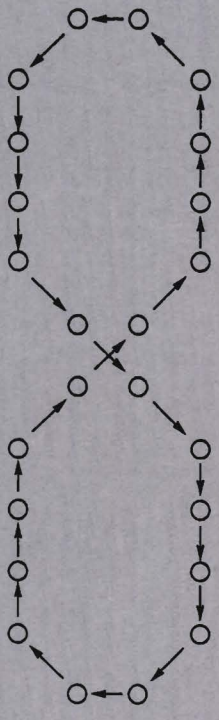
Chain coding in computer graphics

Chain Coding in Computer Graphics



Huub van de Wetering

Chain Coding in Computer Graphics

Chain Coding in Computer Graphics

Proefschrift

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof. dr. J.H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op vrijdag 22 november 1991 om 16.00 uur

door

Hubertus Martinus Maria van de Wetering

geboren te Olland (NBr)

druk: wibro dissertatiedrukkerij, helmond.

Dit proefschrift is goedgekeurd
door de promotoren
prof. dr. dipl. ing. D.K. Hammer
en
prof. dr. F.J. Peters.

copromotor :
dr. ir. C.W.A.M. van Overveld.

Aan mijn ouders.

These investigations were partly supported by
the Netherlands Technology Foundation (STW).

Contents

0

Introduction

0.0 Motivation

A part of computer graphics is concerned with displaying geometrical objects on output devices, such as monitors and printers; this is referred to as *rendering*. The mathematical description of such objects is called *modelling*. In this thesis the objects of interest are curve segments. Curves are commonly-used objects in both computer aided design (CAD) and desk top publishing (DTP), and much literature on curves exists, handling both the modelling as well as the rendering.

In this thesis the problems we tackle for curves are modelling and rendering. Unlike most other approaches that define curves in continuous space, we choose discrete curves as a starting point. Discrete curves are used, e.g. in pattern recognition, where they occur naturally, because of the discrete initial phase of most problems in that field. From pattern recognition we also obtain the notion of chain coding [Fre74], the way in which discrete curves are described in this thesis. One reason for using discrete curves for image synthesis is that this approach dismisses the need for conversion from continuous space to discrete space. Another reason is that discrete curves automatically bring along the benefits of integer arithmetic over floating point arithmetic. These benefits are stated in the sequel.
Below a short introduction to continuous curves and discrete curves is given. Both introductions are divided in a mathematical description (modelling) part and a rendering part. As in the rest of this thesis only continuous curves in $\mathbb{R}^2$ and discrete curves in $\mathbb{Z}^2$ are considered. Nevertheless, the vast majority of the definitions and properties is straightforwardly generalised to $\mathbb{R}^n$ for arbitrary n.

Mathematical Descriptions for Continuous Curves. A continuous curve may be given by a continuous one-parameter function. In case of polynomial curves of degree n this parameter function takes the following form.

$$\mathbf{p}(u) = \mathbf{a}_0 + u\mathbf{a}_1 + \cdots + u^n\mathbf{a}_n \qquad (u \in [0, 1])$$

with $\mathbf{a}_i \in \mathbb{R}^2$. This form of writing a parameter function, called the algebraic form, does not give an intuitive feel for the shape of the curve: there is no obvious relation between the coefficient-vectors $\mathbf{a}_i$ and the shape of the curve. This can be improved by using the, so called, geometric form:

$$\mathbf{p}(u)=F_0(u)\mathbf{p}_0+F_1(u)\mathbf{p}_1+\cdots+F_n(u)\mathbf{p}_n \qquad (u\in[0,1])$$

with $\mathbf{p}_i\in\mathbb{R}^2$ and $F_i\in[0,1]\rightarrow\mathbb{R}$. This form is called a geometric form since, for well-chosen functions F_i, a geometric interpretation may be given to the points $\mathbf{p}_i$. These points then are called control points and the functions F_i are called blending functions.

Below we show two examples for $n=3$.

1) The points $\mathbf{p}_0$ and $\mathbf{p}_1$ are the end points of the curve, and $\mathbf{p}_2$ and $\mathbf{p}_3$ are the derivative vectors of the curve in the end points. This results, in case of polynomial curves, in the cubic Hermite curves.

2) The points $\mathbf{p}_0$ and $\mathbf{p}_3$ are the end points of the curve and $\mathbf{p}_1-\mathbf{p}_0$ and $\mathbf{p}_3-\mathbf{p}_2$ are tangent to the curve. This is the case of the cubic Bezier curves.

The control points $\mathbf{p}_0$ up to $\mathbf{p}_n$ form the characteristic polygon of a curve. A very useful property for modeling a curve, is that the shape of the curve is independent of the orientation, size, and position of the characteristic polygon; this property is fulfilled if a curve is affine invariant. A curve is affine invariant if the blending functions sum to 1, for all $u\in[0,1]$.

Rendering of Continuous Curves. Nowadays this is practically equivalent to rasterisation of curves; that is, computing a pixel set representing a curve on a raster. The existent algorithms are only suitable for curves of a special form. Famous examples are: Bresenham's straight line algorithms [Bre65] , the midpoint circle algorithm [Fol90] , and De Casteljau's algorithm [Boe84] for rendering Bezier curves.

Mathematical Descriptions for Discrete Curves. The notion of discrete curve is the analogy in $\mathbb{Z}^2$ of the notion of continuous curve in $\mathbb{R}^2$. A discrete curve may be described by a starting point and a string of relative vectors as shown in figure 0.1(a).

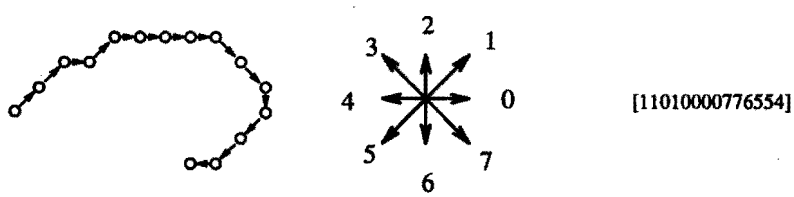


Figure 0.1:
(a) a discrete curve
(b) encoding of the vectors
(c) the corresponding chain

Such a string may be represented by a string of numbers, each of which denotes one of the relative vectors (see figure 0.1(a) and (b)); this string of numbers is called a *chain* and the numbers are called *codes*. Hence, a chain and a starting point form a

mathematical description of a curve. From the modeling point of view, it is, of course, not a handy one. For discrete curves, a more compact description and one more suitable for modeling purposes, may be obtained by using control points (in $\mathbb{Z}^2$). In this thesis we define a large class of discrete curves, which may be specified by control points. These curves, called w-curves, are defined in discrete space and are, hence, not directly related to curves in continuous space. If a w-curve is given by control points $\mathbf{p}_0$ up to $\mathbf{p}_n$, its definition uses the chains, called $bresh(\mathbf{p}_i - \mathbf{p}_{i-1})$ associated with the line segments $\mathbf{p}_i\mathbf{p}_{i-1}$, for $0 < i \leq n$, together with a so-called weaving operator $\underline{w}$ for combining these n chains into one chain. In chapters 3 and 4 several useful definitions for such an operator $\underline{w}$ are given. For all these definitions the operator $\underline{w}$ mixes the codes of the chains $bresh(\mathbf{p}_i - \mathbf{p}_{i-1})$ $(0 < i \leq n)$ in a well-chosen order.

The notion of affine invariance cannot be used in discrete space; e.g, rotations of discrete curves can only be done correctly for angles that are a multiple of 90 degrees. It is shown, however, that w-curves can be related to continuous curves that are invariant under affine transformations.

Rendering of Discrete Curves. The rasterisation of discrete curves is relatively easy. If a discrete curve is given by a chain $\mathbf{c}$ and a starting point $\mathbf{p}$, the pixels may be found by starting at $\mathbf{p}$ and adding the vectors belonging to the elements of the chain one after another. If the curve is a w-curve, the corresponding chain and starting point can easily be computed, as we will see in chapter 3.

Continuous versus discrete curves. We already indicated that curves are commonly used in both CAD and DTP. The use of curves in CAD, if used in combination with computer aided manufacturing (CAM), is restricted to continuous curves. This is due to the nature of the required result: a model in $\mathbb{R}^2$ or $\mathbb{R}^3$. However, the discrete curves, as defined in this thesis, can be related to continuous curves, as will be shown in chapter 3. For DTP the required result is an image on a raster device. In this case the advantages of discrete curves as stated below may be used to their full extent.

- discrete curves have relatively simple rendering algorithms.
- The algorithms for discrete curves use only *integer arithmetic*, which is not only faster than floating point arithmetic but can also be realised easier in hardware. Furthermore integer arithmetic allows for exact operations; hence, there is no need for elaborate numerical analysis on the robustness of the algorithms.
- Since all discrete curves can be represented by a chain and a point, a unified algorithmic approach is possible. Instead of having several algorithms for, e.g., filling a (closed) discrete curve or computing an associated thick curve, one algorithm that is based on the representation by chains and points, suffices. Among these algorithms are also algorithms for linear transformations of discrete curves.

0.1 Research position

The work presented in this thesis is inspired by the thesis "Digitisation functions in Computer Graphics" of Marloes van Lierop [Lie87] . In her thesis van Lierop gives a sound theoretical basis for digitisation in general and for digitisation of straight line segments in particular. The latter was the motive to think about general curve digitisation. Furthermore, the emphasis in van Lierop's work on properties of geometric objects in discrete space resulted in this study of *discrete* curves.

For the representation of a general discrete curve the notion of chain coding, has been introduced by Freeman in 1961 [Fre61] . This notion is still heavily used in the realm of image processing. In computer graphics (image generation) chain coding, as is indicated in this thesis, also turns out to be a useful representation method.

Discrete curve generation algorithms have been a popular subject for investigation ever since the introduction of raster devices. In [Fol90] some well-known examples of these algorithms are referenced. All these algorithms render a given continuous curve. The algorithms given in this thesis, however, use chains, i.e. representations of discrete curves, either to render or to generate other chains.

Finally, curve modelling is mostly done in continuous space. Summaries of the results in this area may be found in for instance [Boe84] or [Mor85] . In contrast the modelling of the discrete curves defined in this thesis is based on a discrete approach by the so-called distribution functions.

Concluding we can state that although the subjects in this thesis are well-known from other investigations, the approach chosen here is totally different from relevant other work.

0.2 Overview

The remainder of this thesis consists of 7 chapters. In chapter 1 the basic definitions are given; the notions of discrete curves and chain coding are introduced here. In chapter 2 not only Bresenham's line algorithms but also algorithms for linear transformations of discrete curves given by chain codes are derived. Furthermore, properties with respect to linear transformations of discrete curves are given. Chapter 3 contains the main part of this thesis. In it operators on chains are defined; these operators are combined into a definition for discrete curves given by three control points. These discrete curves are related to continuous curves by computing them for different resolutions and having this resolution to go to infinity. Chapter 3 concludes with some algorithms for computing the so defined discrete curves. In chapter 4 the themes of chapter 3 are extended for curves with more than 3 control points. For this situation two new operators are introduced. The notion of a canonic chain is introduced. Chapter 5 is a prelude for chapter 6; an algorithm is given for computing

the set of interior points of a given closed discrete curve. In chapter 6 this algorithm is used for computing thick versions of discrete curves. Offset curves are also discussed in this chapter. Finally, some concluding remarks are made in chapter 7.

0.3 Notations

SETS:

$\mathbb{N}_0$ the set of non-negative integer numbers
$\mathbb{N}_1$ the set of positive non-zero integer numbers
$\mathbb{Z}$ the set of integer numbers
$\mathbb{R}$ the set of reals
$[a..b] = \{i \in \mathbb{Z} \mid a \le i \le b\}$
$[a..b) = \{i \in \mathbb{Z} \mid a \le i < b\}$
$(a..b) = \{i \in \mathbb{Z} \mid a < i < b\}$
$(a..b] = \{i \in \mathbb{Z} \mid a < i \le b\}$
$[x, y] = \{r \in \mathbb{R} \mid x \le r \le y\}$
$[x, y) = \{r \in \mathbb{R} \mid x \le r < y\}$
$(x, y) = \{r \in \mathbb{R} \mid x < r < y\}$
$(x, y] = \{r \in \mathbb{R} \mid x < r \le y\}$

$|S|$ denotes the number of elements of the set S.
$A \rightarrow B$ denotes the set of functions with domain A and reach B.

PREDICATE NOTATION:

$(\forall i : R(i) : P(i))$ universal quantification
$(\exists i : R(i) : P(i))$ existential quantification

P_E^x the predicate resulting from substituting in the predicate P the expression E for x.

ARITHMETIC OPERATIONS:

$(\Sigma i : P(i) : f(i))$, $max\{f(i) \mid P(i)\}$, $min\{f(i) \mid P(i)\}$
the sum, the maximum, and minimum, respectively, of $f(i)$ for all i satisfying the predicate $P(i)$.
$(\mathbf{N} i : P(i))$ the number of i's satisfying $P(i)$.
For $x \in \mathbb{R}$:
$/x/$ absolute value
$[x]$ rounding
$\lfloor x \rfloor$ the floor function.
$\lceil x \rceil$ the ceiling function.
The operators *div* and *mod* are defined such that for $i \in \mathbb{Z}$ and $n \in \mathbb{N}_1$.
$i = n(i \text{ div } n) + i \text{ mod } n$ with $i \text{ mod } n \in [0..n)$.

VECTORS:

For $\mathbf{p} \in \mathbb{R}^2$: $\mathbf{p}=(\mathbf{p}_x, \mathbf{p}_y)$.

$\|\mathbf{p}\|$ Euclidean length of vector $\mathbf{p}$.

$/\mathbf{p}/=(/\mathbf{p}_x/, /\mathbf{p}_y/)$

CHAINS:

ε	the empty chain
$\mathbf{c}_i$ or $\mathbf{c}(i)$	the ith code of chain $\mathbf{c}$.
[823]	a chain with the codes 8, 2, and 3.
$\mathbf{c}^n$	concatenation of n-times chain $\mathbf{c}$.
$\mathbf{p}(\mathbf{c}, i)$	the ith point on the chain $\mathbf{c}$.
$\#_c(\mathbf{c}, i, j)$	the number of times the code c occurs in the subsequence $[\mathbf{c}_i \cdots \mathbf{c}_{j-1}]$ of the chain $\mathbf{c}$.
$\|\mathbf{c}\|$	the length of the chain $\mathbf{c}$ (= the number of codes)
$\mathbf{c} \otimes \mathbf{d}$	the concatenation of two chains $\mathbf{c}$ and $\mathbf{d}$
$(\prod i : R(i) : \mathbf{c}_i)$	the continued concatenation of the chains $\mathbf{c}_i$ satisfying $R(i)$ for increasing i.

1

Basic Definitions

1.0 Introduction

In this chapter we give definitions of the notions that are used in the subsequent chapters. The most important definitions are those of discrete curves and chains.

1.1 Continuous Curves

We usually use reals to describe the objects in the real world ($\mathbb{R}^2$ or $\mathbb{R}^3$) that we want to convert to discrete objects on a raster device. One of these objects is a curve segment in $\mathbb{R}^2$, which is defined in definition 1.1. A curve segment is henceforth just called a curve. Note that only parameterised curves are considered.

1.1 **Definition** : continuous curve

A continuous curve C is a subset of $\mathbb{R}^2$ that can be described by two continuous one-parameter functions x and $y \in [0, 1] \rightarrow \mathbb{R}$ as follows

$$C = \{ (x(u), y(u)) \mid u \in [0, 1] \}.$$

□

In case of polynomial curves of degree n the parameter functions x and y in the definition of a continuous curve, take the following form.

$$(x(u), y(u)) = \mathbf{a}_0 + u\mathbf{a}_1 + \cdots + u^n\mathbf{a}_n \qquad (u \in [0, 1])$$

with $\mathbf{a}_i \in \mathbb{R}^2$. This form of writing a parameter function, called the algebraic form, does not give an intuitive feel for the shape of the curve: there is no obvious relation between the coefficient-vectors $\mathbf{a}_i$ and the shape of the curve. This can be improved by using the, so called, geometric form:

$$(x(u), y(u)) = F_0(u)\mathbf{p}_0 + F_1(u)\mathbf{p}_1 + \cdots + F_n(u)\mathbf{p}_n \qquad (u \in [0, 1])$$

with $\mathbf{p}_i \in \mathbb{R}^2$ and $F_i \in [0, 1] \rightarrow \mathbb{R}$. This form is called a geometric form since, for well-chosen functions F_i, a geometric interpretation may be given to the vectors $\mathbf{p}_i$.

This may be done in several ways; below we show some examples for $n=3$.

1) The points $\mathbf{p}_0$ and $\mathbf{p}_1$ are the end points of the curve, and $\mathbf{p}_2$ and $\mathbf{p}_3$ are the derivative vectors of the curve in the end points. This results, in case of polynomial curves, in the cubic Hermite curves.

2) The points $\mathbf{p}_0$ and $\mathbf{p}_3$ are the end points of the curve and $\mathbf{p}_1-\mathbf{p}_0$ and $\mathbf{p}_3-\mathbf{p}_2$ are tangent to the curve. This is the case of the cubic Bezier curves.

The points $\mathbf{p}_i$ are called control points and the functions F_i are called blending functions.
The control points $\mathbf{p}_0$ up to $\mathbf{p}_n$ form the characteristic polygon of a curve. A curve has a very useful property for modeling if the shape of the curve is independent of the orientation, size, and position of the characteristic polygon; this property is fulfilled if a curve is affine invariant. An *affine transformation* A is a function in $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ that satisfies

$$(\forall x : x \in \mathbb{R}^2 : Ax = t + Lx),$$

for some $t \in \mathbb{R}^2$ and some linear transformation L.

1.2 **Definition :** affine invariant

Let $C(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n)$ be continuous curves, for all $\mathbf{p}_i \in \mathbb{R}^2$.
$C(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n)$ is affine invariant iff for all affine transformations A,

$$A(C(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n)) = C(A(\mathbf{p}_0), A(\mathbf{p}_1), \cdots, A(\mathbf{p}_n)).$$

□

The following two properties state relationships between affine invariancy and the blending functions of the form in which a curve is given.

1.3 **Property :**

Let $\mathbf{p}_i \in \mathbb{R}^2$ and blending functions $F_i \in [0, 1] \rightarrow \mathbb{R}$. The continuous curves given by

$$(x(u), y(u)) = F_0(u)\mathbf{p}_0 + F_1(u)\mathbf{p}_1 + \cdots + F_n(u)\mathbf{p}_n,$$

for all $u \in [0, 1]$, are affine invariant iff for all $u \in [0, 1]$,

$$(\Sigma i : i \in [0..n] : F_i(u)) = 1.$$

□

Apart from the above mentioned geometric form we also use the following form

$$(x(u), y(u)) = \mathbf{p}_0 + F_0(u)(\mathbf{p}_1 - \mathbf{p}_0) + F_1(u)(\mathbf{p}_2 - \mathbf{p}_1) + \cdots + F_{n-1}(u)(\mathbf{p}_n - \mathbf{p}_{n-1}).$$

Using property 1.3 we can now prove property 1.4.

