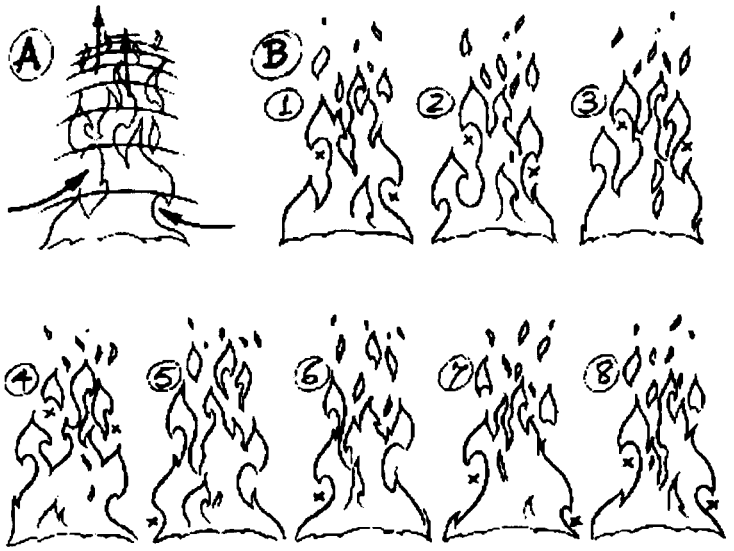


Figure V.3: A hand drawn fire cycle

From this we can separate out the fire body into the following parts:

1. Flames positions,
2. Flame shapes,
3. Connection curves.

For each part we define matching parameters which are used to build the fire body.

1.2 Flame Position Skeleton

The flame position skeleton is used to control flame positions, and is composed of a number of vectors, originating from the centre of the fire base and pointing to the positions of the flames. The end-points are derived from guide lines drawn in Figure V.3(a) which govern the flame movements. Since

the movements of air currents play an important role in the movements of the flames, especially those at the sides, it is important to ensure these flames are positioned as indicated by the crosses in Figure V.3(b).

1.3 Flame Skeleton

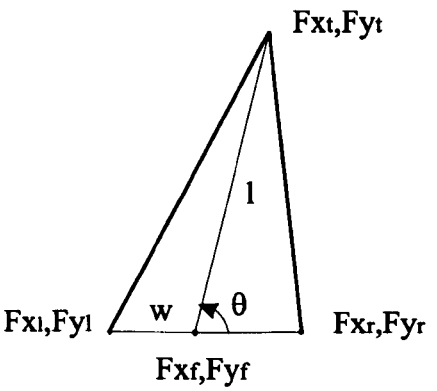


Figure V.4: Flame skeleton

The flame shapes shown in Figure V.3, suggest that it is natural to use a triangle as the skeleton of the flame, as shown in Figure V.4. For each triangle we first define the coordinates Fx_f, Fy_f which serve as the reference point of the skeleton and determine the position of the flame. Then we define the parameters of length l, width w and the angle θ which are sufficient to define a triangle, see Figure V.4. It is easy from the figure to calculate the coordinates of the triangle vertices, i.e.

$$\begin{aligned} Fx_t &= Fx_f + l \cdot cos(\theta), \\ Fy_t &= Fy_f + l \cdot sin(\theta), \\ Fx_l &= Fx_f - w, \\ Fy_l &= Fy_f, \\ Fx_r &= Fx_f + w, \\ Fy_r &= Fy_f. \end{aligned} \tag{V.1}$$

where Fx_t, Fy_t refer to the coordinates of the top vertex, Fx_l, Fy_l and Fx_r, Fy_r refer to that of the left and right vertices respectively.

1.3.1 Skeleton Types

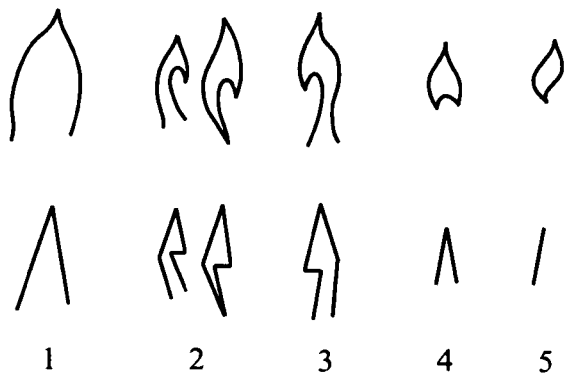


Figure V.5: Skeleton types

Analysis of Figure V.3 shows that flames have varying shapes, so it is necessary to define different types of skeletons to match different flame shapes. In our model we use $FlameTP = i, (i = 1, ...5)$ to define five types of skeletons which are shown in Figure V.5. The first three are used for the fire body and the last two are used for individual flames at the top of the fire. In Figure V.5 there are two skeletons associated with $FlameTP$ 2. This is because the two are similar and we use one parameter to define their types and an additional flag *open* to indicate whether the skeleton keeps open or not. From the figure we can see that $FlameTP$ 2 and 3, 1 and 4 are also similar, but we define them differently because they represent different shapes and their shapes are determined according to their $FlameTP$ values. From the figure it is not difficult to calculate the coordinates of the skeleton.

1.3.2 Skeleton Symmetry

Flames on the left and right side can be derived from the same skeleton if symmetry to the central vertical line on the fire body is taken into account. Here we define $FlameTP$ as positive when it corresponds to the left side and negative when for the right side, to carry the symmetry information.

1.4 Flame Shape

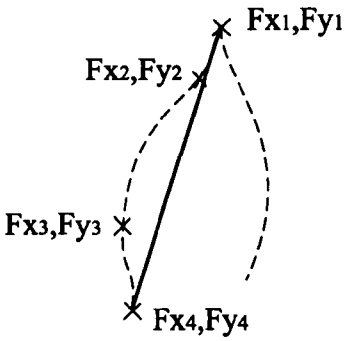


Figure V.6: Flame shape

From the skeleton it is quite easy to generate flame shapes. We take $FlameTP$ 1 as an example, as in Figure V.6. Using the coordinates of the two extreme points Fx_1, Fy_1, Fx_4, Fy_4 which coincide with Fxt, Fyt and Fxl, Fyl in Figure V.4 respectively, we can calculate the other two intermediate points Fx_2, Fy_2, Fx_3, Fy_3:

$$\begin{aligned} fap_1 &= 0.2, \\ Fx_2 &= Fx_1 + fa_1(Fx_4 - Fx_1) + \Delta, \\ Fy_2 &= Fy_1 + fa_1(Fy_4 - Fy_1), \\ fap_2 &= 0.45, \\ Fx_3 &= Fx_1 + fa_2(Fx_4 - Fx_1) + \Delta, \\ Fy_3 &= Fy_1 + fa_2(Fy_4 - Fy_1). \end{aligned} \tag{V.2}$$

where fap_2 and fap_3 are position parameters which control the positions of the intermediate points, Δ and Δ are parameters related to the width of the skeleton and determine the shape of the flame. We take these points as control points and then interpolate them by a spline to get the final curve that draws flame shape. Similarly we can generate flame shapes corresponding to the right side of the skeleton.

From the foregoing process we can see the shape information is determined by control points which are related to skeletons derived from simple models. These models are accessed through parameter $FlameTP$, and because this

parameter is closely related to the flame shape we use $FlameTP$ rather than $SkeletonTP$ in section 1.3.1.

The foregoing method can generate a perfect, symmetric flame shape, but in Figure V.3 we see that flames are not strictly symmetric, for example the first flame in frame 1, where the left side curve is longer and the right side curve is shorter. To deal with this problem we encode another parameter with the $FlameTP$, in the form, for example $FlameTP = 15$, where the first number 1 indicates the flame type as previously defined, and the second serves as a control parameter which becomes 0.5 after decoding and controls the length of the flame side curve.

1.5 Connection of Flames

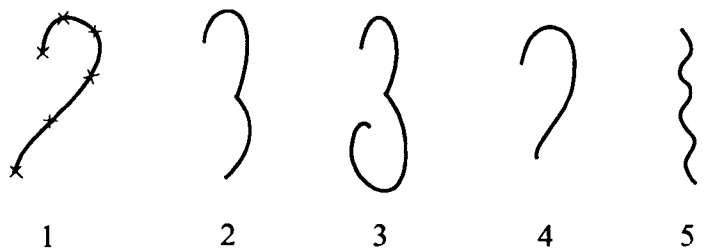


Figure V.7: Connection types

Once flames are generated, we need to connect them in series to complete the drawing of the fire body. After the flame skeletons are placed in their proper positions, we note the coordinates of two terminal points of the flame skeleton Fx_l, Fy_l, Fx_r, Fy_r, which are now used for connection. From Figure V.3 we can see that connection curves also have different shapes. Similar to flame types, we also define connection curve types by $ConnectTP$. In our model we define five connection types and their corresponding shapes are shown in Figure V.7.

Here we describe only how to generate connection curves of type 1. Similarly to generating the flame shape, we introduce four points between the two extreme points to control the shape of the connection curve, as indicated by the cross in Figure V.7. From the figure we can calculate their coordinates as follows:

$$
\begin{aligned}
Fx_3 &= Fx_1 + lft \cdot wd, \\
Fy_3 &= Fy_1, \\
Fx_2 &= Fx_1 + 0.5(Fx_3 - Fx_1), \\
Fy_2 &= Fy_1 + 0.5wd, \\
Fx_4 &= Fx_3 + fap1(Fx_6 - Fx_3) + lft \cdot 0.2 \cdot wd, \\
Fy_4 &= Fy_3 + fap1(Fy_6 - Fy_3), \\
Fx_5 &= Fx_3 + fap2(Fx_6 - Fx_3) - lft \cdot 0.3 \cdot wd, \\
Fy_5 &= Fy_3 + fap2(Fy_6 - Fy_3),
\end{aligned}
\tag{V.3}
$$

where Fx_1, Fy_1 and Fx_6, Fy_6 coincide with the right and left terminal points of the two successive flames respectively, $fap1 = 0.2$ and $fap2 = 0.5$ are positional parameters which control positions of the relevant points, wd is a parameter related to the width of the skeleton, lft is a parameter related to the orientation of the skeleton with a value of 1 or -1 corresponding to the sign of the parameter $FlameTP$. We take these points as control points and then interpolate them with a spline to get a connection curve of type 1.

Like flame symmetry, the connection curves can also be used symmetrically, and we use the same method to indicate the direction of the connection curve, i.e. define $ConnectTP$ as positive on the left and negative on the right.

With the foregoing method we can generate all types of connection curves but one, which needs to be dealt with differently. Looking at connection curves between the third and fourth flame (the second and third flame overlap each other partially) in frame 1, this in fact is composed of two connection curves of types defined previously. To generate this connection curve we set $ConnectTP$ to 9. When the model interprets this it generates another point Fx_d, Fy_d between the right extreme point of the third flame and the left extreme point of the fourth flame and below both of them, and we then take this point and the right extreme point of the third flame as a pair to generate the first connection curve (of type 1 in the case of frame 1), finally we take this point and the left extreme point of the fourth flame as a pair to generate the second connection curve (of type 5 in the case of frame 1).

1.6 Construction of the Matrix

The foregoing described how to generate flame and connection curve shapes. We can assign parameters to each step which can be expressed as a matrix with elements $p_{i,j} = P_{i,j}(FlameTP, x, y, l, w, \theta, ConnectTP)$ which represents the base fire model, where i represents ith frame which is related to

time t, and j represents the jth frame in the fire body when indexed from the left side in one frame.

1.7 Top Flames

Modelling top flames is more difficult than it looks. Flames at the top of the fire body consist mainly of flames of type 4 and 5, and occasionally of type 2 where some flames in the middle of the fire body are small. The movements of those flames are also governed by the guide-lines shown in Figure V.3(a). Here we divide top flames into three parts: left, middle and right, then we detect the lowest flames on the base fire body (they are in the different frames for the left, middle and right parts), take them as the reference positions, and then generate corresponding vertical positions of the flames which are proportional to time t. Sizes and widths of the flames are controlled with a stochastic function, i.e. they decrease with time t and small random components are added to them.

