

**A Region-Based Approach
for
Digital Cel Painting**

by

Guno Sutiono

**A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science**

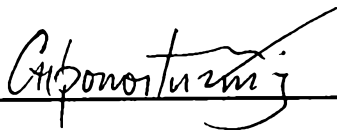
Waterloo, Ontario, Canada, 1989

© Guno Sutiono 1989

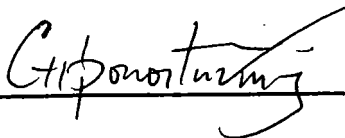
Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.



I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.



Borrower's Page

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Recently, a number of computer-aided paint systems have been developed to improve the colour painting process, traditionally one of the most time-consuming steps in two-dimensional cel animation production. However, much improvement remains to be made in the design of these new systems.

In this thesis, we describe a region-based cel painting design which uses the "paint-by-number" method and is based on a virtual frame buffer model. In this region-based design, the regions are recognizable, defined entities. This allows us to define functions for conceptually-based, interactive painting techniques, which can be easily implemented. Furthermore, the availability of this additional region information introduces the possibility of automatic multi-frame colour tracking, which can be implemented by transforming this problem into the well-known maximum matching problem in graph theory. Thus, this new region-based design builds upon techniques used in earlier systems and paves the way for future improvements.

Acknowledgements

I would like to acknowledge the advice and help of many people who contributed to the successful completion of this thesis.

The research has been supervised by Dr. Kellogg Booth. Financial support was provided by the Natural Sciences and Engineering Research Council of Canada, the Institute for Computer Research at the University of Waterloo, and Digital Equipment Canada.

Dr. Kellogg Booth, Dr. Helen Shen, and Dr. Derick Wood were the faculty readers for this thesis. Mr. Ron Pfeifle served as the student reader. I sincerely appreciate their constructive criticism of this document.

I would like to thank the National Film Board of Canada for providing a collection of traditional cels. Their assistance is greatly appreciated.

I would also like to thank the members in the Computer Graphics Laboratory for their help and friendship.

Thanks to Victor Hui and Mrs. Sharon Lee for offering help during the preparation of this document.

I am also grateful to friends in the CCF for their love and prayers.

Special thanks to Eva Yum for her love, support, and much more ...

Last but not least, I would like to thank my family for their encouragement, understanding, and patience.

To My Dear Mom and Dad

Table of Contents

<i>Author's Declaration</i>	ii
<i>Borrower's Page</i>	iii
<i>Abstract</i>	iv
<i>Acknowledgements</i>	v
1. Introduction	1
1. Traditional Cel Animation	2
2. Digital Cel Painting Systems	3
1. SoftCel — A Scan & Paint System	4
2. Palette — A Virtual Frame Buffer System	6
3. A Region-Based Approach	8
4. Overview of the Thesis	9
2. A Region-Based Virtual Frame Buffer	12
1. Conceptual Analysis	12
2. Specification of the Region-Based Model	15
1. Region-Based Image Representation	15
2. Region Labelling Information	17
3. Palette	17
4. Colours	17
3. The Model's Assumptions	18
4. Region Segmentation	18

3. Region-Based Painting Techniques	21
1. Region-Pointing Technique	21
2. Stroking Technique	22
3. Fencing Technique	23
4. Radiating Technique	24
5. Bounding Scope Technique	26
6. Functional Unification	29
7. Tool Development Issues	30
1. Colour Mapping	30
2. Redundant Painting	31
3. Fencing	32
4. Error Handling	34
8. Parallelism	37
1. Multi-Tasking Approach	38
2. Time-Slicing Approach	40
4. Boundary Completion Techniques	42
1. The Nature of Broken Boundaries	42
2. The Problem of Leaking	43
3. Automatic Boundary Completion	44
4. Interactive Boundary Completion	46
1. Boundary Plane	46
2. Tools	47
3. Region Relabelling	49
1. Region Splitting	49
2. Region Merging	53

5. Multi-Frame Region Tracking	55
1. Formal Description of Tracking	55
2. Algorithmic Complexity	58
3. Measurement of Region Similarity	60
1. Size Coherence	60
2. Positional Coherence	61
3. Image Transformation	62
4. Birth and Death of Regions	64
5. Future Research	65
6. Conclusions	66
Appendix A. Thinning Algorithm	68
Appendix B. Grid Transformation Algorithm	73
Appendix C. Boundary Tracing Algorithm	76
References	79

List of Illustrations

1.1. Traditional cel animation production	2
1.2. An opaqued cel	4
1.3. Proposed digital cel animation production	10
1.4. A cel from the film AMUSE-GUELULE [NFB87]	11
2.1. Direct mapping between regions and colours	13
2.2. Mapping with the additional colour reference mediator	14
2.3. General configuration of a virtual frame buffer	16
2.4. Grid lines	20
2.5. Formation of grid boundary	20
3.1. A palm tree	22
3.2. Demonstrating the stroking technique with a fence	23
3.3. Examples of applying the radiating technique	25
3.4. Region with textural lines	26
3.5. Bounding scope	28
3.6. Effect of changing the fence during painting	33
3.7. An error in stroking	37
3.8. Unblocked interactions	38
3.9. Multi-tasking example	39
3.10. Serialisation of painting interaction	40
4.1. Perceptual regions without closed boundaries	43
4.2. Perceptual boundary completion	43
4.3. A line drawing [from WALT87b]	45
4.4. Example of boundary drawing	48
4.5. Example of end-point gravity	48
4.6. Example of erasing	50
4.7. Region splitting	52

4.8. Boundary tracing	53
4.9. Region merging	54
5.1. Example of region tracking	56
5.2. Bipartite graph matching	57
5.3. Network flow problem	59
5.4. Centre of mass	62
5.5. Incorrect matching of rotated object	63
5.6. Incorrect matching with the birth of a region	64
A.1. Neighbourhood arrangement used by the algorithm	70
A.2. Illustration of conditions (a) and (b) in COND_A	71
A.3. Illustration of violation of condition (b)	72
B.1. Definition of grid lines	74
B.2. Formation of grid boundary	75
C.1. Neighbourhood of a boundary line	76
C.2. Decision path for tracing	77

1. Introduction

In recent years, with the introduction of digital colour image display hardware, researchers have made much progress in using computers to assist the production of *cel*¹ animation. Cel animation consists of producing sequences of cels that are painted so that when viewed in sequence an animated picture is seen. However, there are still areas in which improvements need to be made. One of these areas is the process of *cel painting*.

There are many digital paint systems that can be used to paint the cel, but these systems are still too limited to allow efficient interactive techniques for painting. To build a user interface, that is conceptually easy to use, demands a thorough understanding of the nature of the interface application. In this thesis, we discuss the problem of cel painting. Based on a new conceptual approach and a new painting system, a *region-based virtual frame buffer model* will be described. With this model, efficient techniques for cel painting can be implemented easily.

In the following sections, we briefly review the process of traditional cel animation and provide a survey of existing digital cel painting systems. These materials serve as a reference so that the problem of cel painting, addressed in the thesis, can be better understood.

¹ The word *cel* comes from *celluloid*, from which early plastic sheets were made. Nowadays, any transparent plastic sheet is called a cel regardless of its material.

1.1. Traditional Cel Animation

Cel animation is a form of *two-dimensional character animation* that was invented to avoid redundant drawing. Often, the motion of a character in an animation sequence involves only parts of the character's body. By separating the moving parts and the static parts of the character and by drawing them on different cels, the animator need not redraw the static part of each frame. During the filming process, the relevant cels are composed to create the final image. By combining different moving parts with the same static part, an animation sequence can be produced.

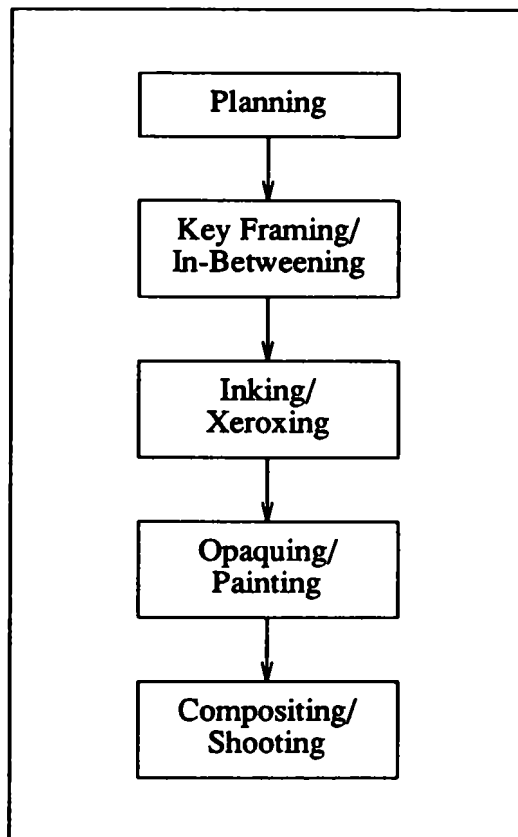


Figure 1.1. — *Traditional cel animation production*

Creating a cel animation film usually involves five major processes (Figure 1.1). Common to all motion picture production, the *planning* of script, characters, music, sound-track, scene, colour, etc. is carried out first. This results in the production of the *storyboard*. Secondly, animators draw rough paper sketches of the characters in pencil in order to preview the movements of the characters. Usually, a senior animator draws these *key frames* and the junior animators draw the frames in-between the key frames. This part of the process is known as *in-betweening*. Thirdly, the pencil drawings are copied onto a sheet of clear acetate by tracing the outline of the drawing using solid paint (usually black). This process is referred to as *inking*. Currently, most production houses use photocopiers to copy line drawings onto the cels. Fourthly, colours are applied to the reverse side of the cel (Figure 1.2). This is the *opaquing* or *painting* process. Note that painting on the reverse side of the cel is more efficient than painting directly on the top of the cel, because the painter can ignore the interior lines of the objects (e.g. the horizontal line segments of the tree trunks in Figure 1.2) while paints are being applied. Finally, filming takes place by overlaying the background and opaqued cels together to make up the final image in a process known as *compositing*.

The aspect of cel animation that is the most tedious, takes the most time, and is the most expensive is the process of opaquing ([LAYB79] pp.142). Therefore speeding up this process is very desirable.

1.2. Digital Cel Painting Systems

In recent years, in order to speed up the opaquing process, researchers have replaced the traditional cel painting process with digital painting systems and have introduced many different painting system designs. Although each new design is meant to overcome the shortcomings of the previous designs, often useful characteristics of the former systems cannot be retained and a tradeoff is necessary. In the following section, two existing painting systems are described to demonstrate the evolution in digital cel painting technologies. The tradeoffs between the two systems are then discussed.

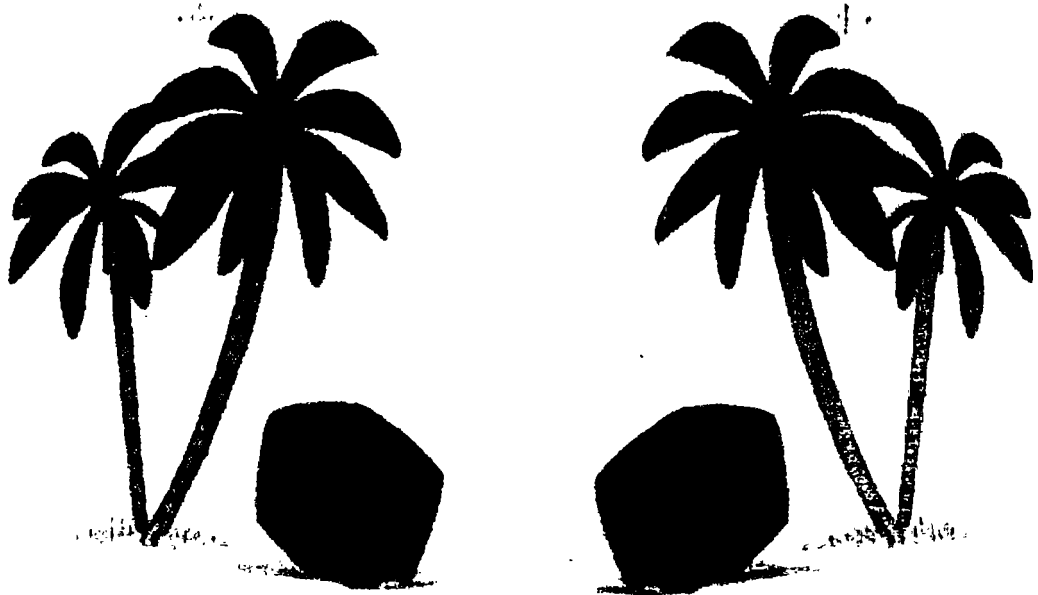


Figure 1.2. — *An opaqued cel*

1.2.1. SoftCel — A Scan & Paint System

By 1978, a group of researchers at The New York Institute of Technology had laid a basis for simulating the traditional inking and opaquing process by using a computer painting system that had a raster scan graphics device with a frame buffer. The system is named *SoftCel* [STER79] because it replaced the traditional cel by a digital raster image referred to as a *digital cel* and substituted computer software for the operations performed by the human animator. The computer-simulated inking and opaquing process is commonly referred to as a *scan and paint* system because of the nature of this process.

SoftCel is one of the earliest scan & paint systems. It replaced the traditional inking and opaquing by scanning pencil drawings with a video camera, digitizing the video signal into a raster image, and painting it with video colour components. The system uses an 8-bit frame buffer for both temporary storage and the viewing of the

image. The values of the three colour components (red, green, and blue) are determined from the 8-bit pixel values by looking up three 256-element tables, known collectively as the colour lookup tables (CLUT). Since the representation of the cel image does not include the visual colour components but an index to the CLUT instead, we refer to this type of painting system design as *paint-by-number*.

Initially, when a line drawing is digitized in the frame buffer, it may contain the full range of pixel values that represent 256 grey levels. As a result, all entries of the CLUT are required to represent the different levels of grey shades. In order to provide interactive painting of a drawing, it is necessary to reduce the number of pixel values used for the representation of lines so that some entries of the CLUT can be freed to represent colours. However, one cannot use too few pixel values to represent the lines because it may create visual aliasing² artifacts. So there is a tradeoff between the quality of the lines and the availability of colours.

In the SoftCel system, the 8 bit-planes are divided into two halves to represent the lines and colours. The four least significant bits are used to represent the line drawings, and the four most significant bits are used to represent the 16 colours available for painting. The separation of the bit planes into two functional representations is particularly interesting. This helps in designing painting tools that focus on modifying the colour fields without altering the line representation. The techniques *tint fill*³ and *tint paint*⁴ are used to colour the cel image. Furthermore, this design is consistent with the conceptual view of the painting process because the painter need only be concerned with the colour to be assigned to a region, and not be concerned whether the colour will blend into the boundary lines smoothly. Also, with this design, the smooth blending of the colour to the lines is automatically handled by

² Lines appear to be staircases (not smooth).

³ Tint fill, developed by Smith [SMIT79], is a colour flooding operation that algorithmically searches all pixels within an area enclosed by a closed line boundary and individually assigns the colour to the colour field of each pixel it visits.

⁴ Tint paint is a brushing technique which alters only the colour field of the pixel it visits.

a CLUT lookup.

In many cases, the colours chosen for the painting may be unsatisfactory and require modification. The paint by number design allows easy modification of the colour. Once an area is painted, the colour field of each pixel in the area has the same value. Since the actual visual colour of the pixel value is determined by the CLUT, a simple change of the corresponding CLUT entries is all that is required to change the colour of the area with the same colour values. This change is much faster than applying a fill operation once again. On the other hand, as noted in [STER79], the SoftCel system has a limited number of colours due to hardware constraints. As there are only 16 colours, the animators are forced to put more effort into the planning of the earlier drawings to ensure that no more than 16 colours are needed in the same cel.

1.2.2. Palette — A Virtual Frame Buffer System

The digital painting system described so far uses the video frame buffer to serve both as the viewing device and the temporary storage of the image. The coupling of these two functions in the same frame buffer is unnecessary. In [LEVO82] and [TANN83], it has been pointed out that having the image information stored in the frame buffer slows down the performance of the painting process. The inefficiency is due to the slow I/O access and heavy data traffic between the host and the frame buffer.

