ANIMATION OF TEXT COMPRESSION ALGORITHMS

A thesis
submitted in partial fulfilment
of the requirements for the degree
of
Master of Science in computer science
in the
University of Canterbury
by
Timothy David Wilson

University of Canterbury
1992
To the glory of God, who gave me a mother and father who have continually shown their love for me, and in their lives, words, and prayers given me the most valuable thing, a faith that is real
Abstract

It has been said that, there is no particular mystery in animation ... it's very simple and like anything that is simple, it is about the hardest thing in the world to do. Text compression is about finding ways of representing text with the smallest amount of data such that it can be restored to its original state. Text compression algorithms are abstract concepts and bringing them into the visual domain is difficult but the effort can bring insight both to the student and to the researcher. This thesis presents some animations of text compression methods and observations about producing good educational and research animations. Several algorithm animation systems were used in the animation production and it was found that although there are several good animation systems fulfilling different functions, little is known about what makes good algorithms animation. A better way of defining animations and some practical principles for animation that were discovered while producing these animations are discussed.
Acknowledgements

I would like to thank,
   Tim Bell for being a supervisor who always encouraged and understood me.
   My good friends, my Mum and Dad, who supported me all the way through, in all I have been doing, and prayed for me.
   David Conradson who dropped his own very busy work and wrote prolific messages to me in his proofs, always encouraging. Thanks David for being the practical help I needed when I needed it, for giving the spiritual encouragement I needed, and for those killer aerobic-circuit training sessions.
   John Stasko who kept up constant dialog when I was using his animation system and was so friendly, enthusiastic, and helpful.
   Jon Bently for accidently sending me the Anim system by Email.
   Dr Mandelberg and Edgar Leon of Emory University, Atlanta for providing the specifications for the currently available video projectors.
   Neil Sharman for being a great fellow thesis sufferer and for writing the scripts that put all my images into a \LaTeX document to produce the flip-book animation.
   Alan Finlay, where-ever and whoever you are, for answering my desperate question to the USENET and helping me work out why I could not print out my animations.
   Rick and Helen Spencer who encouraged me, gave me good food to eat, a place to sleep and shoulder to cry on, for last hard three months.
   And finally, and most importantly my Lord, and God Jesus Christ, without whom this could never have possibly come together at all.
   Nehemiah 6:15 So the wall was completed ...6:16 ...they realised that this work had been done with the help of our God.
Contents

1 Introduction 1
  1.1 Introduction to Algorithms Animation .......................... 1
  1.2 Definitions .................................................................. 3
  1.3 Why study Algorithms? .............................................. 4
  1.4 The Importance of Visualisation ................................... 8

2 Text Compression 11
  2.1 Introduction ............................................................. 11
  2.2 Compression, Prediction and Modelling ......................... 12
  2.3 Models and coding ...................................................... 13
  2.4 The importance of text compression ............................. 15
  2.5 Information and models ............................................. 15
  2.6 Adaptive Models ....................................................... 16
  2.7 Arithmetic coding .................................................... 17
  2.8 Dictionary techniques ............................................... 17
  2.9 The corpus .............................................................. 19

3 Algorithm Animation systems 22
  3.1 Movies ..................................................................... 22
  3.2 Data Structure Display .............................................. 23
    3.2.1 Incense .............................................................. 23
    3.2.2 PV ..................................................................... 24
    3.2.3 GDBX .............................................................. 25
    3.2.4 PROVIDE ......................................................... 25
    3.2.5 GAIGS ............................................................. 25
  3.3 Algorithm Animation Systems .................................... 26
    3.3.1 Toronto ............................................................ 26
## CONTENTS

3.3.2 Balsa .................................. 27  
3.3.3 Using Smalltalk .............................. 29  
3.3.4 Animus .................................. 29  
3.3.5 Anim .................................. 30  
3.3.6 ALADDIN .................................. 31  
3.3.7 Xtango .................................. 31  
3.3.8 Polka .................................. 32  
3.4 The best algorithm animation system ........... 33  
3.4.1 Choosing the best tool ..................... 34  

### 4 Text Compression animations

4.1 Animating Models ...................................... 37  
4.1.1 Models .................................. 37  
4.1.2 Animation of the order-0 model .............. 42  
4.1.3 Higher Order model display .................. 51  
4.1.4 Adding the compressed file size to animation .. 52  
4.1.5 Research Idea .................................. 55  
4.2 Ziv-Lempel animation .................................. 60  
4.2.1 LZ77 .................................. 61  
4.2.2 LZ78 .................................. 65  
4.2.3 Other findings .................................. 70  
4.2.4 Displaying animation to fourth year class .......... 70  
4.3 Arithmetic coding animation ......................... 71  
4.3.1 Progression of creation ......................... 73  
4.3.2 Principles of animations discovered .......... 78  
4.3.3 New features for Xtango ....................... 82  
4.3.4 Concept animation verses interactive animation .... 85  
4.4 Future directions ...................................... 88  
4.4.1 Three Dimensional Animation .................. 88  

### 5 Animating: So what do you want to see?

5.1 An animation system doesn’t make animations ........ 90  
5.2 Algorithm animation design is difficult .......... 93  
5.3 Algorithm animation to static diagram ............. 96  
5.4 Defining animations ................................. 98  
5.4.1 Macro and Micro detail ......................... 98
CONTENTS

5.4.2 Brown’s 3-D definition ........................................ 98
5.4.3 Alternative 3-D definition .................................... 99
5.5 Algorithm animation versus visual programming ............. 100
5.6 Automatic animation ............................................ 100
5.7 Metamouse ...................................................... 101

6 Algorithm Animations and teaching ............................... 103
6.1 Flip book ....................................................... 103
6.2 Video/Movies .................................................. 104
6.3 Seminar or large group situations .............................. 105
6.4 Scripting ....................................................... 106
6.5 Interactive animations ......................................... 106
6.6 Knowledge based help ......................................... 107
6.7 The need for empirical data .................................... 107

7 Conclusions ....................................................... 109

Appendix

A Probability graphs of the Calgary corpus files ................ 113
B The LZ78 animation trie development ........................... 117
C Colour and 3D algorithm visualization ........................... 121
D Specifications of available large screen computer projectors ... 125

Bibliography ....................................................... 129
Chapter 1

Introduction

The purpose of computing is insight, not numbers. Richard Hamming

An algorithm must be seen to be believed... Donald E. Knuth

1.1 Introduction to Algorithms Animation

In 1981 Baeker's movie “Sorting out Sorting”, was shown at SIGGRAPH '81\(^1\). This was one of the first systems to animate algorithms — to show and describe an algorithm in a clearer and more dynamic way. Algorithms that were once described and explained using static diagrams in books and on blackboards now came to life. People could now see and appreciate algorithms visually where they had been accessible only to those who had access to expensive graphics equipment and lots of time.

Recently, with the advent of cheap workstation technology, the description and presentation of algorithms has taken a major step forward. The viewer is no longer passive and restricted to viewing what the movie maker decided to animate, but can interact with the animation to try and retry, stop, go backward, change parameters and generally explore the algorithm in a far more active manner.

The topic of this thesis is the exploration of text compression algorithms in this manner—using animation to allow visualization and interaction with text compression methods, and the insights that this can bring. The research reported in this thesis not only gives insights into text compression, but also insights into algorithms animation.

This chapter is an introduction to the area of algorithm animation. The second chapter will introduce text compression algorithms and how understanding of the algorithms can

\(^1\)Baeker was one of the first to see the possibilities of algorithm animation, and produced several movies of which this is the best known. A fuller description of this movie and the systems he developed are detailed in section 3.1 and 3.3.1
be improved using animation. Chapter 3 is a survey of animation systems, past and present. It introduces the animation systems used in the text compression animations described and discussed in the fourth chapter. The last two chapters discuss concepts of algorithm animation and animation's usefulness to teaching.

Put simply, algorithm animation is making moving pictures of a computer programs as an aid to learning. Programs can be an amorphous mass of writing in a strange language that magically produce results in some form. At the heart of programs are algorithms. To show the workings of program visually seems to be helpful to understanding that program. People have experienced many times in normal life how it is easier to see something happen than to describe what happened. We will discuss later whether this has been shown to be true for algorithms.

As an example of algorithm animation, consider the algorithm displayed in figure 1.1 from the MacBalsa animation system. This is easily described as a bar graph that gets itself into order— in fact, figure 1.1 shows a list partially sorted by insertion sort. Each successive frame shows the elements more sorted than the last. In insertion sort each element is shuffled through to its correct position by successively comparing its value to that of its left neighbour until the neighbour is smaller than itself (as shown in frame (b) by the bar labelled ‘j’ comparing itself to its neighbour, the greyed left box). The animation as a whole portrays concepts such as number of iterations, order of processing, speed, and number of comparisons made.

This example has already raised an important problem of this thesis: portraying animations using the static medium of print. Wherever possible a series of shots from the animation will be shown to give an idea of the effect, but this will never give a full appreciation of what movement brings to a diagram. This is why—for probably the first time ever in a thesis—some flip animation is included on the back pages of the text. This not a complete answer to this problem, but it is better than no animation at all.
Chapter 1. Introduction

The subject of algorithm animation should not be confused with animation algorithms. Animation algorithms are concerned with better, faster, and more beautiful graphics animations. Animation algorithms themselves could be animated, and conversely, the graphics that the algorithms produce may themselves be visually instructive on how the algorithm is working, but this is dependent upon the situation. Graphics algorithms are needed for the output of algorithm animations, but this is not domain of this thesis, and is only mentioned briefly in relation to a single type of graphical transformation.

This thesis concentrates on the animation of text compression algorithms. Text compression involves reducing the size of some stored file without removing redundant information. Text compression will be introduced more fully in the next chapter.

In this chapter definitions and questions that are basic to algorithm animation are given. We also consider the importance of visualisation, and why visualising algorithms is worthwhile. The use of the visual machinery of the brain is discussed and how people benefit from visual means of communicating information.

The reader with an understanding of the concepts of text compression and keen to get their teeth into the animations and the animation principles discovered should head straight for chapter 4, whereas the reader unfamiliar with the field of text compression should read chapter 2, where text compression is overviewed. Chapter 3 is especially interesting for those wanting to animate algorithms—here the different animation systems available are described, and what would suit the needs of the type of visualization you want to produce. Chapter 5 discusses the the needs of the animator, the difficulties involved in animation, and producing static pictures of animations. It also sets out a way of classifying algorithm animations, and discusses the possibility of automatic generation. Chapter 6 outlines the various ways that algorithm animations can be presented for teaching.

1.2 Definitions

The word *algorithm* originates from the word *algorism* which is derived from the name of a famous Arabic mathematician, Abu Ja'fer Mohammed ibn Músú al-Khowárizmí (AD825). Al-Khowárizmí, which means native of Khowárizmí, wrote the book "Kitab al jabar w'al-muqabala" ("Rules of restoration and reduction") [Knu73]. In this he first suggested a mechanical method for adding two numbers represented in the Hindu numerical system. This is the method of addition taught in schools—adding of operands and a previous carry to form the sum, and continuing until there are no more operands to add [Kri85]. This

2The word "algebra" derives from the *al jabar* in the title of Al-Khowárizmí's book
method has all the aspects of an algorithm for which Al-Khowârizmî became the namesake for. A definition of an algorithm has been stated as:

"a finite set of rules which gives a sequence of operations for solving a specific type of problem with the important features of:

- finiteness
- definiteness
- input and output
- effectiveness" [Knu73].

Finiteness requires that an algorithm terminates after a finite number of steps. Definiteness means that each step it is unambiguous in its purpose. There must be some sort of input and output data particular to the algorithm that is being performed, although not necessarily both. The algorithm must consistently perform correctly, that is it must be effective for the job it is supposed to do.

An algorithm could be described as a methodical way of doing something, like a recipe, with a series of steps ("add butter, melt and mix with dry ingredients"), carefully defined ("add 1 cup of milk....bake at 180 C for 35 mins..."), and an expected result ("...until lightly brown", see figure 1.2).

Animation is defined in the Collins Concise English Dictionary as, "to give life to; bring to life; to give motion to."

The word animation is most frequently associated with cartoon animations. Here imaginary characters come to life as the skilfully drawn (or computer generated) pictures are shown in quick succession. Combining the definition of an algorithm and animation, we can define algorithm animation as:

*Taking methods and bringing them to observable life.*

1.3 Why study Algorithms?

An important question to consider before looking at the animation of algorithms as a tool for teaching them is why we would want to teach algorithms.

Algorithms are central to computer science, therefore they are a central part of teaching it. Computer science is about using computers. All that a computer does can be reduced to sequence, selection, and iteration on data of some form, that is, perform some algorithm.
Chapter 1. Introduction

Figure 1.2: An algorithm has been performed, giving the desired result

So for computer science it is natural to want to understand the workings of algorithms because they are such an important part of all that is computer science.

Practically every field in computer science involves algorithms and therefore animation could be applied. For example, operating systems concepts involve algorithms for scheduling processes and for memory management. Figure 1.3 shows a series of stills produced by the Anim algorithm animation package of the memory usage of a program during execution.

Another area of computer science is Data Communication, where protocols (like sliding window) are algorithms for exchange of information between computer systems. Of course, teaching algorithms and data structure design would be naturally appropriate for the use of animation. Courses in areas such as compilers, graphics, databases, parallel programming, and artificial intelligence, all involve algorithms which could be taught using animation.

Another reason we teach algorithms is to enable students to be able to implement algorithms. This is more than just the programming skill, but includes the ability to translate known algorithms into working code. It requires the understanding of algorithms so that the characteristics of a problem can be reflected in the implementation.

Another important aspect of algorithms is their analysis. That is, which algorithm is better than others, and when. We therefore need to teach algorithms to introduce the idea that there are good and bad algorithms, and to encourage students to design efficient
Figure 1.3: Some frames from the animation of memory allocation during runtime. The black shows the memory allocated, with the low addresses at the bottom. The series of frames show initially (a) the memory being filled until at (b) where memory begins to be freed.
Chapter 1. Introduction

Since the teaching of algorithms has a central place in computer science, animation can be used to improve the learning environment. Furthermore, researchers, as well as students are learners—they learn about the new. Algorithms are being created continually and therefore the need to be able to effectively study them is important. Animation can give new insight into how an algorithm works. My first taste of algorithms animation was to animate sorting algorithms for use in teaching a second year algorithms course.

When viewing these for the first time I was struck by how we discovered things about the sorting algorithms that we hadn’t fully realised before. Behaviour that was not obvious from learning about the algorithm from a lecture, became apparent from the animation, which frequently raised the question, “Why does it look like that?” and prompted further exploration.

Figure 1.4 shows the first partition step in a Quicksort of two different random lists. Figure 1.4a is how Quicksort was expected to look from how I understood the algorithm worked. The list divided into two separate lists either side of a partition element with all items less than the partition element to the left, and all those greater to the right. Then the same partitioning process recursively performed on the two lists. What I expected to see was two lists split in half like figure 1.4a, instead I was surprised to see what is shown in figure 1.4b. Reflecting on why it looked this way brought a fuller understanding of how the Quicksort algorithm worked. The partition element chosen from the list will be some random value which will on average partition the list in half. But there will be frequent occurrences of “bad choices” that make the partitions unbalanced.

Figure 1.4: Quicksort shown (a) a "good" partition, and (b) a "bad" partition
Chapter 1. Introduction

It is quite possible that I knew about this behaviour of the algorithm before I saw it, but it was not obvious. This type of discovery of the non-obvious is an important motivation for algorithm animation, and this indicates that animating text compression algorithms could lead to new observations and insights.

There are examples in the book "Algorithm Animation" [Bro88a] of how animation has been profitable in research into algorithms. Through experimenting with an animation of Knuth's dynamic Huffman trees Vitter noticed strange behaviour of the trees with a particular type of input. This lead to a new improved algorithm for dynamic Huffman trees [SV84]. A variation of Shellsort was discovered in conjunction with static colour displays of Bubblesort, Cocktail-Shaker Sort, and the standard Shellsort [Inc86], and an early version of the Balsa animation system was used to study a then newly discovered stable Mergesort [HS82].

1.4 The Importance of Visualisation

Algorithm animation is a tool for the visualisation of algorithms—taking a concept and forming a picture of what is happening. But why is visualisation important if we can grasp the concepts with theory and deduction?

One answer is that we are very much visual beings, and rely heavily upon sight. What the scientific visualisation community is saying is that we think visually, that we comprehend, digest, and discover using both the visual centres and the analytical parts of our brains.

In a special issue on visualisation SIGGRAPH states that an estimated 50 percent of the brain's neurons are associated with vision [McC87]. Visualisation in the scientific community aims to put that neurological machinery to work. If this is the case then it is important that we seek out ways to make use of this "neurological machinery".

Springer and Deutsch's book "Left Brain Right Brain" [SD85] develops the concept that the two hemispheres have quite different tasks, based on studies of patients who have had the hemispheres of their brain surgically separated. Their studies cleverly directed sight and sound information to chosen hemispheres of the brain. The results strongly indicated that the left side of the brain is more verbally oriented and the right side more visually oriented.

From studies of computer science students Scanlan [Sca88] claims that the needs of perhaps only 25% of students are being met by totally verbal methods of teaching algorithms. Scanlan tested student preference to a graphical method (structured flowcharts) compared with a verbal method (pseudocode) when learning short, relatively complex,
Chapter 1. Introduction

His findings show the need for visual methods for algorithm learning.

This idea of illumination via graphical interpretation is stressed in Tufte’s book “The Visual Display of Quantitative Information” [Tuf83]. This book describes the way data has been displayed over the years and presents pointers towards better graphical presentation. It speaks of the importance of using graphical methods. Rather than being presented with a table of values that give no lasting impression, a visual representation “imprints upon the mind a distinct idea”. It is as if when someone sees something, it becomes a picture more readily remembered and linked with the concept that they once just knew by name and concept. This idea is expressed well by Paul Valery:

“seeing is forgetting the name of the thing one sees.”[Tuf83]

In the course of working with algorithm animation I have experienced this quite strongly. I have used several animation systems that all had the sorting algorithms animated, so I have seen the sorting algorithms animated many times and in various ways. I now have pictures in my mind of all the major sorting methods, rather than just names. For example, when I think of Quicksort and Heapsort I think of the distinctive patterns made by the cloud of dots getting sorted, as in figure 1.4 and 1.5.

If a picture is worth a thousand words, then a movie, which shows 25 pictures per second, must be even better. But interactive movies, available on affordable workstations, are even more valuable. It is the purpose of this thesis to explore algorithms animation
and how this brings insight to text compression.
Chapter 2

Text Compression

2.1 Introduction

This thesis is about making algorithms, particularly text compression algorithms, easier to understand by animating them. What follows is a brief introduction to the field of text compression.

Compression is making something more compact by pressure. Text compression is about finding ways of representing text with the smallest amount of data such that it can be restored to its original state.

Text compression has a long history. For as long as humans have communicated they have often subconsciously shortened their speech and writing so that information could be communicated in fewer words. This is all part of formal text compression: removing redundancy so that only that which is necessary to correctly decode the information is stored.

Text compression foreseeably will always be necessary, for even as technology advances and more can be stored and retrieved at greater speed, there are always applications advancing that require greater storage and increased speed. And it is desirable that information be stored optimally because compression allows increased storage without extra storage media.

Bell, et al.'s 'Text Compression' [BCW90] is the basis for this overview and it discusses how compression has been used in the past. This gives a useful introduction to text compression for it shows how intuitive it is to compress information—to represent with short codes the most common and with long codes the infrequent.