1.8 Structure of the Model

The structure of the model can be expressed by the following:

Step 1: Base fire body

1. Specify the matrix $MP(i,j) = p_{i,j}$.
2. Read parameters $Fx_{i,j}, Fy_{i,j}, l_{i,j}, w_{i,j}, \theta_{i,j}, lft_{i,j}, FlameTP_{i,j}$ from the matrix.
3. Decode $FlameTP_{i,j}$ to get direction flag and original $FlameTP_{i,j}$ value.
4. Generate flame shape according to corresponding $FlameTP_{i,j}$ value.
5. Save coordinates of two extreme points of the flame skeleton Fxl_j, Fyl_j and Fxr_j, Fyr_j.
6. Read Fxr_{j-1}, Fyr_{j-1} and Fxl_j, Fyl_j.
7. Read $ConnectTP_{i,j}$ from the matrix.
8. Decode $ConectTP_{i,j}$ to get direction flag and original $ConnectTP_{i,j}$ value.
9. If $ConnectTP_{i,j}$ does not equal to 9 then generate connection curve according to corresponding $ConnectTP_{i,j}$ value.

10. If $ConnectTP_{i,j}$ equals to 9, then generate another point Fxd, Fyd, and then generate connection curves between Fxr_{j-1}, Fyl_{j-1} and Fxd, Fyd, Fxd, Fyd and Fxl_j, Fyl_j respectively.

Step 2: Top flames

1. Detect the lowest positions of the flames on the base fire body for the left, middle and right top flames, take down their coordinates $xlmin, ylmin$, $xmmin, ymmin$, $xrmin, yrmin$ as reference positions.
2. Generate vertical positions of top flames by linear interpolation between their corresponding reference positions and the highest position $ymax$ which is specified in advance ($ymax$ is the maximum value for the fire amplitude from the base to the top flames).
3. Repeat 1 and 2 in this step to generate second layer of top flames.

1.9 Analysis of Parameters

We are able to adjust the parameters defined above to change the flame shapes until the they are comparable to hand-drawn shapes. This means that the parameters we defined for the model are sufficient to control the flame shapes.

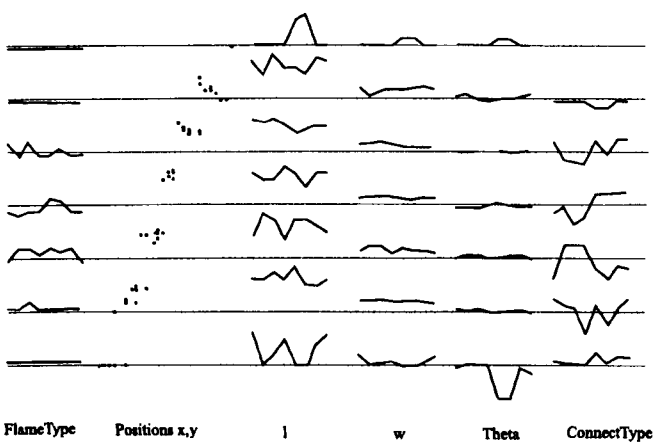


Figure V.8: Parameters associated with the model

Figure V.8 shows parameters associated with the model. The horizontal axis corresponds to time t and the vertical axis corresponds to amplitudes of the parameters for flames counted form the left to the right side in the base fire body. Horizontal lines represent zero value of amplitude for most of parameters of each flame with an exception for the parameter θ where they represent 0.5π radians. It is clear from the figure that these parameters vary irregularly with time t but their values are limited to certain ranges, only in the first, sixth and seventh flames the corresponding $FlameTP$ always stays as 1 or -1. This means that flames situated at the two bottom sides of the base fire body always consist of the same type. Also the y coordinate for the first and seventh flame stays as zero, and the x coordinate plus the value of w are constant thus ensuring the two extreme points at the base of the fire body stick to the predefined points (in other word, these two points should always be the same regardless of the variation in width of the flames). Since fire involves random movements it is natural for us to add some random control into the model. With the exception of the first and seventh flames, almost all parameters for other flames could be controlled stochastically. Also it is possible to control some parameters stochastically in the previous formulae, for example, to calculate the coordinates of the skeleton (see Figure V.4), we can use the following formulae:

$$\begin{aligned} Fx_t &= Fx_f + (l + rnd(l))cos(\theta), \\ Fy_t &= Fy_f + (l + rnd(l))sin(\theta), \\ Fx_l &= Fx_f - (w + rnd(w)), \\ Fy_l &= Fy_f + rnd(y), \\ Fx_r &= Fx_f + (w + rnd(w)), \\ Fy_r &= Fy_f + rnd(y). \end{aligned} \tag{V.4}$$

where rnd is random function which controls corresponding parameters varying within a certain range. We should point out that not all parameters allow random control, particularly for those of the first, second, sixth and seventh flames. Because those flames play important role for animating fire effect, some parameters associated with those flames such as $FlameTP$ and $ConnectTP$ should stay the same. However, we still can vary their l, w and θ to some degree. With stochastic control of the relevant parameters the model can generate flames which change in shape all the time, we can thus avoid using repeated cycles and animate fire for as long as we wish.

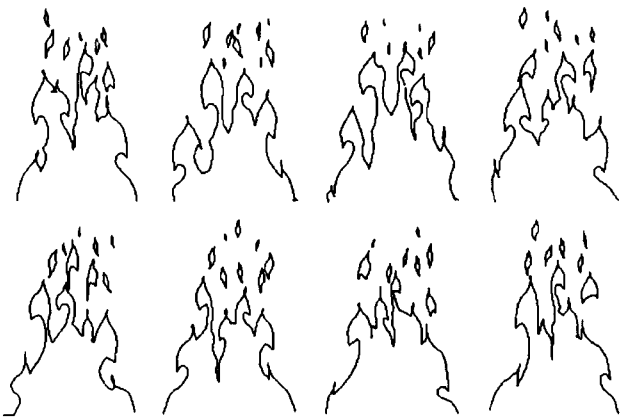


Figure V.9: A fire series generated by the model

1.10 Colouring

Since the hottest part of the fire is in the centre and the fire is surrounded by cold air at the sides, we use a colour gradient which varies from yellow in the hottest part to red representing the coolest part in the flame colours, and the visual effect seems satisfactory.

Figure V.9 shows a computer generated fire series and Figure V.10 shows a coloured frame of fire.

1.11 Simulation of Wind Effect

Simulation of the effects of wind on flames can make animation vivid and this can be achieved just by adding one parameter to our model. When a fire body is blown by wind from the side, flames are moved toward the other side but the bottom of the fire remains fixed, in effect flames look "bent". If we can generate a "bent" fire body then we can simulate the effect of wind. This is done by introducing an angle parameter $\Delta\theta$ which represents the angular amount changed compared with original "still" (i.e. no side wind) flames. The "bent" flames are derived in the following two steps:

Figure V.10: A coloured fire frame

1. Calculate the displacements for the reference points of the skeletons by the following formula,

$$Fx' = Fx + ymax \cdot tang(\Delta\theta) \cdot (1 - cos(fa \cdot 0.5 \cdot \pi)) \qquad \text{(V.5)}$$

where x, y are original position parameters specified in the matrix, and Fx' is the new parameter for Fx, $\Delta\theta$ is an angular value which controls the bending degree of the fire, the nonlinear coefficient $(1 - cos(fa * 0.5 * \pi))$ makes flame bends look more natural and $fa = y/ymax$,

2. Calculate the displacements of control points in a flame with the same formula as before, but with Fx and Fy replaced by the coordinates of the respective control points .

Figure V.11 shows the result of computer generated fire frames for wind effect. Since the degree of fire bending is controlled by $\Delta\theta$, this parameter can also be used as a interpolation variable to animate the process of fire moving from being in still air to being blown about by the wind. If, further, we modulate this parameter with a sinusoidal function in time t, then the model can generate the effect of a swaying fire. It should be pointed out that our model is devised from line drawing pictures rather than being physically based, so the simulation of wind effects is limited to small values of $\Delta\theta$. A

Figure V.11: Effect of wind on fire

serious distortion will arise if we allow $\Delta\theta$ to become too big. With this limitation our model can only simulate light winds but, even so, this is a significant improvement because it is so difficult for traditional animators to create this effect.

1.12 Summary

In this section we presented a cartoon fire model for 2D computer animation. The present model can be used directly for representing bonfires, torches etc, and the animated effect looks correct when we play back fire series generated by our model. We hope the model could be extended to some other situations such as objects burning, fire spreading *et al* and this is a topic for a future work.

2 Smoke

In traditional animation, smoke can be treated in many ways but the main timing problem is how to plan a repeat which does not appear too mechanical. One way of doing this is illustrated in Figure V.12(c) [Har81] (which we refer

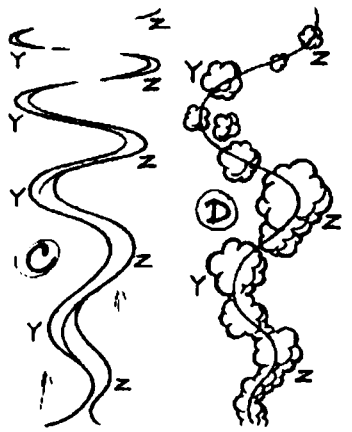
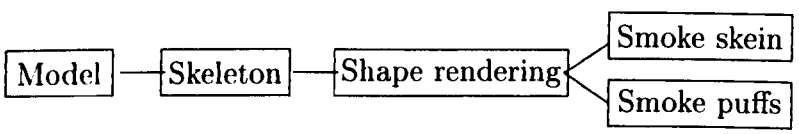


Figure V.12: Hand-drawn smoke

to as smoke skein). A variation on this basic idea is to animate puffs arranged on a wave pattern as shown in Figure V.12(D) (which we refer to as smoke puffs). These may remain as individual puffs or merge into one another to form an irregular column.

In this section we would like to develop the smoke model synthesising the cartoon smoke skein and smoke puffs as shown in Figure V.12 with the following structure :



We begin with abstracting the smoke skeleton into a mathematical expression and render the smoke in two fashions to get smoke skein and smoke puffs.

2.1 Skeleton

Figure V.12 suggests that the wave pattern of the smoke displays a sinusoidal shape with its amplitude and frequency increasing as smoke moves upward

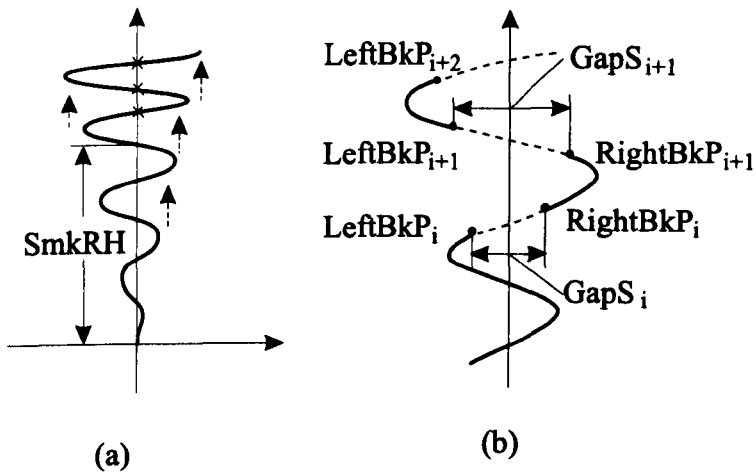


Figure V.13: Smoke skeleton

therefore we could use the following formula to model the skeleton emulating the wave pattern:

$$SmkSKLT_{u,t} = SmkAM(u)sin(\alpha(t) + \omega(u))$$

where $SmkSKLT_{u,t}$ is a point on the skeleton, u is a spatial variable and t is time. $SmkAM(u)$ is the amplitude and $\omega(u)$ is the radian frequency, both of them are linear functions of u and determine the static shape of the skeleton. $\alpha(t)$ is the phase, a linear function of t controls the dynamics of the skeleton that, as t increases, the skeleton moves upward as Figure V.13(a) indicated.

2.2 Shape Rendering

In Figure V.12, the feature of smoke (c) is well reflected by its skien shape while smoke (d) is characterised by a number of puffs. As the smoke moves upward, in smoke (c), the smoke skein becomes progressively thinner and breaks up into small pieces and then disappears as the result of dispersing. In smoke (d) puffs are drawn dense near the bottom and thin near the top of the smoke. The size of each puff increases and then decreases along the skeleton

with variations within certain a degree, again, as the result of dispersing.