To solve this contention problem, a *virtual frame buffer* model was introduced by Levoy [LEVO82]. A virtual frame buffer is a data configuration that contains information about an image but resides outside the video frame buffer. Thus, it is an attempt to decouple the two functions, the storage and the viewing of an image, from the video frame buffer. With the virtual frame buffer, the traffic between the host and the video frame buffer is reduced significantly because the system no longer needs to read from the video frame buffer.

Since the virtual frame buffer concept is independent of the video frame buffer, it also allows us to design abstract pixel representations which are free from the restrictive notion of red, green, and blue. This abstraction subsequently enables us to extend functionalities for the painting application. *Palette* is an example of a paint program that demonstrates the functional power of a virtual frame buffer [HIGG86].

The *Palette* paint system uses a RGBA virtual frame buffer design. The RGB fields are the three colour components (red, green, and blue) and the A field (alpha channel) provides a measure of colour opacity [PORT84]. The opacity feature allows the compositing of cel images because the unpainted portion of a cel can be represented by a totally transparent colour which permits any underlying cel images to show through. In addition, the specification of partial opacity provides for translucent coloured objects, such as fog and clouds, that are not normally found in traditional cel animation.

Since *Palette* is a virtual frame buffer based system, it is independent of the hardware specifications of the frame buffer. Therefore, the system can be implemented with any video frame buffer as long as it gives a visual feedback image. Portability is one of the major advantages of the system. In addition, the number of colour is less restricted because it does not depend on the hardware. Conceptually, there can be up to 16 million colours available if 8 bits of each RGB channel are used to represent a colour. Also, as mentioned above, the model is able to simulate the cel compositing process by introducing a measure of colour opacity; this has demonstrated the power of functional abstraction in a virtual frame buffer. However, this design does not allow the abstraction of the colour attributes from the image as the *paint-by-number* system does. Hence, modifying colours on a cel is not as efficient.

1.3. A Region-Based Approach

Observing the trends in the design of painting systems, we find that there has been a departure from the paint-by-number (or paint-by-pseudo-colour) design towards the use of direct-colour (e.g. RGB components) in the cel painting process. However, when we examine the paint-by-number design closely, we observe that some of its characteristics are desirable in interactive cel painting. In particular, because of the separation of the colour attributes from the frame buffer representation using a colour index, the painter is able to modify the colour of a painted region easily. He can do so by changing the colour attributes of the corresponding CLUT entries which are associated with the colour index stored in the frame buffer. Moreover, by altering the CLUT entries, the corresponding area can be changed to the desired new colour in one video retrace cycle.

The ability to update the colour of a region in real-time is a desirable feature, but in existing paint-by-number designs, it cannot be applied to painting a region for the first time. The reason for this is that the unpainted region has not yet been assigned a colour index, which is necessary for real-time colour updating. Instead, the initial painting of a region requires a slower colour filling operation to assign a colour index to every pixel inside the region. It would be ideal if existing designs could be modified so that the initial painting can also be done in real-time. We can achieve this if distinct pseudo-colours can be assigned to different regions ahead of time, and this can be done algorithmically through the use of a *region-labelling process* (the details are explained in the next chapter). This modified paint-by-number design is referred to as a *region-based design*.

There may be some skepticism about the region-based design because it seems to be beneficial for painting interaction only if the specific hardware (i.e. the CLUT frame buffer) is available. However, in this thesis, we show that the region-based design is a better model for cel painting regardless of the availability of the specific hardware. We have done an analysis of the cel painting process and have discovered that the region-based design can be derived directly from the results of the analysis.

Based on this analysis, we have isolated, functionally, the cel painting interaction as a *region-colour assignment* process, simply by constructing a mapping between a set of regions on the cel and a set of available colours. This assignment process requires that the *region* of a cel image be a defined entity which is made available to the paint system by the region-labelling process.

The additional region-labelling information enables us to define new techniques to improve the efficiency of the interaction. For instance, the time spent in waiting for the completion of a filling operation is significant; this is because all fillings are carried out serially. One possible way of improving filling is to provide parallel filling. The reason this method has not been put into practice previously is that two filling operations may be working in the same region simultaneously resulting in a colour collision. It is impossible to avoid this problem with existing system designs because the two operations are unable to determine if they are in the same region. However, with the extra region labelling information, the collision problem can be avoided by checking whether the region to be filled is being processed by another operation.

Because of the advantages mentioned above, the *region-based* model is preferable to other designs for cel painting.

1.4. Overview of the Thesis

In this thesis, we describe a *region-based* design for achieving efficient cel painting. This design is based on a combination of a paint-by-number concept and a virtual frame buffer construction.

In Chapter 2, we describe the specifications and the assumptions of a model for *region-based* design. In addition, we show how the proposed design can be derived based on the functional descriptions of a conceptual analysis of the cel painting process. The region-based design model is a new digital cel animation production pipeline depicted in Figure 1.3.

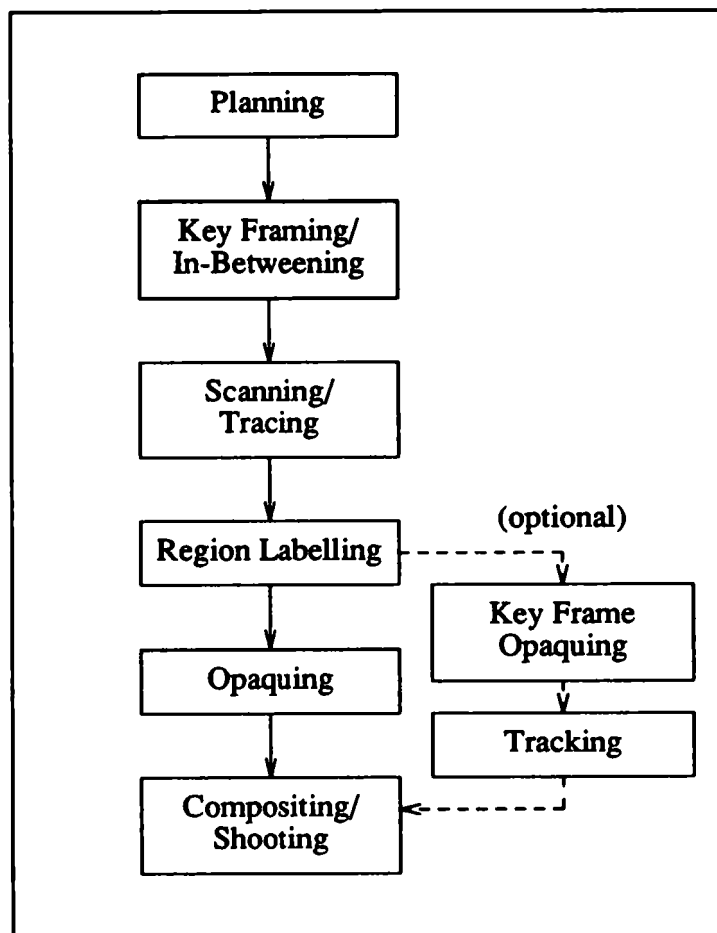


Figure 1.3. — *Proposed digital cel animation production*

Because the region-based model is compatible with the conceptual model for cel painting, useful conceptual painting techniques can be developed with the region-labelling information. In Chapter 3, we describe a new set of painting techniques that can be defined with the *region-based* model. Some issues that are encountered in the development of such tools are discussed. The model assumes that all distinct regions are bounded by closed line boundaries. For, if this does not hold, the *region labelling* process is not able to identify the region correctly. However, in reality, an image may contain broken boundaries. There are at least two reasons for broken boundaries to

occur. First, in the original drawing, a line may be so thin that when the cel is digitized the line disappears. Second, a perceived region may not have an explicit closed boundary; for example, the grass shown in Figure 1.4. In Chapter 4, we describe methods used to complete the missing boundaries, a process known as *tracing*, so that the region labelling information can be corrected.

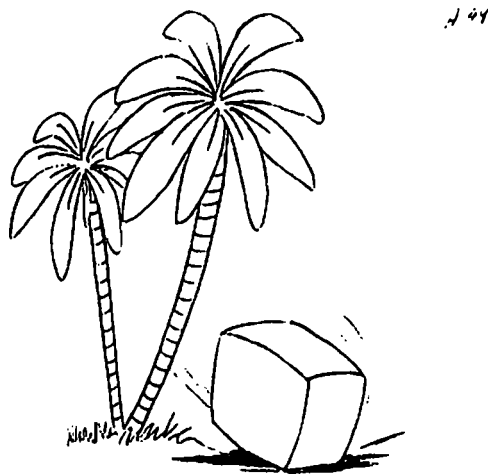


Figure 1.4. — A cel from the film *AMUSE-GUELULE* [NFB87]

It has been noted that the positions of an animated object in a short animated sequence differ by only a little. It is desirable to take advantage of this frame coherence by automatically *tracking* the regions from one frame to another so that the corresponding colours can be copied correctly. With the region labelling information, the formulation of a region *tracking* problem becomes possible. In Chapter 5, we discuss issues encountered in tracking and introduce some ideas that may provide a source of future research directions.

2. A Region-Based Virtual Frame Buffer

The region-based design (a modified paint-by-number model) can be derived directly from the analysis of the cel painting process. In this chapter, we follow the conceptual approach to obtain a description of the region-based model. Furthermore, the construction of a *region-based virtual frame buffer*, which is an implementation of the region-based design, is described.

2.1. Conceptual Analysis

The traditional cel painting process can be divided into the following three procedures.

- A. *Colour Mixing* — The painter prepares the paints required for a sequence of cels according to the specifications given in the storyboard. These paints are usually arranged in a palette.
- B. *Region-Colour Assignment* — The painter identifies a region on the cel. Then, based on the colour specification, an appropriate colour is chosen from the palette and applied to the region.
- C. *Colour Filling* — The painter continues to work on a single region until the whole area is opaqued.

Functionally, *Colour Mixing* is a procedure for finding a mapping from the palette set, P , to the set of available colours, C . Similarly, the *Region-Colour Assignment* is the task of defining a functional mapping from the set, R , of cel regions to the palette set, P . In the *Colour Filling* process, we opaque a region with the paint

with which it is indirectly associated through these two sets of mappings.

Some may think that the separation of procedures B and C is unusual because the traditional painter may not perceive them in such a disjoint way. However, in the context of digital cel painting, the introduction of the *filling* algorithm changes the painting interaction. Procedure C, colour filling, is exactly what a filling algorithm can do. Hence, painting a cel has been reduced to a simple region-colour assignment as described in procedure B.

Some may also argue that the set P , which serves to separate procedures A and B, is redundant because eventually we are only concerned with the mapping from the set R of regions to the set C of paints. This would be a valid argument if we were dealing with the traditional cel painting process. However, in digital cel painting, since it is possible to modify the colour of a painted region, we may also want to design the system in such a way that colour modification is simple. The introduction of the additional set P can be used to achieve this.

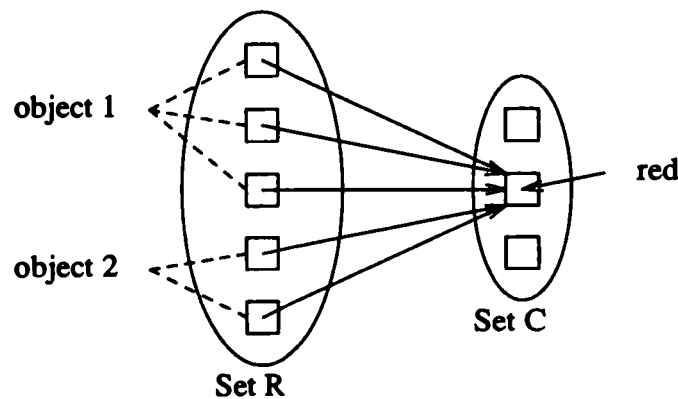


Figure 2.1. — *Direct mapping between regions and colours*

Consider the example depicted in Figure 2.1. Assume that there are some regions in the scene that belong to two separate objects which are all painted with the same colour (red, say). If we decide to change the colour of one object, the colour of all regions in that object must be modified. With the design shown in Figure 2.1, the only option is to re-assign the colour indices of all regions in that object. Although it is possible to modify the colour of all regions in the same object by simply changing the colour attributes to which these regions are mapped, this forces the colour of another object to change as well. However, with the use of the additional set P as an indirect⁵ colour reference, the region-colour assignment can be carefully planned so that these two regions map to two different colour indices, and the scenario in Figure 2.1 can instead be represented by Figure 2.2. In such a way, changing the colour of an object can be achieved easily by re-assigning the corresponding colour index in P with a new colour in C .

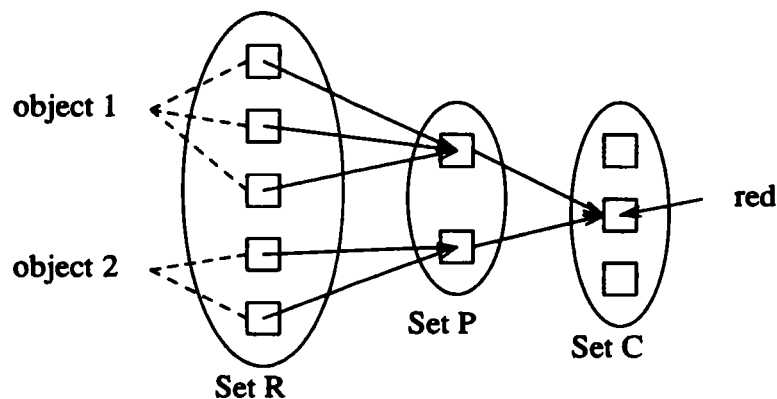


Figure 2.2. — Mapping with the additional colour reference mediator

⁵ "All problems in computer science can be solved by one more level of indirection." — Roger Needham

Since colour filling is a non-interactive procedure, and colour mixing is an interactive process that is performed prior to cel painting, we can regard cel painting as equivalent to the region-colour assignment task and focus on improving this procedure.

2.2. Specification of the Region-Based Model

In this section, we describe the data structure that is required to represent the region-based model. The model consists of four sets of data representations, which are derived from the conceptual analysis. They are the region-based image representation, the region labelling information, the palette, and the colour attributes.

2.2.1. Region-Based Image Representation

In the region-colour assignment procedure, each colour assignment requires a colour selection from the palette and a region selection from the canvas. In digital cel painting, region selection is not possible without defining the set of regions. We need to preprocess the cel image, to determine how many regions there are and give each of them a distinct label. Furthermore, each pixel of the image buffer should indicate to which region it belongs so that region selection can be done easily. It is necessary to have the region labelling information available in the cel image in order to allow the region-colour assignment to proceed.

In addition to the labelling information, the image buffer should contain the grey-level representation of the digitized line drawings. The grey-level is important in the region labelling process because it is used to delimit regions. Also, similar to the SoftCel system described in the previous chapter, the grey levels are used to obtain smooth colour blending at the boundaries.

An optional field for recording the colour of a line may also be useful. Normally, black paint is used to draw the boundaries, occasionally, however, artists like to use other colours. To allow this, we need an additional field in the frame buffer to distinguish the line colour. In our model, we do not store the colour attributes for a

line in the frame buffer because we have a separate data set for the colour attributes which is described later. Therefore, a pointer to an appropriate entry in the colour set is adequate to represent the line colour.

In order to avoid the memory constraint imposed by the video frame buffer, we will adapt the virtual frame buffer method to incorporate these three types of image information: see Figure 2.3.