This overview introduces the concepts of the algorithms that are animated. The descriptions of what was involved in the animations is discussed in chapter 4. It also presents
some points about the importance of text compression. After this introduction we look briefly at the concepts of information and models, moving directly to talk about adaptive models. Next we overview coding with an introduction of the optimal coding technique, Arithmetic coding. The group of techniques called Dictionary text compression methods are then looked at. For the purpose of testing compression techniques and—in this thesis—for the viewing of models, a corpus of standard text is used and this will be introduced. Finally in this chapter we give reasons why text compression should be taught and studied, and therefore why animation is useful and necessary.

2.2 Compression, Prediction and Modelling

"Compression is inextricably bound up with prediction" [BCW90]. By prediction Bell et al. mean the probability of an event happening. Working out the probability of a character occurring in a given text determines the length of code that it should be given.

Weather forecasting provides a useful analogy. A weather forecaster makes a prediction based on current weather data. One simplistic weather model is called persistence forecasting [Bat79], where you predict that tomorrow's weather will be the same as today's. You are making a prediction based on the current context. This is also true of text compression. Often the preceding characters in a text are a good predictors of the characters that follow, for example. 'animatio' from the previous text would almost completely predict that the next character is 'n', especially if you take into the account the common usage of the word in the previous text.

To construct some sort of probability distribution for the characters in the text you require a model. The model for most compression schemes is dynamically based on the data gained about the text from the text seen so far. This can be compared with the climatological information that meteorologists collect over the years. They can then predict what each month's weather will be like on average for different localities where they have collected data, and similarly in text compression we build up 'climatological' data about the text that is being compressed. The frequencies of characters in character context 'localities' make up a model of the text. From this developing model probability distributions are produced to be used for compression.

Text compression must be exact. Unlike picture or sound compression where inexact compression can be tolerated, in text compression there is no such thing as approximately an 'e'—is this closer to being a 'd' or an 'f'? In general text compression means exact compression.

Text stored in computers is generally stored as 8 bit ASCII (America Standard Code
for Information Interchange) or EBCDIC (Extended Binary Coded Decimal Interchange Code). If the 256 symbols that can be represented were all equally likely to occur in a file, then 8 bits/character would be the minimum that each character could be encoded in. But for something like English text in a book, this is never the case and it can be reduced by a good compression scheme down to as little as 2.5 bits/character on average.

To assign a code to each character so that the whole file is compressed requires estimating the probabilities of all the possible characters and from these probabilities calculating codes to these characters. The shortest code is assigned to the most probable character and codes of increasing length for those of decreasing probability. This is seen in Morse code where, after studying the probabilities of characters in English text, Morse assigned ‘e’ = ⋅ and ‘q’ = – – ⋅ ⋅ ⋅ so that the more common characters (like ‘e’) have shorter codes, while the uncommon (like ‘q’) have longer, making the average length of a transmission shorter. The task of assigning these codes from the probabilities is called coding. The task of calculating the probabilities is called modelling.

The distinction between modelling and coding is an important one, because coding has been solved optimally and modelling cannot be. The source of most text is people and therefore we are ultimately trying to model human thought processes, so our models will always be approximate unless ideas from science fiction are realised and computers can think just as humans. The better the model resembles the source of the text the better the message can be compressed.

### 2.3 Models and coding

The conceptual separation of modeller and coder was a major advance in data compression in the last decade. As seen in figure 2.1 the modeller feeds the coder, which generates the compressed version of the file. The construction of the model is as Bell et al. put it “an artistic endeavour which is not soluble in the same definitive way as constructing an encoder” [BCW90].

In some compression schemes, like the Ziv-Lempel dictionary methods, the modeller and coder are hard to see as separate entities. But this is not the case in statistical methods.

The idea of a model is that it provides a probability distribution for each symbol coded, which is generally one character. The coder then takes the probability and the character and devises a code for that character. At the other end the decoder gets a probability distribution from its model (which is exactly the same as the encoder’s model) and the code and works out what the character is.
These models can be either static, semi adaptive or adaptive models.

Morse's model is a static model. From studying English text he came up with the likelihoods of characters and assigned them codes. These were then adopted by all stations. Morse code achieves some reduction in transmission time for English texts because the more common characters have shorter codes. But a static model will expand the transmission time when it gets a message that does not fit the model. This might occur if, for example, the football results were transmitted in Morse, because in Morse code numbers have long codes.

A semi-adaptive Morse code would require you to calculate the symbol probabilities for the message you wanted to transmit, and encode the symbols with reference to these probabilities. You would then transmit the codes for the symbols of the message, and then the message itself. This has two disadvantages: transmitting the codes takes more space and time, and two passes of the file are required to transmit it.

An adaptive model changes the codes relative to the file being compressed as it is sent. Codes for characters need not be sent explicitly as the sender and receiver have the same system of assigning new codes to characters so that the most common characters always have the shortest codes. This also may achieve better compression as the model adapts to
the message changing as it changes, compared to the semi-adaptive model, which assigns codes once only for the entire text.

2.4 The importance of text compression

Over the years there has always been a need for larger storage in computer systems and a greater need for increased communications systems bandwidth. It has been said that you always need 110\% of the disk storage that you currently have, except for the first week after you get a bigger disk. Even with the increased capacity of many storage systems (Compact Discs are able to store $4.8 \times 10^9$ bits), there is still a demand for greater and faster storage systems. With the approach of the paperless office requiring high quality pictures and voice mail as well as text storage, this demand will continue. Even in the event of a breakthrough in storage capacity that went way beyond what we currently have, applications that were once inviable would soon make use of it, and require more space.

It is not only in storage that text compression has an important role, it may be even more important in data transmission. Communication channels are costly to set up and maintain, and national and international links are vital to many businesses. In fact, the loss of connection with the outside world can be incredibly costly. This communications need will continue to expand, and as compressed information is quicker to transmit, transmission costs are reduced, and line capacity increased, making it highly desirable.

Many applications of compression are used daily. Modems have been enhanced with dedicated compression chips that can increase throughput by a factor of two. Facsimile machines use a simple line by line compression scheme that speeds them up by a factor of seven. Netnews is sent daily among many UUCP sites and it has been demonstrated [BCW90] that compressing, transmitting and then uncompressing batches of news can be substantially quicker, and use less processor time, than just transmitting a batch. For pure storage, compressing archive files has become popular as a way of storing information that is not accessed often. Also, the widespread use of compression programs like Unix's compress, Macintosh's Stuffit, and the PC's ARC means that users of these systems can efficiently transmit compressed files to each other.

2.5 Information and models

Information theory is the foundation of compression. Information theory gives us hooks into the information content, which is the limit for compression. That is, when you have just the information that a file contains, you have the most compressed version of the file.
Entropy is a fundamental concept of information theory. Entropy is most commonly heard in relation to order and disorder in the physical realm. It is large where there is disorder and small where there is order. For example, a tidy room is said to have lower entropy than a messy room. Entropy of a message is a measure of order or redundancy in the message.

Shannon (1949) describes entropy as $E = - \sum_{i=1}^{n} p_i \log_2 p_i$ bits, where $E$ has units of bits and the log is base 2.

This entropy is a measure of the quantity of information in a given message. Often we think that the length of a message portrays its information content but more often than not this is incorrect. For example, a long thesis does not necessarily mean it conveys a lot of information, and it may well be the reverse. Albert Einsteins original relativity paper is just 30 pages (approximately 560 lines of text, 6000 words) but the information it contained has caused years of research and large numbers of books and papers to be produced.

Information is inextricably bound up with choice. The more choice, the more information is needed to specify the result of that choice [BCW90].

2.6 Adaptive Models

Adaptive compression methods are flexible and are therefore more often used than purpose built compression methods. One adaptive method can be used for many different types of file, and files that change radically within, like an executable file that has strings of text stored at the end, or a book that includes encapsulated postscript images.

There are two parts to an adaptive model: the statistics unit, and the structure. Compression algorithms use simple structures and they are simple to update, whereas the updating of the statistics presents two problems: the zero-frequency problem, and the best match problem. The former is about how much to assign from the overall probability to events that have not yet happened. That is, there are novel or unexpected events for which you have no information about yet, but they must be given a probability that is greater than zero. The best match problem is about the trade-off between frequency and recency. The model needs to have accurate frequency statistics but also needs to use a large enough recent context as a predictor. But the contexts stored require exponentially larger previously seen text\(^1\) before you have any statistics for all of those states. This problem is solved by blending contexts of different lengths.

\(^1\)A 4 character context has $27^4 = 531441$ states to collect statistics for, a 5 character context has approximately $14 \times 10^8$ states
Chapter 2. Text Compression

This thesis focuses on animating adaptive methods because they only perform slightly worse than a purpose built non-adaptive method for a particular text, whereas in the worst case a non-adaptive method can do far worse than an adaptive method.

Animations of single character and two character frequency statistics show vividly how some files statistics seem to be stable in their growth whereas others have periods of change. These animations are dealt with in detail in Chapter 4, and they show the value of adaptive techniques.

2.7 Arithmetic coding

Arithmetic coding is an optimal coding method; given a model, an arithmetic coder will produce a message equal to the entropy of that model. The output can be thought of as a real number within the range 0 to 1. In practice it works with fixed-precision arithmetic, and can transmit the compressed data incrementally. This means that as the file is being read and compressed it can be sent and uncompressed simultaneously.

Arithmetic coding works by taking the probabilities given by its model and divides up the range \([0,1)\) in proportion to these probabilities. The range of the first symbol of the text to be compressed then becomes the new range (see figure 2.2). This range is similarly divided up with the probability distribution given for the next character. This continues until the whole message has been seen whereupon a terminating symbol is the final range to be expanded. Any real number that lies within this final range can be transmitted.

The decoder starts with the transmitted number and the same probability distribution and the same starting range of zero to one. It can tell what the first character is from the range that the number occurs within. From here on the process is the same as the encoding: expanding the range, then dividing up the range with the probability distribution, and finding the next character range that the decimal fraction occurs in.

As seen in figure 2.2, as the characters are read and the range tightens the first digits of the range bounds become the same and therefore can be transmitted, allowing the precision of the bounding values to be decreased. This means that the encoder and decoder can use finite precision arithmetic and the message can be decoded incrementally. This avoids having to wait until the whole message is sent before the receiver can begin to decode it.

2.8 Dictionary techniques

Compression can generally be categorized into statistical or dictionary. Statistical methods retain statistics about the text and from these produce a probability distribution for the
Chapter 2. Text Compression

Figure 2.2: (a) Representation of the arithmetic coding process; (b) arithmetic coding with the interval scaled up at each stage
characters that is then handed on to an arithmetic or Huffman encoder. This section describes Dictionary techniques.

Dictionary compression methods use a dictionary of phrases that are in the text, replacing the occurrences of these phrases in the text with pointers into a dictionary. They are an intuitive way of compressing information that has been used for centuries. For example we talk of chapter three of the gospel of John rather than "the section of the bible written by the apostle John, about Jesus's meeting with Nicodemus". People have ID numbers that give a lookup into a databases that describes relevant details about that individual. The most obvious example is, of course, a dictionary that gives a word's full definition.

Non-adaptive techniques use a fixed dictionary, while semi-adaptive methods transmit the dictionary of phrases first. Adaptive techniques are almost exclusively based on the Ziv-Lempel or 'LZ'\(^2\) methods [ZL77, ZL78]. The dictionary in an adaptive technique is built from phrases in the previously seen text. A coded phrase is replaced with a pointer to where it has occurred previously. There are two major families of methods known as LZ77 and LZ78 named from the year that the original methods were published.

### 2.9 The corpus

To compare compression methods a standard group of test files has been collected [BCW90]. These files include English text, pictures, geographical data, and programs both source and object files.

For animation purposes they are also useful as a test suite that give a varied range of observations for different situations that text compression methods have to handle. For use with animations these corpus files were sometimes truncated to make the animation file manageable size, and yet make the length of animation long enough to give a feeling for the characteristics of the file. It was generally found that first of 20,000 characters was representative of the total file.

Figure 2.3 and 2.4 describes the corpus files used in the animations in this thesis (especially with the order-0 and order-1 model animations). A sample of each file is shown, on the right hand side is a graph of character probabilities in the file.

\(^2\)The letters are switched because of a historical error
Chapter 2. Text Compression

bib : bibliography in Unix "refer" format, ASCII,
725 references for books and papers on Computer Science
111,261 characters

%A Witten, I.H.
%D 1985
%T Elements of computer typography
%V 23

book1: fiction book in Unformatted ASCII,
Thomas Hardy's: Far from the Madding Crowd
768,771 characters

a caged canary -- all probably from the windows of the
house just vacated. There was also a cat in a willow
basket, from the partly-opened lid of which she gazed
with half-closed eyes, and affectionately-surveyed the
small birds around.

book2: non-fiction book in Unix troff format, ASCII,
Witten: "Principles of computer speech"
610,856 characters

Figure 1.1 shows a calculator that speaks.
Whenever a key is pressed,
the device confirms the action by saying the key's name.
The result of any computation is also spoken aloud.

geo0: geophysical data as 32 bit numbers,
Seismic data
102,400 bytes

d3c2 0034 12c3 00c1 3742 007c 1e43 00c3 2543
d5c2 0020 4442 00b8 1b43 00a2 2143 00a2 1143
1143 000a 1843 0032 e142 0050 36c2 004c 10c3
1143 0081 ad42 0060 e2c2 001c 1fc3
e042 0020 00c3 2543 00a2 1143 004c 10c3

news: electronic news in USENET batch file,
A variety of topics
377,109 characters

In article <8168@ism780C.UUCP> jimh@ism780C.UUCP (Jim Hori)
writes:
> The Lessing is probably Doris who has
> written several futurist/SF novels ...
> Her SF novels are serialized, and from what

obj1: object code as executable file for VAX,
compilation of "progp"
25,004 bytes

0b3e 0000 efdd 2c2a 0000 8fdd 4353 0000 addd
0007 dd00 f0ad 8ed0 d051 c0a1 dd50 9850 7e0a
9003 b150 d604 04al efde 235a 0000 f0ad addd
8fdd 4357 0000 04fb d5ef 000a 7000 c5ef 002b
a1d0 500c 50dd 04fb e7ef 0006 6e00 9def 002b

obj2: object code as executable file for Apple Macintosh,
Knowledge Support System program
246,814 bytes

0004 019c 0572 410a 7474 6972 7562 6574 0073
6e69 6f64 0077 0000 0000 00aa 0091 00ba 06ff
01ba 06ef 0000 0000 0000 00c3 0050 00d3 0687
01d3 90e0 0000 0000 0015 0021 00e0 01f0 00f6
e800 e000 0000 0000 0500 9f01 1900 e501 0204

Figure 2.3: The files of the corpus including a graph of character's frequencies
Such a fixed model is communicated in advance to both encoder and decoder, after which it is used for many messages. Alternatively, the probabilities the model assigns may change as each symbol is transmitted, based on the symbol frequencies.

Programs can be written which spread bugs like an epidemic. They hide in binary code, effectively undetectable (because nobody ever examines binaries). They can remain dormant for months or years, perhaps quietly and imperceptibly infiltrating their way into the very depths of a system.

If $E > \text{Maxexp}$ then \{overflow-set to most negative value\}

\begin{verbatim}
if E > Maxexp then {overflow-set to most negative value}
begin
 S:=MinusFiniteS;
 Closed:=false;
end
\end{verbatim}
Chapter 3

Algorithm Animation systems

...your old men will dream dreams,
your young men will see visions. Joel 2:28

To animate algorithms requires an animation environment—some way of translating the concepts and actions of the algorithm into visual entities. The following is a description of different past and present algorithm animation systems. It is a useful survey in that it covers nearly all possible animation systems that have been produced, mentioning what each of them bring to the field of algorithm animation. Most of these systems are surveyed in greater depth by Brown [Bro88a], others have been produced since his dissertation added. The original algorithm animations were put frame by frame onto movie film, they are mentioned first, along with animated data structure display systems, which can animate algorithms to a certain degree. The main part of this chapter surveys the algorithm animation systems available, and describes what the best possible system would have.

3.1 Movies

At the end of the 1960's and the beginning of the 70's the high cost and low availability of computer graphics hardware meant that animation was rarely used as a teaching tool. Some saw that movies were a more appropriate medium and produced movies of graphics animations so that many more people could see them. Brown [Bro88b] mentions three such movies.

The first of these was "L6: Bell Telephone Laboratories Low-Level Linked List Language", produced by Knowlton [Kno66] at Bell Labs in 1966. It showed the workings of an assembly level linked list. The first to show an algorithm was Hopgood's movie on hashing
algorithms [Hop74]. This had several important features. It showed three simultaneously updated views of the algorithm: a hash table with the number of collisions per entry, a graph of the number of probes needed to insert each item, and a maximum number of collisions to insert an item. The movie showed a level of simulation outside the possibilities of hand simulation and showed the algorithm running with different parameters. Booth’s “PQ-Trees” [Boo75] showed the effects of various algorithms on a PQ-tree data structure. It used colour.

The pinnacle of the algorithm animation movies was Baecker’s “Sorting out Sorting” [BS83] which showed several different scenes of sorting algorithms running, using colour, a voice over commentary describing the characteristics of the algorithms, and large data sets (sorting 10,000 items!). Two pictures from the movie are shown in Appendix C.

The movie used these large sets to make a strong point about how algorithms differ in complexity (speed). The view showed a race between several sorting algorithms\(^1\). Quicksort won, with bubblesort trailing a long last. The movie did not change view after quicksort finished, but showed the entire painfully slow sort of the elements by bubblesort. The voice over explaining that you were being forced to watch the entire slow process (which took several minutes) to remind you to never use an algorithm that is $O(n^2)$ (bubblesort) when there is another algorithm that has order $n \log n$ (quicksort). The movie also shows various views of the sorting algorithms, with small data sets portrayed as sticks of different length and large data sets shown as clouds of dots, which are basically the ends of the sticks rather than the whole thing. Baecker’s work is the best of the movies but had a substantial cost, as it took three years to produce this thirty minute movie of sorting algorithms only.

The distinct disadvantage with movies is that they can only be viewed passively. There is no ability to change parameters and see effects (other than playing the movie backward\(^2\)).

### 3.2 Data Structure Display

The next class of algorithm animation came with the following group of systems which displayed updated views of algorithm data structures.

#### 3.2.1 Incense

Incense [Mye83] was a prototype system developed by Myers at Xerox PARC in the late '70s. It was developed for the strongly typed language of Mesa, which allowed for easy

---

\(^1\)Very similar to top left picture in Appendix C

\(^2\)Played this way the commentary is not as good however
identification and display of variables. No change was needed to the code, and the variables you wished to display could be chosen at runtime.

For each datatype there was an associated format that displayed an appropriate picture of the datatype. It was supposed to have many formats to choose from, and offer the possibility of user defined formats, but in the prototype only one default format was available for each datatype. For each format there was an artist that was a collection of procedures and data (including memory address and type of data item) for handling the display of the chosen variable. The display of pointers and therefore linked structures used a special addition to the artist called a layout. This ensured that recursively defined structures would stay within the area of the screen defined by the user for the display of that variable by drawing referent structures smaller and smaller. Incense was not all that it could have been because it was never integrated with a full debugger. It was also disadvantaged by the speed of the code and machine when displaying data. Without user defined displays and the defaults only giving non picture representations of the variables, this system was more of a runtime data viewer than an algorithm animator.