2.2.1 Smoke Skein

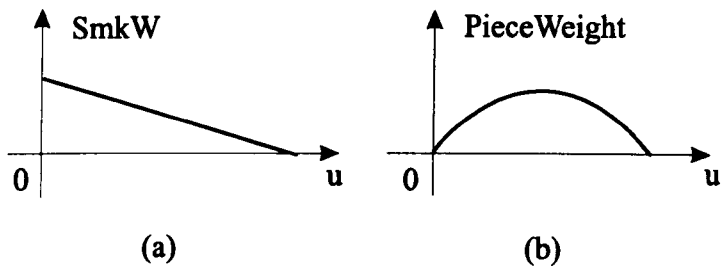


Figure V.14: Smoke shape rendering functions

The rendering of smoke skein involves following parts:

1. Detection of breaking points.
2. Determination of the gap size corresponding to those points.
3. Determination of two terminal points of gaps.
4. Shape rendering of broken skein pieces.
5. Colouring.

We proceed to describe them in more detail beginning with detecting points where the skeleton breaks up. Figure V.12 (c) suggests that breaking up happens at those points on the skeleton where the following conditions apply (see also Figure V.13(a)):

1. They are above a height *SmkRH* (drawn according to the scene).
2. They intersect with the vertical axis.

Once the breaking points are detected with above conditions, they are put into $BreakP_i, (i = 1, ..BrkN)$ where $BrkN$ is the number of breaking points detected. Figure V.12(c) also shows that the gap corresponding to each breaking point becomes wider and wider as the smoke skein goes upward. In the model we define the gap size $GapS_i, (i = 1, ..BrkN)$ as the function of the amplitude of the skeleton as well as the difference between the height of the current breaking point of interest and the height $SmkH$, i.e.

$$GapS_i = C_w \cdot SmkAM(u)(BreakP_i - SmkH), (i = 1, ..BrkN)$$

where C_w is a factor derived through experiment to tune $GapS_i$.

With $GapS_i$ we are able to determine the two terminal points of the gap $LeftBkP_i$ and $RightBkP_i$ with great ease. Those points between $LeftBkP_{i+1}$ and $LeftBkP_{i+2}$ or $RightBkP_i$ and $RightBkP_{i+1}$ form the skeleton of the corresponding piece of smoke skein (see Figure V.13(b)).

The shape of the smoke skein is rendered in two phases. In the first phase we choose to use a linear function $SmkW(u)$ defining the skeleton overall as shown in Figure V.14(a) to synthesis monotonous decrease in the width of the smoke skein. In the second phase, in order to characterise dispersing, we employ a shape weight of a sinusoidal form as shown in Figure V.14(b) to render the shape of each broken skein piece.

Since the hottest part of the smoke is at the bottom, we use a colour gradient which varies from background colour in the hottest part simulating the transparent effect and light gray representing the cool part in the smoke skein colours, and the visual result looks satisfactory.

2.2.2 Smoke Puffs

In smoke (d) each puff exhibits a form of a plum blossom, which can be modelled by a number of arcs with variations in position and size drawn along a base circle as shown in Figure V.15. The base circle may be thought as the skeleton of the puff which in turn controls the size of the puff.

Since smoke puffs are liable to be used for representing a billowing smoke effect, we choose to use colours of a certain range such as a gray scale or a brown scale to colour the area covered by each arc randomly in order to enhance the visual effect.

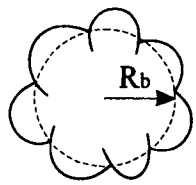


Figure V.15: Smoke puff skeleton

The number and sizes of puffs distributed along the skeleton are controlled by a simple stochastic model with the deterministic component defined by a sinusoidal function as shown in Figure V.14(b).

2.3 Model Structure

The structure of the model can be expressed by the following:

Step 1: Generate skeleton $SmkSKLT(u,t)$.

Step 2: Shape rendering:

(a) Smoke skein:

1. Detect breaking up points $BreakP_i$.
2. Calculate the gap size $GapS_i$ corresponding to $BreakP_i$.
3. Determine the two terminal points $LeftBkP_i$, $RightBkP_i$ of $GapS_i$.
4. Use the linear function $SmkW(u)$ to render the width of the smoke skein along the skeleton.
5. Use the weight to render each of smoke piece respectively.

(b) Smoke puffs:

1. Assign the spatial distribution of the puffs over the skeleton.
2. Determine the number and sizes of the base circles stochastically.
3. For each base circle, determine the number and sizes of arcs stochastically.
4. Generate smoke puffs.

5. Determine positions of smoke puffs stochastically.
6. Place smoke puffs accordingly.

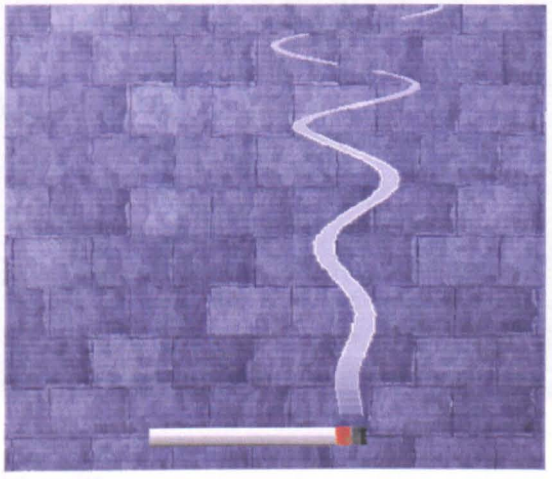


Figure V.16: Smoke skein

Figure V.16 and Figure V.17 show the examples of our model for smoke skein and smoke puffs respectively in which the skeleton of the smoke skein is specified in 2D and that of the smoke puffs is in 3D.

3 Conclusion

In this section we described models of smoke skein and smoke puffs. Smoke skein can be directly used to represent cigarette smoke or steam coming from a coffee cup in a room, while smoke puffs are more suitable for representing smoke coming from the chimney of a house or steam boat, etc. In some cases, the animator draws a really big column of smoke rising from a fire forming a mushroom or smoke ring, a doughnut-shaped eddy with a hole in the middle as shown in Figure V.18 [Har81] and this is a topic for a future work.

Figure V.17: Smoke puffs

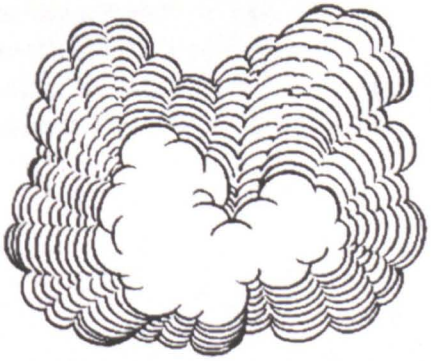


Figure V.18: Billowing smoke

Chapter VI

Rain and Snow

Rain and snow in animation not only simulate the natural phenomena but also express the mood. For instance, a miserable mood can be achieved by slow vertically moving rain, and a more violent mood is expressed by a fast rain with great tilt from the horizontal. This chapter presents models of rain and snow. In the rain model we begin with the light rain model by showing how to use our brush model described in Appendix to simulate the appearance of hand-drawn light rain effect, in particular how to devise and distribute individual skeletons with associated brushes corresponding to their hand-drawn counterparts. Next, we proceed to describe the heavy rain model by showing how to construct multiple levels of rain drops moving in the air and drops hitting the ground, in particular how to design drop shapes and assign dynamic constraints to them respectively. Finally, we describe a snow model with a similar multiple level structure but different dynamic attributes assigned to snow flakes. Examples of the relevant cases are given simultaneously.

1 Rain

In hand drawn animation one problem with animating rain is that it can appear very mechanical. Although rain actually falls in more or less parallel lines, it must be animated with a more random slope if a realistic effect is to be achieved. It is best to time it moving quickly down. If it is drawn on more than one level, the distant rain should move more slowly, to give depth. Foreground rain should cross the screen in about six frames while more distant rain should move progressively slower.

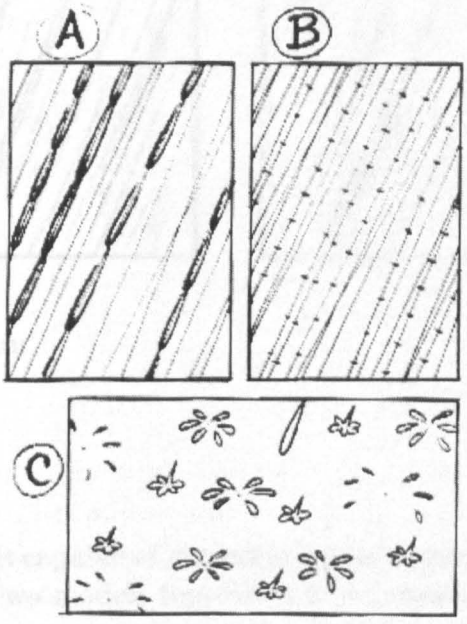


Figure VI.1: Rain

(a) Tracks for foreground rain cycle, (b) Tracks for more distant rain
(c) Cycle of raindrops hitting the ground

Individual drops are drawn as straight lines with consecutive drawings overlapping each other slightly. Rain almost certainly needs single frame animation for a realistic effect. It also needs a fairly long repeat if it is not to appear too mechanical–24 frames at least.

For really heavy rain, the effect is enhanced by animating a cycle of drops hitting the ground. These can be quite random and unrelated to the falling rain. Each drop should animate out in about six frames. See Figure VI.1 [Har81].

Another effect to animate light rain is to draw a number of diagonal lines across the screen, each set on a separate cel, these can then be doped at random, so there is no repetitive pattern, as shown in Figure VI.2 [Whi86].

Obviously, the rain effect drawing involves both deterministic components such as drops moving down and random components evident in drop positions and tracks. In this section we proceed to describe the light rain

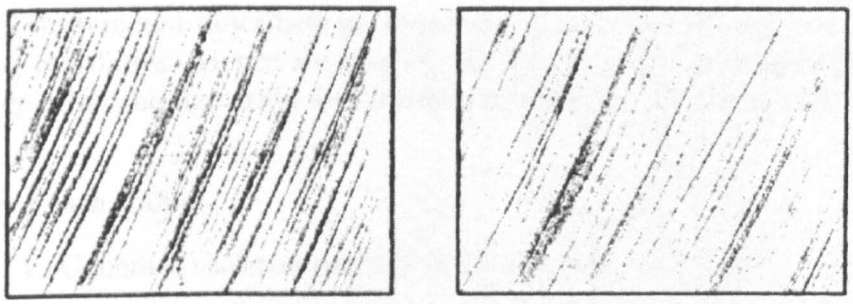
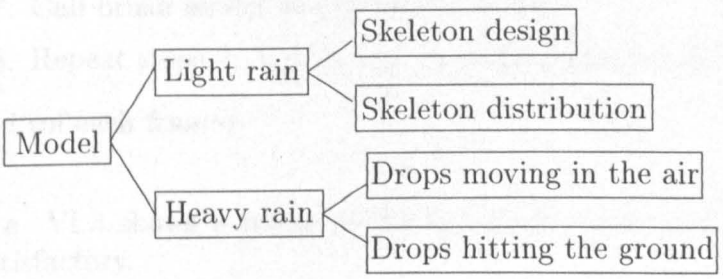


Figure VI.2: Light rain

and heavy rain models capable of generating rain sequences which match the hand-drawn series. Two models together can be expressed by the following hierarchical structure:



The light rain model consists simply of one level in which we simulate random doping of separate cels in hand drawn animation by using a simple model. The heavy rain model comprises two sub-models, rain drops moving in the the air consisting of three levels (front, middle and back levels, aiming at showing the the depth as doing in hand-drawn series) and drops hitting the ground.

2 Light Rain

Modelling of light rain shown in Figure VI.2 is concerned with the design and distribution of individual lines. Each line drawn in Figure VI.2 is characterised by its length, orientation and width which, from the point of view

of our brush model described in Appendix, can be readily taken as brush skeleton attributes. Hence we employ our brush model to synthesise these lines depicting the light rain and implementation details are as follows:

For each frame:

1. Generate skeleton number stochastically.
2. Generate skeleton attributes stochastically:
   a. Angle $LtAng$.
   b. Length $LtLG$.
   c. Position $LtPOS$.
3. Generate one skeleton which is slightly bent.
4. Choose curve (b) in Figure A.2 (see page 136) as $BWLeft(t)$ and $BWRight(t)$.
5. Define $BDensity(t)$ proportional to the brush width attribute.
6. Define $BColor(t)$ with a simple stochastic model.
7. Call brush model to generate a stroke.
8. Repeat steps 2, 3, 4, 5, 6, 7 until the number of skeletons is met.

End (of each frame).

Figure VI.3 shows a frame of the light rain animation and the result seems satisfactory.

3 Heavy Rain

3.1 Drops Moving in the Air

Tracks of drops moving in the air resemble lines of the light rain in skeleton attributes and distribution hence we take advantage of a partial modelling procedure of the light rain model to determine them.