1.4 **Property :**

Continuous curves given, for $u \in [0, 1]$ by

$$(x(u), y(u)) = \mathbf{p}_0 + F_0(u)(\mathbf{p}_1 - \mathbf{p}_0) + F_1(u)(\mathbf{p}_2 - \mathbf{p}_1) + \cdots + F_{n-1}(u)(\mathbf{p}_n - \mathbf{p}_{n-1})$$

with $\mathbf{p}_i \in \mathbb{R}^2$ and $F_i \in [0, 1] \rightarrow \mathbb{R}$, are affine invariant.

□

1.2 Discrete Curves

Before defining the notion of discrete curves we need some introductory notions.

Elements of $\mathbb{Z}^2$ are called *points*, and the coordinates of a point $\mathbf{p}$ are denoted by $\mathbf{p}_x$ and $\mathbf{p}_y$. Multiplications of points with an integer and real factor and addition of two points, are defined as usual in the vector spaces $\mathbb{R}^2$ or $\mathbb{Z}^2$. In $\mathbb{Z}^2$ we define two distance functions: D_4 and D_8. These functions are used, among other things, for the definition of neighbourhoodship of pixels; D_4 and D_8 result in every pixel having 4 and 8 pixels, respectively, at distance 1. In the next definitions these functions are defined on $\mathbb{R}^2$ enabling one to make a more flexible use of them.

1.5 **Definition :** D_4, D_8

For all $\mathbf{p}$ and $\mathbf{q}$ in $\mathbb{R}^2$ the two distance functions D_4 and D_8 (both in $\mathbb{R}^2 x\, \mathbb{R}^2 \rightarrow \mathbb{R}$) are defined by

$$D_4(\mathbf{p}, \mathbf{q}) := |\mathbf{p}_x - \mathbf{q}_x| + |\mathbf{p}_y - \mathbf{q}_y|$$
$$D_8(\mathbf{p}, \mathbf{q}) := max\{|\mathbf{p}_x - \mathbf{q}_x|, |\mathbf{p}_y - \mathbf{q}_y|\}.$$

□

In the remaining of this paragraph we use $\delta \in \{4, 8\}$ to make our definitions and properties valid for both the D_4 and the D_8 distance function. The properties 1.6(a) up to (d) show that D_δ is indeed a distance function.

1.6 **Property :**

For all $\mathbf{p}, \mathbf{q}, \mathbf{r} \in \mathbb{R}^2$ and all $k \in \mathbb{R}$,

(a) $D_\delta(\mathbf{p}, \mathbf{q}) \geq 0$
(b) $D_\delta(\mathbf{p}, \mathbf{q}) = 0$ iff $\mathbf{p} = \mathbf{q}$
(c) $D_\delta(\mathbf{p}, \mathbf{q}) = D_\delta(\mathbf{q}, \mathbf{p})$
(d) $D_\delta(\mathbf{p}, \mathbf{r}) \leq D_\delta(\mathbf{p}, \mathbf{q}) + D_\delta(\mathbf{q}, \mathbf{r})$
(e) $D_\delta(\mathbf{p}+\mathbf{q}, \mathbf{r}+\mathbf{s}) \leq D_\delta(\mathbf{p}, \mathbf{r}) + D_\delta(\mathbf{q}, \mathbf{s})$
(f) $D_\delta(k\mathbf{p}, (0, 0)) = kD_\delta(\mathbf{p}, (0, 0))$
(g) $D_\delta(\mathbf{p}-\mathbf{q}, \mathbf{r}) = D_\delta(\mathbf{p}, \mathbf{r}+\mathbf{q})$

□

1.7 **Definition :** δ-neighbour

Two points **p** and **q** are called δ-neighbours iff $D_\delta(\mathbf{p}, \mathbf{q})=1$.

□

Figure 1.1 shows the δ-neighbours of a point.

Figure 1.1 :
(a) the 4-neighbours of ●
(b) the 8-neighbours of ●

1.8 **Definition :** δ-path

A sequence $\pi=<\mathbf{p}_0, \cdots, \mathbf{p}_{n-1}>$ of points in $\mathbb{Z}^2$ is called a δ-path iff

$$(\forall i : i \in (0..n) : \mathbf{p}_i \text{ and } \mathbf{p}_{i-1} \text{ are δ-neighbours})$$

A δ-path $\pi=<\mathbf{p}_0, \cdots, \mathbf{p}_{n-1}>$ is said to start at $\mathbf{p}_0$ and end at $\mathbf{p}_{n-1}$.

□

1.9 **Definition :** $P(\pi)$

The point set $P(\pi)$ of a δ-path $\pi=<\mathbf{p}_0, \cdots, \mathbf{p}_{n-1}>$ is given by

$$P(\pi) := \{ \mathbf{p}_i \mid i \in [0..n) \}$$

□

1.10 **Definition :** δ-connected set

A set $P \subset \mathbb{Z}^2$ is called connected iff

$$(\forall \mathbf{p}, \mathbf{q} : \mathbf{p}, \mathbf{q} \in P : (\exists \pi : \pi \text{ is a δ-path } : P(\pi) \subseteq P \text{ and } \pi \text{ starts at } \mathbf{p} \text{ and ends at } \mathbf{q}))$$

□

1.11 **Definition :** discrete curve

A discrete curve D is a connected subset of $\mathbb{Z}^2$ that can be described by two one-parameter functions x and $y \in [0..n] \rightarrow \mathbb{Z}$ as follows

$$D=\{(x(i),y(i)) \mid i \in [0..n]\},$$

for some $n \in \mathbb{Z}^+$.
A δ-connected discrete curve is called a δ-curve.

□

This definition has been formulated in accordance with definition 1.1 of continuous curves. The notion continuity in "continuous one-parameter function", however, has been replaced by connectivity in "connected subset". In fact the parameter functions in the definition of discrete curves are not obliged, as may be seen from the following property.

1.12 **Property :**

For all sets $D \subseteq \mathbb{Z}^2$ the following holds.

D is a discrete curve
$\Leftrightarrow$
D is a finite connected subset of $\mathbb{Z}^2$.

Proof

$\Rightarrow$:
This follows directly from the definition of discrete curve.

$\Leftarrow$:
Let D be a finite connected subset of $\mathbb{Z}^2$.
Since D is a finite connected set, its points may be gathered in a path $\pi=<\mathbf{p}_0, \cdots, \mathbf{p}_n>$ ($n \in \mathbb{Z}$), such that $P(\pi)=D$. (Note that in such a path some points may occur more than once. Hence, n may be larger than $|D|$.) The parameter functions x and y for D may then be given by

$x,y: [0..n] \rightarrow \mathbb{Z}$ with
$x(i)=\mathbf{p}_{ix}$ and
$y(i)=\mathbf{p}_{iy}$.

Hence, $D=\{(x(i),y(i)) \mid i \in [0..n]\}$ is a discrete curve.

□

In the conversion of a continuous curve to a discrete one, rounding of reals to integers is a frequently occurring action. Below we give the definitions and notations for rounding and we state some properties.

1.13 **Definition :** $\lfloor x \rfloor$, $\lceil x \rceil$, $[x]$

For all $x \in \mathbb{R}$ we define,

(a) $\lfloor x \rfloor := max\{ i \in \mathbb{Z} \mid i \leq x \}$
(b) $\lceil x \rceil := min\{ i \in \mathbb{Z} \mid i \geq x \}$
(c) $[x] := \lceil x - \frac{1}{2} \rceil$

□

1.14 **Property :**

For all $x \in \mathbb{R}$ and all $i \in \mathbb{N}_0$,

(a) $0 \leq x - \lfloor x \rfloor < 1$
(b) $-1 < x - \lceil x \rceil \leq 0$
(c) $-\frac{1}{2} < x - [x] \leq \frac{1}{2}$
(d) $[i+x] = i + [x]$
(e) $x + \frac{1}{2} \notin \mathbb{Z} \Rightarrow [i-x] = i - [x]$
(f) $x \notin \mathbb{Z} \Rightarrow \lceil x \rceil - \lfloor x \rfloor = 1$
(g) $\lceil -x \rceil = -\lfloor x \rfloor$

□

1.3 Chain coding

In the previous paragraph we stated that a discrete curve can be represented by a path. A path is an expensive representation of a discrete curve since it stores all the coordinates of all the pixels in the curve. Two successive points in a path are neighbours and hence storing the absolute coordinates of both is redundant; knowing the first point and the relative position of every other point with respect to its predecessor is enough to yield the corresponding discrete curve. The number of δ-neighbours of a point is δ and hence at most eight. These possible 8 neighbours are encoded with the numbers 0 up to 7 according to figure 1.2(a). Hence, a discrete curve can be represented by a start point **p** and a chain (a sequence) of codes (the numbers 0 up to 7). This coding convention for discrete curves was introduced by Freeman [Fre61] in 1961, and he elaborated it in [Fre69] . Here we introduce a small extension to this convention: we not only encode each neighbour of a point **p** with a number but also the point **p** itself; it is encoded by the number 8 (see figure 1.2(b)). Below the formal definitions of chains and codes are given.

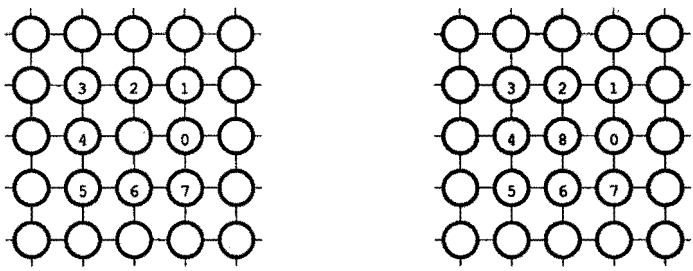


Figure 1.2: encoding of the basic directions in $\mathbb{Z}^2$

1.15 **Definition :** C_4, C_8

$$C_4 := \{0,2,4,6,8\}$$
$$C_8 := \{0,1,2,3,4,5,6,7,8\}$$

The elements of C_4 and C_8 are called codes. An element i of C_δ is called an i-code.

□

Every i-code is related to a displacement vector $\mathbf{v}_i$ according to the following definition.

1.16 **Definition :** basic vector

The basic vectors, denoted by $\mathbf{v}_i$ or $v(i)$, of the codes i in C_8 are given by

$$\mathbf{v}_3=(-1,1),\quad \mathbf{v}_2=(0,1),\quad \mathbf{v}_1=(1,1),$$
$$\mathbf{v}_4=(-1,0),\quad \mathbf{v}_8=(0,0),\quad \mathbf{v}_0=(1,0),$$
$$\mathbf{v}_5=(-1,-1),\quad \mathbf{v}_6=(0,-1),\quad \mathbf{v}_7=(1,-1).$$

□

1.17 **Definition :** δ-chain

A δ-chain is a finite sequence of elements of C_δ.

□

Below we define the length of a chain. Notice that this length is not the sum of the Euclidean lengths of the basic vectors of its codes, but merely the number of codes in the chain.

1.18 **Definition** : length of a chain

The length of a chain $\mathbf{c}$, denoted as $|\mathbf{c}|$, is the number of codes in the sequence of $\mathbf{c}$.

□

A chain is denoted by either a bold face roman letter, typically in the range c to e, and its i-th element is denoted by a subscript to such a letter, e.g. $\mathbf{c}_i$, or by $\mathbf{c}(i)$. A sequence of chains is also denoted by a subscript to this letter but in this case a bold font is used for the subscript, e.g $\mathbf{c}_\mathbf{i}$ is the i-th chain in a sequence of chains. The j-th element of $\mathbf{c}_\mathbf{i}$ is denoted by $\mathbf{c}_{\mathbf{i}j}$ or $\mathbf{c}_\mathbf{i}(j)$. Furthermore, square brackets are used as delimiters for a sequence of codes. Hence, a chain $\mathbf{c}$ can be notated by

$\mathbf{c}=[\,\mathbf{c}(0)\mathbf{c}(1)\cdots\mathbf{c}(|\mathbf{c}|-1)\,]$ or $\mathbf{c}=[\,\mathbf{c}_0\,\mathbf{c}_1\cdots\mathbf{c}_{|\mathbf{c}|-1}\,]$.

The empty chain, the chain with length 0, is denoted by ε.

1.19 **Definition** : concatenation of chains

The concatenation of two chains $\mathbf{c}$ and $\mathbf{d}$, denoted by $\mathbf{c}\otimes\mathbf{d}$, is defined by

$$(\mathbf{c}\otimes\mathbf{d})_i := \begin{cases} \mathbf{c}_i & i\in[\,0\,..\,|\mathbf{c}|\,) \\ \mathbf{d}_{i-|\mathbf{c}|} & i\in[\,|\mathbf{c}|\,..\,|\mathbf{c}|+|\mathbf{d}|\,) \end{cases}$$

The Product $(\Pi\, i : i\in[\,0\,..\,n\,) : \mathbf{c}_\mathbf{i})$ of a sequence of chains $(\mathbf{c}_\mathbf{i})$, denotes the continued concatenation $\mathbf{c}_0\otimes\mathbf{c}_1\otimes\cdots\otimes\mathbf{c}_{n-1}$.
The chain $\mathbf{c}^n$ is defined, for $n\in\mathbb{N}$, by

$$\mathbf{c}^n := (\Pi\, i : i\in[\,0\,..\,n\,) : \mathbf{c}).$$

□

1.20 **Definition** : $rev(\mathbf{c})$

For all chains $\mathbf{c}$ the reversed chain $rev(\mathbf{c})$ of $\mathbf{c}$ is defined by

$$rev(\mathbf{c}) := [\,\mathbf{c}_{|\mathbf{c}|-1}\,\mathbf{c}_{|\mathbf{c}|-2}\cdots\mathbf{c}_0\,]$$

□

The individual points referred to by a chain are given by the following definition.

1.21 **Definition** : $\mathbf{p}(\mathbf{c}, i)$

For all chains $\mathbf{c}$ and all $i\in[\,0\,..\,|\mathbf{c}|\,]$ the ith point on the chain $\mathbf{c}$ is defined by

$$\mathbf{p}(\mathbf{c}, i) := (\Sigma\, j : j\in[\,0\,..\,i\,) : v(\mathbf{c}_j)).$$

□

Notice that since the sum over an empty interval is zero, $\mathbf{p}(\mathbf{c}, 0) = (0, 0)$.
The total displacement vector *end*(**c**) of a chain **c** is defined as follows.

1.22 **Definition :** *end*(*c*)

For all chains **c** the end point *end*(**c**) is defined by

$$end(\mathbf{c}) := \mathbf{p}(\mathbf{c}, |\mathbf{c}|)$$

□

1.23 **Definition :** $P(\mathbf{c})$

The point set $P(\mathbf{c})$ of a chain **c** is defined by

$$P(\mathbf{c}) := \{\, \mathbf{p}(\mathbf{c}, i) \mid i \in [0 .. |\mathbf{c}|] \,\}$$

□

In the introduction of this section we introduced chains as a way to represent, together with a starting point, discrete curves. In the next definition we give this a concrete form by defining the notion $DC(\mathbf{p}, \mathbf{c})$, with **p** a point and **c** a chain, which is equally expressive as the notion of discrete curves.

1.24 **Definition :** $DC(\mathbf{p}, \mathbf{c})$

For all $\mathbf{p} \in \mathbb{Z}^2$ and all chains **c** the discrete curve $DC(\mathbf{p}, \mathbf{c})$ is defined by

$$DC(\mathbf{p}, \mathbf{c}) := \{\, \mathbf{p} + \mathbf{p}(\mathbf{c}, i) \mid i \in [0 .. |\mathbf{c}|] \,\}$$

□

1.25 **Definition :** $\#_c(\mathbf{c}, i, j)$

For all chains **c**, for all codes *c* and for all $i, j \in [0 .. |\mathbf{c}|]$ with $i \leq j$ $\#_c(\mathbf{c}, i, j)$ is defined by

$$\#_c(\mathbf{c}, i, j) := (\mathbf{N}\, k : k \in [i .. j) : \mathbf{c}_k = c)$$

□

As an example for the above definition we can write: $\#_c(\mathbf{c}, 0, |\mathbf{c}|)$ is the number of codes *c* in the chain **c**.

2

Linear Transformations of Discrete Curves

2.0 Introduction

In this section we present Bresenham's algorithms [Bre65] for approximating a line segment by a discrete curve; the line segment is given by two points. Based on these line algorithms algorithms for linearly transforming discrete curves are given. These two topics both come down to computing approximations of linear functions of one and two variables, respectively.
All algorithms use integer arithmetic only. Furthermore they consist of only additions, subtractions, and shifts.

2.1 Linear transformations of discrete curves

If a discrete curve D is the result of a rasterisation algorithm for the continuous curves $C(\mathbf{p}_0, \cdots, \mathbf{p}_{n-1})$ with $\mathbf{p}_i \in \mathbb{R}^2$, it may be transformed with a function A in $\mathbb{R}^2 \to \mathbb{R}^2$ in the following two ways.

(1) A is applied to all the points of D. The result $A(D)$ of this application is, hence, defined by

$$A(D) := \{ A(\mathbf{p}) \mid \mathbf{p} \in D \}$$

(2) A is applied to the control points $\mathbf{p}_i$ of the continuous curve. The transformed control points $A(\mathbf{p}_i)$ are used by the algorithm to compute the rasterisation of the curve $C(A(\mathbf{p}_0), \ldots, A(\mathbf{p}_{n-1}))$.

Here we consider affine transformations A. From definition 1.2 we see that affine invariant curves $C(A(\mathbf{p}_0), \ldots, A(\mathbf{p}_{n-1}))$ equal $A(C(\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}))$. Hence, for affine invariant curves the methods (1) and (2) lead to, possibly different, rasterisations of the same continuous curve.

In image processing it may well be so that a discrete curve is given, which cannot be related to continuous curves, let alone to curves invariant under affine transformations. In such a case the transformation of D must be done with the first method.

The main disadvantage of the first method is that the resulting set of points is not always a discrete curve. The conditions by which it is a discrete curve, are discussed in the sequel.

Let A be a linear function in $\mathbb{R}^2 \rightarrow \mathbb{R}^2$. Let D be a discrete curve. From the definition 1.11 of discrete curves we know that $D \subseteq \mathbb{Z}^2$ and that D is a connected set. $A(D)$, however, is not necessarily a subset of $\mathbb{Z}^2$. Hence, in general, $A(D)$ is not a discrete curve. If, however, for all $i \in C_8$, the discrete curve $\{(0,0), \mathbf{v}_i\}$ is mapped by the linear function A on a discrete curve, $A(D)$ is also a discrete curve. This is shown in the following property, both for 4- and 8-connected curves..