3.2.2 PV

PV was a system which was to “Open the side of the machine and watch the programs run”. This was an ambitious project as the above quote reflects. They wanted to set up an automated Program Visualisation system (hence the name) for use in large programmes (circa $10^6$ lines of code) for use in of development, maintenance, and training with the DoD’s official language ADA. They state 10 different categories of visualisations for programs. Starting from system requirements to function and structure diagrams, protocols between modules, programs text and comments, diagrams of flow control, structured data, persistent data, and the program in the host environment. Of these, it is the diagrams of structured data which give, as they put it, “the most tractable and powerful approaches we have presented”, refering to movies such as “Sorting out Sorting” as examples. This was as stated in their first paper [HBC+82]. By 1985 [HBC+85] they had narrowed the focus of the project to that one category because of the challenge presented by the many forms that data can take and due to funding. The prototype was written in C, for C, under Unix. The area they narrowed their focus to is of interest because the animation of structured data is what we need for research into text compression algorithms. PV took unmodified code and compiled it with hooks that enabled it to get control in much the same way a debugger does. Dynamic graphics did not need additions to the code, but the user could “bind” variables to graphic objects either from a predefined template or by
defining their own.

3.2.3 GDBX

The Graphical Debugger, GDBX was developed by Baskerville [Bas85], and Brown's review of it [Bro88a] states that it was a production tool that ran integrated with the standard UNIX debugger, DBX, on a Sun Microsystems workstation. It was similar to Incense but did not have the generality envisioned in Incense. It differed from Incense in that instead of having dynamic data shrunk to fit the window, the referent links outside the capacity of the window could be viewed by scrolling the window both vertically and horizontally. It also supported changing variables at runtime, including the dragging of pointers. It is really only what it purports to be—a debugger—and only monitors data. It does not provide interesting displays that would give insight into the algorithm.

3.2.4 PROVIDE

Mohers PROVIDE [Moh85], also reviewed by Brown, used Macintosches as a graphical front end to a Vax 11/780. An extensive execution trace was stored on disk by running the program on the Vax after which the Macintosh was used to view the stored data via queries to the stored database. The user would specify what variables they wanted to animate before runtime. Again this was not an true algorithm animator because it only gave very simple direct views of the data.

3.2.5 GAIGS

Naps developed the visualization system for use in student instruction in laboratories [Nap90]. GAIGS3 could animate nine abstract data structures: one dimensional arrays, two dimensional arrays, linked lists, stacks, queues, binary trees, general trees, graphs, and networks. GAIGS reads in text snapshots of the data that are produced by prerunning the algorithm with tracer outputs in it like:

\[ \text{ShowStructure(StructureParameter,Caption)} \]

where the \text{ShowStructure} would output the data structure given by \text{StructureParameter} in the prescribed format. This text file input allowed the algorithms to be written in any language and on any machine, and it could then be transferring for GAIGS to show. Also noteworthy of this system is its reasonable display of trees and networks.

\footnote{Generalized Algorithm Illustration through Graphical Software}
Chapter 3. Algorithm Animation systems

It was originally coded for display on VAXStations but by this time will have been ported to PC-compatible and Macintosh computers. It supplies a low-cost visualization environment that is well suited to a student laboratory situation.

All the above data structure display systems are not well suited to algorithm animation as they can only monitor data. They could not show views of non-data structure information, such as the history of swaps made in a sorting algorithm, or the amount of wastage in bin space during binpacking, without special additions to the algorithm. They are also unsuited to multiple views of different algorithms.

3.3 Algorithm Animation Systems

The next group show aspects of true algorithm animation.

3.3.1 Toronto

These were special purpose animations of special classes of algorithms. They were developed at Toronto University in the mid 1970’s under the visionary for algorithm animation, Baecker, [Bae75, Bro88a] who produced the “Sorting out Sorting” movie. He and his students produced several specific purpose built algorithm animations and also developed several algorithm animation systems. These systems were developed to encourage the use of animation in computer science teaching and allow “short 'quick and dirty' teaching clips can be made with only a few hours effort”.

The hardware at this stage was not available to large numbers of people so movies had to be made. There were several systems produced at Toronto.

Yarwood This gave a picture display of PL/I variables but had only primitive graphics and was limited to monitoring a few types of data structures [Yar74].

De Boer This was used to produce movies of PL/I programs [dB74]. De Boer found that you had to add to the semantics of an algorithm to get good animation, rather than taking the route of easier simplified animation by automated methods. It has no ability to show multiple views. (These are described more fully by Brown [Bro88a])
3.3.2 Balsa

Balsa, developed by Marc Brown [BS84, BS85, Bro88a, Bro88b, Bro89] animates algorithms in a dynamic interactive environment. Balsa originates from a desire to bring the movie "Sorting out sorting" to life. Its interactive nature was influenced by Smalltalk, Balsa I having a lot of Smalltalk's windowing features, which is in contrast to Balsa II which is implemented on the Macintosh and therefore inherited the Macintosh look and feel.

Programs are animated by a programmer annotating the algorithm with "interesting events" which gives an abstraction of the program execution. From this, "views", implemented in modular fashion, now act in classical object orientated fashion (with the addition that some have both incoming and outgoing message protocols). A modeller computes a model based on the update messages passed to it, via an adapter, and a renderer produces a view window on the screen (see figure 3.1).

Balsa did not inherit Smalltalk’s Model, View, and Controller paradigm (MVC) but rather the end users interact with the algorithm through a view, but only when the algorithm requests data. Many systems influenced Balsa. Balsa I's feature (not included in Balsa II) of zooming in to see more detail of an algorithm and simultaneous views came from SDMS [Her80]. EXDAMS [Bal69] gave the idea for the illusion of multiple running programs. This important feature allows the dynamic comparison of similar algorithms. Balsa I preran multiple algorithms producing an execution trace for each algorithm. These were then displayed with elements from each trace being used in round robin fashion. Balsa II used cooroutines so there was no waiting for execution traces, but this took away the ability to run backward that Balsa I had. Balsa's structure of input generators, algorithms, and views is patterned after UNIX's notion of pipes and streams.

Displays in BALSA are non trivial to code as they require the generation of several communicating modules and renderers (the graphic routines, which are the hard part). Although coding and animating an algorithm that is functionally similar to one already coded is not as difficult for the modeler, and renderers are already there, you still have to cope with the work of getting the Macintosh interface right. In the author's experience this was too time consuming for the desired end result of quick exploratory animation. Balsa II is the most well developed and used algorithm animation system. It has most of the necessary features of a great algorithm animations environment. Unfortunately from the animator's side the Macintosh implementation has both a steep learning curve and a

---

4 Brown University Algorithm Simulator and Animator
5 This could be cited as a case of recursive algorithm visualization, a algorithm visualization that led to an algorithm visualization system
Relationship among components that a client-programmer implements to animate algorithms. Boxes represent components implemented by the programmer, and ovals represent parts of the algorithm animation system that are responsible for routing information among components. Solid arrows indicate unidirectional flow of information, and dashed arrows indicate bidirectional flow. The renderer associated with the view window having the keyboard focus is highlighted.

Figure 3.1: From Brown's book on Balsa [Bro88a]
Chapter 3. Algorithm Animation systems

 tediously slow turn around time for animation refinement. Although well suited for micro animations, it does not handle more data intense macro animations. As Brown states:

Our environment for algorithm animation is oriented to sequential programming in the small. It has been tuned for "algorithms" such as those found in a typical textbook or journal article. [Bro88a]

[Bro88a] or [Bro88b] should be referred to for a full description of this well thought out system.

3.3.3 Using Smalltalk

London and Duisberg at Tektronix developed an animation system that was totally Smalltalk based. It was also based to some extent on Balsa's ideas but used the MVC 6 paradigm with a special implementation of Smalltalk that was sufficiently fast to produce good animations. The Smalltalk system would notify all views of an object of any changes. The MVC paradigm was found to be deficient in several ways. Sequential notification of views meant that multiple views were noticeably not simultaneous. Composite views caused repeated redraws, and there is no ability for a view to converse with the algorithm, such as if a user wants to select and affect something. Tektronix was a useful development however, for it showed both the strengths and the limitations of Smalltalk. The result was the next animation system, Animus, also described by Brown [Bro88a].

3.3.4 Animus

After producing the previous animation system, Duisberg's Ph.D. thesis [Dui86] investigated constraint based animation to "describe the appearance and structure of a picture as well as how those pictures evolve over time". The temporal contraints allowed for simultaneous updates of different animation views, which was what the previous system lacked.

Animus also claims to be able to animate an algorithm without altering the algorithm code, by the animator choosing trigger constraints. This is a limited form of automatic animation, as it is dependent on how representative the messages in the Smalltalk code are of the action being performed.

6Model View Controller
3.3.5 Anim

Movie/Stills (or Anim7 was produced by Bentley and Kernighan from Bell Labs [BK87b, BK87a, BK91]). Anim uses filters to take graphic events and either produce a movie which can be viewed in the X windows system, or a series of stills which can be incorporated into a document. Their idea was that animations should be quick to produce. The output is limited but its open ended nature with no particular algorithm language required makes it available for many uses, and allows simple animations to be produced in a matter of an hour or less.

It was extensively used for the animations that will follow in later chapters and was found to be an excellent tool for exploratory algorithms animation. It has several features that should be part of all algorithm animation systems:

Ease of use The learning curve is shallow and, more importantly, the turn around time (design,code,view,design,...) was much shorter than other systems. The only place where this is not true is in the production of still images from large data set animations. The coding of the “stills” processor in C rather than awk would produce the necessary speed improvement.

Speed of display Anim can handle much larger animations than other more complex systems. This means it can be used for algorithm animation at a macro level (see section 4.1).

Ease of printing It is easy to capture the desire frames of an animation and convert them to good quality PostScript or TExXoutput (via groff or troff). Most other animations systems force you to spoil animations by output of bitmapped screen dumps.

Good animation control The clicks that define the possible stop points in an animation are easy to add, and when the animation is running allow the viewer to step between chosen algorithm events.

There are other features that are also part of Anim which are more common to animation systems, such as independent views and object manipulation (rather than graphical calls) which have been taken for granted by the author but would greatly increase user effort if not present.

Anim is a self stated “simple system” but is definitely useful for teaching, research and debugging8.

7The name that will be used in this thesis
8Bentley and Kernighan used Anim to find a memory allocation bug in Anim itself
3.3.6 ALADDIN

Hettula et al. [HHR89] designed a system based on Modula-2 that would let the animator free from the arduous task of programming the graphical output of algorithms animation, ALADDIN (ALgorithm Animation Design and Description using INteraction). The animation is graphically specified in an animation editor that is used as a syntax-directed Macintosh text editor when writing the algorithm code in Modula-2, and can also be animation specific allowing the animator to graphically show what certain parts of the animation should do. The animation specification is made up of: graphical items, which is constructed using a MacDraw type editor, graphical variables, specified by their type, and animation statements which are interactively described to the system by the animator.

Having the ability to easily design animations graphically is most attractive. This could lead to more synthetic views than more rigid systems will allow, but animations will likely be restricted mainly to micro algorithm animation, as macro animation cannot always be easily graphically predefined. ALADDIN was not trialled as part of this thesis so it not possible to comment more fully on its use (unlike other systems that are described here and by their use in animating text compression).

3.3.7 Xtango

Xtango, developed by Stasko [Sta90], is a general purpose animation system which gives smooth animation of algorithms—that is, it shows transitions as smooth changes rather than sudden jumps. This thesis uses Xtango extensively, and adds to the X windows implementation of Xtango, which was used for the animation of arithmetic encoding (described in section 4.3).

Like Balsa, and the Smalltalk system, algorithms animated with Xtango involve minimal change in the actual algorithm code other than placing ‘interesting event’ calls to the graphical animation part. Just as in Balsa these graphical parts are non-trivial to code, but as Xtango is a C source X windows system it has the quick turn around time for compile and testing that Balsa lacks. It also does not have the steep learning curve present in programming the Macintosh present for Balsa animations. Writing the graphical description of the animation is much easier due to the four data types that are included:

- locations,
- images,
- paths, and
Chapter 3. Algorithm Animation systems

- transformations

Xtango's location data type is a real valued logical coordinate into the viewing window that allows the window to be rescaled and have the animation automatically (even during runtime!) correctly adjusted.

There are a nine varieties of primary image types (including line, rectangle, circle, and text), and a composite type that allows the combination of any of the primary images.

Paths designate the magnitude of change in an image's attributes from one frame to the next. Paths are the only way to change an image. All transformations are performed along a path, whether that is moving an image, resizing it, changing its colour, or zooming in on it. The length of the path defines the smoothness of the transformation—a path of length one will cause an image to jump.

A large set of procedures are given to manipulate paths, and create paths of different types. This makes the animator's job of getting objects moving around much easier.

Transitions are the actual animation actions which affect images on the screen. There are eleven basic types: visibility, change fill style, change in colour, movement, resizing, raising, lowering, grab, delete, refresh, delay, and zoom. These can be iterated, concatenated, and composed together to form more complex transitions.

Xtango also has the added macro features of TWIST and TAP. TWIST will create for the animator arrays or groups (binary trees or graphs) of images and locations. TAP gives high-level access to certain Xtango calls for things like exchanging or making an image flash. This is a great ease to animator coding load.

Xtango provides natural primitives for describing actions in algorithm animation. It is an accessible system (the source is freely distributed!) and approachable by a variety of different users. It well suited to micro animations, but the author of this thesis discovered speed limitations on his computing system with macro animations requiring high-throughput.

3.3.8 Polka

Polka was implemented mainly for the animation of parallel programming algorithms. Polka is very similar to Xtango but has the added advantage of multiple simultaneous views, and is coded in C++ which brings the flexibility of being able to link both C
and C++ (as object orientated algorithms animations) algorithms. A basic version of the system has been released and continues to be worked on, with the exciting prospect of three dimensional program views becoming available. This is again available free via anonymous ftp\textsuperscript{14}.

### 3.4 The best algorithm animation system

There will always be improvements that can be made to an algorithm animation system, as with any piece of software. The production of a new algorithm animation system was outside the scope of this thesis, but by using several systems the following were found to be the basic needs and wishes of an algorithm animator.

- **Multiple simultaneous views** Dynamic comparison of different algorithms and different data sets is a powerful tool for algorithm visualisation.

- **Choice of runtime animation or data animation** Run time animation will eventually be limited by the amount of data that can be represented at a reasonable speed. Having the ability to animate preprocessed algorithm output files can aid throughput for algorithm investigation for research.

- **Automatic animation generation possibilities** Some algorithms (or parts thereof) are easily animated and the ability for the system to ask if you would like the algorithm animated automatically would ease the beginning process of animation design.

- **Libraries of graphic routines** For those algorithms that the animator wishes to show in ways beyond an automatic animation system (of which there are many), it would be helpful to have a large library of different animation transformations\textsuperscript{15}.

- **Animation execution control** The viewer of an algorithm should be able to specify algorithm events, and for the system to animate or step between. Balsa [Bro88a] has the best implementation of this, with even the ability to weight certain events so that they take more time, like for the situation where a comparison is performed using faster memory in one view than in another.

\textsuperscript{14} Email: stasko@par.cc.gatech.edu

\textsuperscript{15} As begun with Tango's TAP routines, as mentioned in section 3.3.7
Text manipulation It should be easy to be able to place, move, and pop up text as both objects and windows of scrollable information. The ability to easily manipulate text would be a great aid to text compression animation. It would also aid all types of instruction when added as help information that would pop up during the algorithm at the viewers command.

Stills output Eventually all animation systems need to be output in static form. There needs to be laser quality output available.

Quick turn around time Turn around time is the length of time and effort required by an animator to go from viewing a possible animation, altering the algorithm or animation, to viewing the new display of the algorithm. This should be as short as possible, for this cycle is repeated hundreds of times during the production of an animation.

Colour features The use of colour will definitely enhance the visual appeal of algorithm animation, as it adds a extra dimension. There are important principles that must be learnt by the animator for its effective (rather than distractive) use.

3D possibilities? Whether algorithms can be effectively presented using three dimensions is yet to be shown. It holds many problems and challenges but could lead to some exciting visualizations.

3.4.1 Choosing the best tool

Having described a wish list of what the best system would have, you are still left with the current possibilities mentioned in this chapter. To find the system that will suit your needs here are some simple guidelines.

Debugging For simple debugging use GDBX, or for more user control of what is animated, use Anim.

Animation for demonstration to students, next week! Anim is the easiest system to learn and use for simple presentations.

16 See Appendix C for some initial attempts
Research of macro detail of an algorithm Initially use Anim but you may wish to move to using scientific visualization tools, such as IRIS, aPe, AVS, or IBM's Data Explorer/600 if the animation data sizes are extra large. The book edited by Nielson and Shriver [NS90] is a good introduction to the scientific visualization field.

Smooth animation of algorithm objects The Xtango[Sta90] system is built around all elements being able to smoothly transfer. Polka is a cousin of Xtango that should be used if you require multiple simultaneous views.

A collection of different algorithms, on a well known platform MacBalsa [Bro88a, Bro88b] has most features of a great animations system, with the familiar Mactintosh user interface. You will need to purchase the source if you want to add to the many animations already shown. This would well serve an author wanting to have animations along with his text, simply by including the application along with the book.
Chapter 4

Text Compression animations

The reader should always take pen and paper and work through an example of each algorithm immediately upon encountering it in the text. Donald E. Knuth, “Fundamental Algorithms”, page 4

It is the purpose of this chapter to describe the animations of several text compression methods. The animations will be shown (obviously in static form\(^1\)) and the insights that these give to text compression, and algorithm animation, will be discussed.

Text compression methods are not easily animated. This is because text compression data structures do not easily lend themselves to being animated, and animating the actual process of text compression is hard to conceive simply. To illustrate this, consider a picture of text being fired through some sort of black box and the resultant compressed information being shown coming out the other side. This gives very little idea of how the algorithm works\(^2\). In fact if you are able to read the compressed result the reality is that the compression algorithm is doing a poor job. Even if the file was shown at varying stages of the compression process, again it would give little insight. It is inside the black box, and how this is affected by different inputs, that needs to be shown. This gives rise to a conflict over whether to show detailed algorithm steps or overall algorithm trends, which we will call the macro versus micro detail decision\(^3\). It is difficult, if not impossible, to do both at the same time. Therefore the animator must decide which is appropriate.

What follows are descriptions of the animations, the text compression concepts they represent, the systems used, and algorithm animation insights gained from producing text compression animations.

\(^1\)You can put your pen and paper away now
\(^2\)See compression by elephant figure 5.1
\(^3\)Discussed more fully in section 5.4

36
The first of the animation areas discussed is the model, which holds the most promise for advancing the realm of text compression.

4.1 Animating Models

The most important part of compression is the model used to compress the information. It follows therefore that it is the most important part of compression to animate. It is the model that dictates the compression that is possible for a given text. Given a model, producing the optimal text compression for that model has been solved to the extent that only implementation issues still exist. The modelling of text is not solved however, and cannot be solved in the same way. Therefore better understanding of models, how they form, and how they change may bring better compression techniques.

In this section we look at animations of simple models of text, and the details about the source text that are shown by them. Watching simple models animated affords a variety of interesting perspectives on the information content of files.

The following animations are macro animations. These will be discussed in more depth in section 5.4, but for these examples it will suffice to say that macro means we watch the model change rapidly (not step by step) as the input is processed, looking for peaks, exceptions, and general trends. But first some points of introduction to models will be considered.

4.1.1 Models

As an introduction to the model animations we will consider what models are and why they should be studied. We will specifically look at what the order-0 model is and what it looks like, and consider the possibility of views of higher order models. Also, in this section the Calgary corpus of compression files is introduced as the data set used for the animations, and the speed to display animations is considered.