Individual moving drops, however, differ dramatically from the light rain in shape in that they are modelled as straight lines rather than brush strokes. Take into account that foreground moving drops cross the screen in six frames and each drop in connective drawings overlaps slightly to avoid possible

Figure VI.3: Light rain generated by the model

strobing, we model the length of the rain drop in the front level $FrtLG = \beta ScrH/6$, where $ScrH$ is screen height (which is actually the frame height) and the factor $\beta = 1.1$ ensures overlapping of each drop in connective drawings by 10%. While drops of medium and back levels represent more distant rain, their drop lengths $MidLG$ and $BckLG$ are defined progressively shorter that $FrtLG > MidLG > BckLG$ and through our experience we have found that $MidLG = 0.55FrtLG$ and $BckLG = 0.35FrtLG$ could achieve a satisfactory result.

Relevant speeds of moving drops, namely $FrtSPD$, $MidSPD$ and $BckSPD$, are assigned to three levels respectively: $FrtSPD = 2MidSPD = 4BckSPD$, and as a result, the timing of moving drops becomes slower and slower from front to back levels through which we enhance the more distant rain effect. Appearance or disappearance of a signal moving drop is controlled by a positional controller using a boolean function defined on a time axis which is slid downward along the moving drop tracks, if the value of the controller is 1 and the current position of interest falls into the display window, then a rain drop is drawn, otherwise not.

3.2 Drops Hitting the Ground

3.2.1 Modelling of Individual Drops

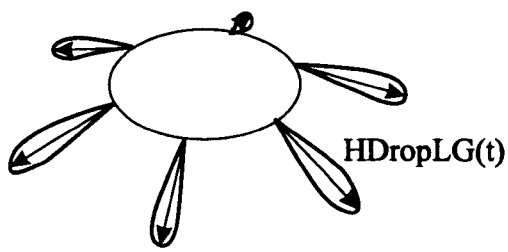


Figure VI.4: Skeleton and shape of the drop hitting the ground

We have mentioned in section 1 of this chapter that animating out each drop hitting the ground takes about six frames (see Figure VI.1). In more detail, the first frame corresponds to the moment that the drop is just hitting the ground and going to split while having a "tail" in the centre. In the consequent frames, the drop splits into six surrounding drops which then become longer, then shorter and thinner, and finally disappear.

In the model we employ a time varying ellipse (which becomes bigger with time) to control positions of the splitting drops, as shown in Figure VI.4. The ratio of the major and minor axes of the ellipse controls the perspective of the drops. Upon the ellipse we choose six points nearly equally spaced and calculate their angles with respect to the centre of the ellipse from which we define six vectors with their lengths varying over time as the skeletons of splitting drops, the length of each vector is modelled by $HDropLG(t) = L_0 \cdot sin(i\pi/6)$, $(i = 1, 2, ...6)$, where L_0 is the longest length specified by the user that $HDropLG(t)$ can reach, i is a relative temporal index associated with time t in the life controller which will be described shortly. The "skin" of the splitting drop is put onto the vector skeleton in a similar fashion to generating a drop shape described in section 2.4 of Chapter IV.

3.2.2 Distribution of Hitting Drops

Since the rain drops hitting the ground are quite random and unrelated to the falling rain drops, they are distributed independently in the area of interest.

3.2.3 Life Controller

Each hitting drop exists in six frames, i.e. it has a "life" of six frames. In the model we define a life controller to control whether a drop becomes "alive" or "dead", which is implemented as a function of a sawtooth form with the number 1, 2, 3, 4, 5, 6 corresponding to the "alive" period and 0 corresponding to the "dead" period. The duration of the "dead" period is controlled stochastically.

Actually this life controller functions in a similar manner to a "particle" attribute described in [Ree83] for it follows the procedure of being born, changing and death. Our hitting drop model defers from the particle in that it has a structure hence we may refer to it as a *structural particle.*

3.2.4 Structure of the Model

The structure of the 2D heavy rain model can be expressed by the following:

Initialise the model:

1. Generate three levels of moving drop track skeleton attributes stochastically:
   a. Number: $FrtN$, $MidN$, $BckN$.
   b. Angle: $FrtAng$, $MidAng$, $BckAng$.
   c. Position: $FrtPOS$, $MidPOS$, $BckPOS$.
   d. Length: LG.
2. Generate three levels of track skeletons.
3. Specify moving drop lengths $FrtLG$, $MidLG$, $BckLG$.
4. Specify moving drop speeds $FrtSPD$, $MidSPD$, $BckSPD$.
5. Initialise randomly positional controller for individual moving drops of three levels.
6. Determine distribution of drops hitting the ground.

7. Generate life controller of drops hitting the ground.

For each frame t:

For each level:

For each skeleton:

1. Read a value from the positional controller.
2. If the value is 1 then generate a moving drop with corresponding length and position (speed).

End (of each skeleton).

End (of each level).

For each drop hitting the ground (constrained by distribution):

1. Read a value from the life controller.
2. If the value is great than 0 then generate corresponding splitting drops.

End (of each drop).

End (of each frame).

Implementation of a 3D rain model is relatively simple in comparison with the implementation of its 2D counterpart because we need to define only one set of parameters such as moving drop length and speed thus avoiding using parameters associated with three levels. In the 3D model we distribute moving drop positions on a plane which is above and parallel to the $X - Z$ plane, the height and size of the plane are specified according to the scene. Moving drop skeletons are positioned vertically with perturbation in orientations in the 3D space of interest. Positions of drop hitting the ground are distributed on the $X - Z$ plane where we define the scene ground. With the positional controller of moving drops and the life controller of hitting drops, we can achieve heavy cartoon rain effects in 3D using a similar procedure.

3.3 Result

We have implemented both 2D and 3D heavy rain models and there is no obvious difference in the visual effect between them if we keep the virtual camera still to view the rain fall from the side in the 3D model. Figure VI.5 shows an example of a series of 3D rain. In the animation, the camera movement is designed standing still first, followed by a rotating and then zooming closer.

Figure VI.5: Heavy rain generated by the model

3.4 Summary

In this section we presented a rain model for computer animation which is able to generate the cartoon light and heavy rain effects. Although we built an individual light rain model, the light rain effect is also achievable through the heavy rain model, if we remove moving drops at the front level and drops hitting the ground, with a different visual appearance compared with that displayed in the former light rain model. As for timing control of moving drops, in contrast with hand drawn animation in which changing timing may require the entire rain series be re-drawn, we can simply deal with it by performing modifications on speed parameters $FrtSPD$, $MidSPD$ and $BckSPD$ in the 2D rain model and one speed parameter in the 3D rain model. Furthermore, with skeleton angular attribute modification, we are able to simulate the greater tilt from the horizontal for a more violent mood.

4 Snow

Gently falling snow drifts in wavy lines and needs even longer cycles than rain to avoid the audience noticing the same flake flowing itself down the same track. Two seconds may be too short if the repeat is run more than

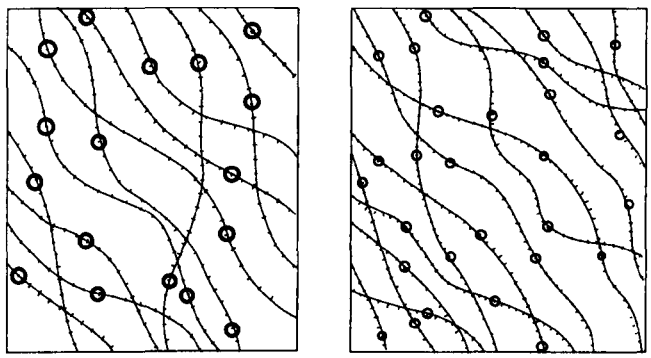


Figure VI.6: Snow

a few times. A foreground flake may take about two seconds to cross the screen, but of course may take less.

To give depth to the snowfall, at least three different sizes of flake are usually needed, the smaller more distant ones travelling more slowly than those in the foreground. Distant snow is not generally animated out of the bottom of the screen but fizzles out in a random way somewhere on the screen. The exact point where it disappears depends on the background on which it is used. See Figure VI.6 [Har81].

The snow model we present in this section resembles the heavy rain model in structure and controlling mechanism therefore we focus on describing the shape designing of snow track skeletons only.

4.1 Snow Track Skeleton

Figure VI.6 suggests that we may first use a diagonal line to construct a base skeleton and take 3 to 5 points randomly nearly equally spaced on the base skeleton from which we draw vectors perpendicular to the base skeleton pointing alternately to the opposite directions with variations in length as shown in Figure VI.7. Next, we take the two end points of the base skeleton together with those arrowheads of vectors as control points and interpolate them with a spline to get the final wavy snow track.

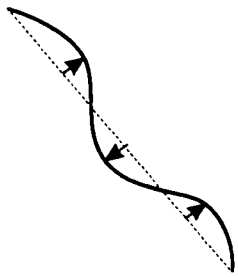


Figure VI.7: Snow track skeleton

4.2 Snow Flakes

Snow flakes can be simply approximated by circles with variations in size and the mean values of the circle sizes corresponding to three levels $FrtSZ$, $MidSZ$, $BckSZ$ are specified according to the scene, where $FrtSZ > MidSZ > BckSZ$. Similarly, speeds associated with snow flakes at three levels are specified that $FrtSPD > MidSPD > BckSPD$.

4.3 Structure of the Snow Model

The structure of the 2D snow model can be expressed by the following:

Initialise the model:

1. Generate three levels of snow track skeleton attributes stochastically:
   a. Number: $FrtN$, $MidN$, $BckN$.
   b. Angle: $FrtAng$, $MidAng$, $BckAng$.
   c. Position: $FrtPOS$, $MidPOS$, $BckPOS$.
   d. Length: LG.
   e. Vector length: $FrtVLG$, $MidVLG$, $BckVLG$.
2. Generate three levels of snow track skeletons.
3. Specify flake sizes $FrtSZ$, $MidSZ$, $BckSZ$.
4. Specify moving flake speeds $FrtSPD$, $MidSPD$, $BckSPD$.

5. Initialise randomly positional controller for each individual moving drops of three levels.

For each frame t:

For each level:

For each skeleton:

1. Read a value from positional controller.
2. If the value is 1 then generate a moving flake according to corresponding size and position (speed).

End (of each skeleton).

End (of each level).

end (of each frame)

Figure VI.8: A snow frame generated by the model

Figure VI.8 shows a snow frame of an animation series generated by the model. Snow man and trees in the background are designed in 3D.

4.4 Summary

Analogous to the heavy rain model, timing of snow can be controlled easily in the model by varying values of $FrtSPD$, $MidSPD$, $BckSPD$. Furthermore,

storm or snow blizzard effects caused by the random gust of wind can be achieved by simply changing the orientation of the tracks which, in the studio, is very difficult and time-consuming to animate with hand drawn ones and the live action is used instead.

Chapter VII

Conclusion and Future Work

In this dissertation we presented the results of research spanning the fields of CAA and procedural animation. With regard to CAA, we have proposed, implemented, and demonstrated an animation framework that enables the creation of stylised effects with minimal intervention from the animator. In our approach, the effects are animated procedurally in the style of hand-drawn effects. Thus, the strength of our approach to animation lies in the fact that this scheme can be made to give effective results in all the cases we have considered with commensurate savings in animator effort – it turns the role of the animator from a hand-drawing slave to the high level model controller. Our procedural approach has advanced the state-of-the-art of computer animation, as evidenced by the stylised animations shown. With regard to procedural animation, we have successfully modelled hand-drawn effects of nontrivial complexity. The convincing simulation results validate our computational models, which capture the essential features of hand-drawn effects – structure, movements and rendering.

In particular, we have developed a hand-drawing-based animation system that emulates the appearance and motion of cartoon effects. Each effect is an individual model with a hierarchic or distributional structure (or combination of the two in the task-specific way) and dynamic control mechanism. Through initial specification, effects are animated according to its static and dynamic structures. The static structure captures the important characteristics of cartoon effects while dynamic structure carries out controlling tasks. The current implemented system is able to not only reproduce cartoon effects on 2D, but also improve hand-drawn effects significantly so that we can animate cartoon effects in 3D because it is almost impossible to achieve this by hand drawn animations. Furthermore, proceduralism avoids the usage of

repetitive cycles. The flexibility of our approach is suggested most evidently by the high-level controls on shape, colour, timing and rendering over the effects.