2.1 **Property :**

For all linear functions A the following holds.

$$(\forall D : D \text{ is a discrete } \delta\text{-curve} : A(D) \text{ is a discrete } \delta\text{-curve})$$
$$\Leftrightarrow$$
$$(\forall i : i \in C_8 : \{(0,0), A(v(i))\} \text{ is a discrete } \delta\text{-curve})$$

Proof

$\Rightarrow$:
$\{(0,0), v(i)\}$ is a discrete δ-curve and equals $\{(0,0), A(v(i))\}$ after applying A to it; hence, according to the hypothesis, the last set is also a discrete δ-curve.
$\Leftarrow$:
Let D be a discrete δ-curve.
We have to prove that $A(D)$ is a discrete δ-curve and hence, that $A(D)$ is a finite connected subset of $\mathbb{Z}^2$ (see property 1.12).

- $A(D) \subseteq \mathbb{Z}^2$.
  For all $\mathbf{p} \in D \subseteq \mathbb{Z}^2$
  $$A(\mathbf{p}) = \mathbf{p}_x A(v(0)) + \mathbf{p}_y A(v(2)) \in \mathbb{Z}^2,$$
  since A is a linear function, $v(0)=(1,0)$, and $v(1)=(0,1)$.
- $A(D)$ is a δ-connected set.
  For alll $\mathbf{p}, \mathbf{q} \in D \subseteq \mathbb{Z}^2$ the following holds.

  $D_8(\mathbf{p}, \mathbf{q}) \leq 1$
  $\Rightarrow$ { definition 1.16 of $v(i)$ }
  $(\exists i : i \in C_8 : v(i) = \mathbf{p} - \mathbf{q})$
  $\Rightarrow$ { A is a linear function }
  $(\exists i : i \in C_8 : A(v(i)) = A(\mathbf{p}) - A(\mathbf{q}))$
  $\Rightarrow$ { hypothesis: $A(v(i))$ and $(0,0)$ are δ-neighbours or equal }
  $D_8(A(\mathbf{p}) - A(\mathbf{q}), (0,0)) \leq 1$
  $\Rightarrow$ { property 1.6(g) }
  $D_8(A(\mathbf{p}), A(\mathbf{q})) \leq 1$

  Hence, if $\langle \mathbf{p}_0, \cdots, \mathbf{p}_{n-1} \rangle$ is a δ-path in D, $\langle A(\mathbf{p}_0), \cdots, A(\mathbf{p}_{(n-1)}) \rangle$ is a

δ-path in $A(D)$. Hence, D is δ-connected implies, that $A(D)$ is δ-connected.

□

This property can be applied to all linear functions A that map $v(i)$, for all $i \in C_\delta$, on an element of $\{v(i) \mid i \in C_\delta\}$. If a discrete curve D is given by a chain code $\mathbf{c}$ and a starting point $\mathbf{p}$ as $D=DC(\mathbf{p}, \mathbf{c})$, the linearly transformed curve $A(D)$ can be given by

$$A(D)=DC(A(\mathbf{p}), \mathbf{d})$$

where the chain $\mathbf{d}$ is given by

$$\mathbf{d}=(\Pi i : i \in [0..|\mathbf{c}|] : [a(\mathbf{c}_i)])$$

and $a(c)$ is such that, for all codes $c \in C_\delta$, $A(v(c))=v(a(c))$.

2.2 **Example :**

(a) rotation over an angle $k\pi/2$ ($k \in \mathbb{Z}$).
In this case $a(c)$ is given by $a(c)=(c+2k) \bmod 8$.

(b) reflection in a line that has an angle $k\pi/4$ ($k \in \mathbb{Z}$) with the horizontal axis.
In this case $a(c)$ is given by $a(c)=(2k-c) \bmod 8$.

□

If A does not fulfill the requirements of property 2.1, a "good" approximation of $A(D)$ in $\mathbb{Z}^2$ may be obtained by replacing every point of $A(D)$ by a nearest point in $\mathbb{Z}^2$ according to

$$[A(D)] := \{[A(\mathbf{p})] \mid \mathbf{p} \in D\}.$$

In the next property we give a necessary and sufficient condition for $[A(D)]$ to be a discrete curve.

2.3 **Property :**

For all linear functions A the following holds.

$$(\forall D : D \text{ is a discrete } \delta\text{-curve} : [A(D)] \text{ is a discrete } \delta\text{-curve})$$
$$\Leftrightarrow$$
$$(\forall i : i \in C_\delta : D_\delta(A(v(i)), (0, 0)) \leq 1)$$

Proof

$\Rightarrow$:
For all $k>0$ the following holds.

$kD_\delta(A(\mathbf{v}_i), (0, 0))$

$= \{$ property 1.6(f) of D_δ $\}$
$D_\delta(kA(\mathbf{v}_i), (0,0))$
$\leq \{$ property 1.6(d) and (g) of D_δ $\}$
$D_\delta(kA(\mathbf{v}_i) - [kA(\mathbf{v}_i)], (0,0)) + D_\delta([kA(\mathbf{v}_i)], (0,0))$
$\leq \{$ property 1.14 of rounding and 1.6(d) and (g) of D_δ $\}$
$\frac{1}{2} + (\Sigma i : i \in [1..k] : D_\delta([iA(\mathbf{v}_i)] - [(i-1)A(\mathbf{v}_i)], (0,0)))$
$= \{$ property 1.6(g) of D_δ $\}$
$\frac{1}{2} + (\Sigma i : i \in [1..k] : D_\delta([iA(\mathbf{v}_i)], [(i-1)A(\mathbf{v}_i)]))$
$\leq \{$ hypothesis and $\{i\,\mathbf{v}_i, (i-1)\mathbf{v}i\}$ is a discrete curve $\}$
$\frac{1}{2} + k$

Hence, $D_\delta(A(\mathbf{v}_i), (0,0)) \leq 1 + \frac{1}{2k}$, for all $k \in \mathbb{Z}$. Consequently, $D_\delta(A(\mathbf{v}_i), (0,0)) \leq 1$.

$\Leftarrow$:
Let D be a discrete δ-curve.
We have to prove that $[A(D)]$ is a discrete δ-curve and hence, according to property 1.12, that $[A(D)]$ is a finite δ-connected subset of $\mathbb{Z}^2$.

- $[A(D)] \subseteq \mathbb{Z}^2$
  This follows directly from the definition of rounding.
- $[A(D)]$ is a δ-connected set.
  Let $\mathbf{p}$, $\mathbf{q}$ be two δ-neighbours in D. First we prove that $A(\mathbf{p})$ and $A(\mathbf{q})$ have a distance of at most 1.

  true
  $\Rightarrow \{$ $\mathbf{p}$ and $\mathbf{q}$ are neighbours $\}$
  $(\exists i : i \in C_\delta : v(i) = \mathbf{p} - \mathbf{q})$
  $\Rightarrow \{$ A is linear $\}$
  $(\exists i : i \in C_\delta : A(v(i)) = A(\mathbf{p}) - A(\mathbf{q}))$
  $\Rightarrow \{$ hypothesis and property 1.6(g) $\}$
  $D_\delta(A(\mathbf{p}), A(\mathbf{q})) \leq 1$

  Using this we prove that $[A(\mathbf{p})]$ and $[A(\mathbf{q})]$ are neighbours.

  true
  $\Rightarrow \{$ calculus $\}$
  $D_\delta([A(\mathbf{p})], [A(\mathbf{q})]) = D_\delta(A(\mathbf{p}) + [A(\mathbf{p})] - A(\mathbf{p}), A(\mathbf{q}) + [A(\mathbf{q})] - A(\mathbf{q}))$
  $\Rightarrow \{$ property 1.6(e) of D_δ $\}$
  $D_\delta([A(\mathbf{p})], [A(\mathbf{q})]) \leq D_\delta(A(\mathbf{p}), A(\mathbf{q})) + D_\delta([A(\mathbf{p})] - A(\mathbf{p}), [A(\mathbf{q})] - A(\mathbf{q}))$
  $\Rightarrow \{$ the above result and property 1.14(c) $\}$
  $D_\delta([A(\mathbf{p})], [A(\mathbf{q})]) < 2$
  $\Rightarrow \{$ $D_\delta([A(\mathbf{p})], [A(\mathbf{q})]) \in Z$ $\}$
  $D_\delta([A(\mathbf{p})], [A(\mathbf{q})]) \leq 1$

Hence, if $<p_0, \cdots, p_{n-1}>$ is a path in D, $<A(p_0), \cdots, A(p_{(n-1)}>$ is a path in $[A(D)]$. Hence, D is δ-connected implies that $A(D)$ is δ-connected.

□

If we represent the linear function A by the matrix $\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}$, $[A(D)]$ may be given as

$$[A(D)]=\{([xA_{00}+yA_{01}], [A_{10}x+A_{11}y]) \mid (x,y)\in D\}$$

and the requirement of the property 2.3 may be given as follows for an 8-connected curve.

$$/A_{00}/+/A_{01}/\leq 1 \wedge /A_{10}/+/A_{11}/\leq 1.$$

If we consider only 4-connected curves this requirement can be weakened to

$$/A_{00}/\leq 1 \wedge /A_{01}/\leq 1 \wedge /A_{10}/\leq 1 \wedge /A_{11}/\leq 1.$$

That is, if D is a δ-connected discrete curve, and A fulfills the respective requirement, $[A(D)]$ is a δ-connected discrete curve.

2.4 **Example :**

(a) scaling with scaling factors at most 1.

(b) Rotation of 4-connected discrete curves.

□

In the subsequent sections we give algorithms for incrementally computing the rounded bilinear expressions $[xA_{00}+yA_{01}]$ and $[xA_{10}+yA_{11}]$ for all $(x,y)\in D$. In order to obtain algorithms which use integer arithmetic only, we require that the coefficients A_{ij} are in $\mathbb{Q}$. This is not a serious drawback since every real can be approximated infinitely close by a rational.

2.2 Bresenham's line algorithms

Bresenham's line algorithm [Bre65] can be given in two ways, one producing an 8-connected chain and the other one a 4-connected chain. Here we give both algorithms. The 4-connected algorithm is given as a transformation of the 8-connected one.

2.2.1 The 8-connected case

Integer approximation of pairs $(x,y) \in \mathbb{R}^2$ fulfilling the line equation

$$ay = bx$$

for $a, b \in \mathbb{Z}$ is done by Bresenham's line algorithm.

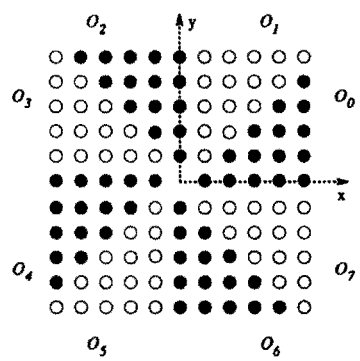


Figure 2.1 : octants

Below we give this algorithm as the function *bresh*8(a, b) that returns an 8-connected chain for a line segment from $(0, 0)$ to (a, b) where (a, b) is a point in the octant O_0 or O_7 (octants are defined in definition 2.5). All other cases of a and b are defined by transformations of *bresh* 8 chains in these two octants.
The definition of octants is illustrated by figure 2.1.

2.5 Definition : octant

For all $i \in [0..7]$ an octant O_i is defined by

$$O_i := \mathbb{Z}^2 \cap \{R^i((x, y)) \mid y \in [0, x) \wedge x \geq 0\}$$

where R is a rotation over $\pi/4$ radians.

□

Notice that every two different octants have an empty intersection and that the union of all the octants is $\mathbb{Z}^2 \setminus \{(0, 0)\}$.

Bresenham's line algorithm produces the best possible digitisation of a line segment ($ay=bx$, $x \in [0..a]$, and $(a, b) \in O_0 \cup O_7$) in the sense that for every $x \in [0..a]$, y is approximated as good as possible in integers, namely by $[\frac{b}{a}x]$. Hence, the function *bresh* 8 has the following property for $(a, b) \in O_0 \cup O_7$

$$P(bresh\,8(a, b)) = \{(x, y) \mid x \in [0..a] \wedge y = [\tfrac{b}{a}x]\}.$$

Notice that this set is indeed both finite and connected for $(a, b) \in O_0 \cup O_7$, and is, hence, a discrete curve according to property 1.12.

Below we give an algorithm for computing the chain $\mathbf{c} = bresh\,8(a, b)$. The pixels $\mathbf{p}(\mathbf{c}, i)$ are given, for $i \in [0..a]$ by

$$\mathbf{p}(\mathbf{c}, i) = (i, [\tfrac{b}{a}i]).$$

or in an equivalent formulation

$$\mathbf{p}_x(\mathbf{c}, i) = i \wedge -\tfrac{1}{2} \leq \mathbf{p}_y(\mathbf{c}, i) - \tfrac{b}{a}i < \tfrac{1}{2}.$$

This results in the following three predicates for $\mathbf{c}$.

$$\mathbf{R_0}: \quad (\forall i : i \in [0..|\mathbf{c}|] : \mathbf{p}_x(\mathbf{c}, i) = i \wedge -\tfrac{1}{2} \leq \mathbf{p}_y(\mathbf{c}, i) - \tfrac{b}{a}i < \tfrac{1}{2})$$
$$\mathbf{R_1}: \quad |\mathbf{c}| = a$$
$$\mathbf{R}: \quad \mathbf{R_0} \wedge \mathbf{R_1}$$

Hence, the function bresh8 is a solution to the following problem:

```
|[
 a, b : int {-a≤ b< a}
      |[
       c : chain;
       c :=bresh8 (a, b)
       { R }
      ]|
]|
```

Below we give the program for the function bresh8. Notice that it uses only integer expressions. To obtain this we multiplied the error term $\mathbf{p}_y(\mathbf{c}, x) - \frac{b}{a}x$ in $\mathbf{R_0}$ by $2a$; the resulting integer expression is called e. Furthermore the program contains no multiplications, apart from shifts, and divisions. Hence, an efficient and accurate result has been obtained. The following invariants, based on $\mathbf{R_0}$ and $\mathbf{R_1}$, are used for its proof.

$$\mathbf{P_0}: \quad (\forall i : i \in [0..x] : \mathbf{p}_x(\mathbf{c}, i) = i \wedge -\tfrac{1}{2} \leq \mathbf{p}_y(\mathbf{c}, i) - \tfrac{b}{a}i < \tfrac{1}{2})$$
$$\mathbf{P_1}: \quad x = |\mathbf{c}| \wedge 0 \leq x \leq a \wedge y = \mathbf{p}_y(\mathbf{c}, x)$$
$$\mathbf{P_2}: \quad e = 2ay - 2bx$$
$$\mathbf{P_3}: \quad -a \leq e < a$$
$$\mathbf{P}: \quad \mathbf{P_0} \wedge \mathbf{P_1} \wedge \mathbf{P_2} \wedge \mathbf{P_3}$$

The if-statement repairs $\mathbf{P_1}$ and leaves $\mathbf{P_2} \wedge \mathbf{P0}^{x}_{x-1}$ invariant. Notice that $\mathbf{P0}^{x}_{x-1} \wedge \mathbf{P_1}$ induce $\mathbf{P_0}$. $\mathbf{P_3}$ is also valid after the if-statement, as is proved for the first alternative by

$$
\begin{aligned}
& e<-a \wedge \mathbf{P3}^{e}_{e+2b} \\
&= \{ \text{calculus} \} \\
& -2b-a \leq e<-a \\
&= \{ b<a \} \\
& -a<e+2a<a
\end{aligned}
$$

Hence, the addition of $2a$ to e in the first alternative indeed induces $\mathbf{P_3}$.

```
func bresh8(a, b:int) : chain          { -a≤b<a }
    e, x, y : int;
    c : chain;
    e, x, y, c :=0, 0, 0, ε; {P}
    do x≠a →
        e, x :=e-2*b, x+1; { P0^x_{x-1} ∧ P1^x_{x-1} ∧ P2 ∧ P3^e_{e+2b} }
        if e<-a          →  e, y, c :=e+2*a, y+1, c ⊗ [1]
        [] -a≤ e<a       →  c :=c ⊗ [0]
        [] e≥ a          →  e, y, c :=e-2*a, y-1, c ⊗ [7]
        fi {P}
    od; { P ∧ x=a; Hence, R }
    bresh8 :=c
cnuf
```

This function differs from the original Bresenham algorithm in two ways:

(1) Originally the algorithm has been defined for only one octant ($(a, b) \in O_1$).

(2) Because of this limitation the algorithm could be made a little more efficient by using $E=e+a-2b$ as the new error term. The body of the loop then is:

```
x :=x+1;
if E<0    →  E, y, c :=E+2*a-2*b, y+1, c ⊗ [1]
[] E≥ 0   →  E, c :=E-2*b, c ⊗ [0]
fi
```

Computing *bresh*8(a, b) in all octants may be done according to the following recipe:

Transform (a, b) using a linear function A such that the resulting point r is in $O_0 \cup O_7$. Compute $\mathbf{c}=$*bresh*8(r_x, r_y) according to the above algorithm and apply A^{-1} to the discrete curve $DC((0, 0), \mathbf{c})$.

Applying the inverse transformation can be done according to property 2.1 and example 2.2. Below *bresh*8 is defined in the other octants in such a way that

property 2.6 holds. Figure 2.2 illustrates the transformations used for this definition by showing the direction in which the rounding of the continuous line segment points to grid points takes place.

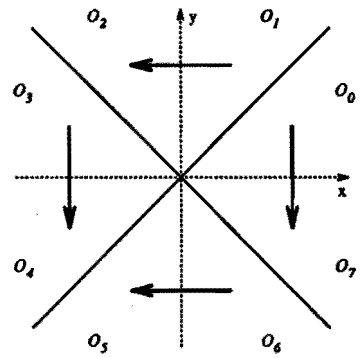


Figure 2.2: rounding direction for line segments resulting from *bresh* 8

The notation $\mathbf{c}\Big|_d^c$ denotes for $c, d \in C_8$ the chain $\mathbf{c}$ with the c-codes replaced by d-codes.

$$bresh\,8(a,b) := \begin{cases} bresh\,8(b,a)\Big|_{1,2,3}^{1,0,7} & \text{if } (a,b) \in O_1 \cup O_2 \\ bresh\,8(-a,b)\Big|_{3,4,5}^{1,0,7} & \text{if } (a,b) \in O_3 \cup O_4 \\ bresh\,8(-b,a)\Big|_{7,6,5}^{1,0,7} & \text{if } (a,b) \in O_5 \cup O_6 \\ \varepsilon & \text{if } (a,b) = (0,0) \end{cases}$$

2.6 Property :

For all pairs $(a, b) \in \mathbb{Z}^2 \setminus \{(0,0)\}$ and for all $k \in [0 .. \max\{|a|, |b|\}]$,

$$\mathbf{p}(bresh\,8(a,b), k) = \left[\frac{k(a,b)}{\max\{|a|,|b|\}}\right]$$

□

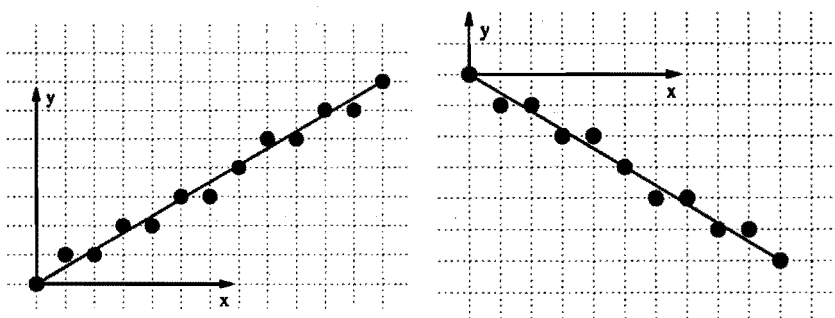
Notice that $\frac{k(a,b)}{\max\{|a|,|b|\}}$ need be rounded for just one of its coordinates and hence the total error is the error in one coordinate and indeed at most ½.