What are Models? A model estimates the probabilities of characters. Models come in many forms but can be classified as:

- **static**, where the probabilities of symbols are fixed,
- **semi-adaptive**, where the probabilities are developed from an initial pass through the text, and then this model is the first thing transmitted before the actual file that it relates to, and
adaptive, where an initial model changes as the file is transmitted to adapt to the symbol frequencies of the text.

Unless otherwise stated any reference to model from here on will be to adaptive models because they are the most reliable at predicting symbols over different file types.

Consider the most basic model: order-0. It consists of the current frequencies of single symbols in the text. The following sections deal with observations gained by animations of this model. From these frequencies the probability of a symbol is calculated and from this probability a code for the symbol is generated.

Why study them? There are several reasons to study models:

• Models are the most important part of text compression because the model is the major factor determining the compression achieved by a method.

• Models themselves are not intuitively visual. They hold the information about the text, but in a different way to how humans model information. Paul Valere [Tu83] states “Seeing a thing is forgetting the name of the thing one sees”. This truth underlies the author's own ability to comment about this process. After seeing models—animated and static—they are no longer abstract, but visual concepts.

• Animated visual interpretations of the data will give new insights into what the information of the file looks like in the model, how the model develops during compression, how the model performs, and how the model adapts to the data.

What is the order-0 view? An order-0 model stores the probability of single characters being found in the text. It gives no contextual information, that is, it takes no account of the context that symbols occur in. But animations of order-0 models still show a dynamic picture of the file and gives us an angle on the information contained within the file. At its simplest, an order-0 animation progressively shows the frequency counts of the symbols in a file as the text is read. Model effects like count halving can be added to give other interesting text compression model views.

The order-0 frequency graphs show a new view of the text, as shown by the figure 4.1. They lead to observations about the information content of the file. They show more than just the character frequencies because they focus on how the content changes over time. They show interesting aspects of the files structure, and also indicate when change occurs. They allow interesting comparisons between texts to be made, the difference between language and picture to be observed, and the similarity of English texts to be examined.
A very interesting observation is that symbols seem to have a characteristic rate of increase which depends on the file content. Some symbols in object files are observed to have bursts of high frequency, while text symbols in files of natural languages show steady rates of change. This rate, or the rate history, could be blended with the character's contextual probability to give better compression.

The order-0 animations were not expected to give much information, and in many cases a static picture of the order-0 distribution of the file would give the same insights. For an ergodic source the final static order-0 model would show little less than watching the animation of it growing uniformly. However, it was the non-ergodic sources that showed dynamic behaviour that would not be picked up solely from the final static picture and gave some of the more interesting observations.

The order-0 animations show peaks at the most common characters, and areas of constant and fluctuating growth. They show the difference between files, and subtle differences between similar file types.

**Are higher order views possible?** Most text compression methods use models more sophisticated than order-0. This is because the context information revealed in higher order contexts leads to a better model of the information in the file.

Higher order animations are not discussed in great detail here as the processing power required to implement and design them was not investigated fully. But if the level of interesting observations generated by the order-0 animations is any indicator, it would be
well worth investigating designing animations of models of greater order than 0.

Order-1 animations were produced of selected characters. These took the same form as the order-0 animations but showed the character counts in the context of a particular character. Another design idea for an order-1 animation will also be described.

In general, showing higher order models is a difficult task. The amount of information required to portray them is much larger than for the order-0 animations and therefore the processing power to display them is much more. To go even higher than order-1 would require inventive and innovative thinking on how to display the frequencies as you are moving into four dimensional data. An example of algorithms display in three dimensions are the initial investigations done by the author and Tim Bell into 3D sorting visualization (shown in appendix C).

The corpus  The corpus used for experiments is a standard set of files collected at Calgary University by Bell et al. [BCW90] for the testing of text compression routines. It is described by the figure 2.3, 2.4, and figure 4.5 later in this chapter. It includes:

- English text in the form of books and technical papers, a bibliography and a batch of articles from Netnews,
- artificial language in the form of programs written in C, Lisp, and Pascal,
- a transcript of a terminal session,
- executable code, of a C program and a large Macintosh application,
- geographic data, and a
- black and white facsimile picture.

The corpus gives us a standard group of files to base our observations on and from here on they will be referred to by the above types and their short names (see figure 2.3 and 2.4).

It would be useful if text compression animators used a common corpus such as this one, so that valid comparisons between animations could then be made. The ability to compare different views of the same algorithm on the same data, or different algorithms on the same data, is a powerful visual tool. Balsa's ability to view several different sorting algorithms running with the same input data gives a good example of this (figure 4.2). For text compression the corpus provides that standard data base.
Chapter 4. Text Compression animations

Figure 4.2: MacBALSA algorithm animation system showing the worth of multiple simultaneous views for a dynamic algorithm comparison

The Calgary corpus is not ideal for animations. What may be required is that a text compression animation corpus be produced, published, provided and then procured by those wishing to produce algorithm animations of text compression algorithms. This corpus could be based on the original Calgary corpus (as are the animations in this thesis) with the files truncated to a manageable and representative size. It is also possible that extra files developed especially for use with text compression animations could be part of such a corpus—files that are both large and small are required for animations of different detail. These files should also have structures that could be expected to produce typical effects.

To show the adaptive methods at work there need to be some files that greatly vary in their internal structure. Today files often incorporate colour pictures, text, sound, and even animated pictures\(^4\). Compressing these documents requires adaptive methods. Adaptive modelling has been shown to be "only slightly worse than the best non-adaptive models" [BCW90]. So it would be good to develop files that put adaptive models through their paces and watch this via animation of the model, and look at the compression achieved.

Micro animations do not suit the use of the corpus as they often only show the processing of a few characters. They can be still useful however if a micro text compression

\(^4\)e.g. Apple Macintosh's Quicktime standard
animation shows the processing of a few steps from a specified interesting point within one of the corpus files.

But the great value in having such a standard corpus for animation is borne out by the usefulness of the Calgary corpus and other such corpora (for example standard images for image compression) for comparing findings of different studies and experiments.

**Animation speed** The model animation was originally tried using the Xtango [Sta90] animation system. This was soon found to be far too slow for any meaningful use. One bar increased in size approximately every half second. Xtango is an excellent animation system and allows for smooth animations, but the model animations require a high data throughput which Xtango is not capable of. The Anim algorithm animation system [BK87a] provided the necessary speed so that models showing files of 20,000 characters could be shown in approximately 45 seconds (dependent on the loading of the machine).

The differences in the speed of the animations due to machine loading raises an important algorithm animation issue. At what speed should animations be displayed? When is faster better than slower and vice versa? Brown [Bro88a] states that it has been shown in [Mod79] that information is lost if an animation is either too fast or too slow.

In the case of micro animations it is always better to go slower. Micro animations show in detail, step by step, what the algorithm does, and usually the viewer will obtain little information if they are shown fast. Macro animations, however, tend to benefit from being viewed fast. The viewer is looking for trends, not single events, so it is desirable that things change rapidly. This will probably mean that the viewer will want to view it many times to be sure that they don’t miss anything.

Another small point that should be mentioned in relation to machine loading is the effect of higher loads on animation interpretation. Viewers need to be aware of wrong interpretations due to machine load affecting the animation. The animation may seem to, stop and start, or go faster and slower depending on the time of day for example, but if macro animations are being used for research they will probably be viewed many times, and it is likely that the viewer will get to know what is part of the algorithm’s behaviour and what is due to machine loading (and this is the author’s experience).

**4.1.2 Animation of the order-0 model**

**Static visible file differences**

Figure 4.5 shows the static order-0 differences between the different files of the compression corpus. The actual animations are produced using the corpus files truncated to 20,000...
Chapter 4. Text Compression animations

bytes, as this creates a one megabyte animation file, and larger files than this are increasingly unwieldy to deal with. This size was found to be sufficient for many interesting observations. For most files this size was also found to be fairly representative of the total file. Figures 2.3 and 2.4 showed similar statistics but with character graphs for the most probable characters over the whole file. Each gives us an order-0 pattern of each file at the 20,000 character point. Each has distinctive peaks and lows, and even as static representations they show a lot about the information content of the files.

These observations are mentioned first because although they are not a function of the animation they are among the first things noticed during the animations.

The difference in file makeup shown by the order-0 animations is useful to visualise. It shows immediately those files with easily discernible structure and those where the structure and information content of the file harder to discern. This intensifies in the mind of the viewer the advantage of adaptive techniques that can handle these different file characteristics. It also highlights what we will call “text peaks”: clusters of frequencies that give the indication that text is present. Figure 4.6 indicates what is meant by “text peaks” in several of the corpus files. The “text peaks” observation is important as we move into discussing the animation aspects of order-0 models so that we differentiate between animation observations and static observations.

All the language files of the corpus, both artificial and natural, have the features of large peaks at the space character and the other generally most common characters in words (etaoinshrdlu). These peaks differ between files, and some may have strong occurrences of characters outside the most common, but the profile is still recognisable as a file that contains language data. It is a similarity that is noticeable in other language files (as shown in figure 4.7).

The profiles of figure 4.7 are similar enough for the viewer to know that language text is present in these files. These peaks are the natural character frequencies that have been know for years. They occur because of the structure of text, and its need for gaps between words which creates the large frequency of space, common use of vowels in words, and common consonants at the beginning of words.

These peaks can also show the peculiarities not only in a particular file but also of a particular writer. For example the prog1 program has bursts of “;” which this particular Lisp programmer used as a border character for a box around the names of each function(see figure 4.14).

Viewing files in this manner leads to many interesting observations. The Lisp program also has many parentheses, as a result of the way parentheses are often used in the
Figure 4.3: The order-0 character frequencies for the files of the Calgary corpus after 20,000 bytes
Chapter 4. Text Compression animations

Figure 4.4: The order-0 character frequencies for the files of the Calgary corpus after 20,000 bytes
Chapter 4. Text Compression animations

Figure 4.5: The order-0 character frequencies for the files of the Calgary corpus after 20,000 bytes

Figure 4.6: The order-0 character frequencies for four of the Calgary corpus files, showing “text peaks” circled
Chapter 4. Text Compression animations

Figure 4.7: The order-0 character frequencies for six non-english languages (collected from soc.culture USENET News groups) showing similar “text peaks”
language. Because of the requirement in Lisp that parentheses are matched it can be observed that they have exactly the same value for left and right parentheses, whereas progc, the C program, has more semicolons than progl, and the facsimile picture file (pic) is dominated by zeros, as most of the file is white space. The object file obj1 produced by compiling the progc program shows a peak at zero, but very little else of any textual structure except for the hint of it from the higher than average occurrences of the space character. The large Macintosh application obj2 is very similar to obj1 but text peaks can be observed. The small cluster of character frequencies shows the presence of words, which occur throughout the compiled code for menu items and dialog box strings, and other messages that are part of the application.

The object file, obj1 in figure 4.5, shows how there are only a few peaks and a wide range of values that have low counts, whereas the text from book1 has an identifiable shape with understandable peaks at the space and the vowel characters. These are the characters that typesetters have known for years as the glyphs they needed most of.

Other than the different word peaks, another even simpler observation is that the language files only take up half the range, whereas the non-language files obj1, obj2, and geo have occurrences across the full 256 values possible for 8 bit ASCII. This should lead to the compression observation that one bit per character is immediately unnecessary in the language files.

**Animation observations**

To stop overflow in the frequency counts text compression models will halve them after a certain number of symbols have been read (as shown in figure 4.8). This sometimes actually improves compression by making the model more locally biased in its predications. This effect on the model can be seen during an animation because the viewer sees the frequency bars being halved.

The effect on the model can be seen from the final picture of the model with, and without halving of the counts taking place. Figure 4.9 shows the difference between obj1 statistics when halving is performed and when it is not. This shows the local bias, for the character counts that betray the presence of words are not seen in figure 4.9(b) where the counts are not halved, whereas they can be noticed (by the fact that the other peaks no longer hide their presence) in figure 4.9(a) where the counts are halved.

Another feature that is apparent in animation is ergodicity. Ergodicity is the property of a stochastic process that is said to be statistically stationary. The Oxford English
Chapter 4. Text Compression animations

Figure 4.8: The book1 corpus file shown before and after a halvecounts has happened

Figure 4.9: Final frequency statistics showing the difference halving the counts makes to bias of the text (the range “etanoi” is more evident). This looked far more obvious in the animation as the “etanoi” range was seen to come rising out when counts were halved.
Dictionary defines it as having the property that the probability of any state can be estimated from a single sufficiently extensive realization, independent of the initial conditions. Bell et al. [BCW90] state that all state-models normally used in connection with natural languages are ergodic. That is, as we progressively develop a model from an ergodic text means it is more likely that is seen, the more that the probabilities of symbols change little; the model is said to be stable. An example of an ergodic source is the coin toss which as the number of tosses increases, the frequency of heads and tails will show the nature of the coin as having even probability for heads and tails. This will not change (unless you had some exotic device planted in the coin that could make it unfair later on), so it is ergodic. This property can be observed in the order-0 animations. All the corpus files based on language, either artificial or natural, display this to some extent, with the files book1, book2, paper1, and paper2 showing it the most. The letter frequencies stay constant relative to each other. The viewer doesn’t see races between different letters, but constant proportional growth. This does not prove that the source is ergodic, but indicates that it is likely to be. A source that is not ergodic, but that displays the same sort of constant steady growth in an animation, is the set of states generated by: (abc)\(^n\)(bca)\(^n\). This is not ergodic as the second half of the series does not have the same structure as the first, but it would look like it is ergodic in an order-0 animation. Figure 4.10 shows the steadiness of an ergodic source (book1).

Observing the English text frequencies animated immediately gives some idea of what is meant by an ergodic source. The text peaks rise out in the graph at generally the same rates (as shown in figure 4.10(a)). The space character is immediately noticed as it shoots out of the graph, the ‘e’, ‘t’, ‘o’, and ‘n’ all rise quickly to form that common form of peaks that occur when text is present. Book1, book2, paper1, paper2, bib, news, trans, progl, progp and progc all show this trend to some degree. A compression scheme should be able to make good use of such consistency. It also exists in obj1 and obj2 but is not easily spotted over the mess of seemingly random values that appear. By halving the counts frequently (every 4096 characters), as some compression schemes do, the text peaks become more obvious in these files (see figure 4.9). They occur where strings are stored in
executable code. These strings are the output messages of the program for example, and
the menu entries. In obj1 these occur in clusters at three distinctive spots (see the graph
of symbol probabilities for obj1 given in appendix A), in obj2 (the Macintosh application)
y they occur spread throughout the code and hence do not have the spurts of common string
characters as happens with obj1.

An interesting feature of the order-0 animation is the rate at which various symbols
grow. For example, in Figure 4.14 we see the Lisp program of the corpus. Over these
figures we see the ';' count advance rapidly and yet as seen by the final frame it is not
the most common symbol. What is shown is an interesting rate of increase of probability
that happens only now and then in the source. This observation is discussed more in
section 4.1.5.

All of these model animations, and especially those that show complex or varying
structure, reinforce the idea that adaptive text compression algorithms are preferable to
static methods. The animations of corpus files that are ergodic sources would seem good
candidates for some sort of static method by their uniformity, but it should be noted that
what is shown are only order-0 models and higher order model representations are likely
to show a more dynamic nature of these files.

4.1.3 Higher Order model display

The previous results raise the question of what the higher order models might show us.
The difficulty is displaying them in a non-confusing manner that shows us the trends in
a few pictures. The method chosen was to display interesting contexts only, so they look
very similar to the previous order-0 model displays, but instead show the probabilities of
characters in the context of a certain character or group of characters.

The initial thought that order-1 model of characters should be shown with a three
dimensional display does not really hold much promise. Characters could make up the
x-y axis and the frequency counts the z. However ordering of characters in the alphabet is
highlighted by this display, yet it is not relevant for textual files (although may be relevant
for some non-textual files, like images).

Overall a three dimensional display will not give a good view of the information in a
text because digram pairs that relate to each other will be clustered with those that bear
no relation to them.

Figures 4.11 shows the display of digrams as bar graphs of the probability of a character
occurring after a particular character. For example, the bar graph in the top right of
figure 4.11 shows the probabilities of characters occurring after an “e”.

In this way we can choose the characters we wish to view and be able to see simultaneously updated views of each of these character's successors as the file is progressed through.

The *obj* file from the compression corpus gives one of the more interesting animations (as shown in figure 4.11). Unlike the book and paper files, the source is not noticeably ergodic, yet half way through has some textual information. It can be clearly identified in the digram display when this textual information occurs in the object file. Textual information like this should be compressed better than the other parts of the object file.

How you choose the digrams to view is dependent upon the file type and so an initial viewing of the final order-0 model will be useful when choosing the diagram distributions to display. But you are still faced with a huge number to view and this becomes exponentially worse for higher order contexts. A better way of displaying higher order contexts needs to be found.

A possible idea for Order-1 animations comes from figure 4.12 (from Tufte [Tuf83]). This clever graphic gives some hope for possible order-1 animations.

The same type of diagram could be used with order-1 models, but instead of the time axis you could have each of the occurring first character contexts, and across the page, instead of light frequency, the letter frequencies that occur following each first character. Just like the example (figure 4.12), it would probably be best if all the x-axis frequencies were drawn as function lines rather than bar graphs for this would mean that lower plots would not be entirely obscured by higher plots that were in front. Just as in the example you could have the sum of the statistics on the right and an average at the top which would be the order-0 model.

The viewing of order-1 was not entirely tested due to the processing power required, but they hold much promise for those who animate them.

### 4.1.4 Adding the compressed file size to animation

Figure 4.13 shows one of the bar animations with a extra view, showing the compressed file size versus the size of the input file. It is simply the number of characters read versus the number of characters output, for order-0 compression. This experimental view could be useful when used with output from several different modelling methods to show a running comparison of the compression achieved. Of course this sort of graph does not need to be animated, because because drawing the graph during the animation gives no more information than the final static view. But when put along side the model animation there should be some interesting observations, especially when the compressed file size
Chapter 4. Text Compression animations

Figure 4.11: Several different digrams views for obj file
Narrowband spectra of individual subpulses. Each point of the intensity $I_c(t)$ plotted on the right is the sum of the distribution of intensities across the receiver bandwidth shown in the center. At the top is plotted the spectrum averaged over the pulse. In the limit of many thousands of pulses this would show the receiver bandpass shape.

Figure 4.12: Figure from Tuft's book [Tu83], which could be an example of how an order-1 model's information might be displayed.
Figure 4.13: The order-0 model animation with an extra view showing the compressed file size versus the total file size.

graph shows several different compression methods.

One should be able to observe when one compression scheme is compressing better than another, and how particular events shown in the model affect the compression. Events such as sudden spurts in the occurrences of certain characters, or a change in the file's characteristics, like in the obj1 file where there are bursts of occurrences of strings, are observable. These events are noticeable in the model animation window and should cause effects in the compression comparisons graph.

This animation could also lead to further study on the ability of humans to model information. Bell et al. [BCW90] states that the most reliable estimate for the entropy of natural language texts as being in the range of 1.25 to 1.35 bits/symbol, from Cover and King's experiments [CK78]. Yet the best computer compression method only achieved 2.25 bits/character, for a natural language source from the Calgary corpus (book2 using DMC from table B-1 in Bell et al. [BCW90]). An interesting change to the graph would be to have the vertical axis being the entropy in bits/character and have the entropy that a human modeller gives the source against the entropy some computer model generates. This along with the model animation, and a view of the text itself could lead to some interesting discoveries of where a human model beat the computer.

Another use of this animation would be to see if humans can spot things in the model animation that lead to compression.