With regard to the implementation, we have pursued a decompositional and compositional approach in which we started by analysing the animator's drawing process. Upon the simulated hand-drawing substrate, we effectively modelled the dynamics of cartoon effects. The compositional nature of our approach to synthesising cartoon effects was proven crucial to achieving the hand-crafted look. Partial solutions that do not adequately model structure, dynamics and rendering, and do not combine these models intimately within the system will not produce convincing results.

In addition to the hand-crafted look, computational costs are far lower. In generation of animations of the effects, most time is spent on stylised rendering on background. Using a PC Contender ATX P5/133, a simulation of flowing water with the picture size 500 × 400 pixels can run at about 5 frames/60 seconds and a simulation of smoke skein with the same picture size can run at about 1 frame/sec.

1 Additional Impact in Animation

The work reported in this dissertation has promoted further research on automatic motion synthesis for computer animation. We have developed a HdS technique that procedurally synthesises the hand drawing process which is able to cope with drawing problems such as trees [Yu93b]. Motivated by this process, we apply this approach to synthesis hand-drawn effects statically and dynamically.

Procedural animation, as we mentioned in Chapter II, means building an object and then using a procedure to control or animate some attribute of the object at it most basic level. In this regard, conventional inbetweening techniques outlined in Chapter II can also be regarded as procedural approaches. However, because of lack of the ability to deal with problems such as leg dynamics in the walk model, 3D information in the head turning model etc, the degree of proceduralism in the conventional inbetweening methods is far lower compared with other procedural animations mentioned in this thesis in terms of abstracting a scene or a sequence into an algorithm (i.e. a procedure), saving in storage as the details are implicit in the procedure as well as gaining the power of parametric control. What is even worse is that automatic methods for inbetweening rely on some degree of continuity

in both geometry and time therefore they fail to deal with many of the effects described in this thesis where there is no planned continuity between elements, and correspondence are difficult to establish or are non-existent.

Most of the models dealing with effects in this thesis have a hierarchical structure. Here the hierarchy goes from static elements (e.g. extent of fire base), then low-continuity elements (like flame elements, which change slowly or not at all in number but may take different orientations), then random elements (e.g. some parameters and connection curves – effectively "fixing up" the art work at the end). The random elements predominately determine the superficial appearance of individual images while the more continuous elements determine the character of the movement associated with the effect.

Timing problems appear here too. When random or stochastic elements are used there is a risk of running into moire' effects, or fairly subtle aliasing effects, where high-frequency elements like rain drop trajectories can lead to low-frequency effects if the trajectories bunch-up regularly in the cycle (once is enough). Essentially care has to be taken to detect and avoid these problems within the stochastic mechanism on a case-to-case basis.

In dealing with effects in a procedural manner questions of shapes (what do the individual images look like?) predominate over questions of timing (frames may be ordered in time but there is no obvious correspondence let alone continuity between elements in successive frames). Continuity considerations are only relevant at low levels of the hierarchy where there is a relationship with the final image but this is not at all obvious. The point is that procedural methods look like other inbetweening methods at a sufficient level of abstraction.

Diversity of model structure and controls in the effects system notwithstanding, the core idea is to ensure movement consistency of effects which is achieved through structural coherence of the models in the system.

In animation, however, movement consistency is undesirable in certain situations such as an explosion. As explosion is intended to shock the audience, it is advisable to use an unpredictable formula for it. There should be a quick anticipation of some sort, then a quick burst or series of bursts which should be kept going for some frames, followed by a slower dispersal. Here the explosion process has two phases, starting with a very fast movement, which is against movement consistency, and tapering off to a slow finish, which retains movement consistency. Therefore it is difficult to deal with explosions with a single model which tends to preserve movement consistency, but we can solve this problem with an anticipation frame plus a movement

consistent model.

2 Impact on Computer Art

Our work opens up several exciting avenues of research in related fields. For example, the system we have developed has made possible interesting new approaches to computer graphics art.

Clearly the stylised animation is a series of pictures rendered artistically which can therefore be regarded as computer graphics art. Our HdS modelling paradigm together with brush models professes effectively competence at computer graphics art.

Both HdS methodology and brush model simulate the hand painting process in a faithful manner. In comparison with Musgraves' approach for his computer art (which he claims to be an obscure process that no one else can hope to use his program!) [Mus95], this approach addresses the needs of computer graphics artists who are motivated to understand and ultimately reverse engineer computer graphics with certain aesthetic quality and, more importantly, could be learned and used by the people having some sort of art background.

3 Potential Applications in Art Education

Art education has long relied on teachers and books. But both of them tend to teach the continuous painting process in a discrete manner: showing a number of pictures that are supposed to be finished at different phases with regard to the complication of a picture and the intermediate painting process between those pictures needs to be completed by the imagination of the students.

It is possible that students will profit to a great extent from the possibilities offered by computer simulation of the painting process. The availability of simulation model makes it possible to show the painting process with a real or slow timing (so that students can see it better) in a fairly continuous manner.

To this end, our currently implemented system can act readily as a prototype model of cartoon effects that provides a novel and potentially useful

educational tool of cartoon. With regard to computer simulation of traditional media, certain innate characteristics, such as visual composition and colour usage, are medium specific and would require the implementation of specific painting routines. However, our HdS modelling scheme remains appropriate.

Certainly we can use video or film to achieve the same goal in the context of art education. However, a significant advantage of computer simulation over the video and film lies in its flexibility that allows different timing control and searching for a particular piece of painting process students would be interested to see.

4 Other Stylised Animations

4.1 Animation of Plants

The title of this dissertation emphasises the generality of our stylised procedural approach to computer animation, rather than its specific application in this thesis to cartoon effects. The core concepts concerned here are stylisation and proceduralism used in animation. The main components of our approach – HdS modelling, controlling and rendering – carry over to the stylistic modelling of other objects like plants, although the detail of each of these components may, to one degree or another, be object specific. Consider, for example, the design of a hand-drawn tree.

When the animator draws a tree, he usually draws the main stem first, then adds limbs to the main stem, and then adds thinner branches to the limbs. He continues drawing in this way until all the branches in a tree image is completed. Finally he adds leaves (in a stylised fashion) to branches.

To capture the complex structure and appearance of the tree, we developed a tree model based on HdS to generate decorative (stylised) tree images [Yu93b]. In the model we use a vector to approximate each branch of the tree thus getting a *vector tree* to model the topology tree. The *branch geometrical constraints (BGC)* are then introduced to render the vector to get the geometrical tree. The main idea is to separate branch shape from nodes, thus the final tree shapes are determined by the branch shapes as well as the positions of branching nodes. By controlling BGC and the branching nodes with deterministic and stochastic elements, the model can generate tree images of certain artistic quality and a large diversity of forms.

The model offers a direct and flexible control over tree topology and shape. Incorporating only one parameter for angular control with the tree model, we can animate with ease swaying tree images blown by wind [Yu93a].

4.2 Character Animation

In Chapter II we mentioned a few procedural approaches dealing with movements of cartoon characters such as human walk, head turning, lipsynch. The popular techniques used in CAA currently still remain laborious keyframing, this is because humans have sophisticated intelligent behaviours and procedural modelling of human actions remains a difficult endeavour.

In principle, at least, we could use our HdS approach for modelling characters because they are drawn by hand as well. But clearly characters, plants and effects differ dramatically, not only in static structure, but also in dynamic structure. With regard to locomotion, there are obvious differences between inanimate objects and animate objects. The former is predictable therefore it is relatively easy to abstract their movements in a procedure, as does our system in this thesis, while the latter is unpredictable because the mental state of characters may consist of several distinct desires which is time varying. For example, at a given moment in time, a character stands in a room, and what the character wants to do next may be sitting down, or raising hands, or doing both simultaneously, etc. If we call sitting down or rising hands as an *action*, the difficulty to deal with character animation in a procedure is that an action may be followed by many possible actions.

Nevertheless, in the context of procedural animation, it is possible to incorporate some automatic and semi-automatic techniques, such as physics based modelling [AG85] [Wil87] [HBO95], behavioral animation [Zel82], motion capture scheme [CCP80], and motion texture [Per95], with stylised rendering to achieve stylised 2D and 3D character animations.

With regard to 2D character animation, although those aforementioned locomotion control techniques originally were developed for 3D animations, there is no reason why they cannot be applied to 2D situations. Consider behavioral animation, for example, perception modelling, control of behaviour, modelling of action selection: all can be implemented on 2D and we could expect further a quick execution compared with their 3D implementations.

With regard to 3D stylised character animation, we can use all existing locomotion control techniques in conjunction with stylised rendering.

In contrast with realistic animation, stylised animation does not depict every detail but allows a certain amount abstraction, or economy of description. This in turn offers much freedom in rendering and one may wish to design ones own desired artistic effects.

5 Future Research Directions

The cartoon effects system that we have proposed and demonstrated can be improved and further developed in many aspects. We are interested in exploring future research in a number of directions.

5.1 A Variety of Cartoon Effects

The current implemented system can produce effects of flowing water, water jet, water ripples, shimmering, reflections, fire, smoke skein, smoke puffs, light rain, heavy rain, and snow. To enhance our system, we want to implement more cartoon effects such as water splash, water absorbed by the earth, explosions, etc.

Based on the existing implementation, more elaborate effects can be achieved by animating initial conditions in some effects models. Consider, for example, how by altering the density of rain drops locally or globally with a deterministic or stochastic model, we can simulate whether the rain appears lighter or heavier. Similarly altering angles of rain drops, we can simulate the effects of wind on rain drops. Another interesting example is animating two boundary curves used in the following water model – moving them closer until they meet – to show the drying process of a running stream.

5.2 Interaction between Objects and Effects

In the current implementation of the system, we model only the cartoon effects in their normal conditions (with an exception that we model the wind effect of fire). An interesting future research would be to examine the interactions between objects and cartoon effects. One example is a water jet hitting a wall, followed by water splashing in different directions, then splitting into drops and falling down. More examples are that of a ship moving on the shimmering surface, smoke meeting some obstacles, etc.

5.3 Hyperprocedure

The flexibility of the procedural approach would allow us to use one or more procedures in a main procedure, which we give a name here as *hyperprocedure*. In our current fire model, the positional skeletons of flames are specified by the user and invariant with time. As a result, the fire series repeat at the positional skeleton level. We can specify different positional skeletons in the model, which means more manual work involved. Therefore, it would be ideal if we could specify positional skeleton procedurally.

A promising approach for accomplishing this level control would be use of Perlin's texture model [Per95] to generate the "texture" of the positional skeletons of flames, and then transit one texture to the next by a *weight*, which is always a scalar value between 0 to 1. Actually this approach falls in the scope of keyframing: texture is the keyframe and weight is the interpolation parameter. In comparison with traditional keyframing, here positional skeletons of flames are generated in a procedural manner rather than specified by the user. By using randomness we can easily build keyframes that are very controllable yet never actually repeat themselves. Therefore, one of the future research directions could be to develop such a procedure to the current animation system. We believe hyperprocedure would allow us to achieve better results in a much more convenient, automatic way.

5.4 From Realism to Stylisation

As we have mentioned in Chapter II, there are diverse procedural approaches to the realistic modelling of natural effects. An obvious way of stylisation of realistic effects is to incorporate a stylised rendering model into those systems. For instance, in those models depicting gaseous phenomena, it may be possible to detect boundary and texture features of a fire or smoke based on density calculated in the models, then render those features to get desired stylised effects.

This hybrid approach has advantages as well as disadvantages when compared with our current implemented system. One advantage is the flexibility in specifying effects which allows us to make animation like burning letters [Sim90] and fire spreading [CMTM94]. Another is the ability to deal with the interactions between objects and effects [CMTM94].

A significant disadvantage is that the computational cost is far higher. For instance, Perlin reported that it took about 8 hours to compute a single

frame of a 640 × 480 × 640 volume for smoke tested on the *AT&T* Pixel Machine [Per95]. Because of this, many researchers suggested using parallel computation [Sim90] [Per95].

5.5 Stylised Animation

The long-term goal of our research is the title of this dissertation – stylised procedural animation. Clearly, computer generated cartoon like animation is a stylised interpretation of the world. We believe, as the movement toward more creative and expressive imagery continues in computer graphics, stylised animation will have a variety of styles as art does today (In a broad sense, photorealistic animation can be regarded as one style of stylish animations, as does photorealism in art).

The process of making stylised animations juxtaposes the deterministic formalism of the scientific method with the subjective aspects of visual aesthetics. Musgrave described a model of the creative process where art and science can be brought together–*searching n-space for local maxima of an aesthetic gradient* [Mus95]. Although the model is used for his product of computer art (which he considers to be fine art), we think it would be heuristic for the stylised procedural animation.