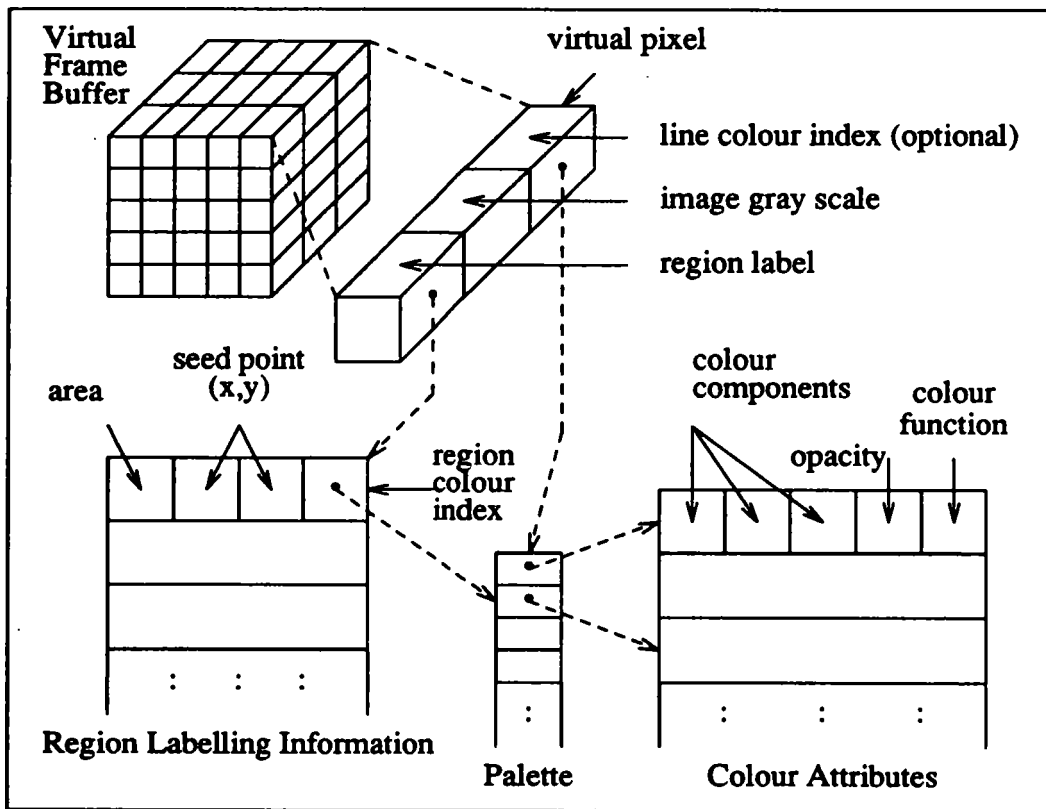


Figure 2.3. — General configuration of a virtual frame buffer

2.2.2. Region Labelling Information

For each label found in the image canvas, there should be a corresponding record which contains all information about that region. Each record should contain at least one pointer to an entry in the palette so that the mapping for the region-colour assignment can be obtained. Moreover, we add extra information to the region labelling data records. They are the area (number of pixels) and a seed point (positional information) of the region (Figure 2.3), which can be obtained, with little computing overhead, during the region labelling process. Although we may not need this additional information in the cel painting interaction, it is useful in the tracking process described in Chapter 5.

2.2.3. Palette

The palette is a set of colour references which serves as an indirect mapping from a region label to the corresponding colour attributes. The reason for this additional mediator between the region labelling set and the colour set was discussed earlier. The data structure for the palette is simply an array of colour indices.

2.2.4. Colours

To represent a solid colour, at least three components are required. There are many methods that can be used to describe a colour, among them, the RGB representation is currently the most popular in computer graphics. The alpha channel, which designates the colour opacity in the Palette cel painting system [HIGG86] has proved to be useful in the digital cel compositing process. Therefore, it is appropriate to incorporate opacity as an attribute of the colour.

Recently, Neely [NEEL88] has developed a filling interpreter that allows the user to define functionally how to colour an area of the frame buffer. This implies that the colour filling is restricted neither to solid colouring nor to specialised hardware configurations as in the case of tint filling. The colour used to paint a region can be considered in terms of a colour function instead of just colour attributes.

For example, if we paint a region using the following colour function:

```
if ( x+y is even )
    colour at pixel (x,y) ← red
else
    colour at pixel (x,y) ← black,
```

the painted region will have a red-black checkerboard colour pattern. Therefore, by defining colour functionally, a wider range of colouring effects is possible in this model.

2.3. The Model's Assumptions

In the region-based virtual frame buffer just described, two assumptions must be satisfied at all times for the model to function correctly. Firstly, no two distinct regions in the image can share the same label. This is to ensure that there will be no confusion when assigning a colour to a particular region. Secondly, each pixel within the canvas must belong to a labelled region so that the whole canvas can be completely segmented into regions.

2.4. Region Segmentation

Until now, we have been assuming that the region labels are constructed magically from a black box called *region labelling*. In this section, we will briefly describe the procedure involved in the region labelling process.

The labelling algorithm is used to segment an image canvas into regions with distinct labels. Since the algorithm is entirely independent from any interaction, it can be applied in a batch mode prior to the interactive painting process. In fact, while a digital cel is undergoing the interactive painting process, the labelling process for the next frame can be initiated and will finish before the current image is painted. Hence, there is no penalty in the real-time performance of the painting process when generating this additional information.

The algorithm is briefly described as follows.

/ Terminology */*

New_Label = Currently used numeric label

$L(x,y)$ = the region label at pixel (x,y)

C = the set of all pixels in the canvas

$F(x,y)$ = the set of pixels visited by the region fill operation initiated at pixel (x,y)

/ Algorithm */*

New_Label ← *NIL*

∀ $(x,y) \in C$, do $L(x,y) \leftarrow \text{NIL}$

∀ $(x,y) \in C$ do

 if $L(x,y) = \text{NIL}$

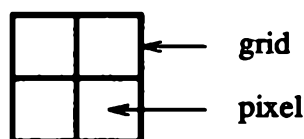
 {

New_Label ← new region record pointer

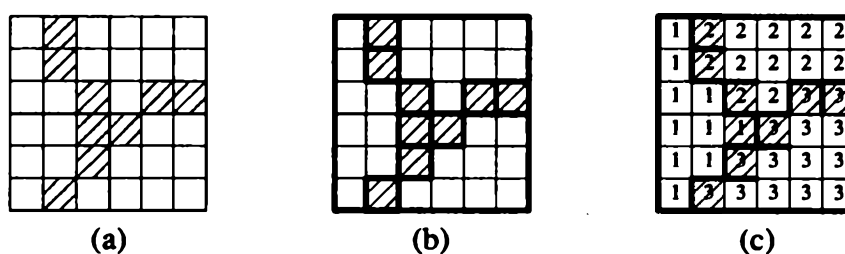
 ∀ $(a,b) \in F(x,y)$, $L(a,b) \leftarrow \text{New_Label}$

 }

As mentioned, the objective of the labelling process is to assign a region label to each pixel in the image frame. However, in order to allow the algorithm to segment the image into regions, it needs to know the region delimiter. Prior to the labelling process, the only information available is the grey-scale digitized line image. However, this information does not give a clear representation of the region delimiters required in the region labelling process because of the various line widths in the image. Therefore, we need a more uniform line representation for the region boundary. A solution is to use a line thinning algorithm to locate the skeleton of the boundaries in the image. A thinning algorithm by Zhang and Suen [ZHAN84] can be found in Appendix A. Some newer thinning algorithms can be found in [HALL89] and [GUO89].

Figure 2.4. — *Grid lines*

Although the thinned line gives a clear separation between regions, it is still unclear as to how to label the pixels that are lying along the thinned line. In order to satisfy the constraint imposed by the design, that is each pixel of the canvas must have a label, we need to further classify the line pixels into some existing region labels. Rather than using the thinned line boundary representation, we use the grid lines that lie between pixels (see Figure 2.4) to represent the region delimiters. With the grid lines as region boundaries, the classification problem is then solved because these grid lines do not occupy any pixels. In Appendix B, an algorithm is given for obtaining the grid line boundaries from a given thinned line boundary representation. Figure 2.5 illustrates the grid boundaries found by the algorithm for the given thinned line image. The thinned lines are presented by the shaded boxes in Figure 2.5 (a).

Figure 2.5. — *Formation of grid boundary*

The first step of the grid line algorithm is to set all the surrounding grid lines of each pixel representing the thinned lines as illustrated in Figure 2.5 (b). Note that the outermost grid of the image frame must always be set in order to define the image frame. Then we open up some grid lines so that the labelling algorithm can later fill in the thinned line pixels (see Figure 2.5 (c)). Using the grid line boundary, all pixels in the image can be segmented completely into distinct regions.

3. Region-Based Painting Techniques

In the previous chapter, we have shown that the cel painting process can be defined conceptually as a region-colour assignment procedure. In this procedure, two selection interactions are involved: a colour is selected from the palette and is assigned to some selected regions. Since the interaction for selecting an entry from the palette is trivial, we focus on techniques for region selection. In this chapter, we investigate some conceptual techniques that help to make region selection much easier. These techniques are developed functionally under the region-based virtual frame buffer model described in the previous chapter.

3.1. Region-Pointing Technique

Region-pointing is a single region selection technique achieved by positioning the on-screen graphics cursor to any point inside the chosen region of the image. The region label can be extracted from the virtual frame buffer at the cursor position. Then, a link is made between the corresponding labelling record and a selected colour.⁶

Functionally, we can define the region-pointing operation as

⁶ In this chapter, colour means a colour index stored in the palette defined in the previous chapter, unless otherwise specified.

$$\text{Map}(\text{colour}, \text{Region}(\text{cursor}_x, \text{cursor}_y)),$$

where *Region* returns the labelling information for a specific pixel location and *Map* associates the region with a *colour*. A detailed discussion of the implementation of these functions is found in a later section.

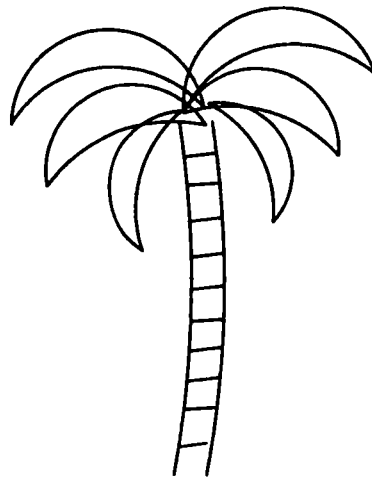


Figure 3.1. — *A palm tree*

3.2. Stroking Technique

The region-pointing technique has limited use. For example, in Figure 3.1, to paint the trunk of the tree, we need to apply laboriously one region-pointing interaction for each sub-region made by the trunk's pattern lines. Ideally, all regions of the trunk should be selected in a single interaction. Observe that these sub-regions have a strong alignment, which implies that there is a connected path through the regions. Hence, a tool that allows the user to go through these regions, neglecting the inter-region boundaries (pattern lines), is desirable. For the palm tree of the example, a painting technique should allow us to go down the trunk by the means of a continuous *stroke* that travels through all the sub-regions. This interaction is called *stroking* and is good for painting a group of regions that are tightly connected, forming a continuous perceptual path.

The functional description of stroking is

$$\forall(x,y) \in S, \text{Map}(\text{colour}, \text{Region}(x,y)),$$

where S is the set of pixels that are stroked.

3.3. Fencing Technique

We all know how hard it is to push a thread through the eye of a needle. This indicates that many people are not manually dextrous. Such a person may be unable to select a small region precisely by the region-pointing technique; they may miss some of the desired regions but include some undesirable ones while using the stroking technique. As a result, some regions may be selected mistakenly for painting; hence, a technique is required to prevent this.

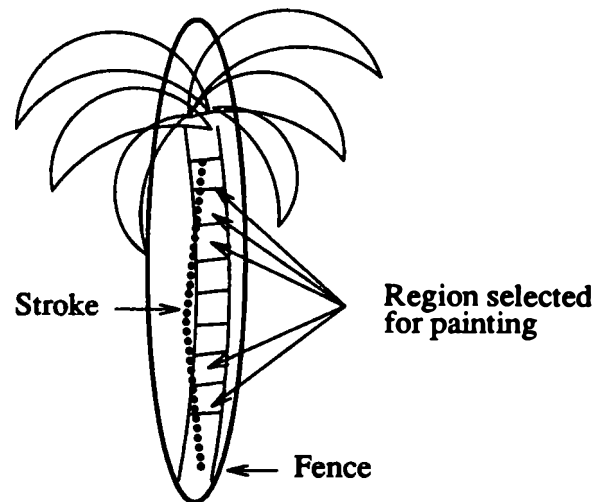


Figure 3.2. — *Demonstrating the stroking technique with a fence*

Note that the undesirable regions are usually some neighbouring regions of the painted regions. So, we introduce the concept of a *fence* to protect the neighbouring regions from being mistakenly selected for painting. A fence is a closed path defined interactively by the user. Unlike the stroking technique, none of the regions that are cut by the fence line are painted while either the region-pointing or the stroking technique is being applied (see Figure 3.2). With the addition of a fence, the functional descriptions of the two region selection techniques, the region-pointing technique and the stroking technique, are redefined. The region-pointing function is now defined as

$$Map(colour, Region(cursor_x, cursor_y)), \text{ if } Region(cursor_x, cursor_y) \notin F$$

and the stroking function as

$$\forall (x, y) \in S, Map(colour, Region(x, y)), \text{ if } Region(x, y) \notin F,$$

where F is the set of regions that are cut by the fence line.

This concept of masking out regions from image modification is similar to Higgins' *masking* operation in his paint program, *Palette* [HIGG86]. The difference between the two techniques is that masking requires the user to define the masked region by hand, but the fencing technique does not. In the cel painting application, the fencing technique is better because of its simpler interaction. Because fencing is region-based rather than pixel-based, the frame buffer needs only the region label to know which pixels are to be masked out.

3.4. Radiating Technique

Although the fencing technique can prevent unwanted regions from being painted, it still does not help the user to select small regions easily. To select a small region for painting, the best the user can do is trial and error until the cursor falls directly within the region. Though the user does not have to suffer the penalty of miscolouring the regions, he must still put in time and effort to paint them. This

shortcoming of the region-pointing technique usually causes user frustration. This problem occurs because the cursor is bound to exactly one pixel, making it harder to hit the target region. The analogy is similar to shooting a bullet into a small bull's eye. If we use a short gun that can scatter many bullets, the chance of hitting the bull's eye is much higher than when using a single ordinary bullet. Similarly, if we expand the area of influence of the cursor, it is easier to locate the desired region, thus reducing the precision required of the user. This technique of increasing the number of examined pixels for region painting is called *radiating*.

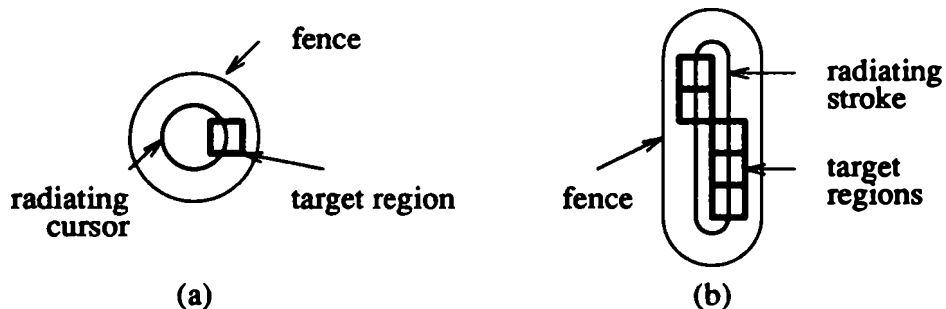


Figure 3.3. — Examples of applying the radiating technique

The problem of the *radiating* technique is that some unwanted neighbouring regions may become candidates for painting. However, if we apply the fencing technique again, the unwanted regions are not selected. Hence, radiating works best with the assistance of a fence. Examples of applying the radiating technique to region-pointing and stroking are illustrated in Figure 3.3.

Now, with the radiating capability, the two painting techniques can be described as follows.

$$\forall(x,y) \in \text{Field}(\text{cursor}_x, \text{cursor}_y), \text{Map}(\text{colour}, \text{Region}(x,y)), \text{if } \text{Region}(x,y) \in F$$

describes the region-pointing technique, where $\text{Field}(x,y)$ is the set of pixels that are within the influence of radiation originating from the location (x,y) . The stroking technique is described as

$$\forall (x,y) \in S', \text{Map}(\text{colour}, \text{Region}(x,y)), \text{ if } \text{Region}(x,y) \in F,$$

where

$$S' = \bigcup_{\forall (x,y) \in S} \text{Field}(x,y)$$

is the set of pixels traversed by the radiating stroke.

3.5. Bounding Scope Technique

So far, the techniques we have described assume that a region defined in the virtual frame buffer always coincides with a region defined perceptually. However, based on our observations described in the following paragraph, this assumption is incorrect. Hence, extended interaction techniques are required so that the region painting techniques become compatible with the human perceptual model.

The human brain can perceptually filter out insignificant information. Using Figure 3.4 as an example, we tend to ignore the pattern lines within the circle and perceive the whole as a single region. The pattern or texture lines inside the region are filtered out perceptually because they are insignificant in defining the region.