2.7 Example :

Figure 2.3a and 2.3b show two 8-connected discrete lines generated by *bresh* 8. Their respective chain codes are:

bresh 8(12, 7) = [101010110101]
bresh 8(10, –6) = [7070770707]

□



(a) (b)

Figure 2.3 : example 2.7

Because $\max\{|a|,|b|\}$ is the cardinality of the pixel set of bresh8(a,b), the time complexity of this algorithm is given by $O(\max\{|a|,|b|\})$.

The chains generated by bresh8 have the following two properties [Wu82] :

* they contain at most two different elements of C_8.
* these elements are distributed as uniformly as possible along the chain.

2.2.2 The 4-connected case

In the previous section we defined the function *bresh* 8(a, b) ($a, b \in \mathbb{Z}$) as the chain of an 8-connected discrete curve that approximates the line segment from the origin to the point (a, b) of the line given by $ay = bx$. In this section we give a 4-connected counterpart of *bresh* 8 called *bresh* 4. We define the chain *bresh* 4(a, b) for integers a and b with $(a, b) \in O_0 \cup O_1$. The other cases of a and b may be obtained by rotation and reflection of chains, but we have no need for them in this thesis.

Instead of giving a treatise similar to the one in the previous section, we define the chain *bresh* 4(a, b) as a chain *bresh* 8(a', b') ($0 \le b' < a'$) in which the codes are systematically replaced by corresponding 4-connected codes. We can do this since a 4-connected Bresenham chain has, because of the symmetry of the continuous line,

properties similar to the above mentioned properties for *bresh* 8 chains: it contains at most two different elements of C_4 and these elements also have a distribution as uniform as possible. By replacing the code(s) of C_8 by code(s) of C_4 the uniform distribution remains the same. We define

$$bresh4(a, b) := bresh8(a+b, b)|_2^1 \quad \text{for } (a, b) \in O_0 \cup O_1.$$

Notice that the chain $bresh8(a+b, b)$ has length $a+b$ and contains a 0-codes and b 1-codes. Hence, for the end point of *bresh* 4 holds that $\mathbf{p}(bresh4(a, b), a+b) = (a, b)$. Notice furthermore, that for a chain with only 0-codes and 1-codes replacing the 1-codes by 2-codes is similar to transforming the chain with a linear function given by the matrix $\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$. Indeed,

$$(1, 0)\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = (1, 0) \text{ and } (1, 1)\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = (0, 1)$$

Hence, the pixel set of the chain $bresh4(a, b)$ is given by the following property (compare property 2.6).

2.8 **Property :**

For all pairs $(a, b) \in O_0 \cup O_1$ and for $i \in [0 .. a+b]$,

$$\mathbf{p}(bresh4(a, b), i) = (i - \lceil i\frac{b}{a+b} \rceil, \lceil i\frac{b}{a+b} \rceil)$$

Proof

For all $i \in [0 .. a+b]$ the following holds.

$\mathbf{p}(bresh4(a, b), i)$
= { the above definition of *bresh* 4 }
$\mathbf{p}(bresh8(a+b, b), i)\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$
= { property 2.6 }
$\lceil (i, i\frac{b}{a+b}) \rceil \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$
= { calculus }
$(i, \lceil i\frac{b}{a+b} \rceil)\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$
= { calculus }
$(i - \lceil i\frac{b}{a+b} \rceil, \lceil i\frac{b}{a+b} \rceil)$

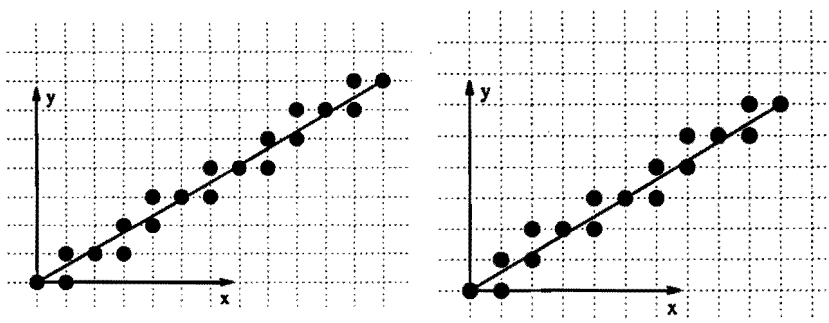
□

2.9 **Example :**

Figure 2.4a and 2.4b contain two 4-connected Bresenham lines. Their chains are:

$bresh\,4(12, 7) = [02002002002002020020]$
$bresh\,4(10, 6) = [02020020020020020]$

□



(a) (b)

Figure 2.4: example 2.9

The time complexity of computing $bresh\,4(a, b)$ is the time complexity of computing $bresh\,8(a+b, b)$ and hence, is $O(|a|+|b|)$.

2.3 Integer approximation of linear functions

In this section we show an integer algorithm for the computation of a linear function on a given interval. It is an introduction to the algorithm for approximating a bilinear function on a given discrete curve; this algorithm is given in section 2.4..
We compute an integer approximation of a line segment

$$ay = bx \text{ with } x \in [0, X],\ X \in \mathbb{N}_0,\ a \in \mathbb{Z},\ b \in \mathbb{Z},\ \text{and } a > 0$$

by computing the points $(i, Y[i])$ for all $x \in [0..x]$. $Y[i]$ is again a best approximation and equals $\lceil \frac{b}{a} x \rceil$.
Notice that in case $-a \le b < a$ this problem can be solved by first computing bresh8(a, b) and afterwards simply computing $Y[i] = \mathrm{p}_y(bresh\,8(a, b), i)$. In general, however, the set $\{(x, y) \mid y = \lceil \frac{b}{a} x \rceil \wedge x \in [0..X]\}$ is not 8-connected and hence, cannot be represented by a chain. Using the predicate **R** given as

$$\mathbf{R}: \quad (\forall x : 0 \le x < X : -\tfrac{1}{2} \le Y[x] - \frac{b}{a}x < \tfrac{1}{2})$$

the problem can now be formulated as follows.
Find a list of statements, called *intlin*, with the following property.

```
|[
 a, b : int; {a≠0}
 X : int; {X≥0}
     |[
      Y(i : 0≤i<X) : array of int;
      intlin;
      { R }
     ]|
]|
```

Invariants leading to the solution of this problem are similar to those for *bresh* 8 in section 2.2.

$$\mathbf{P_0}: \quad (\forall i : 0 \le i < x : -\tfrac{1}{2} \le Y[i] - \frac{b}{a}i < \tfrac{1}{2})$$
$$\mathbf{P_1}: \quad 0 \le x \le X$$
$$\mathbf{P_2}: \quad e = 2ay - 2bx$$
$$\mathbf{P_3}: \quad -a \le e < a$$
$$\mathbf{P}: \quad \mathbf{P_0} \wedge \mathbf{P_1} \wedge \mathbf{P_2} \wedge \mathbf{P_3}$$

The algorithm *intlin* as given below consists of two nested loops. The outer loop has as invariant **P**. The inner loop has invariant $\mathbf{P0}^{x}_{x-1} \wedge \mathbf{P_1} \wedge \mathbf{P_2}$. Notice that if the inner loop ends $\mathbf{P_3}$ holds. In order to proof that this loop ends we distinguish two cases for the value of e. In case $e=a$ the inner loop ends after execution of $e:=e-2a$. In case $|e|>a$, $|e|$ is a variant function of this loop (that is $|e|$ is positive and decreases with each iteration step) since $e<-a$ implies $|e+2a|<|e|$ and $e>a$ implies $|e-2a|<|e|$.

```
|[ e, x, y : int;
  e, x, y, Y[0]:=0, 0, 0, 0; { P }
  do x≠X →
      e, x:=e-2*b, x+1; { P1 ∧ P0^x_{x-1} }
      do e<-a  →  e, y:=e+2*a, y+1
      [] e≥a   →  e, y:=e-2*a, y-1
      od; { P2 ∧ P1 ∧ P0^x_{x-1} }
      Y[x]:=y; { P }
  od { P ∧ x=X; hence, R }
]|
```

The time-complexity of this algorithm is $O(\max\{/X/,/X\,b\,a^{-1}/\})$.

In case $0 \leq b < a$ the above algorithm can be seen to be equal to:

```
|[ c : chain ; i, y : int;
  c :=bresh8(a,b);
  i, y :=0, 0;
  do i< |c| →
     if c_i =1   →   y :=y+1
     [] c_i =0   →   skip
     fi;
     Y[i] :=y
  od
]|
```

From the definition of *bresh*4 we know that *bresh*8(*a*,*b*) is *bresh*4(*a*–*b*,*b*) with the 2-codes replaced by 1-codes. Hence, the above algorithm can also be written as:

```
|[ c : chain ; i, y : int;
  c :=bresh4(a-b,b);
  i, y :=0, 0;
  do i< |c| →
     if c_i =2   →   y :=y+1
     [] c_i =0   →   skip
     fi;
     Y[i] :=y
  od
]|
```

2.4 Integer approximation of bilinear functions on a discrete curve

In this section we show an integer algorithm for the computation of a bilinear function given by

$$cz = ax + by \quad (a,b,c \in \mathbb{Z} \wedge c > 0),$$

on a the pixel set $P(\mathbf{c})$ of a chain **c**. This algorithm is clearly related to the topic of linear transformation of discrete curves, in which two of these equations play a role. At the end of this section we show an example featuring this application. We now formulate the following problem.

Find a list of statements *intbilin* with the following property.

```
|[
 a, b, c : int; {c>0}
 c : chain;
 |[
   Z(i : 0≤i≤ |c|) : array of int;
   intbilin;
   { R }
 ]|
]|
```

where **R** is a predicate given by

$$\mathbf{R}: \quad (\forall i : 0 \le i \le |\mathbf{c}| : -\tfrac{1}{2} \le Z[i] - \tfrac{a}{c}p_x(\mathbf{c}, i) - \tfrac{b}{c}p_y(\mathbf{c}, i) < \tfrac{1}{2})$$

The invariants used in the solution below are similar to those in section 2.3 and read as follows.

$$\mathbf{P_0}: \quad (\forall j : 0 \le j \le i : -\tfrac{1}{2} \le Z[j] - \tfrac{a}{c}p_x(\mathbf{c}, j) - \tfrac{b}{c}p_y(\mathbf{c}, j) < \tfrac{1}{2})$$

$$\mathbf{P_1}: \quad 0 \le i \le |\mathbf{c}| \wedge (x, y) = \mathbf{p}(\mathbf{c}, i)$$

$$\mathbf{P_2}: \quad e = 2cz - 2by - 2ax$$

$$\mathbf{P_3}: \quad -c \le e < \mathbf{c}$$

$$\mathbf{P}: \quad \mathbf{P_0} \wedge \mathbf{P_1} \wedge \mathbf{P_2} \wedge \mathbf{P_3}$$

We denote the inproduct of two vectors **p** and **q** by <**p**, **q**>.

```
|[
 e, x, y, z, i : int;
 e, x, y, i, z, Z[0] :=0, 0, 0, 0, 0;
 do i< |c| →
      (x,y) := (x,y)+v(c_i);
      e :=e-2*<(a, b), v(c_i)>; { P0 ∧ P2 ∧ P1^i_{i+1} }
      do e<-c  →  e, z :=e+2*c, z+1
      [] e≥c   →  e, z :=e-2*c, z-1
      od; { P0 ∧ P2 ∧ P3 ∧ P1^i_{i+1} }
      i :=i+1;
      Z[i] :=z { P }
 od { R }
]|
```

Two remarks can be made on this program:

x and *y* are ghost variables and consequently, may be left out.

The computation of an inproduct of 2 vectors needs in general two multiplications. However, in the above program these may be avoided, since the coordinates of the vectors $v(\mathbf{c}_i)$ are either 0, 1, or -1.

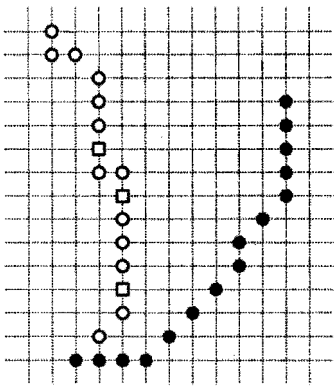


Figure 2.5: example 2.10

2.10 **Example :**

Linearly tranforming an 8-connected discrete curve point by point may result in an unconnected set of points (properties 2.1 and 2.3). We can compute such a set of points by using *intbilin*. In order to obtain a connected set we may interpolate consecutive points in the image set for instance with a *bresh* 8 chain.
In figure 2.5 a rotation over $\pi/4$ of an 8-connected chain round its start point is illustrated. In case of a rotation at most one extra pixel is needed for the interpolation of two consecutive pixels in the image. The 'square' pixels in the figure are the pixels obtained by interpolation.
The original and the image chains are given as follows.

[0 0 0 1 1 1 1 2 1 1 2 2 2 2]
[1 8 1 2 2 2 2 2 2 4 2 2 2 2 3 4 2]

□

3

W-curves

3.0 Introduction

In this chapter a method is defined for generating discrete curves, called w-curves. These curves are defined by three (control) points and two distribution functions. Two points determine the end points of the curve and the line segments towards the third point determine the tangent vectors of the curve in these end points. The distribution functions are used to define the shape of the the curve.

A continuous variant of these discrete curves is given. Smoothing algorithms for the w-curves are given and also algorithms for generating them. The generating algorithms for w-curves have a time complexity worse than linear in the lengths of the chains involved. A slight modification of the definition of w-curves results in (a subset of) curves, called e-curves, which can be computed in linear time.

3.1 Problem Definition

In this section we give a discrete counterpart of the following continuous problem:

Given three (control) points $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$, find a continuous curve that

(1) interpolates the control points $\mathbf{p}_0$ and $\mathbf{p}_2$; that is, it starts at $\mathbf{p}_0$ and ends at $\mathbf{p}_2$.

(2) is tangent to line segment $\mathbf{p}_0\mathbf{p}_1$ at $\mathbf{p}_0$ and to the segment $\mathbf{p}_1\mathbf{p}_2$ at $\mathbf{p}_2$.

In translating this problem to a discrete problem we have to introduce a notion of tangency for discrete curves. We define this notion only for curves tangent in their start points. Two continuous curves are said to be tangent in a point $\mathbf{p}$ if they both contain $\mathbf{p}$, and if their tangent vectors in $\mathbf{p}$ (if any) are collinear. In accordance with this definition, two discrete curves $DC(\mathbf{p}_0, \mathbf{c})$ and $DC(\mathbf{p}_1, \mathbf{d})$ can be said to be tangent (in their start points) if $\mathbf{p}_0=\mathbf{p}_1$ and $\nu(\mathbf{c}_0)=\nu(\mathbf{d}_0)$. In the next definition the notion of tangent vector is not limited to the first basic vector of a curve; if two curves have their first n basic vectors in common they are said to be n-tangent.

3.1 **Definition : n-tangent**

Two chains $\mathbf{c}$ and $\mathbf{d}$ are called n-tangent ($n \in \mathbb{N}_0$) iff

$$(\forall i : i \in [0..n) : \mathbf{c}_i = \mathbf{d}_i)$$

Two discrete curves $DC(\mathbf{p}_0, \mathbf{c})$ and $DC(\mathbf{p}_1, \mathbf{d})$ are said to be n-tangent (in their start points) iff $\mathbf{p}_0 = \mathbf{p}_1$ and the chains $\mathbf{c}$ and $\mathbf{d}$ are n-tangent.

□

Notice that all pairs of discrete curves starting in the same point, are 0-tangent. Furthermore, notice that tangent continuous curves exist that have digitisations that are only 0-tangent (see figure 3.1).

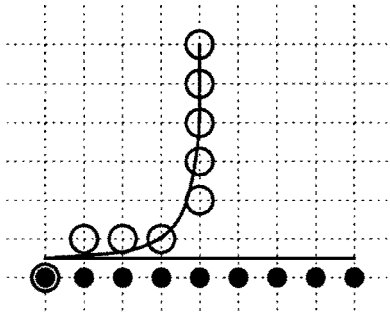


Figure 3.1: Two tangent continuous curves with 0-tangent discretisations

Hence, 0-tangency is a useful notion in discrete space with respect to tangency in continuous space.

The above continuous problem can now be formulated as follows in discrete space:

Given three (control) points $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$ in $\mathbb{Z}^2$, find a discrete curve $DC(\mathbf{p}, \mathbf{e})$ that

(1) interpolates the points $\mathbf{p}_0$ and $\mathbf{p}_2$; that is $\mathbf{p} = \mathbf{p}_0$ and $\mathbf{p} + end(\mathbf{e}) = \mathbf{p}_2$.

(2) is n-tangent to $DC(\mathbf{p}_0, bresh\, 8(\mathbf{p}_1 - \mathbf{p}_0))$ and $DC(\mathbf{p}_2, rev(\mathbf{e}))$ is m-tangent to $DC(\mathbf{p}_2, rev(bresh\, 8(\mathbf{p}_2 - \mathbf{p}_1)))$, for some $m, n \in \mathbb{N}_0$.

The reverse chains appear in requirement (2) because the notion n-tangent is only defined for the start point of curves. Requirement (2) is, as we saw before, a dummy requirement since all pairs of chains are at least 0-tangent. In this chapter we give a method for solving this problem for several combinations of m and n. These solu-

tions are constructed according to the following scheme:

(a) compute the starting point:
$\mathbf{p} := \mathbf{p}_0$

(b) compute the chains **c** and **d**:
$\mathbf{c} := bresh\,8(\mathbf{p}_1 - \mathbf{p}_0)$
$\mathbf{d} := bresh\,8(\mathbf{p}_2 - \mathbf{p}_1)$
The chains **c** and **d** are called *control chains* .