The process can be explained as follows. We have created a procedural, parametrically-controlled model of a synthetic microcosm, say there are n independent parameters in that model and the specification of its projection onto the image plane. As these parameters are independent, we can think of each as representing a *degree of freedom*, or an additional *dimension* or direction in which we can move. Taken together, the n parameters define an *n-dimensional space* or *n-space* for short. In this space we are free to move not just up and down, right and left, or forward and back, but in a whole lot of other abstract directions as well. A *local maxima* is the location in the space from which all directions lead "downhill", i.e. it is a kind of hilltop in n-space. "Downhill" is defined by the "aesthetic gradient function" – the completely subjective (non-deterministic) assessment on the part of the artist of what constitutes a "better" image, in terms of the parameter values. As Musgrave addressed, this so-called "function" is not unambiguous: its value will depend on the criterion by which the image is being assessed, and even upon the mode of the artist at the moment of evaluation. The local maximum is then a point in n-space from which a small move in any direction would result in a "less good" image.

Ambiguity notwithstanding, this n-space gradient-ascent model is more than just entertaining: it points out that a given image represents a *local* maximum of the aesthetic gradient field. Other, more global maxima ("higher hilltops", corresponding to "better" pictures) undoubtedly exist elsewhere in the rich abstract n-space of potential images defined by the formal system.

We think that Musgrave's model, together with our assessment criteria [YP97], would be a good touchstone of stylised procedural animation.

Appendix A

Brush and Painterly Rendering

In cartoon animation, characters are contour drawn and simply shaded, while objects in the background are drawn with the style, realistic (not photo-realistic) or stylised, designed by the art director. Since the main goal of drawings here is beauty, the visual artist searches for an equilibrium between proportions of different shapes and an equilibrium between different colours, to interpret the world by abstracting a scene, real or imaged, to a 2D paper. By varying brush texture, size, and direction, the artist can not only define forms, but also provide rhythm and energy that help direct the viewer's eye. A painter can even use brush strokes to represent light and atmosphere.

Computer rendering provides an easy, automated way to render everything in a scene with fine detail. This creates static images that do not invite the viewer into the process. In particular, when creating images for animation, focus and simplification are essential to showing action in a clear way since the temporal nature of the image gives the viewer much less time to let their eyes wander about the scene. Certainly focus and simplicity can be achieved with computer rendering tools by carefully controlling lighting and surface attributes and unnecessary detail can be obscured using hierarchical modelling, but it is still difficult to obtain the level of abstraction that is evident in a good painting. Hand-drawn and hand-painted animation have an energetic quality that is lacking in most computer-rendered animation. Often when computer methods try to mimic the wavering quality of hand-drawn animation, too much randomness creeps in and makes the animation noisy. A human artist drawing each frame is better to control frame-to-frame coherence, while maintaining a hand-crafted look. The focus of most rendering research in the last two decades has been on the creation of photo-realistic imagery. These methods are quite sophisticated, but tend

to create imagery that is mechanical-looking because detail is represented very accurately. Recently there has been a movement toward more creative and expressive imagery in computer graphics but few techniques that provide ways to achieve different looks, especially for animation. Some computer painting tools can mimic successfully the hand-drawn line quality, painterly looks, and energy of traditional media, but these tools typically work only for still frames. Recently, a painterly rendering technique using particles for animation was proposed in [Mei96] which renders the animated frames with the characteristics of an oil painting.

In this Appendix, we first survey the existing brush models and then address the issue of rendering for animation using brushes, finally presenting a different technique of painterly rendering for animation with a style more adapted to cartoon background.

1 Survey of the Existing Brush Models

The brushes used in early computer painting systems were far simpler than real paint brushes. Usually no more than automated rubber stamps, they build up images by placing repeated copies of some static or simply derived pattern. Some systems offered air-brushes, which simulated a spray of ink by painting pixels in a circle region around the brush.

Before in literature, many powerful and expressive brush models have been proposed. Whitted used antialiased Z-buffer images to create strokes with a 3D appearance [Whi83]. Green described an input device called the "drawing prism" [Gre85] which digitises the image of a real brush (or other object) making optical contract with a transparent prism. Although the resulting images are realistic, the system has no representational abstraction higher than the pixel level. The skeleton based stroke primitives in Berkel's SIAS system allowed the local width of a stroke along the path to vary arbitrarily [vB89]. They might be suitable for specific applications like digital typography. These strokes are however too restrictive in form as a general brush stroke replacement. Strassmann modelled the ink-laying process of bristle brush on paper [Str86]. The image left by a sopping wet brush dragged erratically across a sheet of textured paper can be generated by a representation which keeps track of the physical properties of the materials. Guo and Kunii extended them to include ink-diffusion through the paper fabric mesh [GK91]. Their results are attractive despite the relatively slow computation speed. To achieve ultimate authenticity, Pang *et al* even at-

tached real bristle brushes to plotters and defined the strokes by the paths and pen up/down control parameters [PZ91]. Later on, Wang and Pang developed a computer Chinese calligraphy system [WP91] in which the contour of the Chinese character is defined first, then the stroke speed and ink amount are simulated by sampling points interpolated with splines. Hsu and Lee used the same skeletal idea as in [Str86] and [vB89] to develop a so called skeletal stroke [HL94]. Based on a 2D deformation model defined by an arbitrary path, they can realize the brush and stroke metaphor using arbitrary pictures as "ink".

Combining the skeletal idea with the spray model, we proposed a brush model which allows us to control stroke shape and colour arbitrarily [JJY96]. The model can easily simulate the dry-brush used to present speed lines in cartoon animation, and the effect of *gouache*, furthermore it solves a difficult problem dealing with the *Cao* style of Chinese calligraphy. As indicated in later sections of this Appendix, the model is extended further to painterly line and surface rendering which are amenable to the traditional parametric line and surface models. The following section describes our brush model in more detail.

2 A Skeletal Spray Brush Model

A physical brush is made of a bunch of bristles, such as the brush used for Chinese traditional painting and Chinese calligraphy, or the painting brush used for oil painting. They are made into round or flat shapes in different sizes. The brush leaves a *footprint* when the artist presses the brush and draws a stroke when the artist drags the brush along a *trajectory* on the paper or canvas. The final appearance of the stroke is determined by the local *width* and *colour* of the stroke. We synthesise a brush stroke by employing a *skeleton* to model the trajectory of the brush together with a few skeleton attributes to control the size and colour of the footprint of the brush, as shown in Figure A.1.

2.1 Skeleton

The skeleton controls the trajectory of the brush which is a series of nodes $BGuidP_i, (i = 1, 2, ...GdN)$, where GdN is the number of nodes of the skeleton. We refer them as guiding points because they control the position of the brush. In most applications, the skeleton can be modelled by a spline

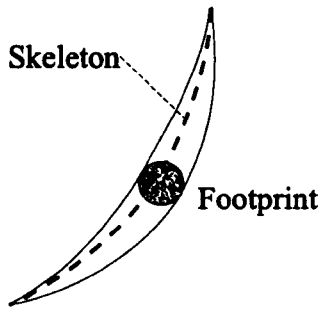


Figure A.1: Footprint, skeleton and shape of the brush

thus requiring only a few control points specified by the user or calculated by some models.

2.2 Footprint

In the model we choose to use a spray within a circle or rectangle to simulate the footprint of the brush. The spray well reflects the feature evident in relative dry colour, as if done with a dry-brush, an effect commonly used in cartoon to represent speed lines, for example, when a character runs off screen [Whi86]. In the case when wet colour is required, we render the footprint with constant shading and particles, which will be described shortly in our painterly rendering.

The density of the spray, i.e. the number of points $BFootN$ contained in a footprint, simulates the amount of the colour left on the paper. For a given size of the footprint, say a radius $BFootR$ of a circle, the bigger $BFootN$ is the denser (heavier) the footprint looks. Furthermore the points within the footprint can be distributed by using different functions such as uniform or gradient distribution to produce a variety of brush looks.

Multiple points used in the footprint allow us to choose from one to a moderate number of colours for presenting the spray. In the case of multiple colours being used, a colour weight can be set in a task-specific way so that some colour may appear stronger than others, say, if a gradient weight function is used.

2.3 Skeleton Attributes

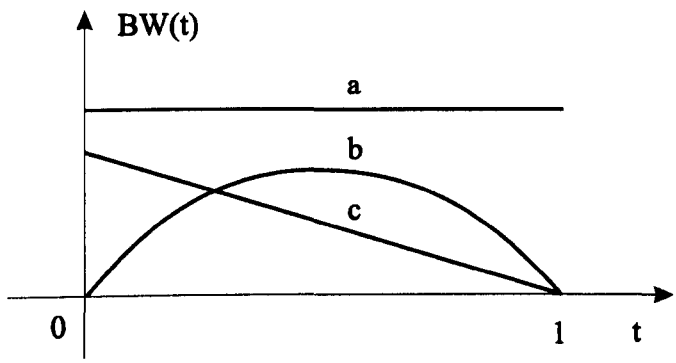


Figure A.2: Brush width attributes

In the model we define a driving parameter $t \in [0, 1]$, which may be thought of time, $t = 0$ corresponds to the starting point $BGuidP_1$ and $t = 1$ corresponds to the end point $BGuidP_{GdN}$ respectively on the skeleton. We take t as a variable to define a skeleton attribute function:

$$BSkltAttrb(t) = BSA(BW(t), BD(t), BC(t))$$

where $BW(t)$, $BD(t)$ and $BC(t)$ are skeleton attributes which control the overall appearance of the brush stroke and defined according to the application: $BW(t)$ controls the size of the footprint (thus in effect determining the width of the stroke) which can be symmetric about the skeleton, or defined as $BleftW(t)$ and $BrightW(t)$ respectively on the two sides of the skeleton. Figure A.2 gives a few commonly used $BW(t)$ curves. Curve (a) corresponds to a stroke of constant thickness, curve (b) may be used to draw shapes like common tree leaves while curve (c) may be used to draw some shape like a bamboo leaf. $BD(t) = BFootN(t)$ controls the spray density and $BC(t)$ controls the colour variance of the footprint along the skeleton.

Once the skeleton attributes are defined, let t vary from 0 to 1, t drives brush model $Brush(t) = \mathrm{B}(BGuidP(t), BSkltAttrb(t))$ move along the skeleton to draw a stroke with desired width, density and colour.

2.4 Structure of the Model

The model of one brush stroke can be expressed as follows:

1. Specify control points for skeleton $BCtrlP_i (i = 1, ...BCtrlN)$.
2. Define shape of the footprint: circle or rectangle.
3. Define spray distribution: uniform or gradient.
4. Define colour distribution: uniform or gradient.
5. Define skeleton attributes $BW(t), BD(t), BC(t)$;
6. Interpolate control points $BCtrlP_i, (i = 1, ...BCtrlN)$ and put interpolated points into $BGuidP_i$ $(i = 1, ...BGdN)$;
7. Set $i = 1$;
8. Define driving parameter $t = i/BGdN$;
9. Pick up current guiding point $BGuidP(t)$ on the skeleton;
10. Determine the size of a footprint with $BW(t)$;
11. Draw a footprint according to $BD(t)$ and $BC(t)$;
12. Increase index i and repeat step 8, 9, 10, 11 until i reaches $BGdN$.

2.5 Applications of the Brush

2.5.1 Computer Art

The current brush model is able to produce the dry brush effect easily that is commonly used in cartoon animation to represent speed lines, light rain (see Chapter VI) etc. In Figure A.3 we show a picture with the characteristics of *gouache* painting generated using our brush model. The skeleton for each lotus petal is a simple vector and a few vectors constitute the skeleton with the form like the mount of a Chinese fan, the orientation and length of each vector randomly perturbed. We use two sets of skeletons to draw the inner and outer petals. The brush width attribute is curve (b) in Figure A.2. The brush density attribute is set to a constant and brush colour attribute varying from pink to white. We use some concentric ellipse as the skeleton of lotus

Figure A.3: An image generated by skeletal spary brush

leaf. Water waves are represented with a rectangle footprint drawn along the horizontal skeletons. This example demonstrates how such a stylised picture can be generated with simply designed models.