4.1.5 Research Idea

Tufte, when talking about graphical integrity states:
"At the core of the preoccupation with deceptive graphics was the assumption that data graphics were mainly devices for showing the obvious to the ignorant. It is hard to imagine any doctrine more likely to stifle intellectual progress in a field" [Tuf83]

This same doctrine can affect the use, and perceived value of algorithms animation. Not only can algorithm animations be used for student instruction, they can lead to research progress.

Probably the most important result of the model animations are that they produced a good idea for further research. We will call this idea 'rates', because it is all about rates of change. The idea is that the rates of change of probability of symbols could be a source of further information about a text and therefore lead to better compression. This idea is not researched by this thesis and its importance as an idea for compression may not be great (time has yet to tell), but the idea came from watching animations of text compression models.

The following is a description of the rates idea, where it came from, and an initial study of probability rates. It is described to fully outline the idea but the overriding goal of this section is to show the importance of visualization for the text compression field and more generally for computer science research.

Most recent study of text compression has involved the use of probability models [BCW90]. Models hold the key to good compression. The closer the model is to the actual structure of the text, the better the compression. This can be difficult to achieve because the original model that created the text may be the thought patterns of some writer.

Compression modelling methods developed so far seem to be static in nature, like looking at the speed of a driver at many different points along the way in a trip and then trying to say what sort of driver that person is, text compression models look at the text at a series of instants. Even constructing complex models of what speed the driver goes at after certain events, like seeing a flock of sheep on the road, or seeing a very large stop sign, will still not give the full picture that the rate of change of speed, or acceleration, will add. Instead of just trying to predict the speed from what the current speed is, we predict the driver's speed from the rate the speed is changing and possibly from previous rates of change that were of similar gradient. Rates in text compression may be less obvious, but they do occur, as shown by the initial study of this idea.

From observing animation of some of the following model animations it was noticed that some symbols occurred in spurts. These spurts were noticed to occur at a certain rate.
This lead to the question: if this rate of change of probability was somehow included into the model, would it give better compression? Or do current models already achieve this through their contexts and blending of contexts? These questions are yet to be answered.

The observation

It was observed that the frequency of some symbols would suddenly spurt up and then stop. This spurt would then occur again later. This was most noticeable in the animation of the progl Lisp program in the Calgary compression corpus.

Figure 4.14 shows several frames from the animation of the order-0 model of progl before and after the spurts of activity.

The rates are even more obvious when the probabilities of the symbols are plotted as the file is read, as shown if figure 4.15. This graph shows the order-0 statistics for the "e", ";", and space symbols (as showing all the characters would cause information overload). The order-0 probability statistics are the probability of that character occurring up to that point in the file. This gives a view that will show the rate of change of probabilities of characters.

These distinctive spurts of semicolons has a very simple explanation. In the Lisp code of progl the comments put in by the programmer are made up of a lot of semicolons! An example from the program is:

```
;;; Named for the buttons on radios in which only one is "in" at a time.
```

This would seem to say that the rate of semicolon activity is a unusual and does not relate to "ordinary" files, but this a wrong view to hold about adaptive text compression. An adaptive model of the source should be able to model such occurrences and in fact it is found that interesting rates occurred in many of the files of the Calgary corpus, as shown in appendix A. All these graphs show the statistics for the characters (the graph of pic being the exception) in the set {space e lf(linefeed) t a n o ( ) ;}. These were chosen as they are some of the most likely characters in English text and when they occur it tends to be a indication that the file does include English text at some point.

The graphs in appendix A hold a lot of interesting information about the rates that probabilities of symbols change at. For example, the graph of objl shows three distinctive bursts of textual activity (shown by the increase in the probability of all the common text characters) where the strings are stored in the executable file.
Chapter 4. Text Compression animations

Figure 4.14: The beginning and end frames of the bursts of ";" activity in the progl Corpus file
The rates idea needs to be implemented and tested against current modelling techniques before it will be known whether it does improve compression. Most importantly, as a result of an observation from a animation this goes some way to show that text compression can be advanced through animation. This animation was at the macro detail level and it is from this level that the author feels the most interesting insights into text compression will come.
4.2 Ziv-Lempel animation

Ziv-Lempel coding achieves the best compression of the dictionary coding schemes. The concept behind dictionary encoding is simple and the animation of it gives a good example of what a micro animation can do compared to blackboard scribbles. The animation teaches the basic concept of LZ coding,

"replacing [a] repeated string by a reference to [an] earlier occurrence" [Whi67].

The LZ animations are good examples of micro animations. They show the step-by-step points of the algorithm, only a small amount of data is processed during the animation, and they show details of the algorithm that are known and understood, not new insights. The animations show the point where the search is up to in the dictionary window, the longest match so far, and it tells you what each output is. All this adds up to an animation that is useful for instruction of students but gives no extra insight to a researcher.

An important aspect of these animations is that they are interactive. That is the user can step slowly through the animation or watch it run through, and Anim's clicks feature allows the user to step through the animation between different points of interest. The viewer can try different input strings to see the effect of the algorithm and the output.

**Anim control**

An advantage MacBalsa and Anim have over Tango is better animation control for the viewer. Most animation systems allow you to step through interesting events. In Anim each different stop point is referenced by an animator inserted "click" and in the LZ77 animations the viewer can easily step through the detail of the algorithm via these clicks. Each click corresponds to a different interesting event that is annotated in the algorithm and is stepped between by a mouse button press.

The LZ77 algorithm the three types of clicks are:

**Search** which steps through string searches,

**MatchSoFar** which steps through string matches, and

**OutputTuple** which steps through output events.

The LZ78 algorithm has one click:
Node which shows each successive node being placed in the trie.

Each click can be turned on or off via a menu in the animation window. This allows the viewer to decide what events to step between. Figure 4.20 shows all 25 frames of a short animation which are the frames a viewer would see if all clicks were turned on. Figure 4.17 shows frames from the separate click “search”, which is what you would see if just that click was turned on.

Anim also allows the viewer to step backwards allowing them to rewind and review points of interest.

4.2.1 LZ77

The first form of LZ compression published in 1977 by Ziv and Lempel [ZL77], LZ77 has a fixed size window within which pointers denote common phrases. A phrase can be up to as long as the lookahead buffer. The compression proceeds as shown in figure 4.17 which are frames from the animation. Figure 4.16 shows the point just after it has encoded the string ababaa as the tuple, \((2, 5, a)\). This means the longest match found was “ababa” which starts at an offset of two from the lookahead window and is five characters long, shown by the thick line above, and has an ‘a’ as the first character that did not match the substring in the window, which is shown by the line below.

The animation shows:

- the window of characters for possible string matches
- the lookahead buffer
- the search position
- the longest matching string so far in search
- the longest match
- the last output tuple.

The animation has several features:

Window of characters

As the characters are read they appear in the animation with a number between zero and nine which denotes their position. Initially the whole lookahead buffer is filled with characters, and from then on characters are read to fill the right end of the lookahead
buffer as strings are matched. The numbers below help the viewer understand the tuple output by giving a visual way of working out the matching string length, and number of characters the matching string is offset from the lookahead buffer.

Lookahead buffer

The lookahead buffer contains the characters not yet coded, shown by a enclosed box in the animation. The previously coded characters, shown to the left of the lookahead buffer, are searched for matching substrings in the lookahead buffer. LZ77 has a greedy parsing scheme which searches for the longest match and when this is found a tuple is output, and the lookahead buffer moves forward by the length of the match. This can be seen in frames 1, 2 and 5, 6 of figure 4.18 which shows the lookahead buffer moving forward after the longest match has been found.

Search Pointer

The search pointer, which is the arrow pointing down on a letter, shows where the algorithm is up to in its search for strings that will match the lookahead buffer.

Using the click feature of Anim the algorithm was annotated so that it output a click named "search" each time the algorithm moved its search position forward. This means that an observer can step through each search watching how the algorithm matches up the strings. This is the lowest level of click added to this animation.

Figure 4.17 shows the first 15 search steps. They show the frames a user would see if they stepped through just the Search click. They are numbered for the following description:
Figure 4.17: The search steps of the LZ77 animation

Search Step 1: The first character has just been encoded as a zero length string with an ‘a’ at the end, as there are no characters to match it with yet.

Search Step 2-5: The pointer is again at the beginning of the window as the next string has just been encoded (as shown by the two thick lines). From 2-5 the search pointer move forward through the available characters looking for the longest match.

Search Step 6-15: The pointer begins again and searches through the available ten characters for the longest match.
Figure 4.18: The match steps of the LZ77 animation

Match Lines

Match lines come in two forms: Firstly as a line showing the longest match so far, shown above characters of the earlier occurrences and below the matched characters in the lookahead window. Secondly they come as a thicker line showing the final longest match, depicted the same way as just described. Figure 4.18 and 4.20 show these match lines.

Figure 4.18 shows the frames a viewer would see if the view stepped through between each of the MatchSoFar clicks. Frames Four to Six show the three matches found that are longer than the previous. These matches give the viewer a sense of the way the algorithm searches and what is found so far. Figure 4.18 shows the longest matches found and the tuple output. In the animation this event is highlighted by the way that the match line becomes thicker when the final match is found.

Tuple output

As strings are encoded the output tuple is displayed. This allows the viewer to check the diagram and understand what the tuples values mean. Figure 4.19 shows the four output events of the example.

The output event is the highest level of click that a viewer can step through. This allows the viewer to step through the animation watching the different output events and watching for different effects. Effects such as the length of the matching strings generally
increases, and the length of matching strings decreases when strange input occurs, may be seen.

Figure 4.20 shows all animation events as they happen.

What this animation needs

This is an example of a micro animation, not a finished teaching tool. There are many features that could be added. Some possibilities are:

- More levels of clicks for the lower levels of searches made for matching strings.
- Pop up text that tells you what each part of the animation is.
- Multiple lines of text or scrolling text that shows the size of both the lookahead window and the already encoded characters.
- Multiple lines of text or scrolling text that has the ability to handle larger files.

4.2.2 LZ78

LZ78 is the second major family of adaptive dictionary compression schemes. Introduced in 1978 by Ziv and Lempel, it has a distinctive difference to the LZ77 family. LZ78 parses the text into phrases, which are the longest match so far plus the new character. Each phrase is given an index number and the text is encoded as tuples of phrase number and new character. A major advantage of LZ78 is that the phrases can be stored as a trie. This means that searching for the longest previous phrase is just a matter of following one arc of the trie and then you are also at the location where the next node is to be
Figure 4.20: The full LZ77 animation
added. Figure 4.21 shows frames from the animation showing the encoding of the string “aaabbabaabaaabab”.

This animation has similar features to the LZ77 animation with the distinctive addition of the phrase trie. The other difference about this animation is the possibilities it exposes for a line of research into the discovery of trie development. Longer phrase tries are shown to expose some of this possible interest.

The animation shows:

• the encoded string
• the match lines
• the tuple output
• the trie.

Encoded string

Unlike LZ77, LZ78 has no lookahead window or window for possible string matches as there is no restriction on how far back a pointer can reach. Therefore the animation shows the characters as they are read. If the click “node” is chosen then the characters appear as they are encoded (phrase, and next character) as shown in the figure 4.21.

Match Lines

Above the encoded characters are lines which denote the matched phrase and next character. As in the LZ77 animation, the matched phrase is the thick line above the character of the phrase and a thinner line at the end showing the next character. This, along with the tuple output above it, emphasizes what the algorithm is generating. New characters stand out as single thin lines.

Tuple output

The output tuples appear above the match lines. They are the longest matching phrase number (which the viewer can match with the number of the last node that flashed on the trie) and the next character. Their positioning at the end of the match line shows at what point the tuple is output.
Figure 4.21: The LZ78 animation
Trie

The trie is the most important concept to grasp when encountering LZ78. The trie (as shown in figure 4.21) shows the storage structure for phrases. The simplicity of Anim and the difficulties involved in showing multiple-way trees made showing just tries involving an alphabet of two characters the best option. The seven frames of Figure 4.21 show character by character development of the trie. The viewer can see at each character read what branch is being followed as the nodes flash, and can also see the new branches added as new phrases are added to the phrase trie.

The animated display of phrase development also holds promise for further study. The figures B.1 to B.3 in appendix B show larger tries that were developed from longer encoded strings. The study of how a trie develops could be well aided by such animations. These figures, which are far from perfect, provided interesting views of trie development. The first (figure B.1) shows the encoding of a single string “abaa” repeated many times. It shows how a trie can get lopsided in such a case. The second (figure B.2) shows the trie made up of a string of coin tosses with the peculiarity that the third toss is always the same as the previous. This in effect means that the first toss is one head or tail and the second and third tosses are head, head or tail, tail respectively. Figure B.3 shows even more frames from the development of a trie for a file made up of ‘a’, ‘b’, and ‘bb’ with the probabilities of 0.5, 0.3, and 0.2 respectively.

It is not part of this thesis to analyse the following trie development, but it is shown in order to provoke interest in the possibilities for further study. Like figure 4.21, the figures shown in appendix B are not good for studying the close detail of when a particular branch is added, but rather give an overall picture of what is happening to the trie. We are moving into the area of the macro view, and hence the shift from a simple student teaching tool to a research orientated study.

Discussion

Originally the matching phrase was denoted by as the characters of the phrase shown above the encoded character string. This was confusing as several lines of information were being shown and it was hard to distinguish each from the other. This illustrates two features of algorithm animation: firstly that graphics need continual refinement, and secondly that using differing visual cues can aid the viewers understanding, by presenting the information in several ways.

The viewer has three visual and four textual cues to the workings of the algorithm. The visual cues are: the trie, which shows where the nodes are added, the match line
Chapter 4. Text Compression animations

with thinner new character line at end, and the positioning of output tuples which shows when the output occurs. The textual cues intermingle with the visual ones and are the encoded characters, the output tuples, the trie node numbers, and branch characters. This combination of different textual and visual cues is important for comprehension.

Crosby and Stelovsky's [CS89] experiments suggest that textual and graphical presentation media differ in the way they facilitate understanding of algorithms. By textual they mean showing the algorithm program text, which would be good as another view. But their result also points to the possibility that more textual information added to an animation may improve the understanding of the graphic.

The display of trie development could hold some interesting discoveries for those who develop more sophisticated animations. This animation only had the ability to show binary trees.

4.2.3 Other findings

Both of these animations (and the animation of arithmetic coding) highlight the need for better animation text handling. In Anim it is difficult to produce a sliding window of characters because elements are always scaled so they can be seen. The animation window shows the entire cast area. This means that difficult manipulations have to be done to produce the text events required for text compression animation. Anim is a good test bed animation system though, and its advantages of easy animation production and animation control make it a useful animation tool.

The LZ78 animation may have been better suited to Xtango where binary trees are handled automatically, but to show LZ78 more fully would require an animation system with good multi-way tree handling. Incorporation of clever tree node placement algorithms into an animation system would also be useful to many other algorithm animations.

These and other micro animations could be improved with inclusion of the program text. Crosby and Stelovsky [CS89] found that comprehension of graphics improved with prior exposure to the program in Pascal. The inclusion of the program text—maybe in a pseudo code form—would help the viewer move between the detail of the code and the visualisation more easily.

4.2.4 Displaying animation to fourth year class

The previous two animations were used in a lecture to honours students on dictionary encoding schemes. After observing the animation they were encouraged to try them out. This was in no way an empirical study with measurable results, but it was valuable
in that it showed the value of interaction with people not involved in the animation’s production. Elements of the animation that were instructive and those that were confusing were quickly identified. Such a group of testers is an necessary part and critical feature in the development and design of an algorithm animation.

4.3 Arithmetic coding animation

Arithmetic coding is a provably optimal way of encoding a given model of a text. The concept itself is at first difficult to comprehend but is actually not complicated. Those teaching it will always resort to diagrams of ranges with expanding subranges. This has the obvious shortcoming of being static, which while perhaps being understandable when first being ‘animatedly’ explained, may prove difficult for the student to remember and understand later.

An animation has many advantages over static display. It can show differing data from several different view points, that can be watched and rewatched, and with explanation of the views along the way can give a far more lasting impression of what is happening. The animation of arithmetic coding is such an animation. It is a micro detailed animation that shows how arithmetic coding works at the symbol by symbol level. The use of the Xtern animation system allowed smooth animation effects well suited to the needs of this animation. Several different animations give the different views and seeing different symbols encoded, and this is just a matter of what file you choose to run with the animation.

This section describes the micro animation of arithmetic coding. Its initial purpose is to describe the animation, its features, and what could be added. But the main purpose of this section is to describe the lessons learned by producing these animations, especially the second animation described as the “zooming” animation. There are also some interesting style ideas and discoveries about algorithm animation that were made.

Animations often arise out of static diagrams, and it is a good place to start for ideas in an animation. The figure in [BCW90] labelled 5.3 (shown as figure 4.22) is where the original idea for the arithmetic animations came from. Decimal ranges with subsections that are shown to expand to become a new range are the most common way of presenting the idea of arithmetic coding, like figure 4.22. The initial range is the entire half-open interval from zero to one, written as [0,1). As each symbol is coded the range is reduced
Figure 4.22: (a) Representation of the arithmetic coding process; (b) arithmetic coding with the interval scaled up at each stage

according to the probability of that symbol that is given by the model\(^7\). This continues zooming in, producing a decimal fraction that represents the entire message. For a full description of the encoding and decoding shown in this figure from [BCW90] see the notes at the end of this chapter.

To animate this method took more than just showing the value of some of the variables in the code. The decimal fraction is not part of the actual implementation of the algorithm as the low and high values of each range are implemented using integers, so that it is easier to do the arithmetic.

Concepts that we wished to be communicated through the animation were:

\(^7\)See figure 2.1
• that there is a range that is ever decreasing in size,
• that each character causes the range to decrease,
• that codes are sent incrementally, and why this is possible,
• that a character may not cause any bits to be sent, or may cause many bits to be sent, and
• that arithmetic coding is suited to adaptive schemes of modelling.

How these concepts were included into the animations of arithmetic encoding is discussed in section 4.3.1.

These are micro synthetic animations. They show arithmetic encoding to a high level of detail. Such animations are useful for teaching. They are an example of an animation that would be very hard to automatically generate. This is because the concept of expanding subranges is not structurally part of the data structures involved but a concept used to describe it. Using Brown's [Bro88a] attributes of animations\(^8\) this is described as a "synthetic" view. The input files for these animations need not be longer than ten characters because the detail shown is such that the viewer will be overloaded (if not the animation system) if more are used.

The animation of arithmetic coding went through several stages, and introduced several interesting animation problems. The progression of the animation also shows that there is a lot for animators to learn as they produce animations. What looks good and what is appropriate for the algorithm are difficult to decide objectively.

4.3.1 Progression of creation

It is useful to view the approximate progression of the creation of this particular animation, for this is also the order that would be used when introducing these concepts to students. What follows is a brief description of the three animations and the different things they show. The second animation is the most detailed and most of the section 4.27 is a discussion is of the algorithm animating problems relating to the creation of these animations.

The following discusses the three animations named:

**disappearing ranges** the ranges seem to disappear as they get too small to see

---

\(^8\) See section 5.4 and figure 5.4
zooming ranges each character read causes the range associated with that character to be zoomed into

expanding ranges each character read causes the range associated with it to expand

Disappearing ranges Early in the animation process an animation of the ever decreasing ranges sizes was shown using subdivided blocks (see “disappear ranges” in figure 4.23). As each symbol was coded (the animation used has ten possible symbols from @-I) the subrange block was divided into 10 other subrange blocks and so on. The animation shows one simple point—that arithmetic animation deals in ever decreasing ranges.