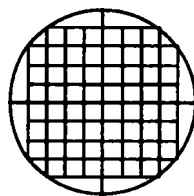


Figure 3.4. — *Region with textural lines*

With the existing techniques described thus far, numerous region-pointing operations are required for painting all sub-regions created by the intersection of the textural lines in Figure 3.4. The stroking technique cannot be applied well here because the sub-regions do not constitute a strong alignment. Therefore, these region-based painting techniques do not always satisfy human perceptual expectations. On the other hand, humans prefer a single region painting operation that selects all sub-regions within the circle for painting because it is more consistent with their perception. To achieve this, we introduce the *bounding scope* technique.

Buxton, *et al.* [BUXT81] developed a demonstrative approach for specifying scopes in an interactive music score editor. The method requires a single gesture to define a *circle of inclusion* graphically on the score. The musical notes that are enclosed by the circle are then grouped together and retrieved from the data base for further processing.

For the texture problem presented in Figure 3.4, we adapt the demonstrative scoping method for grouping together regions that are to be painted by the boundary scope method. This allows the user to outline a closed boundary to enclose all the sub-regions belonging to the perceptual region. As a result, a single region-pointing interaction can be applied to colour all the enclosed sub-regions automatically. This interaction also provides a way to group together some isolated regions that are conceptually related (e.g. stars in the sky). Hence, the bounding scope technique is really a region classification tool. Some scoping examples are illustrated in Figure 3.5.

Primarily, we want to know which regions are entirely enclosed in the bounding scope. However, to acquire that information seems non-trivial and time consuming. Fortunately, with the help of the region labels available in the virtual frame buffer, we can approach the problem in a different way. We find, instead, a dual solution by determining which regions are not entirely enclosed by the scope. In other words, to define the scope is to construct a list of exclusion regions. Therefore, the scope can be defined by the fencing technique.

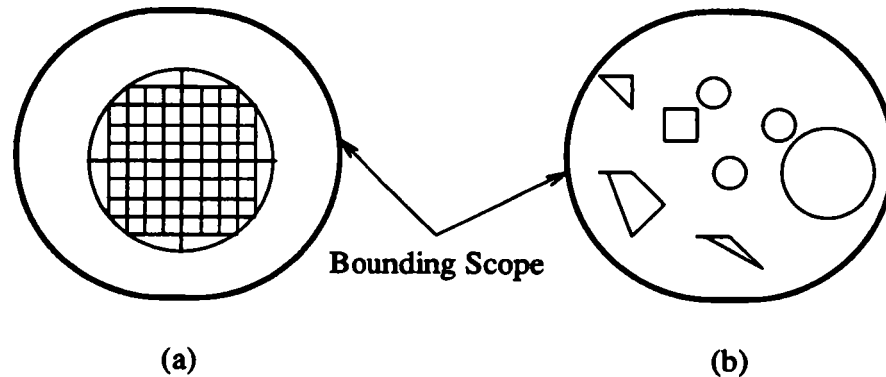


Figure 3.5. — *Bounding scope*

The second part of the interaction is the actual region painting. To be consistent with the other painting interactions, we would like to simulate a region-pointing technique for filling the enclosed region defined by the scope. Fortunately, we can simulate the interaction by applying a special radiating function with the normal region-pointing technique where the radiating field is exactly the area enclosed by the scope. This field can be algorithmically found by applying a fill operation with the scope line as the region's delimiter.

Bounding scope interaction is actually a combination of fencing and a region-pointing interaction with a special radiating function. Actually, with this special radiating function, the use of stroking and region-pointing are equivalent, because both of their radiating fields are identical.

Functionally, boundary scoping can be described as

$$\text{if } Region(x,y) \notin F \text{ then } \forall (x,y) \in B, \text{Map}(\text{colour}, Region(x,y)),$$

where $B = Field(\text{cursor}_x, \text{cursor}_y)$, and $Field()$ is a set of connected pixels delimited by the fence.

3.6. Functional Unification

If we look at the functional descriptions of the various painting tools we have described, they can be given a unified description as

$$\text{if } Region(x,y) \notin F \text{ then } \forall (x,y) \in U, Map(colour, Region(x,y)),$$

where F is the set of regions cut by the fence and

$$U = \bigcup_{\forall (x,y) \in P} Field(x,y),$$

where

$$P = \begin{cases} \{(cursor_x, cursor_y)\} & \text{if using region-pointing} \\ S, & \text{the set of pixels traversed if using stroking} \end{cases}$$

Of course, not all the tools are necessarily used simultaneously. As a result, some of the function variables are undefined if particular techniques are not used. Therefore, we need to assign default values for these conditions. The only conditions that need to be taken care of are F and $Field(x,y)$. If there is no fence present, F should be the empty set and the default for $Field(x,y)$ is $\{(x,y)\}$, because the cursor is bound to exactly one pixel when radiating is not used.

It is easy to verify that each of the functional descriptions mentioned for each tool is a special case of the general description. For example, a simple stroking technique with neither radiating nor fencing implies $P=S$ and $Field(x,y)=\{(x,y)\}$. This implies $U=S$. Since by default $F=\{\}$, the "if" condition is always true. Hence, the general description of the stroking operation reduces to

$$\forall (x,y) \in S, Map(colour, Region(x,y))$$

which is exactly the description mentioned earlier.

3.7. Tool Development Issues

The functional description provides us with a good basis for the development of painting tools. In this section, we address various implementation issues that need to be considered in the process of transforming the functional description into an efficient algorithm.

3.7.1. Colour Mapping

In this subsection, the actions involved in the *Map* function are discussed. Primarily, *Map* is used to create a link between the corresponding region record and the selected colour. Furthermore, colour feedback, on the screen, is required to allow the user to evaluate and continue the painting process. Hence *Map* is basically a two-step procedure:

Map(colour, region)

- establish a mapping between the *region* and the *colour*
- colour feedback for the region

There are two general approaches that can be used for feedback. First, we can use a *direct mapping* of a region label to a physical hardware colour. If we want a smooth colour transition from the line colour to the region colour, we need even more hardware colours to map these intermediate intensities. For example, if we allow four colour intensities to represent the transition, we need at least four times as many hardware colours to achieve direct mapping. With this hardware colour implementation, the effect of region painting can be shown in real time because we can alter the attributes of the corresponding hardware colours and have the changes updated in the next video retrace cycle.

However, the problem with this approach is the limitation in the number of hardware colour indices supported by the system. The upper bound is unknown, because the number of regions is directly proportional to the complexity of the scene. In examining the traditional cels obtained from the National Film Board of Canada, we observed that at most 256 regions were found within an image. Hence, for a

system that allows four levels of intensity for anti-aliasing, a system with 1024 hardware colour indices is needed.

We realize that few systems provide so many hardware colour indices in their frame buffers. For a general graphics system, we need a second approach for the colour feedback. It is a slower approach in terms of visual feedback, but a reasonable compromise due to the hardware limitations. Basically, it is a *colour-sharing* approach, where some of the regions share a set of common hardware colours. Because the number of visually distinct colours required in a cel image is generally low (i.e. less than 64), sharing colours definitely reduces the number of hardware colours required. Using this approach, we can no longer bind a hardware colour to a distinct region. For this reason, all pixels within a painted region require pixel-by-pixel modification in the frame buffer and a fill algorithm is necessary to modify the contents of a region in the video frame buffer.

3.7.2. Redundant Painting

Based on the functional description, there may be more than one point in the set U having the same region label. In this case, some redundant painting operations may be applied to the same region. For the *direct mapping* approach, this might not pose a serious problem. However, for the *colour-sharing* approach, we cannot afford the time required for redundant painting of the same region because the fill algorithm for colour feedback is slow.

An easier solution is to allow each painting operation to keep a local record of the regions that have been painted. With this additional book-keeping, we can decide when it is unnecessary for the *Map* function to be invoked. This guarantees that all regions that are to be opaqued in each painting operation only painted once.

The algorithm based on the functional description now looks like the following.

$$\begin{aligned}
 &R \leftarrow \{\} \\
 &\forall (x,y) \in U \\
 &\quad \text{if } (Region(x,y) \notin F) \text{ and} \\
 &\quad \quad (Region(x,y) \notin R) \\
 &\quad \{ \\
 &\quad \quad R = R \cup \{Region(x,y)\} \\
 &\quad \quad Map(colour, Region(x,y)) \\
 &\quad \}
 \end{aligned}$$

where R is the set of regions that are already painted.

3.7.3. Fencing

The set F of excluded regions is obtained during the process of fence construction but not during the painting operation (e.g. stroking, region-pointing). Hence, prior to any painting operation, the user must ensure that the appropriate fencing information is added or removed.

Because of the separation from the painting operation, the fencing set F should be treated as external information that can be accessed at any time by any painting operation. Thus, all painting operations depend on the same storage set F . It is very important, therefore, to ensure that the contents of F are not modified during a painting process.

If we only allow operations to be executed serially, the information set F cannot be changed during an execution of a painting operation. However, as we will discuss in a later section, we may want to execute these operations in a parallel manner. Accessing information from a common storage set F directly from the painting algorithm may then yield different results at different times because this information can be accessed and modified by any operation at any time.

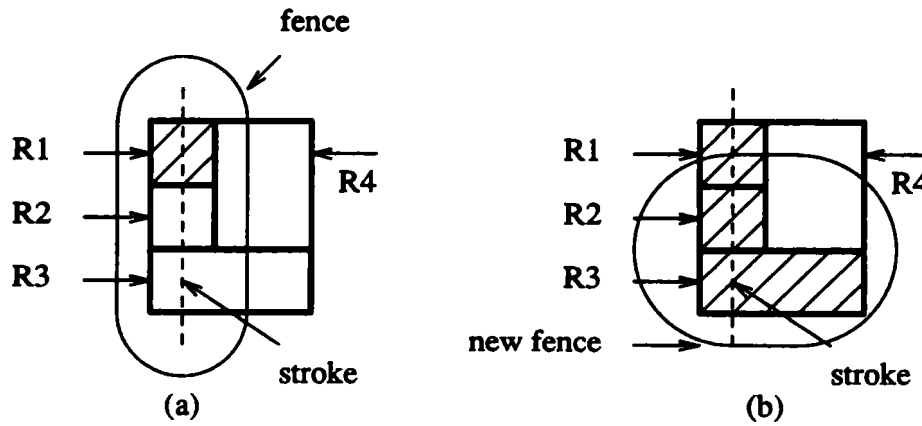


Figure 3.6. — *Effect of changing the fence during painting*

For example, in Figure 3.6, the painting operation should paint regions R1 and R2, and R3 and R4 should be unchanged. Now, if the fence is being changed to Figure 3.6(b) while the painting algorithm finishes painting R1 and starts painting R2, the fencing constraint has been changed. Although the change of the fencing constraint does not affect the painting of R2, it does affect R3. When the stroke comes to R3, since the fence constraint has been changed and the restriction to colour R3 has been removed, a filling operation will be initiated for R3. This will result in R1, R2, and R3 being painted, which will be confusing to the user. To avoid such confusion, each painting operation should acquire a copy of the external information F at the beginning of its execution so that this constraint can be referred to locally.

A modified algorithm can be described by:

```

LF ← F
R ← {}
∀ (x,y) ∈ U
  if (Region(x,y) ∉ LF) and
    (Region(x,y) ∉ R)
  {
    R ← R ∪ {Region(x,y)}
    Map(colour, Region(x,y))
  }

```

3.7.4. Error Handling

When an interactive system is designed, the issue of error handling should not be neglected. In our case, if some regions are mistakenly painted by a painting operation, we need to decide how to allow the user to recover from mistakes efficiently. In general, there are two approaches for recovery that are commonly used. These are the *undo* and *abort* operations.

In order to implement the undo operation, a history of the previous state is maintained so that the system can be restored to its previous condition. In our painting task, this information is the previous colour that a region had. Therefore, an extra field for storing the previous colour record's pointer will be required in the region record.

The algorithm for undo is simple. All we need is a mechanism that can interrupt the painting algorithm. The information stored in the set R is all that is needed to recover the original state, i.e. prior to the painting operation. Since the set R contains the regions that are already painted by the current painting operation at the time of interruption, undoing the operation can be completed by simply re-painting the regions in R with their previous colours.

An algorithm for undo is described in the following:

$$\forall r \in R, \text{Map}(\text{Old_Colour}(r), r),$$

where $\text{Old_Colour}(r)$ retains the previous colour information in the region record r .

The undo operation depends solely on the number of regions that are painted during a painting operation regardless of what technique is used. This algorithm is an optimal solution because only those regions that need to be recovered are included in the undo.

Since the undo uses *Map* to recover the colour, the performance of the operation depends on how efficiently *Map* is carried out. We mentioned that *Map* can be implemented via two approaches: direct mapping and colour-sharing. In the direct mapping approach, the performance of *Map* is in real-time because of the instantaneous colour feedback. The undo operation would be very efficient in this environment. However, in the colour-sharing approach where each *Map* procedure has to initiate a time-consuming colour filling algorithm for feedback, the undo is impractical.

To handle a painting error, we ultimately want to change the colour of the region to some desirable one, not just recover a previous state. Hence, the undo operation is only an intermediate step. But in the colour-sharing approach, having this time-consuming intermediate step seems undesirable since we can change the problem region to the desired colour directly. Therefore, the abort operation would be more desirable in this environment.

Abort is simply an interruption of the painting algorithm so that the operation can be stopped. Without the undo capability, the visual feedback of the regions will fall into one of the following three categories: totally painted, partially painted, and not painted by the painting operation.

Conceptually, we would consider the totally painted and the partially painted regions as the ones that are affected by the painting operation but not the third type. Functionally, we also want the mapping of the region and the colour to be consistent with the user's concept of these functions. Therefore, let us recall the two steps involved in the *Map* function: the function mapping and the feedback procedures. The abort interrupt should only be allowed during feedback but not during mapping. This ensures that while a function mapping has been made, the feedback shows at least some indication of which region is being painted. Conversely, if there is no visual colour feedback in a region, we can be confident that the region does not need to be handled.

The abort operation can be described functionally as:

$$\forall r \in R, \text{Abort_Fill}(r)$$

where we assume that each region filling operation of the colour feedback can be interrupted.

Sometimes, we may not want to abort a painting function entirely if only a few errors occur during the operation. An abort tool that stops only one particular filled region would be more practical. For example in Figure 3.7, when the stroke comes to the region R_x , we know that it is a region that is not intended for painting. However, we do not want to abandon the operation because the other regions that are yet to be filled seem to be correct. If we could simply identify the particular region R_x for the abort function, it would save the user some effort in having to re-initiate a similar painting operation again later.

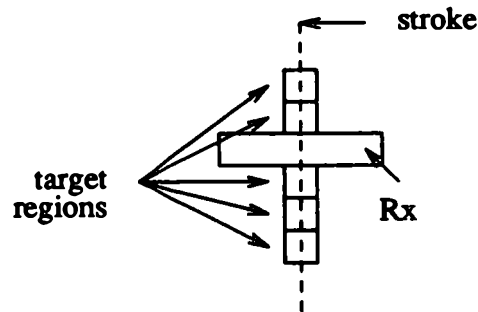


Figure 3.7. — An error in stroking

The region-based abort function can easily be developed. What it requires is a positioning interaction at a point within the region (similar to the region-pointing interaction). The function can be described as:

Abort_Fill(Region(cursor_x, cursor_y))

3.8. Parallelism

A painting operation consists of an interactive component (e.g. drawing a fence, locating a region, and defining a stroke) and an algorithmic component for a colour filling update. One important requirement of the interaction is that the user should not be blocked while interacting with the system. In this case, the user should not be waiting for the filling algorithm to finish before other painting interactions can be initiated. Otherwise, frustration may result because humans do not like to waste time. Therefore, one goal in designing the system is to avoid the blocking of the painting interaction.

In a *colour-sharing* system, all painted regions need to be visually updated by applying the fill algorithm, which is a very time-consuming process. In addition, if we allow the interaction to be continuously applied without being blocked, we would like the feedback to appear as soon as the regions affected by the new interaction are visually updated. This means that all the regions to be painted should be visually

updated in parallel.

For example, in Figure 3.8, if stroke 1 is immediately followed by stroke 2, we would like the regions painted by these two operations to be visually updated simultaneously. Otherwise, if the regions are updated one at a time, that is sequentially, the regions lying underneath the second stroke are not updated until all the regions in the first stroke are filled. This will cause the user to doubt whether the second stroke is doing anything at all. Therefore, a practical painting system should achieve two goals: unblocked interaction and parallel visual updates.