2.5.2 Computer Chinese Calligraphy

Figure A.4: Different styles of Chinese calligraphy

Chinese calligraphy has a long history and there are literally thousands of styles. Basically, they can be categorised into the following scripts: *Jia Gu Wen* (the ancient Chinese words carved on bones, starting time 2000 BC-?),

Figure A.5: A Chinese character generated by skeletal spary brush

Zhuan Shu (seal script, starting time 770 BC-221 BC), *Li Shu* (clerical script, starting time 25-220 AD), *Kai Shu* (standard script, starting time 173 AD), *Xing Shu* (semi-cursive script, starting time 87 AD), and *Cao Shu* (cursive script, starting time about 48 BC). Figure A.4 shows examples of different styles.

In *Cao* style a Chinese character is often so joined-up that it looks like one long, twirling ribbon. Some calligraphists also tend to drag the brush erratically and quickly, thus causing a *Fei-Bai* (hollow stroke), as if done with a half-dry brush, to show the momentum of the brush movement. The final appearance of a *Cao* character is characterised by its trajectory, width variance of the stroke and the ink amount left on the paper (which is both calligraphist and character dependent). To deal with *Cao* style with our brush model, it is essential we design the skeleton, $BW(t)$ and $BD(t)$ to capture the characteristics of a Chinese character of *Cao* style and Figure A.5 shows an example simulated by our model.

3 Rendering for Animation Using Brushes

The constant thickness stroke can be regarded as the simplest brush model and is widely used to draw the contour of characters in cartoon animation. But it puts up a very poor show in the situation when more expressive strokes are required, say, to paint the realistic or stylised cartoon background.

To render objects in the background with a painterly style, we would expect that brushes be applied directly to the traditional parametric line and surface models. Unfortunately, only a few brush models were designed

in such a scheme for the purpose of animation. In the SIAS [vB89], Berkel defined straight lines and arc segments within a 3D Cartesian coordinate system that represented the positions of the eyes, nose, mouth, tongue, etc. These straight lines and arc segments are connected by joints, that allow them to rotate and translate relatively to each other. For each frame in the animation, the straight lines and arc segments are projected onto the view plane, then they are mapped to the virtual screen and strokes are drawn along the straight lines and arc segments on the virtual screen. Here only feature parts on the head are drawn with strokes and the face surface is left transparent for the audience to complete by imagination. Hsu and Lee used the skeletal stroke's compact abstraction to condense the complicated hatching or stipples into simple units with which to further build up characters. Traditional key-frame technique is used to interpolate skeletons to make the stylised animation as shown in Figure A.6 [HL94].

Figure A.6: A stylised keyframe drawing

In order to take advantage of the benefits of a painterly look on computer-rendered animating geometry, Meier proposed a technique which combined core ideas from two areas of previous work: painterly rendering of still images from reference pictures and particle rendering [Mei96]. The goal is to provide a tool that automates the drawing of brush strokes and leaves the artistic decision about lighting, colour, and brush stroke characteristics to the user. The difficulty in using existing still frame methods for animation is getting the paint to "stick" to surface rather than randomly change with each frame, while still retaining a hand-crafted look. Meier solved the temporal randomness problem by using particle rendering methods. The following section will describe the technique in more detail.

4 Rendering Using Particles

In 1990, Haeberli described a system for creating painterly images from a collection of brush strokes that obtain their attributes, such as position, colour, size and orientation, from synthetically rendered or photographic reference pictures [Hae90]. In the system brush position are randomly distributed, so successive frames of an animation would change randomly. Alternatively, the positions and sizes of brush strokes could remain constant over the animation, but this creates the "shower door" effect, in which an animation appears as if it were being viewed through textured glass, because brush strokes are effectively stuck to the view plane not to the animating surface.

The idea to eliminate both the "shower door" effect and random temporal noisiness by Meier is to treat strokes as particles that are stuck to surfaces. She begins by creating a particle set that represents geometry such as a surface. The particles are transformed to screen space and sorted in order of their distance from the viewpoint. A painter's algorithm is used to render particles as 2D brush strokes starting with particles furthest from the view point, and continuing until all particles are exhausted. Each brush stroke renders one particle. The look of the rendered brush strokes, including colour, shape, size, texture and orientation, is specified by a set of reference pictures or by data that is stored with the particles. Reference pictures are rendered pictures of the underlying geometry that use lighting and surface attributes to achieve different looks. The attributes for a particle are looked up in the reference pictures in the same screen space location at which a particle will be rendered finally. The algorithm can be described as follows:

Create particles to represent geometry.

For each frame of animation.

1. Create reference pictures using geometry, surface attributes, and lighting.
2. Transform particles based on animation parameters.
3. Sort particles by distance from viewpoint.
4. For each particle, starting with furthest from view point.
5. Transform particle to screen space.
6. Determine brush stroke from reference pictures or particles and randomly perturb them based on user-selected parameters. Composite brush stroke into paint buffer.

End (for each particle).

End (for each frame).

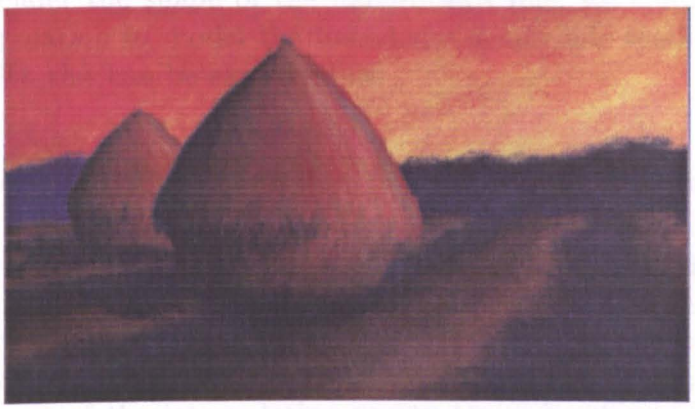


Figure A.7: A frame rendered by particles

Figure A.7 shows a frame from a painterly rendered animation using particles.

5 Painterly Line and Surface Rendering

The technique described in the previous section renders the frames of animation with the characteristics of an oil painting. To achieve a hand-crafted look, using randomness is important in painterly rendering. The image rendered without colour, orientation, or scale variation, as shown in [Mei96], would look mechanical. Similarly a line generated by a simple line drawing command cannot vividly simulate a painterly line drawn by hand. This is because a painter does not use a ruler but controls the brush with his bare hand to accomplish the task, consequently the line drawn has a random appearance both in shape and colour due to slight shaking of the hand. In the following sections we present our method for the painterly line and surface rendering which work straightforwardly with the traditional parametric line and surface models while still retain the look more like water colour.

5.1 Line Rendering

In order to render the shape of the hand-drawn line, we employ a simple random interpolation to model the line shape in our line rendering model. Mathematically, this can be expressed as:

$$LineP = LineP_1 + u(LineP_2 - LineP_1) + rnd(r)V(LineP)$$

where $LineP_1$ and $LineP_2$ are two terminal points of the line, $LineP$ is interpolated points, $u \in [0, 1]$ is the interpolating parameter and $rnd(r)$ is a random variable of variance $V(LineP)$. The final line shape is achieved by joining those interpolated points and the line appearance is determined by the increment of the interpolating parameter and value of the random variable. Smaller increment of the interpolating parameter and big value of the random variable would result a more zigzag line appearance, while a big increment of the interpolating parameter and small value of the random variable would result in a more straight line appearance.

Since the line is actually composed of a series of segments along a straight underline, the use of the random variable controlling the position of each segment may result a break-up between two neighbouring segments which, from the point of view of ordinary graphics application, is undesirable. For the purpose of our line rendering, however, random break-up best reflects the feature of the hand-drawn line resulted from light hand shaking.

Another feature associated with hand-drawn line is that the width of the hand-drawn line varies slightly and randomly, and as a result, some segments may appear light or dark on the line. In most cases the variance in line width is smaller than the pixel size so that it defeats the modelling with pixels, therefore we render each segment with variations in colour instead:

$$LineC = LineBC + rnd(r)V(LineC)$$

where $LineC$ is actual rendering colour, $LineBC$ is the basic colour specified and $rnd(r)$ again is a random variable perturbing $LineC$ with variance $V(LineC)$.

Our line rendering model works directly on a line between two points $LineP_1$ and $LineP_2$. As mentioned in Chapter II that everything in computer animation is in the end represented as polygons or polylines, therefore we can

apply our line rendering model on every segment of a polyline to render a free form curve.

Depth and lighting information can be incorporated with above colour rendering model thus we have:

$$LineC = k_{cb}(\mathbf{NL}) + rnd(r)V(LineC) + LineDC$$

where k_{cb} is a base colour coefficient related to a light source, $\mathbf{N}$ is the unit normal at the point of interest on the line, $\mathbf{L}$ is the vector in the direction of the light source, and $LineDC$ is the colour encoded by the depth information of the object. We should point out that our central concern is painterly rather than realistic rendering, so we define base colour coefficient k_{cb} in our model which actually uses exaggerated hue as well as value variations to distinguish light and shadow parts on the line.

Due to the randomness involved in our rendering algorithm, we use predefined seeds to ensure that the same perturbations in shape and colour will be used for a particular segment on the line throughout an animation. As a result the temporal noisiness is eliminated and the frame-to-frame coherence is maintained. We will show the example of our line rendering in conjunction with our surface rendering described in the following section.

Markosian *et al* [MKT+97] presented a real-time nonphotorealistic rendering which is able to generate artistic strokes such as drawing the polyline directly with slight variations in line width or colour, high-resolution "artistically" perturbed strokes defined by adding offsets to the polyline, and texture-mapped strokes which follow the shape of the polyline. Unfortunately they are generated by modifying the resulting 2D polyline projected into the film plane thus can not avoid "shower door" effects mentioned in section 4. In addition, their method does not take lighting conditions into account.

5.2 Surface Rendering

There are many methods of rendering surfaces, such as those described in [WW92]. Again, for the purpose of painterly rendering, we employ constant shading and particle rendering to produce a look more like water colour that is suitable to render the cartoon background.

Constant shading uses a single intensity for each polygon. In our situation, the intensity can be calculated with the dot product of the unit surface normal and the vector in the direction of the light source, or specified according to the desired effect by the user. Again, a random component can be added to the intensity to achieve a hand-crafted look.

After a polygon is shaded, say, with colour $PBaseC$, we randomly distribute particles within it, the number of particles for the polygon $PrtlN$ is determined by:

$$PrtlN = PrtlN_0 + rnd(r)V(PrtlN)$$

where $PrtlN_0$ is specified by the user and $rnd(r)$ is a random variable of variance $V(PrtlN)$. A bigger $PrtlN$ would produce drier colour effect and inversely a smaller $PrtlN$ would produce wetter colour effect.

Colour diffusion is simulated by distributing particles around the boundary lines of each polygon. The wider the particles are distributed, the heavier the colour diffusion would appear.

Colour for individual particles can be controlled in the same way as described in our former brush model. Actually our painterly line and surface rendering may be regarded as variants of the brush model. In the line rendering the footprint becomes a perturbed segment and in the surface rendering it is represented by blending constant shading and particles with the shape defined with polygon boundary lines.

Applying our surface rendering model to the polygon of a big area would produce a mechanical look because of lacking random variation in constant shading. To solve this problem, motivated by the fact that a painter usually tends to colour a big area with multiple strokes, we sub-divide the polygon into small ones and then apply our rendering model to them individually. Again we distort the polygon shape by locally moving each vertex on the surface randomly after sub-division to achieve a hand-crafted look.

To maintain coherence, as done in the painterly line rendering, pre-defined seeds are stored so that the same perturbations will be used for a particular polygon in every frame of an animation.

In summary, our painterly surface rendering model can be expressed as follows:

For a polygon surface:

1. Determine the base colour $PBaseC$ for each polygon on the surface.
2. Distort each polygon on the surface.

For each polygon:

1. Fill a polygon with colour $PBaseC$.
2. Determine the number of particles $PrtlN$.
3. Determine positions of particles.
4. Determine the colour of particles using $PBaseC+rnd(r)V(PrtlC)$.
5. Draw particles.
6. Determine the width for diffused particles $DPrtlW$.
7. Determine the number for diffused particles $DPrtlN$.
8. Determine positions for diffused particles.
9. Determine the colour for diffused particles using $PBaseC+rnd(r)V(PrtlC)$.
10. Draw disused particles.

End (for each polygon).