(c) compute a chain **e** of the new discrete curve by combining all the codes of the chains **c** and **d**. An exact definition of combining is given in the sequel.

This scheme guarantees that the resulting curve fulfills (1) since

$$\mathbf{p} + end(\mathbf{e}) = \mathbf{p}_0 + end(\mathbf{c}) + end(\mathbf{d}) = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0) + (\mathbf{p}_2 - \mathbf{p}_1) = \mathbf{p}_2.$$

In section 3.2 we define two operators on chains. In section 3.3 these operators are used for constructing the discrete curves called w-curves.

3.2 Operators on chains

In this section two operators on chains are defined; namely, an infix operator $\underline{w}$ for combining two chains and an operator *add* 8 for adding 8-codes to a chain.

3.2.0 The weave operator

As mentioned in the introduction we are going to construct curves by combining the chains of the control lines. The weave operator $\underline{w}$ combines two chains into one and is defined as follows.

3.2 Definition : weave operator

For all chains **c** and **d** the chain $\mathbf{c}\,\underline{w}\,\mathbf{d}$ is defined by defining the codes $(\mathbf{c}\,\underline{w}\,\mathbf{d})_i$ for $i \in [0\,..\,|\mathbf{c}| + |\mathbf{d}|)$ by

$$(\mathbf{c}\,\underline{w}\,\mathbf{d})_i := \begin{cases} \mathbf{c}(\#_0(\mathbf{b}, 0, i)) & \text{if } \mathbf{b}_i = 0 \\ \mathbf{d}(\#_2(\mathbf{b}, 0, i)) & \text{if } \mathbf{b}_i = 2 \end{cases}$$

where $\mathbf{b} = bresh\,4(|\mathbf{c}|, |\mathbf{d}|)$.

□

Hence, the chain $\mathbf{c}\,\underline{w}\,\mathbf{d}$ contains all codes of the chains **c** and **d** where these codes are distributed as the 0-codes and 2-codes in $bresh\,4(|\mathbf{c}|, |\mathbf{d}|)$, respectively. The same distribution of codes, but now of 0-codes and 1-codes, exists in the chain

bresh 8(|c|+|d|, |c|) and we might have used this chain in the above definition, since it only uses the (uniform) distribution of the codes. Notice that *bresh* 4(|c|, |d|) contains only 0-codes and 2-codes and has length |c|+|d|; hence, the above definition indeed defines every element of c $\underline{w}$ d.

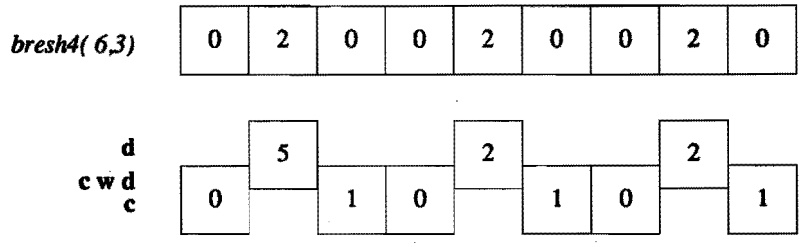


Figure 3.2: example 3.3

3.3 **Example :**

c=[010101] and d=[522]
bresh 4(|c|, |d|)=*bresh* 4(6, 3)=[020020020]
Hence, c $\underline{w}$ d=[051021021] (see figure 3.2).

□

3.4 **Example :**

(a) c=*bresh* 8(10, 10)=[1]^{10} and d=*bresh* 8(−5, 20)=[2322]^{5}
bresh 4(|c|, |d|)=*bresh* 4(10, 20)=[202]^{10}
Hence, c $\underline{w}$ d=[213212]^{5} (see figure 3.3 (a)).

(b) c=*bresh* (p_1-p_0) and d=*bresh* (p_2-p_1)
with p_1-p_0=(445,194) and p_2-p_1=(71,214).
Figure 3.3 (b) shows the chain c $\underline{w}$ d.

□

From the definition of $\underline{w}$ it follows that *end*(c $\underline{w}$ d)=*end*(c)+*end*(d). In property 3.6, stated below, we give, in terms of the pixels in the pixel set of the chains c and d, an expression for all the pixels in the pixel set of c $\underline{w}$ d. The following property is an auxiliary property for the proof of 3.6. It states a relation between the chains c, d, and *bresh* 4(|c|, |d|).

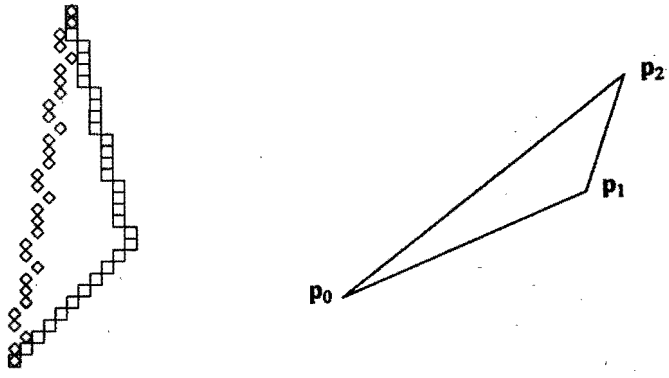


Figure 3.3: example 3.4 (a) and (b)

3.5 **Property :**

For all chains **c**, **d**, and $\mathbf{b}=bresh4(|\mathbf{c}|, |\mathbf{d}|)$ and all $t \in [0\,..\,|\mathbf{b}|]$

(a) $\mathbf{p}(\mathbf{c}, \mathbf{p}_x(\mathbf{b}, t))=(\Sigma i : 0\le i<t \wedge \mathbf{b}_i=0 : v(\mathbf{c}(\#_0(\mathbf{b}, 0, i))))$

(b) $\mathbf{p}(\mathbf{d}, \mathbf{p}_y(\mathbf{b}, t))=(\Sigma i : 0\le i<t \wedge \mathbf{b}_i=2 : v(\mathbf{d}(\#_2(\mathbf{b}, 0, i))))$

Proof

We prove property (a) by induction. Property (b) may be proved similarly.

base: $t=0$

$(\Sigma i : 0\le i<0 \wedge \mathbf{b}_i=0 : v(\mathbf{c}(\#_0(\mathbf{b}, 0, i))))$
= { calculus }
(0, 0)
= { $\mathbf{p}(\mathbf{c}, 0)=(0, 0)$ and $\mathbf{p}_x(\mathbf{b}, 0)=0$ }
$\mathbf{p}(\mathbf{c}, \mathbf{p}_x(\mathbf{b}, 0))$

step: $t>0$

The chain **b** contains only 0-codes and 2-codes. We only give the proof for the induction step if $\mathbf{b}_{t-1}=0$.

$(\Sigma i : 0\le i<t \wedge \mathbf{b}_i=0 : v(\mathbf{c}(\#_0(\mathbf{b}, 0, i))))$
= { calculus }
$(\Sigma i : 0\le i<t-1 \wedge \mathbf{b}_i=0 : v(\mathbf{c}(\#_0(\mathbf{b}, 0, i))))+v(\mathbf{c}(\#_0(\mathbf{b}, 0, t-1))$
= { induction hypothesis }
$\mathbf{p}(\mathbf{c}, \mathbf{p}_x(\mathbf{b}, t-1))+v(\mathbf{c}(\#_0(\mathbf{b}, 0, t-1))$
= { definition 1.21 of **p** and $\mathbf{p}_x(\mathbf{b}, t-1)=\#_0(\mathbf{b}, 0, t-1)$ }
$(\Sigma i : 0\le i<\#_0(\mathbf{b}, 0, t-1) : v(\mathbf{c}_i))+v(\mathbf{c}(\#_0(\mathbf{b}, 0, t-1))$
= { calculus }

$(\Sigma i : 0 \le i \le \#_0(\mathbf{b}, 0, t-1) : v(\mathbf{c}_i))$
$=$ { $\#_0(\mathbf{b}, 0, t) = \#_0(\mathbf{b}, 0, t-1)+1$ because $\mathbf{b}_{t-1}=0$ }
$(\Sigma i : 0 \le i < \#_0(\mathbf{b}, 0, t) : v(\mathbf{c}_i))$
$=$ { $\#_0(\mathbf{b}, 0, t) = \mathbf{p}_x(\mathbf{b}, t)$ and definition 1.21 of $\mathbf{p}$ }
$\mathbf{p}(\mathbf{c}, \mathbf{p}_x(\mathbf{b}, t))$

□

The expression in the next property for the t-th pixel of $\mathbf{c} \underline{w} \mathbf{d}$ is asymmetric in $\mathbf{c}$ and $\mathbf{d}$. This is due to the inevitable asymmetry of the round operator. For $\mathbf{c}$ and $\mathbf{d}$ with $\frac{|\mathbf{d}|}{|\mathbf{d}|+|\mathbf{c}|}+\frac{1}{2} \notin \mathbb{Z}$ the expression is symmetric, since then, according to property 1.14(e), the following holds.

$$t-\lfloor \frac{|\mathbf{d}|}{|\mathbf{d}|+|\mathbf{c}|} t \rfloor = \lfloor \frac{|\mathbf{c}|}{|\mathbf{d}|+|\mathbf{c}|} t \rfloor .$$

3.6 **Property :**

For all chains $\mathbf{c}$ and $\mathbf{d}$, and for $t \in [0 .. |\mathbf{c}|+|\mathbf{d}|]$,

$$\mathbf{p}(\mathbf{c} \underline{w} \mathbf{d}, t) = \mathbf{p}(\mathbf{c}, t-\lfloor \frac{|\mathbf{d}|}{|\mathbf{c}|+|\mathbf{d}|} t \rfloor) + \mathbf{p}(\mathbf{d}, \lfloor \frac{|\mathbf{d}|}{|\mathbf{c}|+|\mathbf{d}|} t \rfloor).$$

Proof

Let $\mathbf{b}$ be *bresh* $4(|\mathbf{c}|, |\mathbf{d}|)$.

$\mathbf{p}(\mathbf{c} \underline{w} \mathbf{d}, t)$
$=$ { definition 1.21 of $\mathbf{p}$ }
$(\Sigma i : 0 \le i < t : v((\mathbf{c} \underline{w} \mathbf{d})_i))$
$=$ { definition 3.2 of $\underline{w}$ }
$(\Sigma i : 0 \le i < t \wedge \mathbf{b}_i = 0 : v(\mathbf{c}(\#_0(\mathbf{b}, 0, i))))$
$+$
$(\Sigma i : 0 \le i < t \wedge \mathbf{b}_i = 2 : v(\mathbf{d}(\#_2(\mathbf{b}, 0, i))))$
$=$ { property 3.5 }
$\mathbf{p}(\mathbf{c}, \mathbf{p}_x(\mathbf{b}, t)) + \mathbf{p}(\mathbf{d}, \mathbf{p}_y(\mathbf{b}, t))$
$=$ { property 2.8 }
$\mathbf{p}(\mathbf{c}, t-\lfloor \frac{|\mathbf{d}|}{|\mathbf{c}|+|\mathbf{d}|} t \rfloor) + \mathbf{p}(\mathbf{d}, \lfloor \frac{|\mathbf{d}|}{|\mathbf{c}|+|\mathbf{d}|} t \rfloor)$

□

The arguments of the weave operator can be any chain. If, however, both argument chains are control chains, that is to say both chains have been obtained by Bresenham's line algorithm, the pixel set of the resulting chain resembles a straight line segment (see figure 3.3 (a) and (b)). This resemblance can be explained by the following argument: if $\mathbf{c}$ is a Bresenham chain of the line segment $\mathbf{p}_0\mathbf{p}_1$ and $\mathbf{d}$ of the

line segment p_1p_2, we can state, according to property 2.6, that

$$p(c, t) \approx t\frac{p_1 - p_0}{|c|} \wedge p(d, t) \approx t\frac{p_2 - p_1}{|d|}.$$

Hence, with property 3.6

$$p(c \underline{w} d, t) \approx (t - \frac{|d|}{|c|+|d|}t)\frac{p_1 - p_0}{|c|} + \frac{|d|}{|c|+|d|}t\frac{p_2 - p_1}{|d|} = t\frac{p_2 - p_0}{|c|+|d|}.$$

A non-mathematical explanation of this phenomenon is: each of the chains **c**, **d**, and **b** (=*bresh*4(|**c**|, |**d**|)) consists of 2 uniformly distributed codes. Consequently, **c** $\underline{w}$ **d** has a uniform distribution of a maximum of four different codes, and hence, its pixel set resembles a line segment.

3.2.1 The add8 operator

As we saw in the previous section, the uniform distribution of the control chains determines the result of the weave operator. In this section an operator that adds 8-codes to a chain according to some distribution function is given; this operator is called *add*8. By doing this the chain has, in general, no longer a uniform distribution of codes. Consequently, the result of the weave no longer resembles a straight line segment. Notice, furthermore, that by adding 8-codes to a chain the pixel set of that chain does not change. In the sequel, first distribution functions are defined, and thereupon the operator *add*8 is defined.

3.7 **Definition** : distribution function

For all $n \in \mathbb{N}_0$, the set of functions $\mathcal{D}_n$ is defined by

$$\mathcal{D}_n := \{f | f \in [0\,..\,n] \rightarrow \mathbb{N}_0\}$$

The elements of $\mathcal{D}_n$ ($n \in \mathbb{N}_0$) are called distribution functions.

□

3.8 **Definition** : *add*8

We define the operator *add*8 for all chains **c** and all distribution functions $f \in \mathcal{D}_{|c|}$ by

$$add8(c, f) := [8]^{f(0)} \otimes [c_0] \otimes [8]^{f(1)} \otimes [c_1] \otimes \cdots \otimes [c_{n-1}] \otimes [8]^{f(n)}$$

□

3.9 **Example :**

$f:[0..6]\rightarrow\mathbb{Z}$ with $f(i):=i$
$add\,8([000000],f) := [0808808880888880888880888888]$

□

Below a property for the pixels in the pixel set of $add\,8$ is given along with some definitions. First monotonous and continuous functions F are defined such that the $F(i)$-th code in the chain $add\,8(c,f)$ is c_i. The functions F are defined on a real interval since their inverses are needed in the sequel.

3.10 **Definition :** primitive function

A monotonous and continuous function $F:[-1,n]\rightarrow\mathbb{R}$ is called a primitive function of a distribution function $f\in\mathcal{D}_n$ iff

$$F(0)=f(0)\wedge(\forall j : j\in[0..n] : F(j)=F(j-1)+f(j)+1)$$

□

The name primitive function is chosen since, as is illustrated in figure 3.4, $F(j)$ approximates the area under the function $f+1$.

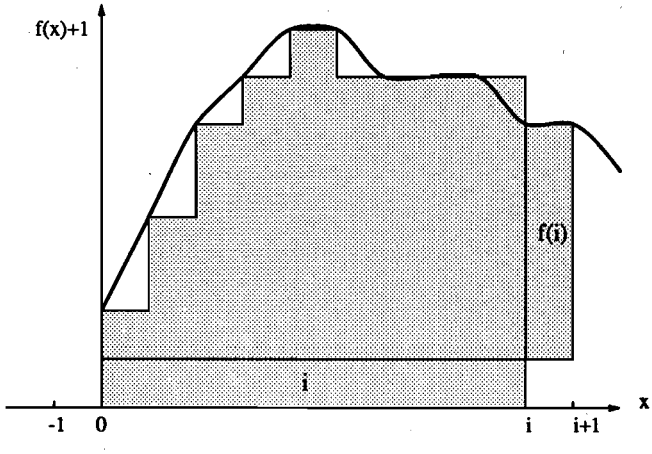


Figure 3.4: primitive function F
The area of the gray surface equals $F(i)$ $(i\in\mathbb{N}_0)$.

Primitive functions are notated by the capital letter of the corresponding distribution function. The following remarks can be made on this definition.

(1) The sequence $F(-1),\cdots,F(n)$ is monotonous because $f(i)+1>0$, for all $i\in[-1..n]$.

(2) The values of $F(x)$, with x not an integer, are not prescribed by the above definition. Hence, in general, several primitive functions of a given distribution function can exist. Because of (1), there are primitive functions for every distribution function.

(3) Since F is both monotonous and continuous, F has an inverse function F^{-1}.

(4) $F(-1)=-1$.

The following property gives a non-recursive expression for F.

3.11 **Property :**

For all chains **c** and all primitive functions F of a distribution function $f \in \mathcal{D}_{|\mathbf{c}|}$ the following holds.

(a) $(\forall i : i \in [-1\,..\,|\mathbf{c}|] : F(i) = i + (\Sigma j : j \in [0\,..\,i] : f(j)))$

(b) $F(|\mathbf{c}|) = |add8(\mathbf{c}, f)|$

□

A proof of property 3.11(a) follows directly from the definition of F; (b) can be deduced from (a) and the fact that the sum in (a) equals, for $i = |\mathbf{c}|$, the number of 8-codes added to the chain **c**. Case (a) in property 3.12 states that the $F(i)$-th code of the chain $\mathbf{d} = add(\mathbf{c}, f)$ is $\mathbf{c}_i$, for $i \in [0\,..\,|\mathbf{c}|]$; this property may be simply checked by using the definitions of $add8$ and F. Since, all other codes in **d** are 8-codes, case (b) also holds.

3.12 **Property :**

For all chains **c**, distribution functions $f \in \mathcal{D}_{|\mathbf{c}|}$, and all $x \in [0, |\mathbf{c}|]$ with $F(x) \in \mathbb{N}_0$ the following holds for the chain $\mathbf{d} = add8(\mathbf{c}, f)$.

(a) $x \in \mathbb{Z} \Rightarrow \mathbf{d}(F(x)) = \mathbf{c}(x)$.

(b) $x \notin \mathbb{Z} \Rightarrow \mathbf{d}(F(x)) = 8$

□

The following property is the equivalent for $add8$ of the property 3.6 for the operator $\underline{w}$. It states the relation between the pixels in the pixel set of $add8(\mathbf{c})$ and those in the pixel set of **c** itself.