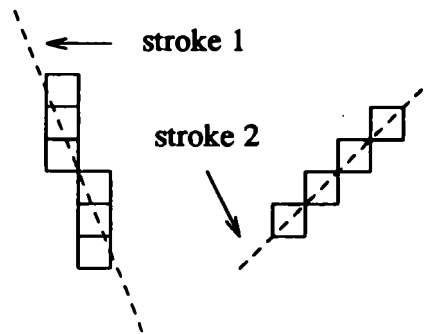


Figure 3.8. — *Unblocked interactions*

3.8.1. Multi-Tasking Approach

In a multi-tasking environment, the painting interaction can be implemented using a non-blocked server. Upon receiving a painting interaction, a new task can be created for executing the painting algorithm. Since each painting algorithm requires only some locally stored information (described in the last section), and the region information from the virtual frame buffer (which is in the shared space), a newly created task does not need any communication link to its parent or to other tasks. This allows the task to run asynchronously and independently. In fact, the task can even destroy itself upon completion without reporting this to other tasks, which makes for a simple implementation.

Each task of a painting algorithm may create additional filling tasks for each individual region colour update. These filling tasks are asynchronous and independent of other tasks because the filling algorithm works locally with only its own fill stack and the shared virtual frame buffer. In this manner, the parallel visual feedback of the painted regions can be accomplished easily. However, we may have a situation where two painting operations try to colour the same region. We need to devise a mechanism so that only one painting operation can have access to a region for filling. Otherwise, the issue of colour collision would have to be faced.

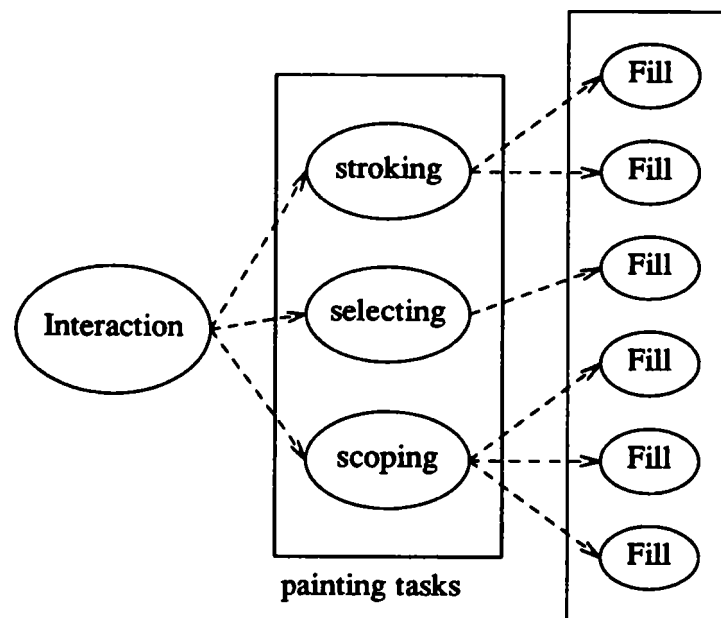
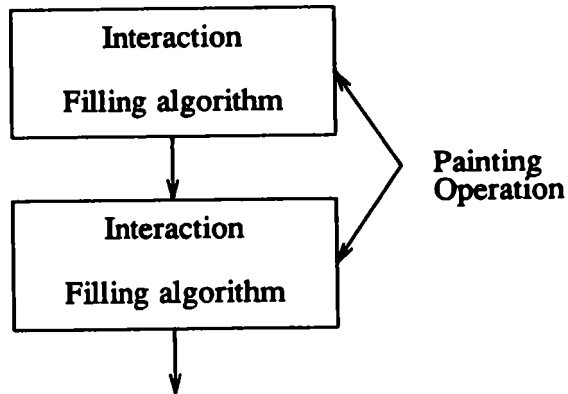


Figure 3.9. — *Multi-tasking example*

A solution to the problem is to implement a monitor for region access. Before each region is to be painted, it should check with the monitor whether the region is currently being painted. If not, the monitor grants access to this particular painting operation; otherwise, access is denied.

3.8.2. Time-Slicing Approach

Figure 3.10. — *Serialisation of painting interaction*

In a single processor without the multi-tasking programming capability, it is more difficult to achieve the two goals stipulated for the fill interaction. An interaction and a painting process have to be serialised as illustrated in Figure 3.10. To reduce the duration of the waiting period for successive interactions caused by the filling algorithm, we may poll the interaction event from time to time, which requires time-slicing the painting operation so that an interactive task can be interleaved with a filling algorithm.

The algorithm can be depicted as follows:

```

loop {
    Poll Event
    If (Event exists)
        put the corresponding painting operation in queue
    Perform one segment of the painting operation
}
  
```

The filling is still carried out sequentially, however, this does not matter because all that is required is that the interaction be allowed to continue without delay. The polling mechanism solves the problem beautifully, provided the segment of the

painting operation is of short enough duration.

The issue of parallel visual feedback has to be addressed as well. Fortunately, this problem can be handled quite easily with a multi-stack implementation for the region filling algorithm. Since the filling of a whole region can be done with a single algorithm, we need only one copy of the algorithm. We have adopted a stack-based filling algorithm [FISH85] because it contains looping which can be used to time-slice the filling operations.

Basically, the fill algorithm acquires the run-length information of a group of horizontally connected pixels from the stack. Then, a selected colour is assigned to these pixels. A few more neighbouring run-length candidates are found and placed in the stack to be processed in future loops. Hence, a region filling operation depends entirely on the information in stack.

If there are many regions to be painted, one stack can be allocated for each region filling process. Then to achieve a parallel update, a simple cyclic scheduling scheme can be used to access the different stacks in some sequential order for each poll loop. For example, assume two regions A and B are being painted simultaneously. If the fill cycle has been used to update a run-length in region A, then the next cycle should process a run-length in region B in an alternating fashion.

```
loop {  
    Poll Event  
    If (Event exists)  
        put the corresponding painting operation in queue  
    Perform one segment of the painting operation  
        allocate stack for each new region painted  
    Perform one loop in filling  
}
```

A further advantage of using the multi-stack implementation is that it allows an easy abort operation for any region; we just empty the corresponding stack.

4. Boundary Completion Techniques

In our region-based model, a region can be uniquely identified by the labelling process if it is entirely delimited by a closed boundary. We assume that every region identified by the labelling process is equivalent to a human perceptual region. However, in reality, some regions without closed line boundaries are still perceived as regions by the human visual system. This broken boundary violates our model's assumptions and prevents our painting operations from functioning correctly. Therefore, it is necessary to complete the missing boundaries of perceptual regions so that they can be identified correctly by the labelling process. In this chapter, we describe two general approaches, using automatic and interactive techniques, that may solve the boundary completion problem.

4.1. The Nature of Broken Boundaries

In a line drawing, the broken boundaries that are most frequently found are those which are not easily detected visually, unless they are closely examined. Although the gaps may not be left intentionally in the image by the animator, their presence is still acceptable because the human brain can perceptually reconstruct the missing information. For example, examining Figure 4.1(a), we can mentally interpolate the missing line that is required to complete the square (depicted in Figure 4.2(a)).

Sometimes, the animator intentionally uses special arrangements of some broken line segments to create an illusory region. For example, in Figure 4.1(b), we perceive an imaginary circle which is circumscribed by the end points of the straight lines (Figure 4.2(b)). Although the circumference of the circle does not exist in the figure,

our minds still regard the illusory circle as a region.

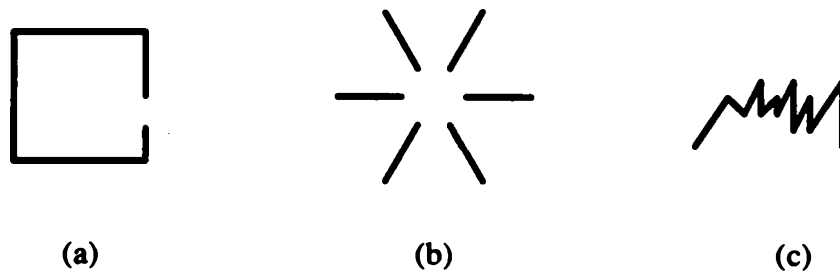


Figure 4.1. — *Perceptual regions without closed boundaries*

Sometimes, the contextual information of the image affects how we interpret the lines and regions. For instance, when we look at Figure 4.1(c), it is not clear whether the figure constitutes a region or not. However, if the figure appears in a scene with trees, we may conclude that this figure represents the grass, according to the contextual information provided. The boundary of this grass figure is then interpreted perceptually by the painter.

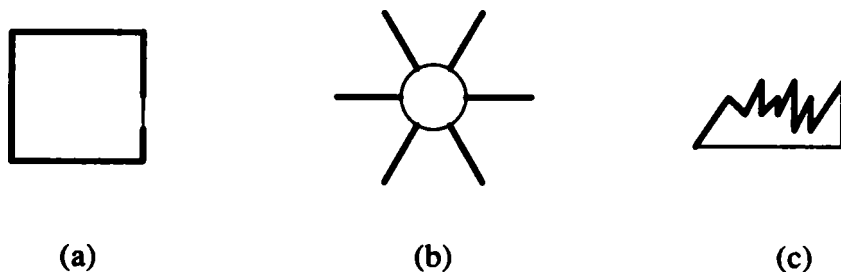


Figure 4.2. — *Perceptual boundary completion*

4.2. The Problem of Leaking

Although all these examples of "open" regions are commonly encountered in line drawing images, our region painting techniques, described in the last chapter, cannot handle them as effectively as humans can. This is because all regions that are recognized by the computer must be delimited by some boundary information, which in our case depends on the lines presented in the image. If a boundary gap exists, the

areas on both sides of the gap are treated as the same region because they are connected areas by definition. So, when a painting operation is applied to this "open" region, the area outside the gap is painted as well. We say that the region has a *leak*.

Therefore, we need to find ways of telling the system that the areas divided by the missing boundary are actually distinct regions so that our painting techniques can be applied correctly.

4.3. Automatic Boundary Completion

It is highly desirable to have an automatic boundary completing algorithm to take care of all the leaks that might appear in the image. If such an algorithm exists, it should be applied before the region labelling process is invoked so that each perceptual region can be uniquely identified. Various algorithms have been used in attempting to solve the problem of broken boundaries. Krishnan, in his PhD dissertation [KRIS88b], gives a thorough survey of existing boundary completion algorithms. In particular, there are three approaches that should be mentioned briefly.

Ullman's algorithm [ULLM76] generates a line between two end-points of a broken curve according to the constraint of minimum curvature, to guarantee the smoothness of the new curve. However, with this smoothness constraint, it is impossible to fix any sharp broken corners that may occur in an image.

Grossberg and Mingolla [GROS87] noticed the problem of illusory contours and suggested a boundary completion method to handle it.

Krishnan and Walters' [KRIS88a] ρ -space approach for boundary completion is by far the most successful one, in our opinion. This approach is motivated by the psychophysical experimental results of Walters [WALT87a] showing how humans perceive end-connection of lines. The algorithm is capable of constructing illusory contours. In addition, the gap linking mechanism is able to connect a broken curve as well as a broken corner within a local area.

Although existing computer paint programs are able to provide tools for assigning colour to the original line drawing, Walters [WALT87b] mentions three problems that need to be handled in order to speed up this interactive process. Firstly, artists often indicate an image region in a manner that does not require a closed boundary, as the human visual system has the ability to perceptually link the gap. Secondly, regions are often subdivided, not only by the contours of occluded objects, but also by texture or pattern (see Figure 4.3). Assigning colour to an object such as this requires many fill operations for each sub-region. Thirdly, an illusory contour often needs to be considered as a distinct region to be filled with colour. However, no filling operation can achieve this because the contour does not exist in the image.

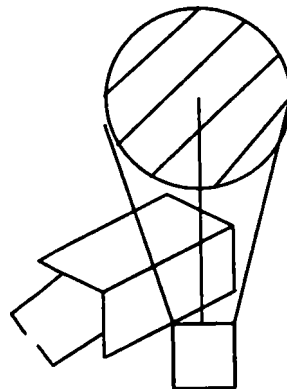


Figure 4.3. — A line drawing [from WALT87b]

The ρ -space algorithm described in [WALT87b], which uses the ρ -space representation, is claimed to solve 80% of the problems mentioned above. Although the ρ -space algorithm seems promising for solving the boundary completion problem, we have not yet incorporated this into our system because the algorithm requires much computation that is impractical without a parallel computing environment⁷.

⁷ The interested individual may refer to Krishnan's PhD dissertation [KRIS88b] for a detailed description of ρ -space implementation in parallel machinery.

Moreover, for the grass region in Figure 4.1(c), the ρ -space approach fails to determine the boundary because this depends on a human contextual interpretation. In fact, we wonder if there will ever be an algorithm that can completely link all the gaps in the image. Therefore, no matter whether an automatic linking process is applied or not, we still need an interactive tool for the user to tell the system how to complete boundaries.

4.4. Interactive Boundary Completion

As we mentioned earlier, some boundary gaps are hard to detect visually. These gaps are only detected if colour leaking occurs when we try to paint a leaky region. Therefore, the boundary completion tools must be incorporated in the cel painting process so that the boundary information of the image can be interactively modified.

4.4.1. Boundary Plane

In our cel painting design, the boundary information used to determine the delimiters of regions for labelling is only required during the labelling process and is not stored in the virtual frame buffer. In order to be able to modify the boundary information for region relabelling, it is necessary to keep the boundary information in the virtual frame buffer. Therefore, we have to extend our virtual frame buffer to include one more data slice called the *boundary plane*⁸.

As mentioned in Chapter 2, the boundary is represented by grid lines. Since these grids are algorithmically constructed based on the one pixel wide thin skeleton representation of the line drawing, we store the thin line skeleton in the boundary plane. Moreover, keeping the thin line skeleton in the virtual frame buffer makes designing of the completion tools simpler.

⁸ The boundary plane is simply a one-bit bit-plane where set bits represent the presence of the boundary.

Observe that the boundary plane is separated from the grey scale line representation in the virtual frame buffer. So, if any modification is made to the boundary plane, the original drawing is not affected. This is indeed desirable because we do not want to add a visible boundary if the artist left out the boundaries deliberately.

4.4.2. Tools

The tools required to modify the content of the boundary plane are simple. All we need is line drawing interaction for adding lines to the boundary plane and an eraser brushing mechanism for deletion.

In solving the leakage problem caused by the discontinuity of a region boundary, we need a drawing technique that can give us a continuous boundary for our boundary completion interaction. There are numerous techniques that can be used to draw a continuous line. In our design, we adopt a line drawing interaction similar to the stroking interaction, where a line is defined with a pen-stroke gesture. In some cases, if stroking is too fast, the computer may not be able to sample the input rapidly enough to produce a continuous line (even though the stroking motion is continuous). In such a case, line interpolation between the sampling points from the input device is necessary to ensure the continuity of the line. The simplest approach is to use linear interpolation, like Bresenham's fast raster line interpolation algorithm [FOLE82].

The second issue that we need to consider in the boundary completion operation is *end-point continuity*. In Figure 4.4(a), a constructed line is required to enclose the region by connecting points P and Q . Because of human limitation in the ability to draw precisely, we may instead obtain the line segments shown in Figure 4.4(b), with the region remaining open.

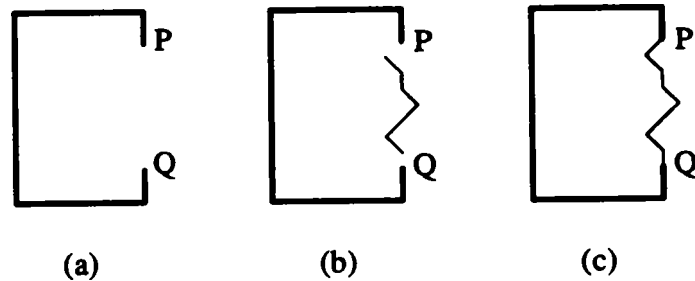


Figure 4.4. — *Example of boundary drawing*

The problem can be solved by applying the concept of *end-point gravity*. Gravity is a field defined at the end points of a newly drawn boundary line. If another boundary line segment exists within the gravity field, the end-point of the newly drawn line is extended to touch that existing boundary line (Figure 4.5).