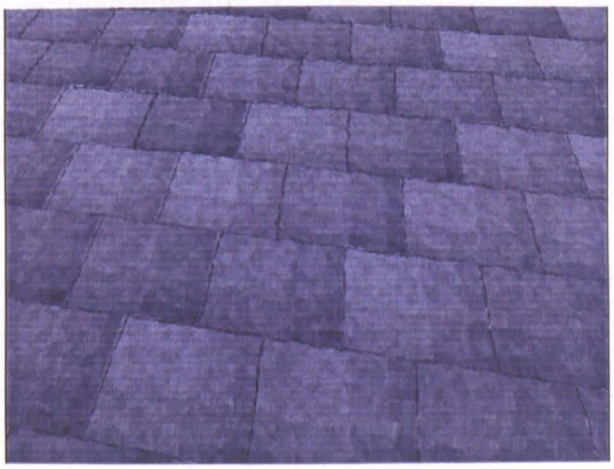


Figure A.8: A pavement with a water color look

Figure A.8 shows a pavement rendered with our line and surface rendering models. The colour specification in this example is made at two phases. The first phase is concerned with the base colour for each paving block which is calculated by adding two components encoded by the lighting and depth information to the colour value specified in advance. The second phase is concerned with the actual colour rendering the small sub-divided, distorted polygons and we use the base colour together with the component encoded by the local spatial information within one paving block and an additional component to fulfil the task. The boundary lines to draw the paving blocks are rendered by our line rendering model in which the base colour is darker in comparison with the base colour used in paving blocks.

In designing the pavement, we specify the colours of paving blocks which are perturbed randomly around a base colour value and encoded with lighting information. Then each paving block is sub-divided into small distorted polygons which are rendered with the above model using the same lighting information. Finally the boundary lines of paving blocks are rendered by our line rendering model with a darker base colour compared with that used for paving blocks.

6 Summary

Both Meier's and our renderers aim at achieving a painterly rather than realistic look for animation and overcome the problem of random frame-by-frame changes in animation involved in the previous painterly rendering techniques. The distinction between the two is obvious that the former renders the frame with the characteristics of an oil painting and the latter renders the frame with a look more close to the water colour. In addition, Meier's rendering scheme uses a reference picture to define 2D brush stroke attributes, while our renderer offers a straightforward rendering which is amenable to the traditional parametric line and surface models. Although our renderers are currently used for the cartoon background, we can image the look of an animation if applying them to both 2D and 3D characters.

Bibliography

[AG85] W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purpose of animation. *The Visual Computer*, 1:231–240, 1985.

[AKN91] T. Agui, Y. Kohno, and M. Nakajima. Generating 2-dimensional flame images in computer graphics. *Trans. IECE of Japan*, 2:184–189, 1991.

[BW71] N. Burtnyk and M. Wein. Computer generated key-frame animation. *Journal of Society for Motion Picture and Television Engineering*, 80:149–153, 1971.

[BW76] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *CACM*, 19:564–569, 1976.

[CCP80] T.W. Calvert, J. Chapma, and A. Patla. The intergration of subjective and objective data in the animation of human movement. *SIGGRAPH'80*, pages 198–203, 1980.

[CMTM94] N. Chiba, K. Muraoka, H. Takahashi, and M. Miura. Two-dimensional visual simulation of flames, smoke and the spread of fire. *Visualisation and Computer Animation*, 5:37–53, 1994.

[EG94] G. Elber and D.H. Gotsman. Multiresolution control for non-uniform b-spline curve editing. *Preprint*, 1994.

[FR86] A Fournier and W. T. Reeves. A simple model of ocean waves. *Computer Graphics*, 20:75–84, 1986.

[Gar92] G. Gardner. Fractal ellipsoid fire. *SIGGRAPH Video Review*, pages 184–189, Issue 81, 14,(1992).

[Ger09] F.J. Gerstner. Theorie der wellen. *Ann. der Physik*, 32:412–440, 1809.

[GG95] E. Goldstein and C. Gotsman. Polygon morphing using a multiresolution representation. *Computer Interface'95*, pages 247–254, 1995.

[GK91] Q. Guo and T.L. Kunii. Modelling the diffuse paintings of 'sumie'. *Modelling in Computer Graphics*, 1991.

[GMM87] P.A. Watterberg G.A. Mastin and J.F. Mareda. Fourier synthesis of ocean scenes. *IEE Computer Graphics and Applications*, 7:16–23, 1987.

[Gre85] R. Greene. The drawing prism: A versatile graphics input device. *SIGGRAPH'85*, 19:103–110, 1985.

[Hae90] P.E. Haeberli. Painting by numbers. *SIGGRAPH'90*, 24:207–214, 1990.

[Har81] W. Harold. *Timing for Animation.* Focal Press Limited, London, 1981.

[HBO95] J. Hodgins, D. Brogan, and J. O'Brien. Animating human athletics. *SIGGRAPH'95*, pages 71–78, 1995.

[HL94] S.C. HSU and I. H. H. LEE. Drawing and animation using skeletal strokes. *SIGGRAPH'94*, pages 109–118, 1994.

[Hun94] Jane Hunter. *Synchronisation of Sound and Animation.* Cambridge University, PhD Thesis, Cambridge, 1994.

[Ina90] M. Inakage. A simple model of flames. *CGI'90*, pages 71–81, 1990.

[JJY96] Yu Jinhui, Zhang Jidong, and Cong Yanqi. A physically-based brush-pen model. *The journal of CAD and Computer Graphics (in Chinese)*, 8:241–245, 1996.

[KTZ94] B.B. Kimia, A. Tannenbaum, and S.W. Zucker. Shapes, shocks and deformations. *International Journal of Computer Vision*, 1994.

[Mal95] H. Mallinder. The modelling of large waterfalls using string texture. *Visualisation and Computer Animation*, 6:3–10, 1995.

[Max81] NL Max. Vectorised procedural models for natural terrain: Waves and islands in the sunset. *Computer Graphics*, 15:317–324, 1981.

[Mei96] B.J. Meier. Painterly rendering for animation. *Computer Graphics (SIGGRAPH'96)*, pages 477–484, 1996.

[MIT67] T. Miura, J. Iwata, and J. Tsuda. An application of hybrid curve generation - cartoon animation by electronic computers. *Spring Joint Computer Conference*, 1967.

[MKT+97] L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and J. F. Hughes. Real-time nonphotorealistic rendering. *SIGGRAPH'97*, 1997.

[Mus95] Lecturer: F.K. Musgrave, editor. Orgniser: D. Bert, Los Angeles, 1995.

[NMN87] T. Nishita, Y. Miyawaki, and E. Nakamae. A shading model for atmospheric scattering considering luminous intensity distribution of light sources. *Computer Graphics*, 21:303–310, 1987.

[NN87] T. Nishita and E. Nakamae. A display method of uniform particles in the atmosphere. *Proc. 35th Annual Convention IPS Japan*, pages 2307–2308, 1987.

[OI88] T. Ohshima and S. Itahashi. Texture animation. *Proc. NICOGRAPH'88*, pages 110–119, 1988.

[Pea86] DR Peachey. Modelling waves and surf. *Computer Graphics*, 20:65–74, 1986.

[Per85] K. Perlin. An image synthesiser. *Computer Graphics*, 19:287–296, 1985.

[Per95] Lecturer: K. Perlin, editor. Orgniser: D. Bert, Los Angeles, 1995.

[PG92] J.W. Patterson and G.Cockton. Composing hierarchically structured images. *Proc. of EUROGRAPHICS'92*, 11:829–839, 1992.

[PT88] X Pueyo and D Tost. Survey of computer animation. *Computer Graphics Forum*, 7:281–300, 1988.

[PW94] J.W. Patterson and P. J. Willis. Computer assisted animation: 2d or not 2d? *The Computer Journal*, 37:829–839, 1994.

[PZ91] Y.J. Pang and H.X. Zhong. Drawing chinese traditional painting by computer. *Modelling in Computer Graphics*, 1991.

[Ree81] W. T. Reeves. Inbetweening for computer animation utilising moving point constraints. *Computer Graphics*, 15:263–269, 1981.

[Ree83] W. T. Reeves. Particle system - a technique for modelling a class of fussy objects. *Computer Graphics*, 17:359–376, 1983.

[RF96] V. Ranjian and A. Fournier. Matching and interpolation of shapes using unions of circles. *Computer Graphics Forum*, 15:C–129–C–142, 1996.

[Sak93] G. Sakas. Modelling and animating turbulent gaseous phenomena. *The Visual Computer*, 9:200–212, 1993.

[SF95] J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion. *Computer Graphics*, pages 129–136, 1995.

[SG92a] G. Sakas and M. Gerth. Sampling and anti-aliasing of discrete 3-d volume density textures. *EUROGRAPHICS'92*, pages 107–117, 1992.

[SG92b] T. W. Sederberg and E. Greenwood. A physically based approach to 2d shape blending. *Computer Graphics*, 26:25–34, 1992.

[SGWM93] T. W. Sederberg, P. Gao, G. Wang, and H. Mu. 2d shape blending: An intrinsic solution to the vertex path problem. *Computer Graphics*, 27:15–18, 1993.

[Sim90] K. Sims. Particle animation and rendering using data parallel computation. *Computer Graphics*, 24:405–413, 1990.

[SR94] M. Shapira and A. Rappoport. On compatible star decompositions. *Tech. Report TR94-15, Institute of Computer Science, Hebrew Univ. of Jerusalem*, 1994.

[SR95] M. Shapira and A. Rappoport. Shape blending using the star-skeleton representations. *IEEE Computer Graphics and Applications*, 19:44–50, 1995.

[Str86] S. Strassmann. Hairy brushes. *SIGGRAPH'86*, 20:225–232, 1986.

[TT85a] N. M. Thalmann and D. Thalmann. *Computer Animation: Theory and Practice.* Springer-Verlag, Berlin, 1985.

[TT85b] N. M. Thalmann and D. Thalmann. *An Indexed Bibliography on Computer Animation.* IEEE CG&A, 1985.

[TT87] N. M. Thalmann and D. Thalmann. *Image Synthesis.* Springer-Verlag, 1987.

[vB89] P. van Berkel. Sias, strokes interpreted animated sequences. *Computer Graphics Forum*, 8:35–45, 1989.

[Whi80] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23:343–349, 1980.

[Whi83] T. Whitted. Anti-aliased line drawing using brush extrusion. *SIGGRAPH'83*, 17:151–156, 1983.

[Whi86] T White. *The animator's book.* Watson-Guptill, New York, 1986.

[Wil87] J. Wilhems. Using dynamic analysis for realistic animaiton of articulated bodies. *IEEE Computer Graphics and Applications*, 7:12–27, 1987.

[WP91] X.Z. Wang and Y.J. Pang. A computer chinese calligraphy system. *Journal of Computer-aided Design & Computer Graphics*, 3:35–40, 1991.

[WW92] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques.* ACM Press, New York, 1992.

[YP96a] Jinhui Yu and John W. Patterson. A fire model for 2d computer animation. *Computer Animation and Simulation'96, Springer-ComputerScience EG*, pages 49–60, 1996.

[YP96b] Jinhui Yu and John W. Patterson. Object deformation using quaternions. *Proc. of Eurographics UK Chapter 14th Annual Conference* ,, pages 75–88, 1996.

[YP97] Jinhui Yu and John W. Patterson. Assessment criteria for 2d shape transformations in animation. *Computer Animation'97*, pages 103–112, 1997.

[Yu90a] Jinhui Yu. Inbetweening for computer animation using polar coordinate linear interpolation. *CS Report Series, CSC 90/R23, University of Glasgow, UK*, 1990.

[Yu90b] Jinhui Yu. A walk model for computer-aided character animation. *CS Report Series, CSC 90/R30, University of Glasgow, UK*, 1990.

[Yu92] Jinhui Yu. A walk model for computer-aided character animation. *Proc. of The 6th National Conference on Imagery and Graphics,Zhengzhou, China (in Chinese)*, 1992.

[Yu93a] Jinhui Yu. Animating sway trees using a model-aided inbetweening methods. *Proc. of The Third International Conference for Yong Computer Scientists*, 1993.

[Yu93b] Jinhui Yu. Computer generation of decorative tree images. *Proc. of The Third International Conference on CAD & Computer Graphics*, 1993.

[Yu93c] Jinhui Yu. A new interpolation algorithm for computer animation. *Applied Science and Technology*, 2, 1993.

[Yu94a] Jinhui Yu. A head turning model for computer-aided character animation. *The Journal of CAD & Computer Graphics (in Chinese)*, 6, 1994.

[Yu94b] Jinhui Yu. A hierarchical flowing water model. *Proc. of The 7th National Conference on Imagery and Graphics, April, 1994, Chengdu, China (in Chinese)*, 1994.

[Zel82] D. Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Application*, 2:53–59, 1982.