3.13 **Property :**

For all chains **c**, distribution functions $f \in \mathcal{D}_{|\mathbf{c}|}$, and $t \in [0\,..\,F(|\mathbf{c}|)]$

$$\mathbf{p}(add8(\mathbf{c}, f), t) = \mathbf{p}(\mathbf{c}, \lceil F^{-1}(t) \rceil).$$

Proof

$\mathbf{p}(add\,8(\mathbf{c}, f), t)$
$= \{$ definition 1.21 of $\mathbf{p}$ $\}$
$(\Sigma i : i \in [0..t) : v(add\,8(\mathbf{c}, f)(i)))$
$= \{$ F is monotonous and continuous, and $t \in [0..F(|\mathbf{c}|))$ $\}$
$(\Sigma x : x \in [-1, |\mathbf{c}|] \wedge F(x) \in [0..t) : v(add\,8(\mathbf{c}, f)(F(x))))$
$= \{$ property 3.12 (a) and (b) and $F(-1) = -1$ $\}$
$(\Sigma x : x \in [0..|\mathbf{c}|] \wedge F(x) \in [0..t) : v(\mathbf{c}_x))$
$+$
$(\Sigma x : x \in [0, |\mathbf{c}|]/\mathbb{Z} \wedge F(x) \in [0..t) : v(8))$
$= \{$ $v(8) = (0,0)$ $\}$
$(\Sigma i : i \in [0..|\mathbf{c}|] \wedge F(i) \in [0..t) : v(\mathbf{c}_i))$
$= \{$ F is monotone increasing $\}$
$(\Sigma i : i \in [0..|\mathbf{c}|] \wedge F(i) \in [F(0)..t) : v(\mathbf{c}_i))$
$= \{$ definition 3.10: F is monotonous and continuous $\}$
$(\Sigma i : i \in [0..|\mathbf{c}|] \wedge i \in [0..F^{-1}(t)) : v(\mathbf{c}_i))$
$= \{$ $F^{-1}(t) \leq F^{-1}(F(|\mathbf{c}|)) = |\mathbf{c}|$ $\}$
$(\Sigma i : i \in [0..F^{-1}(t)) : v(\mathbf{c}_i))$
$= \{$ definition 1.21 of $\mathbf{p}$ $\}$
$\mathbf{p}(\mathbf{c}, \lceil F^{-1}(t) \rceil)$

□

We conclude this section with a property stating two inequalities for primitive functions of the same distribution function and one inequality for the inverse of such primitive functions.

3.14 **Property :**

For all $n \in \mathbb{N}_0$, all functions $f \in \mathcal{D}_n$ and for all primitive functions F_0 and F_1 of f.

(a) $(\forall a, b : a, b \in [-1..n] : /F_0(a) - F_1(b)/ \geq /a - b/)$

(b) $(\forall x, y : x, y \in [0, n] : /F_0(x) - F_1(y)/ \geq /x - y/ - 2)$

(c) $(\forall x, y : x, y \in [0, F_0(n)] : /F_0^{-1}(x) - F_1^{-1}(y)/ \leq /x - y/ + 2)$

Proof

ad (a):

Suppose $a \leq b$.
$/F_0(a) - F_1(b)/$
$= \{$ $F_0(i) = F_1(i)$, for all $i \in [-1..n]$ $\}$
$/F_0(a) - F_0(b)/$
$= \{$ property 3.11(a) and $a \leq b$ $\}$

$$
\begin{array}{l}
/b-a+(\Sigma j : a<j\leq b : f(j))/ \\
\geq \{\ f(j)\geq 0 \text{ and } a\leq b\ \} \\
/b-a/
\end{array}
$$

ad (b):

case $\lfloor x\rfloor \geq \lceil y\rceil$

$$
\begin{array}{l}
/F(y)-F(x)/ \\
= \{\ x\geq y \text{ and } F \text{ is increasing}\ \} \\
F(x)-F(y) \\
\geq \{\ \text{F is increasing}\ \} \\
F(\lfloor x\rfloor)-F(\lceil y\rceil) \\
\geq \{\ \text{property 3.14 (a) and } \lfloor x\rfloor \geq \lceil y\rceil\ \} \\
\lfloor x\rfloor -\lceil y\rceil \\
\geq \{\ /x-y/\leq /x-\lfloor x\rfloor /+/y-\lceil y\rceil /+/\lfloor x\rfloor +\lceil y\rceil /\leq /\lfloor x\rfloor +\lceil y\rceil /+2\ \} \\
/x-y/-2
\end{array}
$$

case $\lfloor y\rfloor \geq \lceil x\rceil$: similar.

case $\lfloor y\rfloor < \lceil x\rceil \wedge \lfloor y\rfloor < \lceil x\rceil$

$$
\begin{array}{l}
/F(y)-F(x)/ \\
\geq \{\ \text{calculus}\ \} \\
0 \\
\geq \{\ /x-y/<1\ \} \\
/x-y/-2
\end{array}
$$

ad (c):

(c) follows directly from (b) after substituting $F_0^{-1}(x)$ and $F_1^{-1}(y)$ for respectively x and y.

□

3.3 W-curves

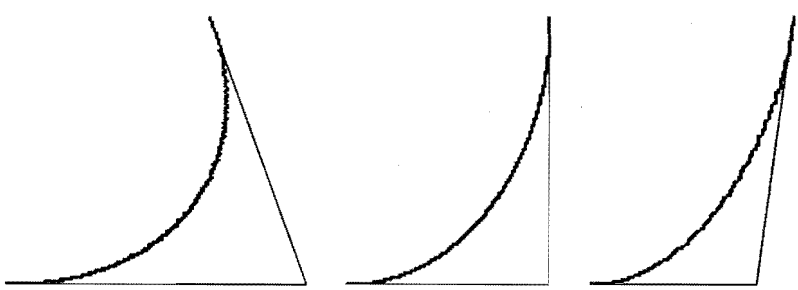
In this section we use the two operators *add*8 and $\underline{w}$, introduced in the previous section, for constructing the so called w(eave)-curve: a discrete curve given by 3 control points and two distribution functions.

3.15 **Definition :** w-curve

For all points $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$ and distribution functions $f \in \mathcal{D}_{|\mathbf{c}|}$ and $g \in \mathcal{D}_{|\mathbf{d}|}$, with $\mathbf{c} = bresh8(\mathbf{p}_1 - \mathbf{p}_0)$ and $\mathbf{d} = bresh8(\mathbf{p}_2 - \mathbf{p}_1)$, we define the w-curve $W(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f, g)$ as the discrete curve $DC(\mathbf{p}, \mathbf{e})$ with $\mathbf{p}$ and $\mathbf{e}$ given by

$$\mathbf{p} = \mathbf{p}_0$$
$$\mathbf{e} = add8(\mathbf{c}, f)\,\underline{w}\,add8(\mathbf{d}, g).$$

□



(a) (b) (c)

Figure 3.5 : w-curves of example 3.16

3.16 **Example :**

(a) $f : [0..6] \to \mathbb{N}_o \quad f(i) = i$
$g : [0..6] \to \mathbb{N}_o \quad g(i) = 6 - i$
$\mathbf{p}_0 = (0, 0)$, $\mathbf{p}_1 = (6, 0)$, and $\mathbf{p}_2 = (0, 6)$.
$\mathbf{c} = [\,000000\,]$ and $\mathbf{d} = [\,222222\,]$.
$\bar{\mathbf{c}} := add8(\mathbf{c}, f) = [\,080808088088880888880888888\,]$,
$\bar{\mathbf{d}} := add8(\mathbf{d}, g) = [\,888882888882888828882888288282\,]$
$bresh4(|\bar{\mathbf{c}}|, |\bar{\mathbf{d}}|) = bresh4(27,27) = [\,02\,]^{27}$;

Hence, in this case weaving consists of taking codes alternately from the chains $\bar{\mathbf{c}}$ and $\bar{\mathbf{d}}$, and $\bar{\mathbf{c}}\,\underline{w}\,\bar{\mathbf{d}}$ can be written as

[088808888808828888088888828808888828888088288888828882]

The 8-codes in $\bar{\mathbf{c}} \underline{w} \bar{\mathbf{d}}$ do not contribute to its geometric interpretation and hence the chain **e** of the w-curve can also be given by

$$\mathbf{e} = [\,000202020222\,]$$

(b) Figure 3.5 shows 3 w-curves with as distribution functions

$f: [0..n] \rightarrow \mathbb{N}_0$ with $f(i)=i$
and $g: [0..m] \rightarrow \mathbb{N}_0$ with $g(i)=m-i$
for suitable n and m.

The control points $\mathbf{p}_0$ and $\mathbf{p}_2$ are given for all cases by $\mathbf{p}_0=(0, 0)$ and $\mathbf{p}_2=(92, 121)$. For the cases (a), (b), and (c) $\mathbf{p}_1$ has the values (92, 0), (75, 0), and (136, 0), respectively.

□

The following property relates the pixel set of a w-curve to the pixel sets of its control chains and to the primitives of their distribution functions. Note that the property is stated for general chains and not only for control chains. In the proof of the property the similar properties 3.6 and 3.13 for the operators $\underline{w}$ and *add*8, respectively, are used.

3.17 **Property :**

For all chains **c** and **d**, and for all distribution functions $f \in \mathcal{D}_{|\mathbf{c}|}$ and $g \in \mathcal{D}_{|\mathbf{d}|}$, the points on the chain

$$\mathbf{e} = add8(\mathbf{c}, f) \,\underline{w}\, add8(\mathbf{d}, g)$$

are given for $t \in [0..|\mathbf{e}|]$ by

$$\mathbf{p}(\mathbf{e}, t) = \mathbf{p}(\mathbf{c}, \lceil F^{-1}(t - \lfloor \tfrac{|\bar{\mathbf{d}}|}{|\bar{\mathbf{c}}|+|\bar{\mathbf{d}}|} t \rfloor) \rceil) + \mathbf{p}(\mathbf{d}, \lceil G^{-1}(\lfloor \tfrac{|\bar{\mathbf{d}}|}{|\bar{\mathbf{c}}|+|\bar{\mathbf{d}}|} t \rfloor) \rceil)$$

with $\bar{\mathbf{c}} = add8(\mathbf{c}, f)$ and $\bar{\mathbf{d}} = add8(\mathbf{d}, g)$.

Proof

We define the short hands $\sigma := \lfloor \tfrac{|\bar{\mathbf{d}}|}{|\bar{\mathbf{c}}|+|\bar{\mathbf{d}}|} t \rfloor$ and $\tau := t - \sigma$.

$\mathbf{p}(\mathbf{e}, t)$
= { definition of **e**, $\bar{\mathbf{c}}$, and $\bar{\mathbf{d}}$ }
$\mathbf{p}(\bar{\mathbf{c}} \underline{w} \bar{\mathbf{d}}, t)$
= { property 3.6 and definition of τ and σ }
$\mathbf{p}(\bar{\mathbf{c}}, \tau) + \mathbf{p}(\bar{\mathbf{d}}, \sigma)$
= { property 3.13, definition of $\bar{\mathbf{c}}$ and $\bar{\mathbf{d}}$ }
$\mathbf{p}(\mathbf{c}, \lceil F^{-1}(\tau) \rceil) + \mathbf{p}(\mathbf{d}, \lceil G^{-1}(\sigma) \rceil)$

□

If **c** and **d** in the above property are control chains on the points p_0, p_1, and p_2, $\mathbf{p}(\mathbf{c}, t)$ and $\mathbf{p}(\mathbf{d}, t)$ can be written , according to property 2.6, as

$$\mathbf{p}(\mathbf{c}, t)=\left[\frac{t}{|\mathbf{c}|}(p_1-p_0)\right] \wedge \mathbf{p}(\mathbf{d}, t)=\left[\frac{t}{|\mathbf{d}|}(p_2-p_1)\right]$$

for $t \in [0..|\mathbf{c}|]$ and $t \in [0..|\mathbf{d}|]$, respectively. Hence, property 3.17 may be rewritten as follows.

$$\mathbf{p}(\mathbf{e}, t)=\left[\frac{\lceil F^{-1}(\tau)\rceil}{|\mathbf{c}|}(p_1-p_0)\right]+\left[\frac{\lceil G^{-1}(\sigma)\rceil}{|\mathbf{d}|}(p_2-p_1)\right]$$

with τ and σ as in the proof of the property. Leaving out the ceiling and rounding operators, the following approximation of $\mathbf{p}(\mathbf{e}, t)$ may be given.

$$\mathbf{p}(\mathbf{e}, t) \approx \frac{F^{-1}((1-\alpha)t)}{|\mathbf{c}|}(p_1-p_0)+\frac{G^{-1}(\alpha t)}{|\mathbf{d}|}(p_2-p_1)$$

with $\alpha=\frac{|\bar{\mathbf{d}}|}{|\bar{\mathbf{c}}|+|\bar{\mathbf{d}}|}$. Clearly, $\frac{F^{-1}((1-\alpha)t)}{|\mathbf{c}|}$ and $\frac{G^{-1}(\alpha t)}{|\mathbf{d}|}$ are blending functions in this approximation. Consequently, $\frac{\lceil F^{-1}(\tau)\rceil}{|\mathbf{c}|}$ and $\frac{\lceil G^{-1}(\sigma)\rceil}{|\mathbf{d}|}$ can be seen as the discrete equivalents of blending functions.

3.4. Smoothing of w-curves

W-curves, as shown for instance in figure 3.5, appear unsmooth. Here two methods are given to improve this, namely

- local smoothing
- global smoothing.

These two methods are fundamentally different. In the first method the chain of the w-curves are filtered in order to appear smoother. In the second method the curve is computed more accurately to obtain the same objective. We clarify these two methods in the following two sections.

3.4.0 Local smoothing

A chain makes an unsmooth impression if two consecutive codes have basic vectors which differ too much. As a result of weaving two control chains, unsmooth combinations of codes can easily occur, since the two control chains can consist of a maximum of four different codes.
Below we give both for 4- and 8-connectedness a definition of a smooth chain. We only allow minimal changes in basic vectors of consecutive codes of a smooth chain. Although 8-codes do not change the appearance of a chain, we require, for reasons of simplicity, that a smooth chain contains no 8-codes. Notice that the definition for a smooth chain as given below depends on the more or less arbitrary choice for the

encoding of the directions.

3.18 **Definition** : smooth chain

A 4-connected chain c is called (4-)smooth if it contains no 8-codes and

$$(\forall i : i \in (0..|c|) : (c_i - c_{i-1}) \bmod 4 \in \{0, 2\})$$

An 8-connected chain c is called (8-)smooth if it contains no 8-codes and

$$(\forall i : i \in (0..|c|) : (c_i - c_{i-1}) \bmod 8 \in \{0, 1, 7\})$$

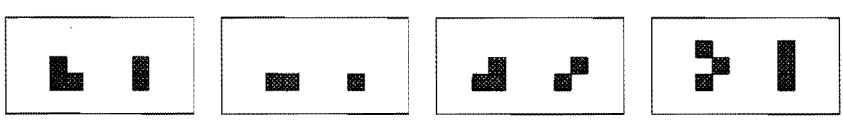
□

The 4-connected and the 8-connected chains resulting from Bresenham's algorithms as given in chapter 2 are smooth chains. The chains of w-curves are, in general, unsmooth; even if the 8-codes are removed.

In the local smoothing method pairs of consecutive codes in the chain which are unsmooth, are replaced by Bresenham chains that have the same end vector as the original codes. This results, for 8-connected chains, in the following conversion table for pairs of codes; notice that the order of the codes is not important. A similar table can be made for 4-connected chains.

[02]→[1]	[13]→[22]	[24]→[3]	[35]→[44]	[46]→[5]	[57]→[66]
[03]→[2]	[14]→[2]	[25]→[4]	[36]→[4]	[47]→[6]	
[04]→ε	[15]→ε	[26]→ε	[37]→ε		
[05]→[6]	[16]→[0]	[27]→[0]			
[06]→[7]	[17]→[00]				

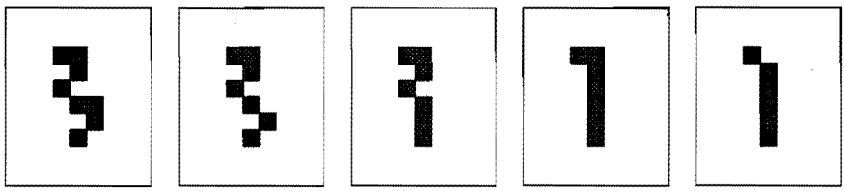
In figure 3.6 four of these conversions are illustrated.



(a) (b) (c) (d)

Figure 3.6: 8-connected local smoothing
(a) [03]→[2] (b) [04]→ε
(c) [02]→[1] (d) [13]→[22]

Replacing, using this table, each time the left most unsmooth pair of codes, the final chain becomes a smooth chain and has the same end point as the original chain. This method ends, since after every step the chain becomes shorter; that is, if the length of a chain is measured as the sum of the Euclidean length of the basic vectors. In [Fre61] Freeman uses a similar method for obtaining a chain with minimal Euclidean length between the end points. In figure 3.7 an example is given. The main disadvantage of this method is, that it is a local method and hence, not related



(a) (b) (c) (d) (e)

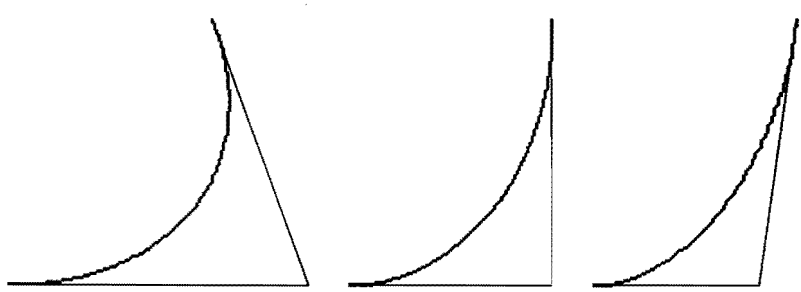
Figure 3.7: steps in the process of 8-connected local smoothing
(a) [1243124] (b) [133124] (c) [223124]
(d) [222224] (e) [22223]

to the global shape of the curve. The deviation of the resulting curve and the original curve may be large, as can be seen from the fact that the chain [22226666] reduces to the empty chain after applying this method. Another disadvantage of the method is that it destroys the symmetry of the curve since it makes a difference whether the curve is smoothed from left to right or the other way around.

3.19 **Example :**

In figure 3.8 the curves of figure 3.5 are shown after local smoothing.

□



(a) (b) (c)

Figure 3.8: local-smoothed w-curves of example 3.19

3.4.1 Global smoothing

Before explaining global smoothing we define subclasses of w-curves. A subclass is a set of w-curves which contains exactly one w-curve for each triple of control points. Moreover, the distribution functions for the w-curves in a subclass are restricted to distribution functions in a so called complete set of distribution functions. Below a complete set is defined by means of $\mathcal{D}_n$, the set of distribution functions with domain $[0..n]$ (see definition 3.7).

3.20 **Definition** : complete set (of distribution functions)

A complete set S (of distributions functions) is a set of distributions functions containing for each $n \in \mathbb{N}_0$ exactly one element of $\mathcal{D}_n$.

□

For complete sets names like Sf and Sg are used. The elements of $\mathcal{D}_n$ ($n \in \mathbb{N}_0$) in these sets are called f_n and g_n, respectively. The corresponding primitive functions are called F_n and G_n.
A subclass may now be defined as follows.

3.21 **Definition** : subclass

For all complete sets Sf and Sg the subclass $W(Sf, Sg)$ of w-curves is defined by

$$W(Sf, Sg) := \{W(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f, g) \mid \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \in \mathbb{Z}^2 \wedge f \in Sf \cap \mathcal{D}_n \wedge g \in Sg \cap \mathcal{D}_m\}.$$

with the abbreviations n and m given by $n = D_8(\mathbf{p}_1 - \mathbf{p}_0)$ and $m = D_8(\mathbf{p}_2 - \mathbf{p}_1)$.