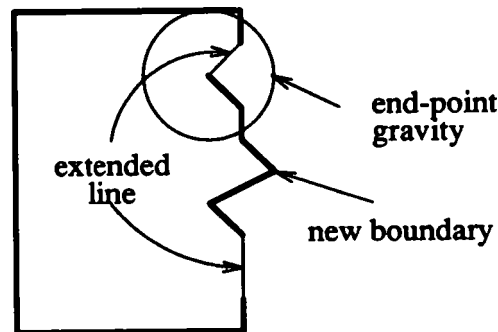


Figure 4.5. — *Example of end-point gravity*

With the technique described above, the boundary linking *P* and *Q* in Figure 4.4(b) may now look like the one in Figure 4.4(c). Notice that the resultant region is not the expected rectangle. If paint is applied to this region, colour will follow the wiggling boundary from *P* to *Q* instead of a straight line. However, to a human being, the colour appears in the correct position if the image is viewed at a "distance." Examples of this can be found in the Sunday comic strips where the colours applied to the regions seldom lie entirely inside the regions. They are still acceptable to the human visual system because when they are placed at a distance, we perceive that the

colour falls correctly into the perceived regions. Therefore, we need not be too concerned as to whether the boundary drawn is exactly the same as the human perceptual boundary. On the other hand, we may sometimes mistakenly draw some undesirable boundary lines. These lines may be unacceptable because they define some additional undesired regions. An additional tool is required to wipe out these boundaries. A simple technique, like eraser brushing, can be used to accomplish this. An example of erasing is depicted in Figure 4.6.

4.4.3. Region Relabelling

The difficulty in designing operations for boundary completion is not how to design the interaction but how to preserve the assumptions required in the region-based model. The difficulty arises because when a boundary line is interactively added, it may separate an existing region into several sub-regions. These sub-regions then share the same region label because they originated from the same region. Also, when erasing is applied, regions may be merged to form a single region and, as a result, the resulting region contains the labels that belonged to the original regions. So in both cases, the assumptions are violated.

In the following two sections, we describe solutions which preserve the assumptions in region splitting and merging respectively.

4.4.3.1. Region Splitting

An additional boundary line usually splits an existing region into smaller regions. In order to preserve the assumption that no two distinct regions share the same label, we need to assign them new labels.

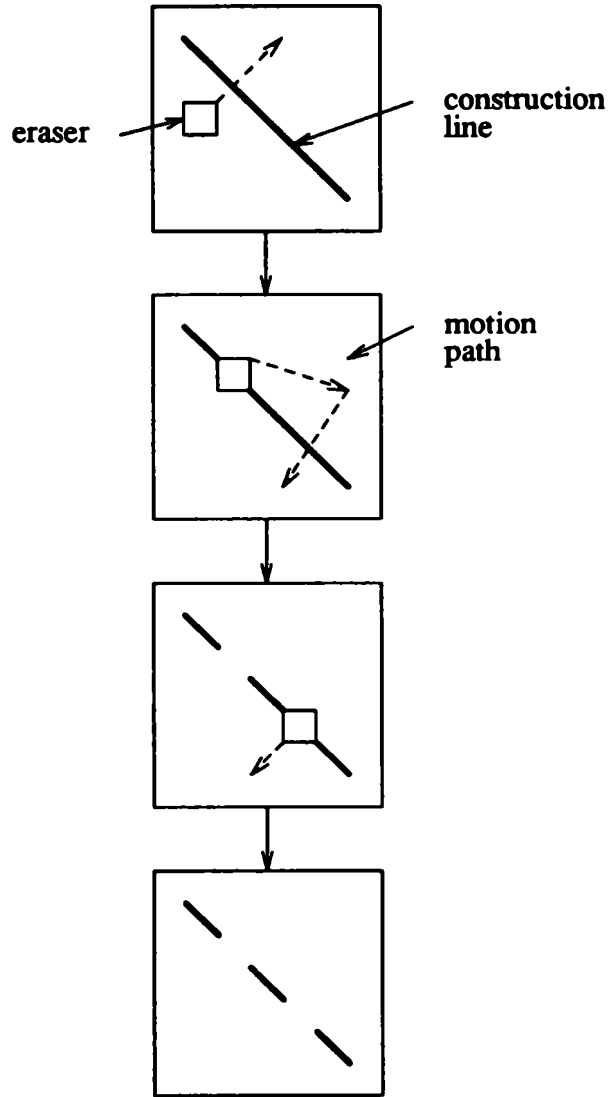


Figure 4.6. — *Example of erasing*

The simplest way to do this is by applying the region labelling process to the whole image again. However, this is not a good approach because the process is too slow to be used in this interactive environment. Instead, we need an adaptive relabelling mechanism that only changes the labels of the affected areas. For example, in Figure 4.7, the new boundary divides the two regions $R1$ and $R2$ into four sub-regions. Ideally, only one of sub-regions a and b and one of c and d need to be relabelled in order to distinguish all four sub-regions. Also, we would prefer to relabel the smaller regions for the sake of minimising the effort in the relabelling process.

We assume that we know how many new sub-regions there are and their seed points. Based on the labelling information at each seed point, we know which sub-regions originated from the same region. For example, in Figure 4.7, since any point in sub-regions a and b have the same label as in $R1$, we know that a and b are from the same region and should be grouped together. Similarly, we can group sub-regions c and d together.

For each group of N sub-regions, we only need to relabel $N-1$ of them. As mentioned, we want to pick the smallest $N-1$ sub-regions for relabelling. However, it is difficult to find the size of each sub-region prior to the labelling process. Therefore, an optimal method of efficiently identifying these $N-1$ sub-regions is impossible. Instead, we use the following less elegant solution.

We initiate the labelling process for all N sub-regions one pixel at a time in a cyclic fashion. While each pixel is being relabelled, we keep track of the number of each label in the image. As soon as the second largest sub-region is entirely relabelled, we refer to the label count and find the number of pixels with the same original label from the unsplit region, X . Also, we can find how many pixels in the largest sub-region Y have been relabelled. If $X > Y$, we undo the relabelling applied to the largest sub-region. Otherwise, the relabelling of the largest sub-region continues.

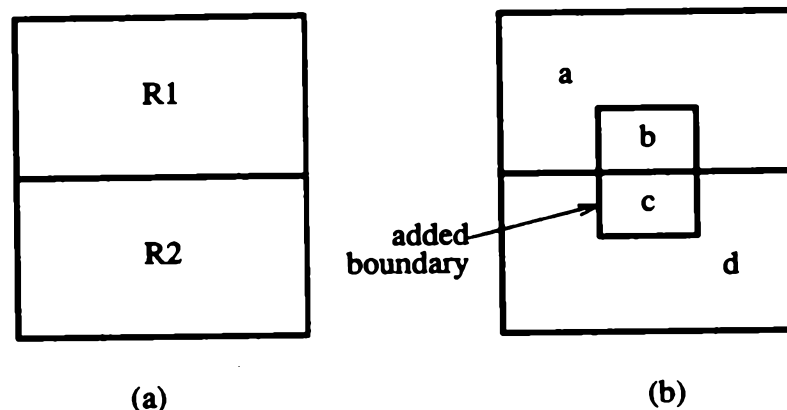


Figure 4.7. — *Region splitting*

Compared with the ideal solution, the above method only involves an extra relabelling of either twice the number of pixels in the second largest region, or the number of pixels in the largest region, whichever is smaller. In most cases, the second largest region is much smaller than the largest region. This is commonly found to hold in the boundary completion of leaky regions that are connected to a generally large⁹ background region (e.g. the base of the house in Figure 4.3). Therefore, this method generally performs well.

Since the method we have described assumes that the number of new sub-regions and each of their seed points are known, we have to show how this information can be obtained. To find these seed points, one only needs to examine the points on either side of one of the new boundary lines, since each new region always uses the new boundary as part of its delimiter. This means that all the points lying on both sides of the new boundary belong to the outermost pixels of the new sub-regions. If we can find how many *traces* of such regions' outermost pixels pass through the points along both sides of the boundary, we will know how many regions are candidates for relabelling. For example, in Figure 4.8, 4 traces can be found along the new

⁹ In a cel drawing, since objects usually occupy only a small area of the image, the background region is usually very large.

boundary.

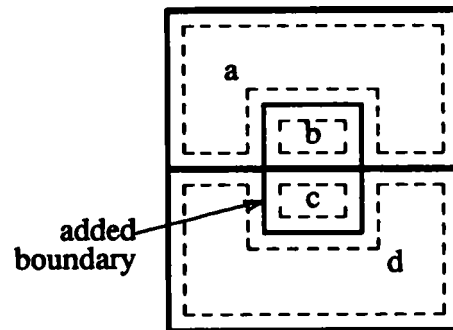


Figure 4.8. — *Boundary tracing*

After the process of finding the traces¹⁰, any point that belongs to the trace can be selected as a seed point of the corresponding region. When these seed points are found, the grouping of regions can proceed, and the relabelling can be carried out.

4.4.3.2. Region Merging

When regions are merged after erasing has removed part of a boundary line, a conflict may be created if more than one label refers to the same merged region. Although applying the relabelling operation to the merged region may solve the problem, a more efficient approach is used.

All that is required is to allow the labels of the original regions to refer to the same region labelling information. This can be implemented using a linked list. When two regions are merged, we choose one of the labels as the primary label, which is used to represent the merged region. The other label has a pointer to this primary label so that pixels containing this other label can refer to the primary labelling information as well.

¹⁰ The tracing algorithm can be found in Appendix C.

For example, in Figure 4.9, the regions $R1$ and $R2$ are merged after the boundary line between these regions is broken. Let us assume that $R1$ has the label 1 and $R2$ has the label 2. If we decide that label 1 is the primary label, there will be a pointer from label 2 to the primary label 1. Now, with this additional data structure, although there may be two or more different labels in the merged region, they all share the same primary label. So, whenever we want to know to what label a specific pixel belongs, we refer to the primary label.

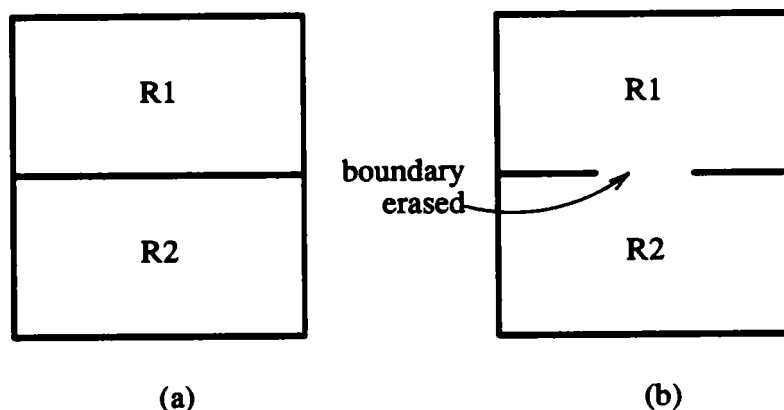


Figure 4.9. — *Region merging*

We consider each region as consisting of a set of labels, with one label identified as the primary label. The merging of the regions is equivalent to the *set union* of two sets of labels. To find the primary label of each label, we need a function which returns the name of the set the element is in. For example, in Figure 4.9, before the two regions are merged, they are labelled set $\{1\}$ and $\{2\}$, called *label sets* 1 and 2 respectively. When the regions are merged, these label sets are removed and a new labelling set $\{1,2\}$ is formed, and it is known as the region label set 1 if we choose label 1 as the primary label. When set finding is applied to either label 1 or label 2, we obtain the name of the region set, 1, which is the same as the primary label. Therefore, label merging can be achieved by applying the *Union* and *Find* set operations. An efficient algorithm for handling both operations, known as *Union-Find*, has been developed and can be found in [AHO74].

5. Multi-Frame Region Tracking

An animated production usually requires 24 to 30 frames for each second of animation. Since the timespan between two consecutive frames is short, there is little movement of an object from one frame to the next. This *frame coherence* property leads us to ask whether an automated *region tracking* system could be developed to find the corresponding regions of two successive frames, so that the appropriate colours from one frame can be copied to the succeeding one. Such a system would reduce the effort required in some interactive painting tasks which in turn would lead to greater productivity. With our region-based design, the tracking process can be transformed into the well-known *graph matching problem*. In this chapter, we give a formal description of the tracking problem in terms of the graph matching problem.

5.1. Formal Description of Tracking

The objective of tracking is to establish a relationship between the set of regions of an already painted frame and the set of regions of the succeeding unpainted frame. This relationship may be based on the similarity of the regions' shapes, sizes, positions, etc. (Figure 5.1). Using this relationship, the regions of the unpainted frame can be assigned the colours of the corresponding regions from the previous frame.

With our region-based model, this task can be formally described as finding a maximal set of functional mappings

$$f:R_{cur} \rightarrow R_{prev} = \{(a,b) \mid a \text{ and } b \text{ are similar regions}, \forall a \in R_{cur}, \forall b \in R_{prev}\}.$$

R_{prev} is the set of regions of the previous painted frame and R_{cur} is the set of regions of the succeeding unpainted frame, i.e. the current frame.

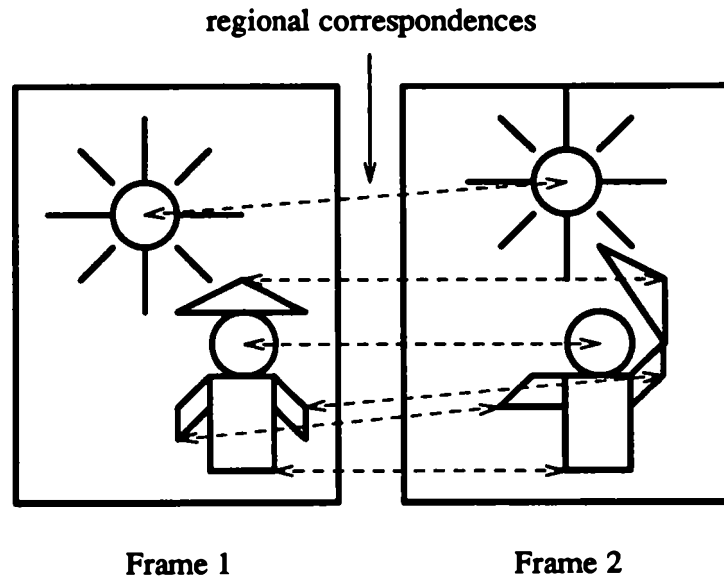


Figure 5.1. — *Example of region tracking*

In order to compare the similarity of two regions, we introduce a *region similarity measure* which is a quantitative measurement of how similar two regions from two different frames are. The smaller the measure is, the greater the similarity of the two regions. To determine whether regions a and b are similar from their similarity measure, $s(a,b)$, we use a similarity threshold value, S . If $s(a,b) < S$, then a and b are considered similar regions. With this definition of region similarity, the function can be described as:

$$f:R_{cur} \rightarrow R_{prev} = \{(a,b) \mid s(a,b) < S, \forall a \in R_{cur}, \forall b \in R_{prev}\}.$$

Methods of defining the similarity of two regions is of extreme importance and needs further study. Later in this chapter, we present some suggestions for the definition of such a function.

The tracking problem stated above can be transformed into a well-known problem in graph theory, called the maximum *bipartite graph* matching problem. A *bipartite graph* is a graph in which the vertices of the graph are divided into two disjoint sets (X and Y), and no two vertices of the same set have an edge connecting them (Figure 5.2(a)).

Let us consider the two sets R_{prev} and R_{cur} to be two sets of vertices in a graph. Also, if two regions, a and b , are found to be similar (i.e. $s(a,b) < S$), we assign an edge between the two corresponding vertices in the graph. Since the similarity measure only applies between two regions of the distinct sets R_{prev} and R_{cur} and not between regions within the same set, it is obvious that the resulting graph is a bipartite graph.