Lines better than blocks

Blocks do not associate naturally with ranges so the animation was developed such that it showed ranges! These are lines capped with shorter perpendicular lines. This was the initial step of the most involved arithmetic animation called “zoom”, which is shown in figure 4.23(b). Without a zoom in transformation the ranges quickly became so small they could not be seen.

It is here that XTango was augmented to. It has user controls for zooming and panning small amounts, and animation transformations for objects, like move, resize, and fill but no way of performing an animation controlled zoom. The author of this thesis added this feature to Xtango. The implementation details of zooming are discussed in section 4.3.3. There is no other known algorithm animation that smoothly zooms in like this and so there were no guidelines to how zooming should operate.

The animation itself (see figure 4.24) always shows the entire current range. When a new character is read the associated range is smoothly expanded up so that it is centred in the viewing area. Its vertical height now zoomed up to be nearly the full height of the animation window (a small amount is left at the top and bottom so the cap ends can be seen). As shown in figure 4.24 there is also textual information displayed along with the ranges

Text display

Animations describing an abstract concept like arithmetic encoding need textual information, or they are hard to interpret without someone along side the viewer explaining what is happening. The textual displays in the “zoom” animation were added to show:

---

9 A couple of other possible applications have been thought of, see section 4.3.3
10 Having someone alongside giving a running commentary on an animation would always be handy
Chapter 4. Text Compression animations

Figure 4.23: The three arithmetic encoding animations

(a) disappearing ranges
(b) zooming ranges
(c) expanding ranges
Figure 4.24: The "zoom" animation showing a series of frames during a zoom
Chapter 4. Text Compression animations

Figure 4.25: The "zoom" animation showing the incremental bit output phase

- character value of each range,
- the integer bit low and high ranges, and
- the incremental bit sent.

The figure 4.25 shows how the animation of the bits being transmitted. The low and high ranges have been moved next to each other to aid comparison. It shows that the left-most bits of the ranges that are the same are transmitted (there are other bits also transmitted which are due to underflow and overflow but the animation does not attempt to explain these yet).
Decimal fraction

Once the viewer has seen the "zooming" animation and the zooming feature has been understood, then moving onto the expanding ranges view should be informative. It gives both a history of all the expansions and shows the decimal fractions that each range correspond to. Figure 4.26 shows frames from the animation with the expanding range shown as a box. The rectangles of increasing size are not permanent in the animation but are put in these static pictures to give an idea of what happens during the animation.

4.3.2 Principles of animations discovered

The animations just described did more for the author's understanding of the intricacies of good animation than they will probably do for any student watching them. The major effort needed to get the smooth zooming of ranges is repaid by the question afterwards "does that actually show very much?" What it has shown is that the animation process
must be focused on a particular group of viewers, in this case either researcher or students. The researcher will usually want to see things in the large (Macro detail), while students usually will need to see things in the small (Micro detail)\textsuperscript{11}. The outcome of this decision then leads to a host of animation decisions about what is to be shown, what is important to show, and how to show it. When thinking of an algorithm like quicksort these descisions may seem trivial. It is easy to know what to show and how to show it. But for more synthetic animations it is more difficult, because algorithm data does not relate closely to the animation elements that are to be shown. These “small” decisions take a lot of time and testing. Here follows an itemization of some of the seemingly trivial decisions that need to be made in the production of arithmetic animations.

**Macro or Micro view**

The detail to which the animation will be shown is the initial decision that the animator must make. This Macro and micro detail is discussed more fully in section 5.4. This was discovered from animating arithmetic encoding and being faced often with a conflict of end goals. The goal of describing the concepts of an algorithm to students conflicted with a goal of researching behaviour of the algorithm for large coding tasks. Both could not be performed simultaneously so that the viewer could view both. Micro detail was finally chosen for these animations.

The principle here is: *the detail of an animation is driven by the needs of the end viewer.*

**Number of ranges displayed**

It was discovered early on that showing even a reduced character set of thirty symbols was too many and that reducing this further aided comprehension. The arithmetic encoding animations show ten character ranges from “@” to “I”. This enables the animation to run at a resonable speed, and more importantly reduces the amount of information in the animation for the users to digest.

The principle here is: *reduce extraneous data to the minimum required to still portray the concepts of the algorithm.*

\textsuperscript{11}The content (i.e. whether direct or synthetic) and persistence (current, or history) do not relate what type of viewer you have, see section 5.4
Chapter 4. Text Compression animations

Figure 4.27: The "zoom" animation from the top showing the ranges getting closer together as they go in

Changing distance moved due to scale

In the "zoom" animation each character read caused a "zoom in" and a new range to be moved to the right of the previous (see figure 4.24). It was found that the amount a new range expansion was moved right of the last had to decrease in relation to the amount zoomed or the previous range would completely disappear from the view, and therefore make it easy to miss the fact that ranges are related, and getting smaller. Figure 4.3.2 shows this by manual zooming back out after the animation.

The principle here is: when zooming (in or out) try to keep previous parts of the animation in view so that visual cues of the change in scale are retained.

This is related to Brown and Sedgewick's finding that "small cases" require a history of execution [BS85]. In the zoom animation the full history of previous expansion can be seen by pausing the animation and using the zoom out button.

Centring problems caused by zooming

The original version of the zoom transformation only approximately centred in on a particular object's position. This was unsatisfactory as a viewer would not know where to look next for the actions of the algorithm. This was improved by making the zoom transformation always smoothly locate a given object in the centre of the animation window. This will not always be possible for all types of algorithm animation, but for micro animations, like these ones it should.
Chapter 4. Text Compression animations

The principle here is: In micro animations, make it obvious where the viewer should look.

The problem of Animation junk

Tufte in [Tuf83] describes the concept of chartjunk—the decoration of data graphics. He concludes that:

The overwhelming fact of data graphics is that they stand or fall on their content, gracefully displayed. Graphics do not become attractive and interesting through the addition of ornamental hatching and false perspective to a few bars.

There is a related problem in animations. Animation junk is a problem of enticing animation effects, which you start to become aware of when you extract yourself from the detail of animating (of which the previous text is a good example). The niceties of animation, the zooming boxes and expanding lines, can become as Feynman [Fey85] puts it, a disease\(^\text{12}\). The animator can be caught by the fun of making objects fly to and fro and forget that this is supposed to portray something. The very animation itself can become more confusing than explaining, but it will be fun to make and watch.

The animator needs to continually ask how well each element of the animation relates to the algorithm. There are elements that weakly relate to the algorithm, and those that have a strong relation to the algorithm. The bit output of the zooming animation is an example of a weak relation. The zero’s and one’s zipping across the screen are only portraying the incremental output, and in fact really should show chunks of zero’s and one’s being output and the actual movement deemphasised. The zooming in on a range is an example of a stronger relation. It is important in that it keeps the actions of the algorithm visible and it emphasises the fact that the range is getting progressively smaller. But here also we have to beware, because the zooming effect itself should not be distracting.

The second version of the zoom transformation caused a swooping effect where the desired centre point would at first move slowly towards the centre of the screen as the zoom in took place, and then towards the end of the zoom, “swoop” in quickly. This is quite an interesting effect of the same sort of scaling problem as mentioned in section 4.3.2,

\(^\text{12}\)Richard Feynman describes the first sort of computer type programs using a series of card calculating machines inflicting the “programmers” with a disease that would get them working on programs for calculating arctan tables just cause it was fun to do, not because they actually needed arctan tables.
but probably injurious to the concentration of the viewer. This was improved so that the display panned smoothly during the zoom.

The principle here then is: *animation elements should relate to algorithm events.*

**Finding principles**

There are more principles than are mentioned here and there are parts to algorithm animation that are hard to find principles for, such as determining the speed of an animation, or deciding how much information to display in one view. These require specific testing.

An essential phase for algorithm animation that is destined for the instruction of others is that it is tested to see if it conveys the correct information. This will identify both the small elements of the animation that are important and those that confuse. It will help the animator focus his efforts on those parts of the animation that are really important. It will also lead to new principles of algorithm animation to be found. This is especially needed by algorithm animators. It is the development of such principles specific to algorithm animation that will lead to advancement of the field.

**4.3.3 New features for Xtango**

Additions to Xtango were a response to the difficult animation of arithmetic encoding. Its expanding ranges required zooming. Initially this was performed by the viewer with the zoom and pan buttons, but what was needed was an algorithm-driven transformation. This section introduces the two features added, pan and zoom, discussing possible other uses of zoom, and also introduces some improvements to Xtango.

**Change of pan**

The pan buttons in Xtango moves the window around the viewing space. They were improved so that the amount panned was relative to the amount zoomed in. This was necessary for the zoom animation magnified the viewing space by a large factor for each character read. If the user wanted to pause and pan around then they would find the pan would take everything away! This was a simple alteration but emphasizes the fact that zooming was never thought of in XTango [Sta90] to the extent to which it has now been used.
Chapter 4. Text Compression animations

The zoom transformation

No other algorithm animation system had smooth zooming before the author added it to the Xtango animation system. The addition of it was not complicated as Tango is a well thought out piece of software, and this is to the great credit of John Stasko [Sta90]. The latest release of the Xtango system now has this feature included along with several other useful features, which are described in section 3.3.7.

All transformations in Xtango occur along a path of offset values. A move is just a matter of progressively relocating the given object by the values of the offsets. The zoom transformation is the same but the offsets are the scale factors for the x and y dimensions (usually the same). This allows stretching and distortions if desired but probably more usefully allows for zooms to increase and/or decrease during the path, on speed up and/or slow down during a zoom. Like other transformations, a zoom path is given an object to operate on and in the case of zoom the location of the given object will be the centre at the end of the zoom.

Implementation details The first version of zoom relocated the window such that it was centred at the given object’s location, after which it zoomed straight in on the object. This meant that the first transition along the path caused the window to suddenly jump to the new location. This can be confusing, and it goes against the principles of Xtango, that of providing transformations that are smooth and continuous.

The zoom transformation was then developed to zoom and pan simultaneously towards the given object’s location. This transformation had three forms; each with different characteristics:

1. John Stasko’s
2. Swoop zoom

John Stasko’s zoom function After discussing the idea for the zoom transformation with John Stasko, Xtango’s creator, he came up with the following function for the relocation of the windows coordinates at each step of the path. The equations 4.1 and 4.3 show how the right and left coordinates of the window are scaled according to the objects location, x, and the zoomfactor.

\[ rx = rx - ((rx - x) \cdot \text{zoomfactor}) \]  

13 Transition-based Animation Generation
This way of zooming is smooth but has the problem that it only tends towards being centred at the given object’s location. This was not satisfactory for the arithmetic encoding zooming animation because after several ranges had been zoomed in on the current range “tends” not to be in the viewing window. The advantage of this method over the next two is that it does not require us know the number of steps in the zoom path, which means that it can be composed together with all the other types of transformations that Tango offers.

The author did calculate a function for determining what offset from a given object’s location can be given to John Stasko’s function such that it will end up centred at the object’s location, but this again required knowing the number of zoom offsets in the path. It was also more difficult to implement than other methods due to the fact that the order of the equation to calculate an offset varied with the length of the path. A much simpler implementation that produced pleasing results was as follows.

**Swoop zoom** This is called the swoop zoom because it seems to swoop in along a curve, and this was simply the combination of a zoom and a pan. Given the number of offsets in the zoom path it calculated a panfactor for the x and y axis that was the distance between the current and what would be the new centre divided by the number of zoom steps. This caused the swooping effect because as the view is zoomed in the panfactor stays the same while the coordinate system of the entire window becomes smaller so the pans look like they are increasing in speed. This zoom transformation could be of use in some animations in the same way that xtango has moves that go along circular paths rather than straight so that objects can seemingly go around one another as they swap.

**Square root** This swooping effect was stopped by panning by the square root of the amount of the transformation done multiplied by the length of the total pan required. Equation 4.3 shows the calculation of a panning amount.

\[
x_{\text{pan}} = x_{\text{pan total}} \times \left( \frac{\text{framenumber}}{\text{TotalNumberFrames}} - \frac{\text{framenumber - 1}}{\text{TotalNumberFrames}} \right)
\] (4.3)

What is not part of the animation transformation is the calculation of how far to zoom. Xtango has path finding functions that given locations will construct a path to move between those locations, and given an example path will make a new path for an object to move along that will look similar. But there is now path finding functions for
the path of scale factors used during a zoom. At the moment it must be calculated by the animator. The calculation is given by:

\[ i = \frac{\ln x}{\ln y} \]  

(4.4)

where \( i \) is the length of the path, \( x = 1 - \text{ScaleStepAmount} \), and \( y \) is the desired length that will be fully zoomed up at the end of \( i \) ScaleStepAmounts.

Zooming could have many other uses for algorithm animation. The animating of the DMC method of text compression modelling could use zooming for zooming out as the number of states increased beyond the ability to show them, or for zooming in on particular parts of the models development to see where and when states clone. Numeric methods that involve convergence would also be candidates for zooming. Simultaneous views (one zooming in, the other constant) of the Newton-Raphson method of root convergence would portray the concept of this method well.

Idea for step menu button

Xtango's animation control would be much improved by a top menu pop up that had each of the 'interesting events' in it and the ability to step between them as Anim has with its clicks.

4.3.4 Concept animation verses interactive animation

The production of the arithmetic and Ziv-Lempel animations, and exposure to the Macro­mind multimedia product with its animation possibilities raised the distinction between two types of animation—concept and interactive. It also raised the question: Are the arithmetic animations anything more than a concept animation that could be more easily explictictly graphically specified in a multimedia package such as Macromind.

Concept animation is like a video tape. It can be rewound and rerun many times at differing speeds but it will always be the same. Interactive animation is algorithm portrayal that allows exploration.

Concept animation is the sort of animation you see with a Macromind animation. That is, it is an animation of the concept, and could successfully portray the workings of an algorithm but is not changeable. Interactive animation is animation that comes straight from the algorithm running.

Concept animation is a fixed animation that is designed. An animator sits down with the animation system and designs each frame of the animation. It maybe a sophisticated system that allows the animator to define just beginning and end positions from items and
Chapter 4. Text Compression animations

the computer then does the 'inbetweening'. The animator decides what the animation looks like completely. The animator's understanding of the concept determines how it will be animated.

Interactive animation is generated from the running of the algorithm. The views that are shown of the algorithm are decided by the animator. The input data can be changed and the animation reviewed. Parameters in the algorithm can be adjusted and they affect what is seen in a subsequent animation.

A concept animation of LZ compression would be look very similar to the animation produced with Anim and discussed in section 4.2. It could have the advantages of popup dialogs about different stages of the LZ pattern matching process. The viewer could step through or run through the animation. But the option of re-running the animation with different input string or a different size lookahead buffer would not be possible. This would be only possible with a interactive animation system like Anim or Xtango.

For the animation of algorithms, interactive animation is best. It gives the flexibility required and the ability to produce animations that give new insight into the workings of the algorithm.

The concept animations like those made using the Macintosh multimedia product Macromind show animated diagrams for teaching and promotional purposes (figure 4.3.4 is an example). They can be of the form of MacBaby, an animation of the physiological processes of a child's development in the womb, or technical drawings with moving arrows showing the flow of water. They are just diagrams with movement and are not interactive animations. They are purpose built for simple teaching and presentational settings.

True algorithm animation provides an exploratory setting. The viewer can be passive, but does have the ability to actively change the animation's variables and inputs. This is well suited for student laboratory settings and for researchers. The student watching an animation for the first time would see very little difference between a concept animation and a true algorithm animation environment\textsuperscript{14} but from here on the similarity ends. A concept animation raises questions in the student's mind, and this is where the true potential of animation begins. The student can begin exploring the algorithm that has been introduced, and this will reinforce concepts portrayed visually by seeing them over and over with different inputs and parameters.

Interactive animation gives flexibility that concept animations do not have. They can be rerun with new parameters and differing inputs. Once you have produced your concept animation you are stuck with it. Despite the speed of display and the ability to slowly

\textsuperscript{14}Balsa's scripting facility could be conceived as a concept animation
Chapter 4. Text Compression animations

Figure 4.28: Macromind animation of the concept behind the operation of a solar water heater
step through, it is this flexibility that caused the author not to use a concept animation system. It may have been possible to produce more “attractive” looking animations, but one is soon frustrated with the lack of flexibility.

4.4 Future directions

I have concluded from the production of these animations that the process of algorithm animation is a lot more complex than one would first think. Brown [Bro88b] says:

Implementing the algorithm involves, for the most part, annotating the code with interesting event markers.

This is given that the input generators and views are available from a library, but in the present time for text compression algorithms there are no such library routines for views and input events. The production of such graphical views was found to be very difficult. This was also found by Brown who described this part of algorithm animation as “the hard part”, as it involves a tight loop of refining the animation, compiling, and viewing it. This may be a significant part of algorithm animation but the most important is that these algorithm events are displayed in a fashion that increases understanding.

The animations of the models of the data identify a need for more in-depth studies of all the different possible view statistics that can be produced for compression. The LZ animations show how an easily accessible system, such as Anim, can produce useful micro animations, and the arithmetic animations illustrate the need for good principles of animation along with good graphics design and critical analysis.

4.4.1 Three Dimensional Animation

The bottom four images in appendix C show a new idea for the static display of sorting algorithms. Using three dimensions we see the sorted location of items, as well as the iteration dimension. Three dimensions and colour (giving four) is the requirement of data intensive visualisations. The production of instructive three dimension algorithm animations could be exciting, or just plain confusing (as some think these three dimensional pictures are).

The second page of the appendix C also shows a static view of sorting algorithms where the sorted values are put into a table at each set of stages of the algorithm and then loaded into a common graphing software which affords a new view of the three sorts that shows all the particular characteristics of each type of sort. We see insertion sort shuffling the
colours into the final sorted order and quicksort swapping elements around in partition steps.
Chapter 5

Animating: So what do you want to see?

"There is no particular mystery to animation....its very simple and like anything simple its about the hardest thing in the world to do." Bill Tytla, at Walt Disney Studios, June 28, 1937

Over recent year several accessible algorithm animation systems have become available. Discussion has centred on what makes a good animation system, and not on what makes good animations¹. This is understandable as you need the "tools" before you can expect to develop any "talent". This chapter begins discussing the area of animating algorithms, and how to define the animations. It is a more general discussion of animating, leading on from the specific difficulties and discoveries about animation of the previous chapter.

The animator faces several important decisions before starting to code the animation, the most obvious being, what do you want to see? This can be harder than expected, depending on the algorithm and the detail and content desired in the animation. This chapter discusses the difficulties of animation, and investigates whether this animation process be automated?

5.1 An animation system doesn’t make animations

People's creative nature is still required to produce good animations. While some animations may possibly be generated automatically these will still be very simple direct representations of data structures in the algorithm. People are indispensable in the loop

¹[BS85] being an exception
of animation production, for without outside interpretation of the purpose and function of the algorithm it is difficult, if not impossible, to produce views that are anything more than just data structure views. These are useful, but "synthetic" views, of which there are relatively few examples, could produce animations of better quality and greater information content. Synthetic views are those which do not have a direct mapping to any program variables, and therefore are not part of the code itself but must be synthesised by the animator with the extra knowledge that they have about the algorithms function.

The animation systems described in chapter 3 have many good features but they still require an animator—someone who will annotate the code, design the graphics, and refine them to the stage where they can be comprehended easily.

The animator requires:

- understanding of the concepts to be animated,
- to have studied ways used previously to describe the concept,
- to have outside influence critiquing the animations produced,
- data graphic design skill, and
- creative thinking to produce interesting insight giving animations.