□

Global smoothing of a w-curve $W_0 = W(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f, g)$ in the subclass $W(Sf, Sg)$ consists of two steps:

(a) compute, for some integer factor k, the w-curve $W_1 = W(k\mathbf{p}_0, k\mathbf{p}_1, k\mathbf{p}_2, \bar{f}, \bar{g})$, where $\bar{f}$ and $\bar{g}$ are the appropriate distribution functions in Sf and Sg, respectively.

(b) scale W_1 to the size of W_0, using the transformation algorithms as given in chapter two.

Within a subclass $W(Sf, Sg)$ we can define, for all $k \in \mathbb{Z}$ the smoothed discrete curve $C_k(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg)$ of a w-curve $W(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f, g)$ by

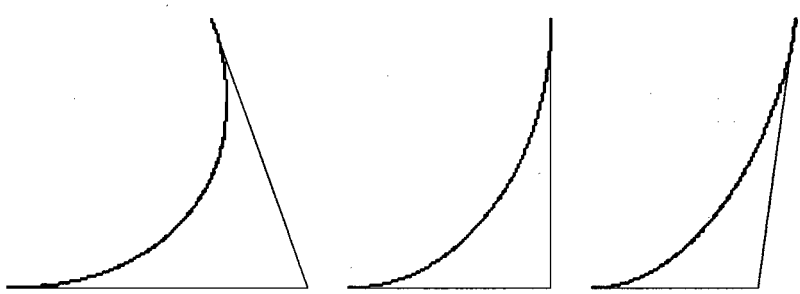
$$C_k(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg) := [A_k(D_k)] \text{ with } A_k = \begin{bmatrix} k^{-1} & 0 \\ 0 & k^{-1} \end{bmatrix}$$
and $D_k = W(k\mathbf{p}_0, k\mathbf{p}_1, k\mathbf{p}_2, f, g)$

where f and g are the appropriate functions from Sf and Sg.

3.22 **Example :**

In figure 3.9 the curves of figure 3.5 are shown after global smoothing with factor $k=3$. The subclass Sf and Sg contain the distribution functions f_n, $g_n \in D_n$ with $f_n(i)=i$ and $g_n(i)=n-i$.

□



(a) (b) (c)

Figure 3.9: global-smoothed w-curves of example 3.22

Notice that global smoothing does not result in smooth chains as defined in definition 3.18. Hence, it can be useful to apply first global smoothing and afterwards local smoothing.

It is unclear how the complete sets Sf and Sg have to be chosen. In the next chapter the global smoothing of this section is used to construct a continuous curve from a subclass of w-curves. A minimal requirement for the complete set of functions Sf and Sg is the convergence of the limit process that results in the continuous curve.

3.5 W-curves : the continuous case

In this section the notions of continuity, convergence, uniform continuity, and uniform convergence are used; as a consequence especially the proofs are rather technical. For an explanation of these notions the reader is referred to a text book on mathematical analysis, e.g. [Cou89] or [Rud76] .

The definition of w-curves is based on a discrete representation of its control lines by means of chains. In this paragraph a continuous counterpart of w-curves is obtained by partitioning the control lines in infinitely small vectors. In terms of the paragraph 3.4.1, the continuous w-curve within a given subclass $W(Sf,Sg)$ is given by

$$C(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg)=\lim_{k\to\infty} A_k(D_k)$$

First of all the limit of a sequence of chains is defined. We can give for each chain $\mathbf{c}$ a function in $[0,1]\to\mathbb{Z}^2$, say P, such that $P(\mathbf{c};\tau)$ is the sum of a fraction τ of the basic vectors of $\mathbf{c}$. P is defined by

$$P(\mathbf{c};\tau):=\mathbf{p}(\mathbf{c}, \lfloor \tau \mid \mathbf{c} \mid \rfloor)$$

and is a parameterisation of a discrete function but with a continuous parameter τ. Consider a sequence of chains $(\mathbf{c}_k)_{k\in\mathbb{N}}$ with $end(\mathbf{c}_k)=k\,end(\mathbf{c}_1)$, for $k\in\mathbb{N}_1$ and suppose that the chains $\mathbf{c}_k$ represent the same curve but at different sizes. Now, by taking the limit of the functions $\frac{1}{k}P(\mathbf{c}_k;\tau)$, we obtain the continuous version of the curve represented by the chains $\mathbf{c}_k$. In the sequel we assume that the sequences of chains are chosen such that their limit as defined in the next definition, exists.

3.23 **Definition :** limit of a sequence of chains

The limit of a sequence of chains $(\mathbf{c}_k)_{k\in\mathbb{N}}$ with

$$end(\mathbf{c}_k)=k\,end(\mathbf{c}_1)$$

is a function $L:[0,1]\to\mathbb{R}^2$ given by

$$L(\tau):=\lim_{k\to\infty}\frac{1}{k}\mathbf{p}(\mathbf{c}_k, \lfloor \tau \mid \mathbf{c}_k \mid \rfloor).$$

□

Instead of saying that the limit of a sequence of chains is L, we just say that the sequence converges to L. The limit L defined in this definition is not necessarily continuous. In case of uniform convergence on $[0,1]$, however, L is continuous on $[0,1]$.

3.24 **Property :**

If a sequence of chains $(\mathbf{c}_k)_{k\in\mathbb{N}}$ converges uniformly to $L(\tau)$ for $\tau\in[0,1]$, L is continuous on $[0,1]$.

Proof

This property follows from the following:

(a) The limit of a uniform convergent sequence of continuous functions on $[0,1]$ is continuous.

(b) For a functions Q, defined for a chain $\mathbf{c}$ and $\tau\in[0,1]$ by

$$Q(\mathbf{c};\tau):=\mathbf{p}(\mathbf{c},i)+(\tau-\frac{i}{|\mathbf{c}|})(\mathbf{p}(\mathbf{c},i+1)-\mathbf{p}(\mathbf{c},i)) \text{ for } \tau\in[\frac{i}{|\mathbf{c}|},\frac{i+1}{|\mathbf{c}|}],$$

it holds that $Q(\mathbf{c}_k;\tau)$ converges uniformly to $L(\tau)$ for $k\to\infty$.

(c) The functions $Q(\mathbf{c}_k;\tau)$ are continuous on $[0,1]$.

□

For $\tau=0$ and $\tau=1$, $L(\tau)$ may be simply computed:

$L(0)=(0,0)$
$L(1)=end(\mathbf{c}_1)$.

Using definition 3.23 we can give the following property concerning the limit of a sequence of chains, where these chains are the result of weaving two chains with 8-codes added according to distribution functions in two given complete sets. This property is used to compute a continuous counterpart of w-curves.

3.25 **Property :**

(a) Let $(\mathbf{c}_k)_{k\in\mathbb{N}}$ and $(\mathbf{d}_k)_{k\in\mathbb{N}}$ be sequences of chains with

$$end(\mathbf{c}_k)=k\,end(\mathbf{c}_1)$$
$$end(\mathbf{d}_k)=k\,end(\mathbf{d}_1)$$

and let these sequences converge uniformly to the functions C_0 and C_1, respectively. Let Sf and Sg be complete sets.
Then the sequence of chains $(\mathbf{e}_k)_{k\in\mathbb{N}}$ given by

$$\mathbf{e}_k=add8(\mathbf{c}_k,f_{|\mathbf{c}_k|})\,\underline{w}\,add8(\mathbf{d}_k,g_{|\mathbf{d}_k|}),$$

converges to a function C in $[0,1]\to\mathbb{R}^2$ given by

$$C(\tau)=C_0(\mathcal{F}(\tau))+C_1(\mathcal{G}(\tau))$$

with the blending functions $\mathcal{F}(\tau)$ and $\mathcal{G}(\tau)$ defined by

$$\mathcal{F}(\tau):=\lim_{k\to\infty}\frac{F_k^{-1}(F_k(k)\tau)}{k} \text{ and } \mathcal{G}(\tau):=\lim_{k\to\infty}\frac{G_k^{-1}(G_k(k)\tau)}{k}.$$

(b) The sequence of chains $(\mathbf{e}_k)_{k\in\mathbb{N}}$ converges uniformly for $\tau\in[0,1]$ to $C(\tau)$ if the limits for the blending functions $\mathcal{F}$ and $\mathcal{G}$ converge uniformly on this interval.

Proof

(a) From property 3.17 we can see that the relation between the points of $\mathbf{c}_k$, $\mathbf{d}_k$, and $\mathbf{e}_k$ can be given as follows:

$$\mathbf{p}(\mathbf{e}_k, [\tau\,|\,\mathbf{e}_k\,|]) = \mathbf{p}(\mathbf{c}_k, t_k) + \mathbf{p}(\mathbf{d}_k, s_k)$$

with t_k and s_k given by

$$t_k := \lceil F^{-1}_{|\mathbf{c}_k|}(t - [\alpha_k[\tau\,|\,\mathbf{e}_k\,|]]) \rceil$$
$$s_k := \lceil G^{-1}_{|\mathbf{d}_k|}([\alpha_k[\tau\,|\,\mathbf{e}_k\,|]]) \rceil$$

and α_k, $\overline{\mathbf{c}_k}$, and $\overline{\mathbf{d}_k}$ by

$$\alpha_k = \frac{|\overline{\mathbf{d}_k}|}{|\overline{\mathbf{c}_k}| + |\overline{\mathbf{d}_k}|} = \frac{G_{|\mathbf{d}_k|}(|\mathbf{d}_k|)}{|\mathbf{e}_k|}$$
$$\overline{\mathbf{c}_k} = add\,8(\mathbf{c}_k, f_k)$$
$$\overline{\mathbf{d}_k} = add\,8(\mathbf{d}_k, g_k).$$

In the sequel only the second term $\mathbf{p}(\mathbf{d}_k, s_k)$ is considered; the other term can be handled in a similar way.
First we define σ_k by

$$\sigma_k := \frac{G^{-1}_{|\mathbf{d}_k|}(\tau G_{|\mathbf{d}_k|}(|\mathbf{d}_k|))}{|\mathbf{d}_k|}$$

Given this definition of σ_k we can now state the following.

$$\begin{aligned}
&\tfrac{1}{k}/\,\mathbf{p}(\mathbf{d}_k, s_k) - \mathbf{p}(\mathbf{d}_k, [\mathcal{G}(\tau)\,|\,\mathbf{d}_k\,|])/ \\
&\leq \{\text{ triangle inequality }\} \\
&\tfrac{1}{k}/\,\mathbf{p}(\mathbf{d}_k, s_k) - \mathbf{p}(\mathbf{d}_k, [\sigma_k\,|\,\mathbf{d}_k\,|])/ \\
&+ \\
&\tfrac{1}{k}/\,\mathbf{p}(\mathbf{d}_k, [\sigma_k\,|\,\mathbf{d}_k\,|]) - \mathbf{p}(\mathbf{d}_k, [\mathcal{G}(\tau)\,|\,\mathbf{d}_k\,|])/
\end{aligned}$$

This result and the two lemmas 3.26 and 3.27, stated and proved below, result in

$$\lim_{k\to\infty} \frac{1}{k}\,\mathbf{p}(\mathbf{d}_k, s_k) - \frac{1}{k}\,\mathbf{p}(\mathbf{d}_k, [\mathcal{G}(\tau)\,|\,\mathbf{d}_k\,|]) = (0,0).$$

From the (uniform) convergence of the sequence $(\mathbf{d}_k)_{k\in\mathbb{N}}$ to C_1 it follows that

$$\lim_{k\to\infty} \frac{1}{k}\,\mathbf{p}(\mathbf{d}_k, [\mathcal{G}(\tau)\,|\,\mathbf{d}_k\,|]) = C_1(\mathcal{G}(\tau)).$$

Hence,

$$\lim_{k\to\infty}\frac{1}{k}\,\mathbf{p}(\mathbf{d}_k, s_k)=C_1(\mathcal{G}(\tau)).$$

A similar result holds for $(\mathbf{c}_k)_{k\in\mathbb{N}}$ and the proof of (a) is concluded as follows.

$$\begin{aligned}
&C(\tau)\\
&=\\
&\lim_{k\to\infty}\frac{1}{k}\,\mathbf{p}(\mathbf{e}_k, [\tau|\mathbf{e}_k|])\\
&=\\
&\lim_{k\to\infty}\frac{1}{k}\,\mathbf{p}(\mathbf{c}_k, t_k)+\lim_{k\to\infty}\frac{1}{k}\,\mathbf{p}(\mathbf{d}_k, s_k)\\
&=\\
&C_0(\mathcal{F}(\tau))+C_1(\mathcal{G}(\tau))
\end{aligned}$$

(b) From the uniform convergence of the sequence $(\mathbf{d}_k)_{k\in\mathbb{N}}$ and the uniform convergence of the limits in the lemma's, it follows that

$$\lim_{k\to\infty}\frac{1}{k}\,\mathbf{p}(\mathbf{d}_k, s_k)=C_1(\mathcal{G}(\tau)).$$

has uniform convergence. A similar result holds for $(\mathbf{c}_k)_{k\in\mathbb{N}}$ and hence, the sequence $(\mathbf{e}_k)_{k\in\mathbb{N}}$ converges uniformly to $C(\tau)$.

□

3.26 **Lemma :**

(a) $\lim_{k\to\infty}\frac{1}{k}/\,\mathbf{p}(\mathbf{d}_k, s_k)-\mathbf{p}(\mathbf{d}_k, [\sigma_k|\mathbf{d}_k|])/=(0,0)$

(b) The limit in (a) converges uniformly for $\tau\in[0,1]$.

Proof

$$\begin{aligned}
&/\mathbf{p}(\mathbf{d}_k, s_k)-\mathbf{p}(\mathbf{d}_k, [\sigma_k|\mathbf{d}_k|])/\\
&\leq\\
&/s_k-[\sigma_k|\mathbf{d}_k|]/\,(1,1)\\
&\leq \{\text{ definitions of } s_k \text{ and } \sigma_k \text{ and property 1.14 of ceiling }\}\\
&/G^{-1}_{|\mathbf{d}_k|}([\alpha_k[\tau|\mathbf{e}_k|]])-G^{-1}_{|\mathbf{d}_k|}(\tau G_{|\mathbf{d}_k|}(|\mathbf{d}_k|))/\,(1,1)+(3/2,3/2)\\
&\leq \{\text{ property 3.14(c) of } G^{-1}_{|\mathbf{d}_k|}\ \}\\
&/[\alpha_k[\tau|\mathbf{e}_k|]]-\tau G_{|\mathbf{d}_k|}(|\mathbf{d}_k|)/\,(1,1)+(5/2,5/2)\\
&\leq \{\text{ definition of } \alpha_k \text{ and property 1.14 of entier }\}\\
&(7/2,7/2)
\end{aligned}$$

□

3.27 **Lemma :**

(a) $\lim_{k\to\infty} \frac{1}{k} / \mathrm{p}(\mathrm{d}_k, \lceil \sigma_k \mid \mathrm{d}_k \mid \rceil) - \mathrm{p}(\mathrm{d}_k, \lceil \mathcal{G}(\tau) \mid \mathrm{d}_k \mid \rceil) / = (0,0)$

(b) The limit in (a) converges uniformly if σ_k converges uniformly to $\mathcal{G}(\tau)$ on the interval $[0, 1]$.

Proof

Ad (a).
Let $\varepsilon > 0$.

• According to the uniform convergence of the sequence of chains $(\mathrm{d}_k)_{k \in \mathbb{N}}$ (see definition 3.23) an integer K_1 exists such that for all $k > K_1$ and for all $\tau \in [0, 1]$

$$/ \frac{1}{k} \mathrm{p}(\mathrm{d}_k, \lceil \tau \mid \mathrm{d}_k \mid \rceil) - C_1(\tau) / < \varepsilon(1, 1)$$

• $(\mathrm{d}_k)_{k \in \mathbb{N}}$ converges uniformly to C_1. Consequently, according to property 3.24, C_1 is continuous on $[0, 1]$. Moreover, C_1 is uniform continuous on $[0, 1]$, since on a closed and bounded interval uniform continuous and continuous are equivalent. Hence, a $\delta > 0$ exists such that for all τ_1, τ_2 in $[0, 1]$ the following holds.

$$/ \tau_1 - \tau_2 / < \delta \Rightarrow / C_1(\tau_1) - C_1(\tau_2) / < \varepsilon(1, 1)$$

• σ_k converges to $\mathcal{G}(\tau)$; hence, an integer K_2 exists such that for all $k > K_2$ holds.

$$/ \mathcal{G}(\tau) - \sigma_k / < \delta$$

Using uniform convergence, continuity and convergence results for all $k > \max(K_1, K_2)$ in :

$$\begin{aligned}
&\frac{1}{k} / \mathrm{p}(\mathrm{d}_k, \lceil \sigma_k \mid \mathrm{d}_k \mid \rceil) - \mathrm{p}(\mathrm{d}_k, \lceil \mathcal{G}(\tau) \mid \mathrm{d}_k \mid \rceil) / \\
&\leq \\
&/ \frac{1}{k} \mathrm{p}(\mathrm{d}_k, \lceil \sigma_k \mid \mathrm{d}_k \mid \rceil) - C_1(\sigma_k) / \\
&+ / C_1(\mathcal{G}(\tau)) - \frac{1}{k} \mathrm{p}(\mathrm{d}_k, \lceil \mathcal{G}(\tau) \mid \mathrm{d}_k \mid \rceil) / \\
&+ / C_1(\mathcal{G}(\tau)) - C_1(\sigma_k) / \\
&\leq \{ \text{ uniform convergence } \} \\
&2\varepsilon(1, 1) + / C_1(\mathcal{G}(\tau)) - C_1(\sigma_k) / \\
&\leq \{ \text{ continuity and limit of } \sigma_k \} \\
&3\varepsilon(1, 1)
\end{aligned}$$

Hence, (a) holds.

Ad (b).

If K_1 and K_2 in the proof of (a) may be chosen independent of τ, (b) holds. K_1 is independent of τ; K_2 may be chosen independent of τ since σ_k converges uniformly to $\mathcal{G}(\tau)$ for $\tau \in [0, 1]$.

Hence, (b) holds.

□

If we apply property 3.25 for w-curves we choose the chains $\mathbf{c_k}$ and $\mathbf{d_k}$, for $k \in \mathbb{N}_1$ as the following Bresenham chains.

$$\mathbf{c_k} = bresh\,8(k(\mathbf{p}_1 - \mathbf{p}_0))$$
$$\mathbf{d_k} = bresh\,8(k(\mathbf{p}_2 - \mathbf{p}_1)).$$

The limits of these sequences of chains are given by

$$C_0(\tau) = \tau(\mathbf{p}_1 - \mathbf{p}_0)$$
$$C_1(\tau) = \tau(\mathbf{p}_2 - \mathbf{p}_1).$$

as can be seen from property 2.6 and definition 3.23. These observations lead to the following definition for a continuous version of w-curves.