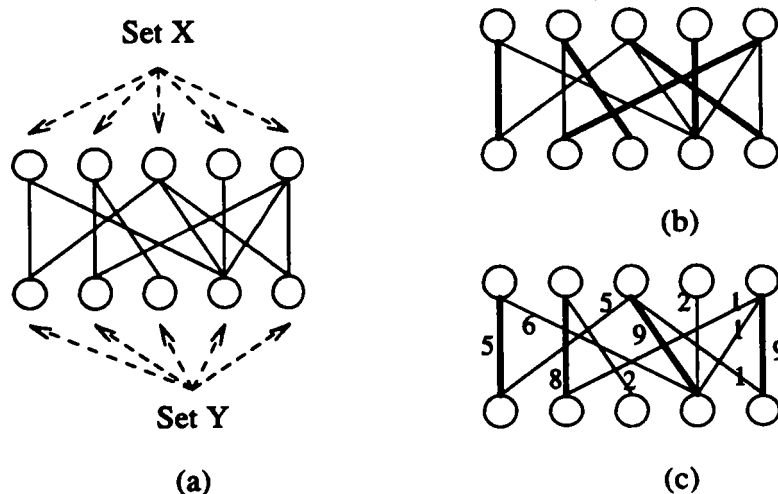


Figure 5.2. — *Bipartite graph matching*

There are two approaches we can use to obtain a set of regional correspondences in a bipartite graph. If we treat all edges in the constructed bipartite graph as equally desirable for potential matches regardless of the differences in their similarity measures, finding the *maximum matching* of the graph will give us the maximum number of correspondences between two sets of regions (Figure 5.2(b)). On the other hand, if we associate a weight designating the significance of each edge according to its similarity measure (e.g. weight function: $w(a,b) = S - s(a,b)$), then finding the *maximum weight matching* of the graph gives us the set of edges that gives the maximal sum of weights, thereby yielding better (or more significant) matches (Figure 5.2(c)).

Note that these two approaches have their tradeoffs. While the maximum weight matching gives a better matching, the number of correspondences found may be fewer than in the other approach. Moreover, the algorithm for maximum weight matching is less efficient. Hence, further study is required to evaluate the practicality of the two approaches to region tracking.

5.2. Algorithmic Complexity

The tracking process, using the graph matching approach, consists of two stages. Firstly, for the given two sets of regions, we need to construct a bipartite graph by applying the similarity measure from one region of one set to every region of the other set. Thus, the time required to construct the graph is $O(n^2)$, where n is the number of vertices in the graph.

Secondly, we need an algorithm which will find the maximum matching given by the bipartite graph. If the graph is unweighted¹¹, there is an algorithm that runs in $O(m\sqrt{n})$ time, where n is the number of vertices and m is the number of edges. This algorithm is called Dinic's algorithm¹² [DINI70], and its time complexity analysis can

¹¹ An unweighted graph is one in which every edge is equally desirable as a match.

¹² A brief description of Dinic's algorithm can be found in [YAO82].

be found in [TARJ83].

Originally, Dinic's algorithm was intended to solve the network max-flow problem, not the maximum matching problem. However, transforming a bipartite graph matching problem into a network max-flow problem can be done easily by adding two vertices s and t , edges (s, x) , for every $x \in X$, and edges (t, y) , for every $y \in Y$, (Figure 5.3) [TARJ83]. If we make the flow capacity of each edge equal to one unit, finding the routing for maximum flow from s to t will yield the maximum matching from X to Y .

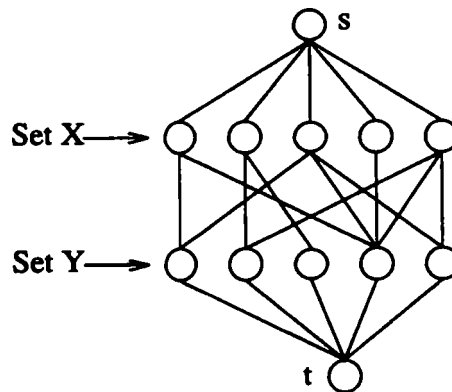


Figure 5.3. — Network flow problem

Similarly, the maximum weight matching problem can be solved by the max-flow problem by viewing the weights as flow capacities. In [TARJ83], it has been shown that the time required to solve maximum weight matching is $O(nm \log_{(2+m/n)} n)$, which is less efficient than the unweighted maximum matching case.

The tracking process as a whole requires $O(n^2) + O(m\sqrt{n})$ time to complete the maximum matching of an unweighted graph and $O(n^2) + O(mn \log n)$ time in the case of maximum weight matching. Notice that $m \geq n$ because of the additional edges required to connect s and t to the bipartite graph. Therefore, the complexity of the tracking process with the maximum weight matching approach is bounded by $O(mn \log n)$.

5.3. Measurement of Region Similarity

As we have mentioned, if we can construct a bipartite graph from the region information of two consecutive frames, an efficient algorithm can be applied to solve the tracking problem. However, the construction of the bipartite graph requires the measurement of region similarity, a function that is not clearly defined. In the following sections, we will examine the region similarity problem and suggest some possible methods for measuring region similarity.

5.3.1. Size Coherence

In most cases, the sizes and shapes of the regions do not vary drastically from frame to frame. A large region usually remains large in the next frame, and the same holds true for small regions. Hence, comparing region sizes is one way to evaluate region similarity.

The size of a region can be measured easily by counting the number of pixels contained in that region. Therefore, a simple form of similarity measure can be described in terms of the difference of the regions' sizes:

$$s(a,b) = \Delta Size = |Size(a) - Size(b)|$$

Of course, this method of measuring region similarity is very limited because usually there are many regions of similar size in the image. For instance, in Figure 5.1, the regions representing the sun and the head are the same size. With only this size similarity measure, we cannot prevent the matching algorithm from matching the sun of Frame 1 with the head of Frame 2. Thus, a more sophisticated method is required to further distinguish the similarity between regions of the same size.

5.3.2. Positional Coherence

In an animation sequence, objects seldom move rapidly around the frame. If there is a position change, it is usually gradual; hence, the differences in positions of the regions from one frame to the next are usually small.

If we can represent a region's position by a point $p=(x,y)$, we can construct a position similarity measure:

$$\Delta P = \text{distance}(p_a, p_b),$$

where p_a and p_b are the positional points of region a in set R_{cur} and of region b in R_{prev} respectively.

For example, in Figure 5.1, it is obvious that the position of the head in Frame 1 is closer to the position of the head in Frame 2 than to the position of the sun. So, by using this second criterion, regions of the same size can be further distinguished by their positional similarity.

The position similarity measure assumes that a positional point is uniquely defined for each region. However, the problem of uniquely defining the position of a region requires further consideration.

If the region is a solid convex polygon, the centre of mass of the polygon, obtained by averaging the positions of all the pixels within the region, can be used to represent the position of the region. The centre of mass always lies within the region, thus, the region's centre can be used to uniquely represent the region if all the regions are also convex in the frame.

However, if there is a region that is not solid and convex (i.e. non-convex or contains holes), the centre of mass calculated by averaging may not lie within the region. For example, in Figure 5.4, regions $R1$ and $R2$ have the same centre of mass calculated as point p . In such a case, the centre of mass can no longer be used to represent the position of a region uniquely.

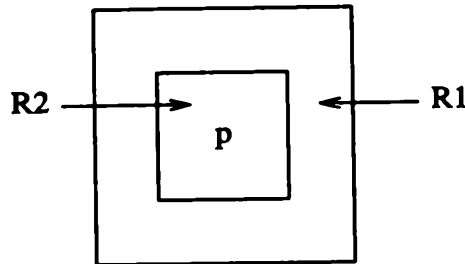


Figure 5.4. — *Centre of mass*

An ad-hoc solution would be to use the same averaging calculation but with some modification. If the calculated centre of mass is within the region, it is used as the positional point, as before. Otherwise, we search for the point in the region which is the closest to the centre of mass, and use this instead as the positional point. This ensures that the positional points are always distinct from each other because each positional point is always inside the region it represents.

This ad-hoc solution is crude, and further refinements to achieve better positional representation are yet to be realized.

5.3.3. Image Transformation

The position similarity measure introduced in the last section only works well if all the regions undergo a simple spatial translation. However, motion in an animated sequence involves more than spatial translation. There may be rotation, scaling, zooming, spinning, etc.

In Figure 5.5, we have a rotation of an object consisting of three regions. If the simple positional measure is applied, it may result in an incorrect matching. On the other hand, if we can apply a rotation transformation on the positional points prior to the similarity measurement, a more accurate similarity measure can be obtained.

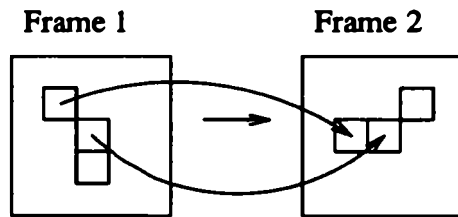


Figure 5.5. — *Incorrect matching of rotated object*

Since we know that any transformation can be expressed in terms of a transformation matrix, we can describe the modified position measure as

$$\Delta P' = \text{distance}(p_a, p_b'),$$

where $p_b' = Tm \times p_b$, (Tm = the transformation matrix). For example, for rotation transformation, the transformation matrix is

$$\begin{pmatrix} \sin\theta & \cos\theta \\ \cos\theta & -\sin\theta \end{pmatrix}$$

where θ is the the angle of rotation.

One may wonder, "How can we know what transformations the objects undergo between two consecutive frames?" In our opinion, the best way to approach this problem is to allow the user to interactively determine the transformation. This will, however, introduce additional human interaction overhead which seems to contradict the original objective of tracking which was to reduce human interaction. Therefore, further experimental study is required to test whether or not the additional tracking process actually increases efficiency in cel painting.

Furthermore, the transformation matrix is limited so far to a two-dimensional transformation because the regions exist in a two-dimensional world. However, in reality, animators usually think of the animated objects in three-dimension and then draw their projected two-dimensional images on paper. Hence, some objects in an animation sequence may appear to undergo some three-dimensional transformations.

For instance, we may see a spinning object; in which case, some of its visible regions may be hidden, and vice versa. In our transformation model, this situation cannot be handled correctly due to the lack of the animators' perceptual three-dimensional modeling information based on the two-dimensional images given. This is a difficult problem that needs to be solved.

5.4. Birth and Death of Regions

Until now, we have assumed that the regions between two frames always have a one-to-one correspondence. In other words, neither the *birth* of a new region nor the *death* of an existing region occurs from one frame to the next. However, the birth and death of regions frequently happens in an animation sequence. For instance, the spinning of an object may introduce some new regions (i.e. birth) and may hide some visible regions (i.e. death).

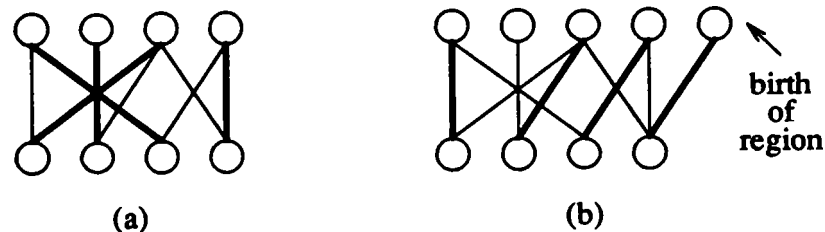


Figure 5.6. — *Incorrect matching with the birth of a region*

This birth and death of regions may cause a serious problem in the region matching process because the one-to-one regional correspondence assumption no longer holds. The maximum matching algorithm, however, can still make a match, but the result would likely be incorrect (Figure 5.6). This is because the available regional information (i.e. size and position) in our system is insufficient for recognizing the birth or death of regions. The algorithm assumes that these regions correspond to some regions in the other frame. As a result, these regions are included as candidates for region matching and affect the outcome of the matching.

Isolating these regions from consideration in matching seems to be a very difficult problem. Nevertheless, this is a problem of extreme interest in many disciplines and is worthy of exploration in future.

5.5. Future Research

The region-based model has allowed us to look at the tracking problem from a new perspective: graph matching. However, more studies are required before the tracking process can be made practical.

A better method for similarity measure must be designed so that the similarity of regions can be evaluated more accurately for the later matching process. Also, further experiments are required to determine what the similarity threshold should be in order for two regions to be candidates for a potential match.

Achieving automatic region tracking appears to be a very difficult problem due to the inadequacy of the information given by the two-dimensional cel images. Nevertheless, adding human interaction may aid in solving the problem by allowing manipulation of the individual images so that the correct spatial transformation can be applied to the regions for accurate matching. Since this kind of tracking involves human interaction, further experimental study is required to compare production efficiency with and without the tracking process.

At present, our approach in region tracking cannot handle the problem of birth and death of regions because the limited regional information available is not sufficient for detecting them. Thus, this problem remains unsolved and needs to be explored.

6. Conclusions

In traditional cel animation production, most tasks involve tedious and repetitive operations. Painting of cels is the major bottleneck of this process because each cel is painted by hand. With the advancement of computer technology, the animation industry is changing to include computer aided production systems, in order to minimize the unpleasant and inefficient tasks that exist in traditional animation. Several computer paint systems have been developed over the years, but improvement can still be made, especially in painting the cels.

We have introduced a conceptual region-based painting model that is based on the human perception of the painting process. The model uses a virtual frame buffer that allows functional description of the painting task. With the region-based model, conceptual tools like region-pointing, stroking, fencing, radiating and bounding scope techniques can be described functionally. We have also shown that these techniques are all subsets of a general description of a unified painting operation. With the unification of these painting techniques, algorithms can be developed easily.

In order to develop painting tools that operate effectively, special attention must be paid to issues such as system limitations, redundant operation avoidance, error recovery, and parallelism in colour filling.

We have developed a prototype of the region-based painting model on a workstation consisting of a single processor, an 8-bit video frame buffer, and a tablet with a two button stylus as the input interaction interface. Among the painting techniques described, the region-pointing, stroking, and bounding scope techniques

have been successfully implemented and tested. Furthermore, we use a time-slicing approach for scheduling the various painting processes so that the system supports parallel region colouring.

Region-based painting techniques do not operate well if the image contains broken boundaries. Techniques are required to overcome the resulting leakage problem. Generally, two approaches to the problem are available. First, some broken boundaries may be linked algorithmically using, for example, the ρ -space algorithm [KRIS88a]. In our system, we adopt the second approach to linking, which is to interactively define the missing boundaries. This uses the concept of a boundary plane which is basically an extended virtual bit-plane that can be easily incorporated into the existing virtual frame buffer model. In the interactive linking process, regions may be split or merged. As a result, the assumptions of the region-based model may not be met, but solutions are described to ensure that the assumptions of the model are met in the cases of region splitting and merging.

Finally, based on the assumption of the frame coherence property of two consecutive frames in an animation sequence, we feel that a region tracking process may be beneficial in speeding up the whole painting process. The region-based model provides us with a way to transform the tracking problem into the well-known maximum matching problem of a bipartite graph. Possibilities for future research in the tracking process have also been outlined.

Appendix A. Thinning Algorithm

In our region-based model, lines are used as region boundaries. However, in a video scanned line drawings, we often find great variation in line widths; with this variation, region boundaries cannot be defined easily. We prefer a more uniform line representation so that region delimiters can be better defined. One way to obtain a uniform line representation is to apply a thinning algorithm to find the skeleton of the lines.

Zhang and Suen [ZHAN84] developed a thinning algorithm that is able to find a one-pixel-wide 8-connected skeleton representation of each line. The resulting skeleton lies relatively close to the centre of its original thick line, and thus can represent the region boundary.

Since their thinning algorithm applies to a binary images only, we need to transform the grey-scale images to binary images by applying an image thresholding process to separate the image value into two classes: background and foreground. Then we can simply mark 0 for the background pixels and 1 otherwise to obtain the binary image.

The algorithm is a two-step iterative process. The first step flags a non-background pixel p for deletion (i.e. replacement by a zero value) if the following conditions $COND_A$ are satisfied:

$$(a) \ 2 \leq N(p_1) \leq 6,$$

$$(b) \ S(p_1) = 1,$$

$$(c) \ p_2 \times p_4 \times p_6 = 0,$$

$$(d) \ p_4 \times p_6 \times p_8 = 0,$$

where $N(p_1)$ is the number of nonzero neighbours of p_1 ; that is,

$$N(p_1) = p_2 + p_3 + \dots + p_8 + p_9$$

and $S(p_1)$ is the number of 0-1 transitions in the cyclic ordered sequence of $p_2, p_3, \dots, p_9, p_2$ (see Figure A.1 for reference of p 's). For example, $N(p_1) = 5$ and $S(p_1) = 3$ in Figure A.2.

In the second step, the only differences in the conditions are (c) and (d). These *COND_B* are:

$$(c') \ p_2 \times p_4 \times p_8 = 0,$$

$$(d') \ p_2 \times p_6 \times p_8 = 0,$$

The iterative algorithm works as follows.


```
continue ← TRUE.
while ( continue )
{
  /* Step 1 */
  ∀ p≠0, flag p for deletion if all COND_A are satisfied.
  if there is no flagged pixel,

else
{
  replace all flagged pixels by zero.
  /* Step 2 */
  ∀ p≠0, flag p for deletion if all COND_B are satisfied.
  if there is no flagged pixel,
    continue ← FALSE
  else
    replace all flagged pixels by zero.
}
}
```

p9	p2	p3
p8	p1	p4
p7	p6	p5

Figure A.1. — Neighbourhood arrangement used by the algorithm

Notice that the flagged pixels are deleted only after the flagging mechanism is applied to the whole image. This is done to avoid changing the image structure.