**Understanding of the concepts to be animated.** It is essential that the animator of an algorithm understand fully what they are animating. They may be producing a research animation that is investigating the characteristics of an algorithm not yet fully studied, but this will probably require greater understanding of the fundamental concepts of the algorithm, not less. To animate with incomplete understanding is sure to lead to incorrect representations of the algorithm, especially when the animation's content is of a synthetic nature. Animations where the content is direct (see section 5.4 and figure 5.4) will still be difficult to animate without understanding the fundamental operations of the algorithm.

**Studied ways used previously to describe the concept.** Most of the animations produced in this thesis began as static graphics. For example the idea for the arithmetic expanding range animation (described in detail in 4.3) came from a figure from the book "Text Compression" [BCW90]. From this starting point the zooming animation eventually developed. Algorithm animation of a static diagrams from books has great appeal,
especially to the authors of the book, for then they can include a disk with the book to bring the algorithms to life for the reader\textsuperscript{2}.

**Outside influence critiquing the animations produced.** Testing that an animation conveys the right information an important stage in the refinement of an animation. The best is an empirical test, with a good range of possible users, [CS89] and [Hay88] are examples of such testing.

**Data graphic design skill** There are principles of good data graphic design that can be applied to algorithm animation. Books like Bertin's "Graphics and Graphic Information Processing" [Ber81] and Tufte's "The Visual Display of Quantitative Information" [Tuf83] and "Envisioning Information" [Tuf90] have a wealth of useful principles for static data graphics that can be usefully applied to algorithm animations. For example, the book "The Visual Display of Quantitative Information" has important principles in relation to the concept of data-ink in graphics, which can be summarized as:

- Above all else show the data
- Maximize the data to ink ratio
- Erase non-data-ink
- Erase redundant data-ink
- Revise and edit.

These can also be applied to the production of algorithm animations, the algorithm events are thought of as the data. Other principles of Tufte's are mentioned throughout this thesis. Such principles as these have been developed from years of study of good and bad data graphics and are ignored at great peril.

**Creative thinking to produce interesting insight giving animations.** This means an ability to dream up new views of algorithms. Depending on the detail, content, and persistence (described in 5.4) required, an animation may require varying levels of creative talent. Animations that lie in the synthetic, macro, history region will require much greater effort and creative skill than direct, micro, current animations (again these terms are defined in section 5.4).

\textsuperscript{2}MacBALSA, [Bro89] is closely linked with Robert Sedgewicks "Algorithms"[Sed88] to the point that the entries in the "Chapters" menu refer to the chapter of the book and the algorithms they contain.
5.2 Algorithm animation design is difficult

Algorithm animation design has several difficulties. Much effort, expertise, and many hours put into the production of an animation still may not add to the understanding of an algorithm, and worse may even give a false impression.

There are few empirical studies of student comprehension improvement using algorithm animation. It has been found that by using structured charts to introduce relatively complex algorithms the comprehension of at least 25% of students could be increased [Sca88]. This is hardly algorithm animation, but is a useful pointer to the improvement that graphics can bring to presenting ideas. Hays [Hay88] found from studying the 103 beginning computer science students that the use of interactive graphics for devising algorithms, combined with simulation of the student’s attempted solutions, can be used effectively to assist students to discover solutions to novel problems. Crosby and Stelovsky [CS89] produced some of the most interesting results with two tests involving students viewing an animation and a textual definition of program. They found that graphical presentations only improved the comprehension of a student when preceded by a view of the textual definition. This would seem the reverse of what was expected. This probably relates to the type of animation, theirs was a micro detail animation of sorting, therefore the viewing of the micro detail explained by the textual information of the program code before actually seeing the animation is understandably better. For macro detail animations this result is unlikely to be true, but it does point to the how all animations can be improved by the inclusion of textual information.

Crosby and Stelovsky second experiment involved using a eye movement monitor to give insight into the users viewing strategies. This also gives interesting results, but more study like this is needed. The eye movement monitor has great possibilities as an animation refinement and testing tool. These sort of findings are important for they indicate to the animator how particular animations should be shown to make best use of the information they portray—graphically, with interaction, and with textual data.

Algorithm animation is conceptual in nature. The animator has to form a visual representation of a non visual algorithm. Algorithms by nature are conceptual in that they deal with non-physical items. Animations of physical phenomena, like the movement of a sail boat, are not difficult to produce (once you have the data). Algorithms, on the other hand, are not naturally visual. They require an animator with an understanding of the fundamental operations of the algorithm to produce a visual representation.

Text compression is a good example in that the processes that go on to produce the compression have no physical shape. It means little to show a compressing document, like
Chapter 5. Animating: So what do you want to see?

Figure 5.1: Text Compression by elephant, nice to watch but not informative

You need to express some of the mind's picture of the algorithms process, for algorithms are generated in minds, not out where you can see them.

Producing pictures of algorithms abstract ideas is hard work. Brown states that:

A major proportion of our effort went into determining precisely how the various data structures should be presented graphically [BS85], and that writing and debugging these graphical presentations is the hardest part of the animating process. BALSA-II [Bro88a] allowed these graphical views to be shared by diverse algorithms, which should mean that you progress towards the situation that new algorithms will not necessarily need any new views created.

Annotating the algorithm is the easy part of animation. Annotating an algorithm comes naturally from an understanding of the algorithm, and a mental picture of the concepts it portrays and you as the animator wish to communicate. Interesting events seem obvious when you know what you want to be displayed.

The process of algorithm animation production is summarized in figure 5.2. The process begins with the animator bringing together the algorithm code, the concepts wanting to be portrayed about the algorithm in the animation, and maybe some static graphic that have been produced to explain the algorithm. Moving into the initial design phase, data graphics design principles should be used. Out of this design phase should come a story board—a high-level sequence of sketches showing the structure and ideas of the animation. This is used in conventional animation. The storyboard could be used effectively during algorithm design, so that the what the animation will hopefully look like is visualized. Once with a story board of the animation the animator can proceed more readily to coding the view with its objects and movements. This then leads to the implementation problems like: Will making an object that is bigger than another give the wrong impression? How fast should we zoom in on that field? Where should that text be displayed? How much should we add to this view before it is overloaded? These are the type of questions faced. This phase of development in an animation's life cycle requires
Chapter 5. Animating: So what do you want to see?

Figure 5.2: Life cycle of algorithm animation production

much trial and discussion, as shown by the refinement part of the coding, evaluation stage in figure 5.2.

There is often a temptation to go for easy methods of animation that may suit the particular animation system but give a wrong impression. An example of this was the beginning stages of the arithmetic animation that used boxes for the ranges. This was an easy way of producing the animation but the later zooming animation used lines which actually looked like ranges. The same sort of temptation could be found with the reusable views that BALSA offers, in that you use a view which nearly fits your requirements, rather than code a new view that would be more correct.

Algorithm animation in its new, accessible form is a young discipline and there is still much to learn, even though it has been around for about two decades. Baecker states:

Computer scientists, however, have produced very little computer animation. In the first decade, ['65-'75] the technique has been used to produce only two computer science teaching films [Bae75].

There are still few practitioners, and few who are pushing the limits of animation. There
are now plenty of systems to do algorithm animations, and we now need those who will learn how to produce good algorithm animations. There are lessons to be learned from the cinematography animators, and also from the new area of scientific visualisation [NS90]. These may have much to offer the macro animator looking for the quirks of algorithms in the "large".

5.3 Algorithm animation to static diagram

Tufte states:

"The world is complex, dynamic, multidimensional; the paper is static, flat. How do we represent the rich visual world of experience and measurement on mere flatland?" [Tuf90]

This text has an obvious problem, related to this, in that it portrays animation with static diagrams. These cannot ever give the same sensation as the actual animation.

The flip book is one of the original forms of animation. This was experienced in the 1800's via machines where a coin was put in the slot and a handle cranked so that the cards were shown consecutively. Each card had another frame of a series of pictures showing the movement of people or things. The encyclopedia Britannica describes the reason that this produces the illusion of motion:

...this is based on the optical phenomena know as persistence of vision and the phi phenomena. The first of these causes the brain to retain images cast upon the retina of the eye for a fraction of a second beyond their disappearance from the field of sight, while the latter creates apparent movement between images when they succeed each other rapidly. Together, these phenomena permit the succession of still frames on a motion picture film [or in an algorithm animation] to represent continuous movement when projected at the proper speed. Britannica 1989, 24, p379

This ability of the mind to perceive motion is put to use by algorithm animations to improve understanding, as algorithms are dynamic by nature and therefore animating them is a useful way of understanding this dynamic nature.

The series of pictures on the reverse pages of this thesis show the animations of several algorithms. This is "low-tech" animation, but is still a marked improvement on the single static frame approach. This idea was reinforced as a possibility after seeing the book "Kayak" by William Nealy [Nea90]. In this book about kayaking techniques the top
corner of each page had a hand drawn picture of a person in a kayak doing an eskimo roll, so that when the pages were flipped through you could observe, and learn how to eskimo roll. The flip pictures for this thesis were generated by the Anim and Xtango animation systems and then fashioned by \LaTeX. It has been observerd during the preparation of this work that animation is superior to static graphics. Although a well designed, data rich, static graphic will lead the viewer to delve and explore, it will not be able to grab the attention as ananimation can, and involve the immediate question: “Why does it look like that?”

The way that animation is portrayed statically most frequently involves the use of small multiples, which are many frames from an animation placed in rows on a single page. Tufte states this as follows:

At the heart of quantitative reasoning is a single question: Compared to what? Small multiple designs, multivariate and date bountiful, answer directly by visually enforcing comparisons of changes, of the difference among objects, of the scope of alternatives. For a wide range of problems in data presentation small multiples are the best design solution.[Tuf83]

For an example of small multiples see the figures in appendix B.

This would seem like the easy answer, just produce a lot of frames from the animation\footnote{This itself maybe very difficult and time consuming in some animation systems that do not have a stills production feature} reduced so they fit on one page. This introduces next problem of going from dynamic to static display\footnote{This has been encountered often by the author.}. Although the obvious approach of just printing frames of the animation would seem sufficient, it is not always, and in fact probably never satisfactory. Frames that are seemingly cluttered as static representations do not look so cluttered when animated because the clutter is moving.

This is an important fact for the animator to be aware of for. Animations may need extra refinement for presentation statically. The animations portrayed statically in this thesis are mainly taken directly from the dynamic display, and have not been refined for static display, other than putting axes and labels on some figures.
Chapter 5. Animating: So what do you want to see?

5.4 Defining animations

5.4.1 Macro and Micro detail

A major distinction found between the different animations described in this thesis is the detail described. Some are at a "micro" level of detail, showing individual steps of the algorithm, while others, like the model animations, show a higher level of information and could be classified as "macro" animations.

This was found to be indicative of the purpose of the animation. Animations made especially for students tended to be of the micro detail type, like the LZ77 and LZ78 animations, whereas when researching an algorithm's behaviour, macro animations are more suitable because the researcher wants to see more global effects, not fine detail. The model animations are of this second type. There may also be points in between the far ends of the scale, or even mixtures of the two, which may prove useful.

5.4.2 Brown's 3-D definition

Brown classifies algorithm animations in using three dimensions [Bro88a]. Content, Persistence, Transformation, as shown in figure 5.3. This is a useful way of classifying animations. The axes are described by Brown as follows:

The content for the displays ranges from direct representation of the program's data to synthetic images information not necessarily inside the program. The persistence dimension ranges from displays that show only the current state of information to those that show a complete history of each change in the information. The transformation dimension ranges from displays that show changes in the pictures discretely to those that show incremental and continuous changes.

This is helpful for classifying an animation. Animations where the content is direct will be easier to produce, like the sorting animations. Synthetic animations, like the arithmetic coding animation, are more difficult to produce and therefore more time consuming. The two different views of arithmetic encoding (zooming\(^5\) and expanding\(^6\)) show the two extremes of the persistence axis—the zooming animation only showing current information while the expanding blocks show history. The transformation axis extremes can be seen in the difference between the smooth zooming of the arithmetic encoding animations and the discrete additions of trie nodes in the LZ78 animation. The type of the transformation

\(^5\)see figure 4.24  
\(^6\)see figure 4.26
Chapter 5. Animating: So what do you want to see?

is more of an attribute of the animation system used and not really (in the authors view) an attribute of the animation itself (therefore I propose an alternative for this axis). Also, an animation can have either discrete or incremental transformations, the more incremental just meaning the smoother the animation. Comparing between discrete and smooth animations is useful but comparing the smoothness of animations is very subjective, and not a very useful as a defining axis.

5.4.3 Alternative 3-D definition

A more useful third attribute of animations is shown in the figure 5.4 and labelled Detail. This defines how “close” you are to the algorithm. Macro level animations are far above the algorithm observing global effects and trends. Micro level animations show step by step instances of change. Micro level animations will often be improved by incremental transformations, like the smooth zooming on the next range in the arithmetic animation. But macro detailed animations will not need smooth transformations because many discrete events will be happening over a short period (although if the computer was fast enough it should not be important whether they were incremental or not, and in fact if they are incremental is better then you can latter zoom in on detail and see smooth transformations)

The zooming feature of Balsa-II is described on page 54 of [Bro88a]. It is a scroll bar on the left of the window which allows the viewer to interactively decide on the level of detail they wanted from the animation. Brown states that this zooming was indeed micro zooming for it not only made the view bigger but “causes more information to be
Chapter 5. Animating: So what do you want to see?

Figure 5.4: Alternative attributes of dynamic algorithm animation displays along three axis

displayed." This is the only reference to the animation principle; that different levels of detail give different information to the viewer. It is an important principle because it is one of the important initial decisions the animator has to make.

5.5 Algorithm animation versus visual programming

Algorithm animation is distinct to visual programming, and as Baecker states [Bae86] animation is a form of program visualization, which is:

the use of the technology of interactive graphics and the crafts of graphics design, typography, animation, and cinematography to enhance the presentation and understanding of computer programs. Program visualization is related to but distinct from the discipline of visual programming which is the use of various two-dimensional or diagrammatic notations in the programming process [Bro88a]

Visual programming is more of an aid to program construction than animation. It is purely data structures that are animated and the concept behind the animation is still in the programmers head. It could be seen as a extreme micro sort of animation.

5.6 Automatic animation

The idea of automatic animation is very attractive, but is very unlikely to be achieved. As stated by Brown:
Chapter 5. Animating: So what do you want to see?

... an algorithm’s operations cannot be deduced from an arbitrary algorithm automatically but must be denoted by a person with knowledge of the operations performed by the algorithm. [Bro88a]

Automatic animation relies on the premise that the algorithm and data input and output give all the information that is needed to visually portray the events in an instructive way. This is generally not so and in general such automatic systems produces very limited results. Brown describes the problems involved in automatic algorithm animation generation [Bro88a].

Algorithm animation is an attempt to portray things which may never have been fully visualised even by their creators. A good idea produces code that produces an interesting result, but how it happened in detail and particularly what is happening, at a macro level is often not known or understood. Also people forget how things work or how they came to solutions. The author has been told that there is code for in sophisticated graphics system commented with:

/* Please do not alter, as even the creator is not sure how it works*/

To somehow automatically generate a picture of that which is not even fully understood and visualized by its creator, and give informative displays is very difficult at this time.

5.7 Metamouse

Metamouse [MWKF91, MW91] is a learning graphical drawing program, which could possibly have an application for algorithm animation. Metamouse follow the user and predicts what their actions will be from watching what they have drawn so far. Metamouse will take over if its prediction is deemed correct by the user. An example that comes with a video about metamouse showed metamouse learning how to order blocks by their height, which looks very much like an animation of sorting. If metamouse could be taught this then what about other algorithms? Could it be taught about the expanding ranges that are arithmetic encoding? Quite possibly.

This isn’t really automatic animation but animation in reverse. Instead of starting with code and producing something visual it is starting with a visual environment and teaching the system the algorithm visually. This has exciting appeal. To take this to an extreme, perhaps we could produce algorithms visually that haven’t been coded yet. This is artificial intelligence and machine learning’s domain—getting machines to learn the way people do things.
From the algorithm animation point of view it also has interesting possibilities. Teaching metamouse to carry out a process means that eventually metamouse will animate that process for you. It would not be a animation of a “standard” algorithm but an animation of a metamouse user generated algorithm. Nevertheless this could animate algorithm concepts. This sounds easier that it is likely in practice. Teaching metamouse to do things is not easy. Teaching it to animate algorithms could be very difficult.
Chapter 6

Algorithm Animations and teaching

To these four young men God gave knowledge and understanding of all kinds of literature and learning. And Daniel could understand visions and dreams of all kinds. Daniel 1:17

The goal of algorithm animation is that it should improve the understanding of the algorithm. Whether this is teaching a beginning computer science student, or a researcher being the first to learn about unknown characteristics, the goal of algorithm animation is the same.

An animation is not really finished until it has been tested in a teaching situation. This is an acid test of its whether it is actually informative.

This chapter describes the various ways algorithm animations can be presented.

6.1 Flip book

This is not in common usage but on the reverse sides of this thesis it is brought to life for the first time as an algorithm animation viewing system. Cheap, and available without sophisticated technology, this new version of the original hand-wound system of flick cards\(^1\) makes a brief reappearance.

To use the flipbook, simply hold the thesis firmly with one hand and use the thumb of the other to flick quickly through the pages\(^2\). Try it at varying speeds, and with the book placed on a flat surface rather than held. Three types of algorithm are depicted:

\(^1\)From which the movie theatre got the nickname “the flicks”
\(^2\)If you are reading this on a medium such as microfiche, a little imagination may be required
• Quicksort
• Bubblesort
• Order-0 model of book
• Order-0 model of obj
• Gerty (the elephant) compression

Quicksort and bubblesort race off against one another with quicksort finishing before it looks as if bubblesort has even started. The two order-0 models develop beside one another, showing the first 20,000 symbols of each respective corpus text, and Gerty does a magical job of making text compression fun to watch.

This flipbook idea could be developed further with aspects such as colour, and maybe even a mechanised way of flicking the pages at a constant rate!

6.2 Video/Movies

When the author was at school, an afternoon during the week was often devoted to films and videos that would bring the study of some subject to life. For example when we plodded through Shakespeare’s Macbeth, we watched the interesting screen production by Polanski\(^3\). Bringing algorithms to life was the original purpose of algorithm animations\(^4\) [Kno66, Hop74, Boo75, BS83] which were produced and put on film so that they could be seen by far more people than had access to the hardware, which at that time was prohibitively expensive. Today graphics workstations are common place in computer science teaching establishments, and the original requirement that algorithm animations be put on film is no longer necessary. Movies (and now videos), however, still can play an important role in presenting algorithm animation.

Movies are well suited to lecture situations where the setting up of computer equipment may prove difficult, and the level of presentation only really requires a few animations, displayed perhaps once with stops for explanation. Interactive animation is not really required for this type of setting.

The introduction of multimedia tools make the production of film and video presentations from computer easier, and will probably therefore increase the use of algorithm animation presentations in the teaching environment.

\(^3\)Perhaps this is not such a good example of ‘bringing a subject to life’ as it involves quite a bit of death
\(^4\)described in section 3.1
6.3 Seminar or large group situations

The presentation of algorithm animations to large audiences is not a straightforward task due to complexities such as compatibility of equipment. However, it is often necessary to display algorithm animations to large audiences. There are four options for doing this:

- connect a large screen display device to a computer that is running the animation,
- produce a video of the animations and use a large screen display device connected to a video player,
- have a class setup for algorithm animation presentation with each person having a computer in front of them, and
- physically act the algorithm out in front of your audience\(^5\)

The first option would seem the simplest: produce the animation, set up the inputs and variables of the particular animation for the presentation, take the computer to the presentation and plug into a video projector. Unfortunately it is hardly ever like this. A computer should have RGB output for most video projectors, and the video projector has to be one that can be adjusted to the higher frequency display rates of computers (60-75Hz). The cost of these projectors is high ($18,000 to $50,000\(^6\)) and the results can be disappointing. Details of the specifications of the available projectors is included in appendix D.

The second option is one of the easiest to use during the actual presentation. It still requires an expensive projector but it does not need to have as high performance as those for computer projection. There are systems for transferring graphics from computer to video (like VEX [Bru90]), or one can simply video the screen with a camera\(^7\) while the animation is taking place. To produce a video that can be distributed takes a lot more effort, as editing and commentary will be necessary.

The third is the best option as it involves the audience more actively in the presentation, which could even allow the audience to explore the algorithms presented in the presentation. A system like this is used at Brown University\(^8\), and is described in section 6.5.

\(^5\)This idea comes from the authors father who describes the antics of a very popular neurology lecturer who acted out the symptoms of certain types of neurological diseases

\(^6\)In 1992 US$

\(^7\)A specialised camera may be necessary to avoid stroboscopic effects

\(^8\)As a result of the work by Brown [Bro88a], but the name is a coincidence
Chapter 6. Algorithm Animations and teaching

6.4 Scripting

There are two possible forms of algorithms animation by scripting. It is possible to develop purpose-built animations with animation scripting programs such as Macromind. These animations are not interactive—just like a video, they show one sequence with only the speed (and direction) of the animation being variable. They also require a high animation effort for something that, when finished, cannot be rerun with different input, as a true algorithms animation can.

The scripting included in BALSA fits into both the video and interactive type of algorithm animation display. It can be used purely like a video (with stops, rewinds, fast forward, freeze frame) that the student runs through, or it can be used as a starting point for interactive exploration of the algorithms.

This has several advantages:

- the simplicity of a video presentation, with far less effort to produce,
- it can be used as an introduction for the beginning student, and
- can be a launching platform for the advanced student to go on an explore the algorithms.

6.5 Interactive animations

The next step beyond a video type animation that has no interactive ability is a system that allows the user to interact with the parameters controlling an animation. An example of this would be the ability to change the data being compressed, or to increase the size of the lookup buffer in LZ77. The ability to adjust parameters and experiment is a major step forward from the passive viewing of movies. Such a system is used in the Electronic Classroom at Brown University.

Brown University has an auditorium equipped with 55 workstations that are used by instructors to teach dynamic simulations of algorithms and programming concepts presented to the student via the workstation under the direction of the instructor, or by the students themselves. The system has been used to teach a variety of courses including an introductory course to computer science, an algorithms course, graphics, assembly language, mathematics courses, and even geography and linguistics.

The instructors use the system in two styles: broadcast and scripting. During broadcast instruction the instructor’s actions are shown on each student’s display, whereas scripting
requires the instructor to set up a script for the session and then the students work through this at their own pace.

This would seem to be the optimal way of introducing and using algorithm animation for teaching and presentation.

### 6.6 Knowledge based help

Mahling et al. [MHHB88] state that visualisation is not enough and that it should be augmented with the knowledge underlying the system being visualised. In their system a visual electronics lab was augmented with an expert component that 'watched' the user, controlled parameters, simplified, and explained concepts.

This sort of system requires even greater effort from the creator of an animation, but has strong benefits at both the micro and macro level of animation.

At the micro level, learning about algorithms is guided by the knowledge based system like a lab tutor would. The video type scripting that Balsa [Bro88a] has simplifies the loading of the animations and sets all the parameters. The knowledge based system of Mahling et al. [MHHB88] adds to this a system that follows the student and helps them out as they go.

At the macro detail level a knowledge based system could help by giving the researcher who is viewing and interacting with the algorithm background on the current understanding and previous work with this algorithm. It could also point out phenomena already noticed and create interesting data sets and parameters to show other effects.

### 6.7 The need for empirical data

As already stated, there are few empirical studies of the effectiveness of the animation of algorithms in teaching. Brown [Bro88a] did not do any controlled experimentation with the electronic classroom, but the results of student surveys showed that students did prefer this interactive environment, and felt that they learned more easily. The only others studies of algorithm animation done are [Hay88, CS89].

The findings of Crosby and Stelovsky [CS89] regarding the effect of the presentation medium on strategies of viewing algorithms are significant. Their experimentation with students involved displaying to one group an animation of an algorithm and then later the Pascal text, after which the reverse was done for another group. They found that "While the text provided a basis for understanding graphics, graphics did not facilitate
understanding of subsequent text. The subjects used the precise detail given in the textual description to foster an understanding of graphical representations\textsuperscript{9} [CS89].

This reinforces the observation made by the author while trying to animate text compression that text is not handled well by the current algorithm animation systems, and that the inclusion of textual “popups” about events in algorithm animations (like the expert system of Mahling et al.) would aid comprehension of the algorithm.

It is also a very significant result in relation to this whole thesis, summed up by the question “Can text compression be advanced through animation?” It is the author’s belief that direct advancement will come through animation at the macro detail level, not at the micro level that was tested in Crosby and Stelovsky’s experiments. Macro animations are for experienced eyes and should prompt questions such as “What is that?” and “Why did that happen?” during exploratory observations. These may result in advancement of understanding in text compression.

Indirect advancement (that is, a greater number of people aware and using compression methods) will come through greater understanding of text compression concepts by micro animations, but this has to be augmented with precise detail and greater ‘in animation’ textual information.

In their second experiment Crosby and Stelovsky, used an eye movement monitor to study what viewers look at in animations and in static program text [CS89]. They used slides from both the animation and text of the algorithm in Pascal. This very interesting study shows the great value of getting students to view algorithms. It also shows the potential value of using an eye movement monitor to critique animations. Used along with the software Crosby and Stelovsky produced for dynamically showing what the viewer was looking at, this could help an animator greatly. They would see what was attracting the most of the viewer’s attention, and how different visual effects affect the viewer.

Empirical testing is required to understand what makes good teaching animations. This is very time consuming, but also very worthwhile if it brings about more effective methods of conveying the abstract concepts of computer science. Many other disciplines could also benefit, such as scientific visualization. There is a need for greater empirical data that demonstrates the educational value of animations to justify the effort involved in animating. Presently only a few studies have been done on this, and still do not have, as a whole, sufficient weight to motivate greater research effort in algorithms animation.

\textsuperscript{9}As mentioned earlier this is probably a function of the type of animation
Chapter 7

Conclusions

This thesis provides a preliminary study of the animation of text compression, showing that animation of text compression can:

- provide direction for research,
- assist in the teaching of fundamental concepts, and
- be difficult and require time and inventive graphical excellence.

This thesis also provides pointers for animation techniques. Knowledge of such techniques are not often found among computer scientists and must be gleaned from the variety of sources mentioned. Some of the points from static data graphical presentation have been related to algorithm animations, but the artistic world of conventional animation also have many lessons to share that could influence information portrayal.

The algorithm animation “exploratorium”

Algorithm animation provides a wonderful environment for learning and exploration. The word “exploratorium” [Bro88a] embodies the idea of exploration for fuller understanding without a required depth of knowledge, but with possibilities for all levels of understanding, like the Smithsonian Institute’s interactive physics exhibits that give the child an incentive to learn by making it interactive and fun.

The problems of animation of algorithms

Animation of algorithms has the distinct difference to other visualisations of physical data in that what you are animating is conceptual. Conceptual ideas require various attempts at
visualisation with constant critiquing. Decisions need to be made about what to animate and what not to animate, and at what detail level the animation will attempt to display the algorithm. It has been found that producing animations of algorithms is difficult and time consuming, as the techniques and principles for algorithm animation are not yet well defined. This thesis provides some principles for algorithm animation.

Can Text Compression be advanced through animation

The "rates" idea (using the rate of change of probabilities as an aid to prediction) goes some way to demonstrating that text compression can indeed be advanced by animation. There could be interesting discoveries waiting around the corner for those who use macro animations of complex text compression models displayed well. But advances are not guaranteed. I have therefore concentrated on producing some interactive and descriptive micro animations that do not directly advance the field of text compression, but with fuller development and use could indirectly advance it by the interest gained in text compression through the animations, and the better understanding gained by students of text compression that will produce an increasing awareness and use of compression methods.

Macro versus Micro

Macro versus micro detail is an important concept discovered during the production of the animations for this thesis. It is the decision an animator must make regarding what view of the algorithm will be presented.

Macro animations are an overall view of the algorithm that shows the overall trends in the algorithm, and has the potential for bringing new insights into an algorithm's behaviour. Macro animations are more difficult to produce because it is not easy to predict what will be useful to observe. Also, it can be difficult to form an animation that gets the right scale of detail—not so far out that data events happen in a blur that gives no ability to comprehend what is happening, and not so close that we miss the overall effects. The view should be creative but not such that it produces incorrect interpretations. The model animations of this thesis showing the developing order-0 models are an example of a macro animation that caused new insights to be gained.

Micro animations are a low level animation of events in an algorithm. They were found to be much easier to produce because the concepts portrayed are known and understood. The events are each individual step in the algorithm, and the animation is likely to be stepped through these events by the viewer who would be someone wanting to understand
the basic concepts behind the algorithm. The Ziv-Lempel and Arithmetic Compression animations and the arithmetic encoding animations are examples of micro animation.

Smooth Zoom

The smooth zoom function developed for the Xtango algorithm animation system is important for "flatland" animation. All though this is but a small part of one of the animations described by this thesis, it is (in this author's opinion) a kind of first. There is no other algorithm animation that has smooth zooming in on the interesting detail\(^1\). The need for careful thought on how zooming affects the end user's perception of the algorithm is an example of the careful thought that is needed behind many aspects of algorithm animations.

Rates

"Rates" is the name given by the author for the idea that the rate of symbol probability change may give more information to a model and lead to greater compression. The most important factor (relating to this thesis) about this idea is that it was caused by the viewing of animations of text compression models. A important point of this thesis is that animation of text compression algorithms will bring about improvement and further understanding of text compression. The rates idea is an example of this.

LZ78 trie

The simple animation of the development of the trie again shows that animation can be a direction finder for research. The way it develops and what this says about the source, how the source effects it could be studied via an extension of this simple animation.

The fundamental needs of the algorithm animator

To conclude, what of the future? What are future needs of the algorithm animator? This is summed up by the following four points.

\(^1\)The original Balsa had an extra scroll bar that zoomed but this was not included in later versions, and the original was not automatic, but user initiated zoom
Right system

The animator requires the right system for their needs. Algorithm animation systems at the moment can fulfil a lot of these needs but new systems are needed that fill gaps they leave (as described in section 3.4).

Goals

The animator requires well defined goals for the animations being produced. As with any software project, a well defined specification is required if the end user expects to get what they want. When producing macro animations these goals may be vague, and what exactly will shown by the algorithm will be uncertain, but the understanding of the concepts of the algorithm, what should be animated, and how, should not be vague.

Plenty of time

Animation takes time. The animator should be realistic about this process and make sure that it includes a testing phase for the audience it is designed for, to be sure that it conveys the correct information. The process as described by Tufte [Tuf83] for static graphics as "revise and edit"; this also sums up the process for animation production.

Diagram talent

There are several algorithm animation system designers but few algorithm animators. Just as the arrival of desktop publishing did not make us all wonderful document designers, so with the arrival of animation systems we do not all become great algorithm animators. It would be useful to gain as much as possible from the lessons learnt by traditional animators and data graphic designers and apply them to algorithm animation.

Good animation is an art that requires skilled practioners to develop well thought out visualizations exposing the true nature of algorithms. This will open new dimensions for both researchers and students.
Appendix A

Probability graphs of the Calgary corpus files
Appendix A. Probability graphs of the Calgary corpus files

Figure A.1: The probability graphs of selected characters in the corpus files
Figure A.2: The probability graphs of selected characters in the corpus files
Appendix A. Probability graphs of the Calgary corpus files

Figure A.3: The probability graphs of selected characters in the corpus files
Appendix B

The LZ78 animation trie development

The LZ78 animation produced the following frames showing the development of the trie data structure with different input alphabets. They are interesting because they show how the different input alphabets affect the trie, and point to possibilities in further study into how the trie develops using animation.
Figure B.1: The LZ78 trie development for a source text made up of repeated occurrences of the string “abaa”.

Appendix B. The LZ78 animation trie development
Figure B.2: The LZ78 trie development for a source of evenly probable heads and tails, with the peculiarity that every third toss is the same as the previous.
Figure B.3: The LZ78 trie development for a source made up of a, b, and bb with the probabilities 0.5, 0.3, and 0.2 respectively.
Appendix C

Colour and 3D algorithm visualization

Colour and three dimensional views add two extra dimensions to simple animation systems. The following two pages show three different colour views of algorithms:

- two photographs taken of the algorithm animation movie, “Sorting out Sorting”,
- four frames showing 3D colour static views of sorting algorithms, and
- three graphs showing sorting in algorithms using colour to define the sort value.

Movie pictures The movie “Sorting out sorting” is described in section 3.1. The two frames shown from it are not high quality but give some idea of what the movie was like. The frame on the left shows nine simultaneous views of the different sorting algorithms—see if you can work out which one is which! Quicksort and Heapsort should be fairly obvious.

The frame on the right shows the heap data structure of heapsort as a tree for three different settings. It also shows the time that each algorithm took.

3D sorts The middle two frames show insertion sort and heapsort as rendered coloured balls, and the bottom two frames show quicksort with the items as pipes wending their way to the correct sorted position.

Graphed colour sorts These are very similar to the previous two frames but have colour as the sorting key rather than three dimensional position. The horizontal axis
shows fundamental steps of the each particular algorithm—insertions for insertion sort, partitions for quicksort, and heapdowns for heapsort.
Two scenes from the movie "Sorting out Sorting"

3D view of Insertion Sort (10 items)

3D view of Heapsort (25 items)

Two 3D views of Quicksort
Appendix C. Colour and 3D algorithm visualization
Appendix D

Specifications of available large screen computer projectors

This is a summary of the projectors that work well with Sun Sparcstations. The projectors were shown at the INFOCOMM International Projection Shoot-Out, which took place February 6-8 1992 at the Washington Convention Center. The specifications are from the brochures (prices are in 1991 $US). Projection equipment is important for algorithm animation, fortunately, both the quality and price of this equipment is improving rapidly.

Dr Mandelberg states:

It is clear that the future of the projection equipment is in LCD technology. The advantages are clear: no convergence problems since three LCD plates are converged internally, and the size and weight of the projectors should decrease. I was impressed by the Barcodata 5000. It has only one lens, and it was extremely bright, 5000 lumens, it uses light valve technology. The maximum horizontal scan rate is 36KHz, and the resolution is 640x480. It will work with a Mac II but it will not work with a Sun. The text of the Mac was shaking a little. The technology is not here yet for a one lens LCD projector that you could use with a Sun, it will take a few years before the resolution is high enough and before the LCDs are fast fast enough to handle the higher scanning rates. Also it will take time before the LCDs can compete in price, the Barcodata 5000 costs $40k. Barco is leading at this point, the other vendors (except Hughes) do not have any LCD machines and would not comment on future products.

In the meantime, the only choices are three gun projectors. If you want to
project Sun images and if you want to get decent results, you need to spend $20k.

In these specifications convergence-zones refers to the number of areas on the screen where you can converge and adjust the projector, and ACON refers to a tv camera that looks at the screen and converges the projector automatically, so that you don’t have to do it manually.

**MODEL** : Barcographics 800  
**HORIZONTAL-SCAN** : 15 - 90 KHz  
**VERTICAL-SCAN** : 37 - 140 Hz  
**BANDWIDTH** : 60 MHz  
**RGB RESOLUTION** : ?  
**CRT SIZE** : 8"  
**BRIGHTNESS** : 800 lumens  
**CONVERGENCE-ZONES** : 25  
**PRICE** : $20k  
**ACON** : $3450  
**PLANS FOR LCD** :  
**WHAT'S NEW** : The Barcographics 1200 has a horiz-scan rate of 15-135 MHz, a bandwidth of 120MHz and the price is $35k. This is probably overkill for a Sun unless you need the bandwidth. The Barcodata 5000 is an LCD projector, more on this model below.

**MODEL** : Electrohome ECP4101  
**HORIZONTAL-SCAN** : 15 - 80 KHz  
**VERTICAL-SCAN** : 45 - 120 Hz  
**BANDWIDTH** : 70 MHz  
**RGB RESOLUTION** : 1280 - 1024  
**CRT SIZE** : 7"  
**BRIGHTNESS** : 650 lumens  
**CONVERGENCE-ZONES** : 45  
**PRICE** : $20k  
**ACON** : $3k
## Appendix D. Specifications of available large screen computer projectors

### PLANS FOR LCD
- General Electric Imager 610
- Hughes Light Valve Series 300

### WHAT'S NEW
- Software that will help you if you use a variety of video sources. Increased reliability, 8200 MTBF
- This projector is in a different category, it is extremely bright, and it is great for video.

### MODEL

<table>
<thead>
<tr>
<th>Feature</th>
<th>General Electric Imager 610</th>
<th>Hughes Light Valve Series 300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal-Scan</td>
<td>15 - 75 KHz</td>
<td>15 - 90 KHz</td>
</tr>
<tr>
<td>Vertical-Scan</td>
<td>38 - 100 Hz</td>
<td>45 - 120 Hz</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>70 MHz</td>
<td>70 MHz</td>
</tr>
<tr>
<td>RGB Resolution</td>
<td>1280 - 1024</td>
<td>1280 - 1024</td>
</tr>
<tr>
<td>CRT Size</td>
<td>9&quot;</td>
<td>uses 3 LCD beams that need to be converged</td>
</tr>
<tr>
<td>Brightness</td>
<td>650 lumens</td>
<td>1000 - 2500 lumens</td>
</tr>
<tr>
<td>Convergence-Zones</td>
<td>150</td>
<td>13</td>
</tr>
<tr>
<td>Price</td>
<td>$18.5k</td>
<td>$40-$50k</td>
</tr>
<tr>
<td>ACON</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Plans for LCD</td>
<td>no comment</td>
<td>obvious</td>
</tr>
</tbody>
</table>

- Increased reliability, 8200 MTBF
- I have not seen this unit connected to a Sun, I could not compare with previous models.
Appendix D. Specifications of available large screen computer projectors

Sun. They showed it connected to a mac and the text seemed to shake.

**MODEL** : Sony 1271Q
**HORIZONTAL-SCAN** : 15 - 85 KHz
**VERTICAL-SCAN** : 38 - 150 Hz
**BANDWIDTH** : 70 MHz
**RGB RESOLUTION** : 1280 - 1024
**CRT SIZE** : 7"
**BRIGHTNESS** : 650 lumens
**CONVERGENCE-ZONES** : 21
**PRICE** : $20k
**ACON** : no
**PLANS FOR LCD** : no comment
**WHAT'S NEW** : This is an improved version of the 1270Q which only handled up to 75 KHz (horiz-scan), had a bandwidth of only 40 MHz, and had 9 convergence zones.
Bibliography


[Boo75] Kellogg Booth. P-Q trees. 16mm colour silent film, 12 minutes, May 1975.


Flip Book
Algorithm Animation

Quicksort
Partition steps

Bubblesort
Racing against Quicksort

Order-0 Model
Obj1 file of Calgary Corpus

Order-0 Model
Book1 file of Calgary Corpus

Gerty
Text Compression by Elephant


[Sta] John Stasko. Animating parallel programs. Georgia Institute of Technology, Atlanta, GA 03332-0280, Email: stasko@cc.gatech.edu.