0	1	1
1	1	0
1	0	1

Figure A.2. — *Illustration of conditions (a) and (b) in COND_A*

In [GONZ87], three standard constraints of a thinning algorithm are stated. These constraints require that if a pixel of the image is deleted, it (1) does not remove end points, (2) does not break connectedness, and (3) does not cause excessive erosion of the region. By examining both conditions of the Zhang's and Suen's thinning algorithm, we find that the algorithm has fulfilled these constraints.

Condition (a) is violated if the examined pixel has less than 2 adjacent pixels. This implies that the pixel is either a point element or an end point of a line. Therefore, this satisfies constraint (1) mentioned above. Condition (a) can also be violated if the examined pixel has 7 or 8 neighbours. Deletion of the pixel will cause erosion into the region. Hence constraint (3) is guaranteed by this. Condition (b) prevents disconnection of an existing line that is one pixel thick. A few examples are depicted in Figure A.3 that violate condition (b). Conditions (c) and (d) delete east or south boundaries and the northwest corner. Conditions (c') and (d') delete north or west boundaries and the southeast corner. Therefore, the difference between the two steps is simply the direction of the thinning applied.

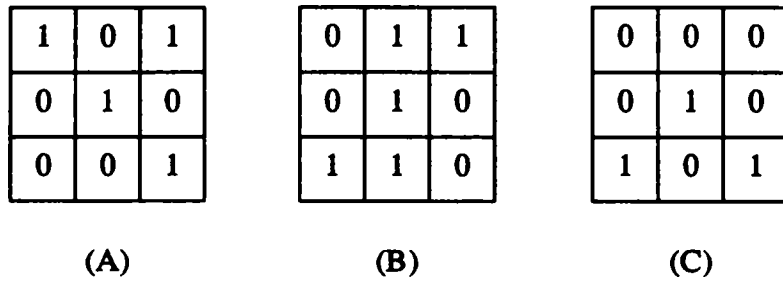


Figure A.3. — *Illustration of violation of condition (b)*

Appendix B. Grid Transformation Algorithm

An assumption stated in our region-based model is that each pixel of the image canvas must belong to one of the regions. This implies that the region boundary representation must not include any pixels; otherwise, there will be pixels that do not belong to any region but to the line boundary. Therefore, we find that the thin line representation for region boundary is not ideal because it does occupy some pixels.

We need a better boundary representation that does not occupy any pixel. We adopt a grid line representation, which exists between two adjacent pixels, to be a new line boundary representation (Figure B.1). In order to obtain the grid line boundary, the thin line representation is still required. An algorithm is described here to give a grid line representation of the region boundary based on the given thin line boundary representation.

The following is an algorithm that determines the grid boundary:

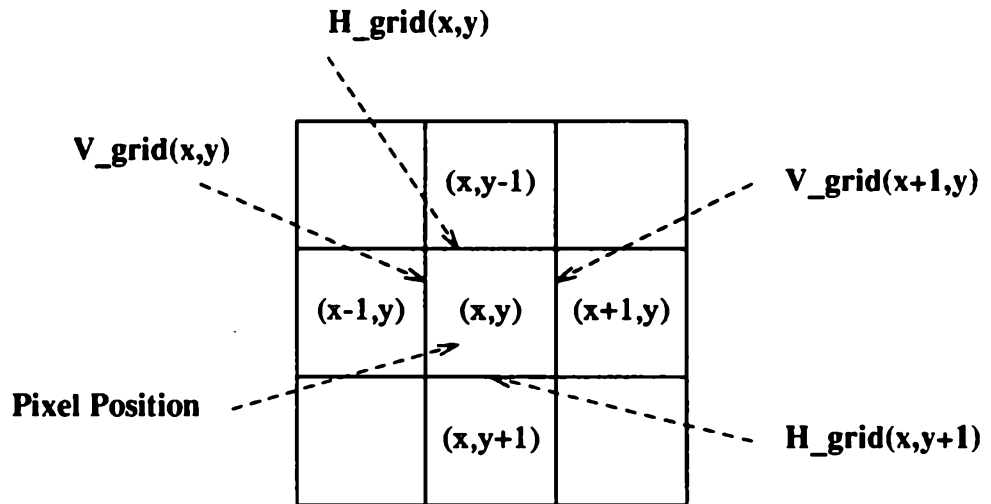


Figure B.1. — *Definition of grid lines*

$\forall(x,y) \in T$ where T = the set of pixels that belong to thin lines

```
{
  Set V_grid(x,y)    /* left grid */
  Set V_grid(x+1,y) /* right grid */
  Set H_grid(x,y)    /* top grid */
  Set H_grid(x,y+1) /* bottom grid */
}
```

$\forall(x,y) \in T$

Subroutine: Grid_Adjustment(x,y)

```

{
  if (x+1,y) ∈ T
    Clear V_grid(x+1,y) /* open right grid */
  else if (x,y+1) ∈ T
    Clear H_grid(x,y+1) /* open bottom grid */
  else if (x-1,y) ∈ T
    Clear V_grid(x,y) /* open left grid */
  else if (x,y-1) ∈ T
    Clear H_grid(x,y) /* open top grid */
  else /* the pixel (x,y) is at the intersection of two lines */
    {
      if (x>0) /* if there is left neighbour */
        Clear V_grid(x,y) /* open left grid */
      else if (y>0) /* if there is top neighbour */
        Clear H_grid(x,y) /* open top grid */
    }
}

```

Figure B.2 illustrates the grid boundaries found by the algorithm on a given thinned line image. The thinned lines are shown in the shaded pixels in Figure B.2(a).

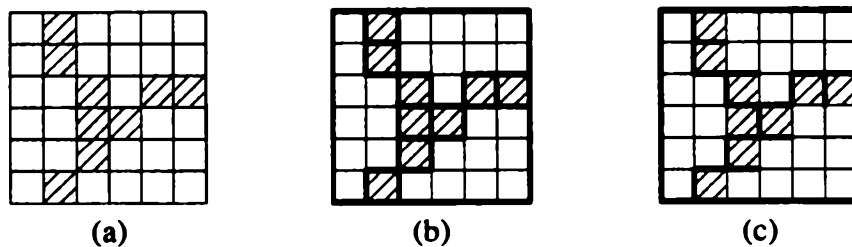


Figure B.2. — Formation of grid boundary

Appendix C. Boundary Tracing Algorithm

In Chapter 4, when we refer to the two sides of the boundary line created by the boundary completion technique, we are referring to the pixels that are horizontally and vertically adjacent to the pixels of the boundary (Figure C.1). The reason for this is that the region delimiter in our model uses the grid line representation. So, only the horizontally or vertically adjacent pixels can be considered as the sides of a boundary segment.

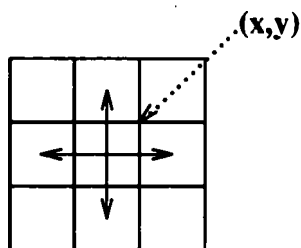


Figure C.1. — *Neighbourhood of a boundary line*

We will now describe a tracing algorithm that can find all the traces that pass through the neighbourhoods of the boundary line. Any pixel lying on the trace can be used as the seed point because the trace lies entirely inside the region.

$\forall (x,y) \in C$, where C = set of pixels of the new boundary line
 {
 if (x,y) not traced, $Trace(x,y)$
 if $(x,y-1)$ not traced and $L(x,y-1)=L(x,y)$, $Trace(x,y-1)$
 if $(x-1,y)$ not traced and $L(x-1,y)=L(x,y)$, $Trace(x-1,y)$
 if $(x,y+1)$ not traced and $L(x,y+1)=L(x,y)$, $Trace(x,y+1)$
 if $(x+1,y)$ not traced and $L(x+1,y)=L(x,y)$, $Trace(x+1,y)$
 }
 where $L(x,y)$ is the label of the pixel (x,y) .

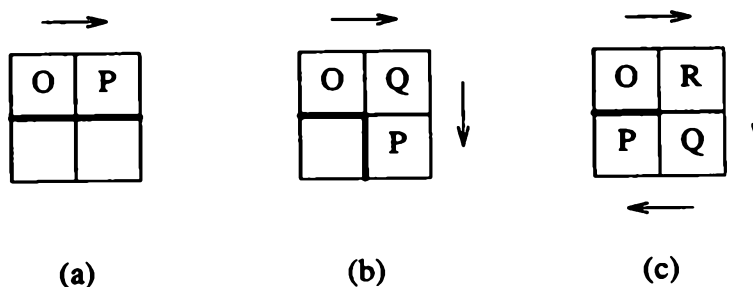


Figure C.2. — Decision path for tracing


```

Subroutine: Trace(x,y)
{
  Record (x,y) as the seed point
  mark (x,y) traced
  Initialise the direction for tracing
  Do
  {
    if the grid line in the trace direction is ON
      change direction to its RIGHT
    else
      if case 1 (Figure C.2(a))
        mark P traced
        O ← pixel P
      else if case 2 (Figure C.2(b))
        mark P and Q traced
        change direction to its RIGHT
        O ← pixel P
      else if case 3 (Figure C.2(c))
        mark P, Q, R traced
        reverse direction
        O ← pixel P
  }
  Until O = (x,y)

```

After the trace routine, the seed point recorded for each trace can be used to initiate the relabelling algorithm for the region. During this process, an area count for each region must be updated so that the labels having a zero area count can be deleted from the region labelling set. This is to ensure that a null region does not exist in the image.

References

- [AHO74] Aho, A.V., Hopcroft, J.E., Ullman, J.D., *The Design And Analysis Of Computer Algorithms*, Addison-Wesley Publishing Company, pp.124-139, 1974.
- [BOND76] Bondy J.A., Murty, U.S., *Graph Theory With Applications*, North Holland Publishing Company, pp.80-85, 1976.
- [BUXT81] Boxtton, W., Patel, S., Reeves, W., Baecker, R., "Scope in Interactive Score Editors," *Computer Music Journal*, 5(3), pp. 50-56, 1981.
- [CATM78] Catmull, E., "The Problems of Computer-Assisted Animation," *Computer Graphics*, vol. 12, No. 3, pp. 348-353, August 1978.
- [CHOW72] Chow, C.K., Kaneko, T., "Automatic Boundary Detection of the Left Ventricle from Cineangiograms," *Comp. and Biomed. Res.*, vol. 5, pp. 388-410, 1972.
- [DINI80] Dinic, E.A., "Algorithm for solution of a problem of maximum flow in a network with power setimation," *Soviet Math., Dokl.*, vol. 11, pp. 1277-1280, 1970.

- [FISC87] Fischetti, M.A., "The Silver Screen Blossoms into Colour," *IEEE SPECTRUM*, pp. 50-55, August 1987.
- [FISH84] Fishkin, K.P., Barsky, B.A., "A Family of New Algorithms for Soft Filling," *Computer Graphics*, vol. 18, No. 3, pp. 235-244, July, 1984.
- [FISH85] Fishkin, K.P., Barsky, B.A., "An Analysis and Algorithm For Filling Propagation," *Proceedings Graphics Interface '85*, pp. 203-212, May, 1985.
- [FLOY75] Floyd, R.W., Steinberg, L., "An adaptive algorithm for spatial gray scale," *SID 75, Int. Symp. Dig. Tech. Papers*, pp. 36, 1975.
- [FOLD73] Peter Foldes, *Hunger (La Faim)*, National Film Board of Canada, Montreal, PQ, Canada (1973). Film available from the NFB.
- [FOLE82] Foley, J.D., Van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, pp.433-436, 1982.
- [GONZ87] Gonzalez, R.C., Wintz, P., *Digital Image Processing*, Addison-Wesley Publication Co., 2nd edition, 1987.
- [GROS87] Grossberg, S. and Mingolla, E., "Neural dynamics of form perception: Boundary completion, illusory figures, and neon colour spreading," *Advances in Psychology II*, vol. 43, pp.82-142, 1987.
- [GUO89] Guo, Z., and Hall R.W., "Parallel Thinning with Two-Subiteration Algorithms," *Communications of the ACM*, vol. 32, No. 3, pp.359-373, 1989.

- [HALL89] Hall R.W., "Fast Parallel Thinning Algorithms: Parallel Speed and Connectivity Preservation," *Communications of the ACM*, vol. 32, No. 1, pp.124-131, 1989.
- [HARD87] Hardtke, Ines, *Kinetics For Key Frame Interpolation*, Master's Thesis, Dept. of Comp. Sci., University of Waterloo, 1987.
- [HIGG86] Higgins, T., *Painting A Cel -- Digital Painting in a Virtual RGBA Frame Buffer*, Master's Thesis, Dept. of Comp. Sci., University of Waterloo, 1986.
- [KOCH82] Kochanek, D., Bartels, R.H., and Booth, K.S., *A Computer System for Smooth Keyframe Animation*, Dept. of Comp. Sci., Technical Report CS-82-42, University of Waterloo, 1982.
- [KRIS88a] Krishnan, G., Walters, D., "Segmenting Intersecting and Incomplete Boundaries," *Proceedings of SPIE's Applications of AI conference*, April, 1988.
- [KRIS88b] Krishnan, G., *Machine Segmentation and Recognition of Line Drawings*, PhD Dissertation, State University of New York at Buffalo, 1988.
- [LAYB79] Laybourne, K., *The Animation Book*, Crown Publishers Inc., New York., 1979.
- [LEVO82] Levoy, M., "Frame Buffer Configurations for Paint Programs," *tutorial notes for course 9 of the SIGGRAPH '82 conference*, pp. 19-30, 1982.

- [MATU72] Matula, D.W., Marble G., and Isaacson J.D., "Graph Colouring Algorithms," *Graph Theory & Computing*, Academic Press, pp. 109-122, 1972.
- [NEEL88] Neely, S.R., *The Fill Interpreter: A Unified View of Brushing, Filling, and Compositing*, Master's Thesis, Dept. of Comp. Sci., University of Waterloo, 1988.
- [NFB87] National Film Board of Canada, *Amuse-Guelule*, Animation video 113C-0264-059, 1987.
- [PAET85] Paeth, A.W., *The IM Toolkit: A Comprehensive Raster Manipulation Package Presented through Design and Example*, Master's Essay, Dept. of Comp. Sci., University of Waterloo, 1985.
- [PORT84] Porter, T., Duff, T., "Compositing Digital Images," *Computer Graphics*, Vol. 18, No. 3, pp. 253-259, July, 1984.
- [SMIT79] Smith, A.R., "Tint Fill," *SIGGRAPH Proceeding*, pp. 276-283, 1979.
- [STER79] Stern, G., "SoftCel - An Application of Raster Scan Graphics to Conventional Cel Animation," *SIGGRAPH Proceeding*, pp. 284-288, 1979.
- [TANN83] Tanner, P., Cowan, W., Wein, M., "Colour Selection, Swath Brushes and Memory Architectures for Paint Systems," *Proceedings of Graphics Interface '83*, Edmonton, pp. 171-180, May 9-13, 1983.

- [TARJ83] Tarjan, R.E., *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, 1983.
- [ULLM76] Ullman, S., "Filling-in the Gaps: The Shape of Subjective Contours and a Model for Their Generation," *Biol. Cybernetics* 25, pp. 1-6, 1976.
- [WALT87a] Walters, D., "Selection of Image Primitives for General-Purpose Visual Processing," *Computer Vision, Graphics, and Image Processing*, 37(3), pp. 261-289, 1987.
- [WALT87b] Walters, D., "Automated Fake Colour Separation: Combining Computer Vision and Computer Graphics," *Proceedings of SPIE's Applications of AI*, vol. 786, pp.541-548, 1987.
- [WHIT83] White, J.M. and Rohrer, G.D., "Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction," *IBM J. Res. Devel.*, vol. 27, no. 4, pp. 400-411, 1983.
- [YAO82] Yao, F., "Maximum Flows In Networks," *Proceedings of Symposia in Applied Mathematics*, vol. 26, pp. 31-43, 1982.
- [ZHAN84] Zhang, T.Y. and Suen, C.Y., "A Fast Parallel Algorithm for Thinning Digital Patterns," *Comm. ACM*, vol. 27, no. 3, pp. 236-239, 1984.