3.28 **Definition :** CW

The function $CW(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg)$ is defined for all points $\mathbf{p}_i$ $(i \in [0..2])$ and all complete sets Sf and Sg by

$$CW(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg) := \mathbf{p}_0 + \mathcal{F}(\tau)(\mathbf{p}_1 - \mathbf{p}_0) + \mathcal{G}(\tau)(\mathbf{p}_2 - \mathbf{p}_1),$$

for all $\tau \in [0, 1]$.

□

A *CW* curve is of the form of property 1.4. Hence, the following property holds.

3.29 **Property :**

The curves $CW(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg)$ $(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \in \mathbb{Z}^2)$ are affine invariant, for all sets of complete functions Sf and Sg.

□

Conditions for the continuity of *CW* follow from the properties 3.24 and 3.25 and

result in the following property.

3.30 **Property :**

The curves $CW(p_0, p_1, p_2, Sf, Sg)$ $(p_0, p_1, p_2 \in \mathbb{Z}^2)$ are continuous on $[0, 1]$, if

$$\frac{F_k^{-1}(F_k(k)\tau)}{k} \text{ and } \frac{G_k^{-1}(G_k(k)\tau)}{k}$$

converge uniformly on $[0, 1]$ for $k \to \infty$ (to $\mathcal{F}(\tau)$ and $\mathcal{G}(\tau)$, respectively).

□

3.5.1 Blending functions

The functions $\mathcal{F}$ and $\mathcal{G}$ of property 3.25 are the blending functions as introduced in section 1.1. In this section some properties and definitions concerning these functions are given.

3.31 **Property :**

For the blending functions $\mathcal{F}$ and $\mathcal{G}$ of property 3.25 the following holds.

(a) $\mathcal{F}(0) = \mathcal{G}(0) = 0$
(b) $\mathcal{F}(1) = \mathcal{G}(1) = 1$
(c) $\mathcal{F}$ and $\mathcal{G}$ are both monotonous functions on $[0, 1]$.

Proof

(a) $$\mathcal{F}(0) = \lim_{k \to \infty} \frac{F_k^{-1}(0)}{k} = 0$$

Since, $$\frac{-1}{k} \leq \frac{F_k^{-1}(0)}{k} \leq \frac{F_k^{-1}(F_k(0))}{k} = 0.$$

(b) $$\mathcal{F}(1) = \lim_{k \to \infty} \frac{F_k^{-1}(F_k(k))}{k} = 1.$$

(c) follows directly from the monotonicity of F_k^{-1} and G_k^{-1}.

□

Hence, the continuous curve $C(p_0, p_1, p_2, Sf, Sg)$ lies within the parallelogram with vertices $<p_0, p_1, p_2, p_0 - p_1 + p_2>$.

In the remaining of this section we give two special cases for distribution functions and their primitives.

3.32 **Definition** : multiplicative

A function f is called multiplieative if

$$(\forall s, t : s, t \in dom(f) : f(st)=f(s)f(t))$$

□

In appendix 3.A we show that a multiplicative function with domain $\mathbb{R}^+$ is either constant 0 or 1, or equals x^r for a fixed r and for all x in its domain.
Property 3.34 shows that in case the primitives of the functions in a complete set Sf are all multiplicative, a much simpler expression for $\mathcal{F}$ may be given. The property follows directly from the definition of $\mathcal{F}$, the notion of multiplicative functions, and the following property of multiplicative functions.

3.33 **Property** :

The inverse function of a multiplicative function, if it exists, is a multiplicative function.

□

3.34 **Property** :

For all complete sets $Sf = \{f_k \in \mathcal{D}_k \mid k \in \mathbb{N}_o\}$, with F_k a multiplicative function, for all $k \in \mathbb{N}_o$, the following holds.

$$\mathcal{F}(\tau) = \lim_{k \to \infty} F_k^{-1}(\tau)$$

□

Often blending functions $\mathcal{F}$ and $\mathcal{G}$ must have some kind of symmetry. The next definition for distribution functions supplies in this.

3.35 **Definition** : symmetrical

Two distribution functions f and g, both in $\mathcal{D}_n$, are called symmetrical iff

$$(\forall i : i \in [0..n] : f(i)=g(n-i)).$$

□

The blending functions resulting from symmetrical distribution functions are also called symmetrical and their symmetry is described by property 3.36(b).

3.36 **Property :**

Let Sf and Sg be complete sets with the elements of $\mathcal{D}_n$, called f_n and g_n, respectively. If f_k and g_k are symmetrical functions for all $k \in \mathbb{N}_0$, the following holds.

(a) $(\forall k,i : k \in \mathbb{N}_0 \wedge i \in [-1..k] : G_k(i) = F_k(k) - F_k(k-i-1) - 1)$

(b) $(\forall \tau : \tau \in [0,1] : \mathcal{F}(\tau) = 1 - \mathcal{G}(1-\tau))$

Proof

(a) Let $k \in \mathbb{N}_0$ and $i \in [-1..k]$.

$G_k(i)$
= { property 3.11 (a) }
$i + (\Sigma j : j \in [0..i] : g_k(j))$
= { f and g are symmetrical }
$i + (\Sigma j : j \in [0..i] : f_k(k-j))$
= { calculus }
$i + (\Sigma j : j \in [k-i..k] : f_k(j))$
= { calculus }
$i + (\Sigma j : j \in [0..k] : f_k(j)) - (\Sigma j : j \in [0..k-i-1] : f_k(j))$
= { property 3.11 (a) }
$F_k(k) - F_k(k-i-1) - 1$

(b) Let $\tau \in [0,1]$.

$\mathcal{G}(\tau)$
= {definition of $\mathcal{G}$ }
$\lim_{k\to\infty} \dfrac{G_k^{-1}(G_k(k)\tau)}{k}$
= { from (a) follow both $G_k(k) = F_k(k)$ and $G_k^{-1}(i) = k - 1 - F_k^{-1}(F_k(k) - i - 1)$ }

$\lim_{k\to\infty} 1 - \dfrac{1}{k} - \dfrac{F_k^{-1}(F_k(k)(1-\tau)-1)}{k}$

= { calculus and 3.14 (c) }
$1 - \lim_{k\to\infty} \dfrac{F_k^{-1}(F_k(k)(1-\tau))}{k}$
= { definition of $\mathcal{F}$ }
$1 - \mathcal{F}(1-\tau)$

□

Finally, we give in this section a property for primitive functions of a polynomial form. In the sequel these primitive functions turn out to be useful. The property states that under certain conditions the blending function $\mathcal{F}$ is independent of the coefficients in the polynomial and that it depends only on the highest degree

occurring in the polynomial.

3.37 **Property :**

For a primitive function $F_k(x)$ satisfying

(a) $F_k(x)=C_k x^{\alpha}+O(x^{\beta})$ for $x\to\infty$
(b) $F_k(k)=C_k k^{\alpha}+O(k^{\beta})$ for $k\to\infty$
(c) $C_k=O(k^{\gamma})$ for $k\to\infty$

with $\alpha+\gamma>\beta$, the blending function $\mathcal{F}$ is given for $\tau\in[0,1]$ by

$$\mathcal{F}(\tau)=\tau^{\frac{1}{\alpha}}$$

Proof

Let $y=F_k(k)\tau=F_k(x)$. It follows that $x\leq k$ since F_k is increasing and $\tau\in[0,1]$; hence, $x=O(k)$ for $k\to\infty$. For $k\to\infty$ the following derivation holds.

$$\begin{aligned}
&\left(\frac{F_k^{-1}(y)}{k}\right)^{\alpha}\\
&=\{\text{ (a) with } x=F_k^{-1}(y) \text{ and } x=O(k)\ \}\\
&\frac{y-O(k^{\beta})}{C_k k^{\alpha}}\\
&=\{\ y=F_k(k)\tau \text{ and (b)}\ \}\\
&\tau+C_k^{-1}O(k^{\beta-\alpha})\\
&=\{\text{ (c)}\ \}\\
&\tau+O(k^{\beta-\alpha-\gamma})\\
&\to\{\ \beta-\alpha-\gamma<0\ \}\\
&\tau
\end{aligned}$$

□

3.5.2 A subclass consisting of circle and ellipse segments

In this section we deduce the distribution functions for a subclass of w-curves with as continuous curves circle and ellipse segments.

An equation of a circle, given by its center $\mathbf{m}$ and radius R is

$$(x-\mathrm{m}_x)^2+(y-\mathrm{m}_y)^2=R^2$$

By substitution in this equation, the following may be shown to be a parameterisation of a quarter of a circle.

$$(x,y)=\mathbf{m}+R\sqrt{\tau}\,\mathbf{a}_1+R\sqrt{1-\tau}\,\mathbf{a}_2 \text{ with } \tau\in[0,1]$$

where $\mathbf{a}_1$ and $\mathbf{a}_2$ are two vectors such that for their Euclidean distance holds that $\|\mathbf{a}_1\|=\|\mathbf{a}_2\|=1$ and furthermore the angle of $\mathbf{a}_1$ and $\mathbf{a}_2$ is $\pi/2$ (see figure 3.10 (a)). If we

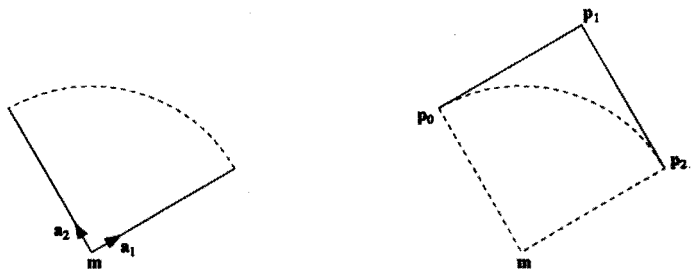


Figure 3.10: parameterisation of a circle

use the following three control points, the control points for generating this circle segment are given by (see figure 3.10(b))

$$\mathbf{p}_0=\mathbf{m}+R\mathbf{a}_2$$
$$\mathbf{p}_1=\mathbf{m}+R(\mathbf{a}_1+\mathbf{a}_2)$$
$$\mathbf{p}_2=\mathbf{m}+R\mathbf{a}_1$$

and we may write $\mathbf{m}=\mathbf{p}_0-\mathbf{p}_1+\mathbf{p}_2$ and give the following parameterisation of a quarter of a circle in terms of these control points.

$$(x,y)=\mathbf{p}_0+\sqrt{\tau}(\mathbf{p}_1-\mathbf{p}_0)+(1-\sqrt{1-\tau})(\mathbf{p}_2-\mathbf{p}_1)\,.$$

This parameterisation has the form of a continuous weave curve with the blending functions $\mathcal{F}$ and $\mathcal{G}$ given, for all $\tau\in[0,1]$, by

$$\mathcal{F}(\tau)=\sqrt{\tau}$$
$$\mathcal{G}(\tau)=1-\sqrt{1-\tau}.$$

$\mathcal{F}$ and $\mathcal{G}$ are symmetrical. Therefore it suffices to determine only the distribution functions f_k for generating $\mathcal{F}$.

$\mathcal{F}$ is also a multiplicative function. We try to find a sequence of multiplicative primitive functions F_k such that

$$\mathcal{F}(\tau)=\lim_{k\to\infty} F_k^{-1}(\tau)$$

By choosing all primitive functions F_k equal to $\mathcal{F}^{-1}$ on $[0,1]$, we find

$$F_k(\tau)=\tau^2, \text{ for all } \tau\in[0,1].$$

Since F_k is multiplicative this also holds on $[0,k]$. Given a primitive function F_k we can compute the distribution function f_k by using definition 3.10.

$$f_k(j)=F_k(j)-F_k(j-1)-1$$

for $j\geq 0$ and with $F_k(-1)=-1$. Hence, the f_k's may be defined by

$$f_k(j):=\begin{cases}2j-2 & (k\geq j>0)\\ 0 & (j=0)\end{cases}$$

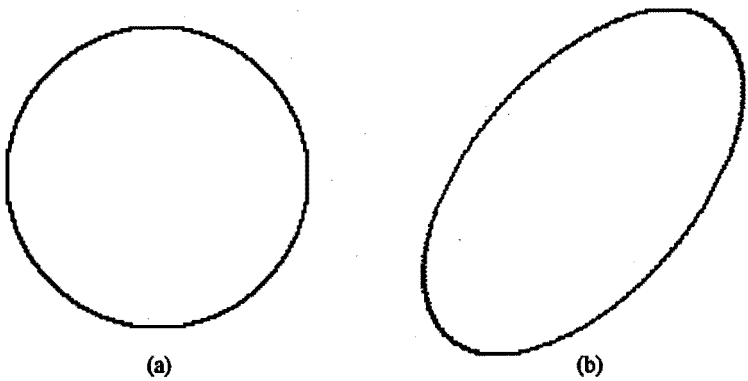


Figure 3.11: Example 3.38

The symmetrical version g_k of f_k is given by (see 3.35)

$$g_k(j) := \begin{cases} 2k-2j-2 & (k > j \geq 0) \\ 0 & (j=k) \end{cases}$$

With these f_k and g_k we can make a quarter of a circle and since an affine transformation of a quarter of a circle is an ellipse segment, the subclass $W(Sf, Sg)$ with the complete sets Sf and Sg given by

$$Sf := \{f_k \mid k \in \mathbb{N}_0\}$$
$$Sg := \{g_k \mid k \in \mathbb{N}_0\},$$

contains only circle and ellipse segments.

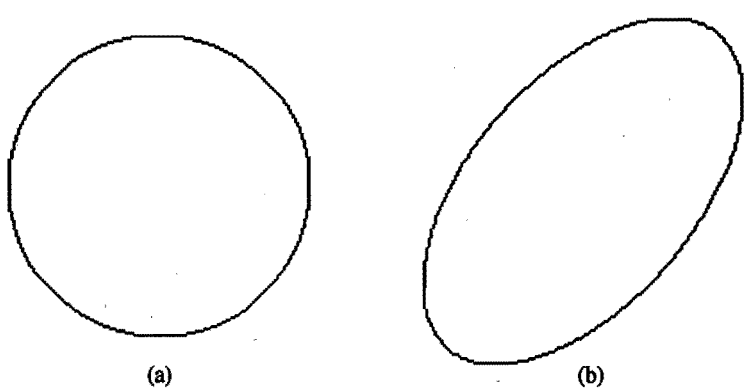


Figure 3.12: Example 3.38

3.38 **Example :**

Figure 3.11 contains both a circle and an ellipse generated by four w-curves. The circle is given by a 4-connected chain, since the control chains that were used, were either horizontal or vertical.
Figure 3.12 shows the same curves after applying the local smoothing method.

□

3.6 Algorithms for computing w-curves

In this section we give algorithms for computing w-curves. According to its definition, a w-curve may be computed by computing the chains of the control lines, adding the 8-codes to them according to some distribution function, and finally weaving the results. However, this approach does not result in an algorithm with a time complexity linear to the length of the two control chains. This is due to the fact that the number of 8-codes added to a control chain need not be linear to the length of the control chain. In this section we give an improved algorithm. And finally, we give a linear algorithm for the cases where the distribution functions are symmetrical.

3.6.0 Algorithm for weaving

In this section we give an algorithm for the function *weave* which solves the problem given below.

```
|[
 chain     c, d, e;
 e:=weave(c, d);
 { e=c w d }
]|
```

A solution to this problem is readily found from the definition 3.2 of weaving and may be constructed according to the following invariant.

$$\begin{aligned} \mathbf{P}: \quad & 0 \le i \le |\mathbf{c}| + |\mathbf{d}| \wedge ic = \#_0(\mathbf{b}, 0, i) \wedge id = \#_2(\mathbf{b}, 0, i) \\ & \wedge \mathbf{e} = (\Pi j : 0 \le j < i : (\mathbf{c} \underline{w} \mathbf{d})(j)) \\ & \wedge \mathbf{b} = bresh4(|\mathbf{c}|, |\mathbf{d}|) \end{aligned}$$

```
func weave(c, d: chain) : chain
    chain     b, e;
    int   i, ic, id;
    b, e :=bresh4( |c| , |d| ), ε;
    i, ic, id:=0, 0, 0; { P }
    do i < |c| + |d|    →
        if b_i =0    →   e :=e⊗ [c_ic];ic :=ic+1
        [] b_i =2    →   e :=e⊗ [d_id];id :=id+1
        fi;
        i :=i+1 { P }
    od;
    weave :=e
cnuf
```

The total time complexity of this algorithm is given by $O(|\mathbf{c}|+|\mathbf{d}|)$. The loop in this algorithm and the loop in the algorithm for *bresh*4 can be combined into one; the if-statements in the two loops distinguish the same cases.

3.6.1 Algorithm for add8

Here we give the function *add*8 which solves the following problem.

```
|[
 chain     c, e;
 array of int: f;
 e :=add8(c, f);
 { e=add8(c, f) }
]|
```

This problem can be solved using the invariant **P** defined as follows.

$$\mathbf{P}: \quad 0 \le i \le |\mathbf{c}| \wedge 0 \le j \le F(i) \wedge Fi = F(i)$$
$$\wedge\, \mathbf{e} = (\Pi k : 0 \le k < j : add8(\mathbf{c}, f)(k))$$

Notice that the length of $add8(\mathbf{c}, f)$ is $F(|\mathbf{c}|)$ and hence, $\mathbf{P} \wedge i = |\mathbf{c}| \wedge j = Fi$ implies $\mathbf{e} = add8(\mathbf{c}, f)$.

```
func add8(c: chain, f: array of int) : chain
    chain e;
    int i, j, Fi;
    i, j, Fi, e:=0, 0, f(0), ε;
    do i<|c| ∨ j<Fi →
        if j<Fi  →  e:=e⊗[8]
        □ j=Fi   →  e:=e⊗[c_i];
                    i, Fi:=i+1, Fi+f(i+1)+1
        fi;
        j:=j+1 {P}
    od
    add8:=e
cnuf
```

The total time complexity of this algorithm is given by $O(|\mathbf{e}|)=O(F(|\mathbf{c}|))$.

3.6.2 Algorithm for computing the pixel set of a w-curve

In this section we derive an algorithm for computing the w-curve $W(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f, g)$. This problem can be formally specified by

```
|[
 point     p0, p1, p2;
 array of int   f, g;
 set of point   S;
 compute_w_curve;
 {S=W(p0, p1, p2, f, g)}
]|
```

A simple solution to this problem has the following form

(1) compute the control chains:

$\mathbf{c}:=bresh\,8(\mathbf{p}_1-\mathbf{p}_0)$; $\mathbf{d}:=bresh\,8(\mathbf{p}_2-\mathbf{p}_1)$;

(2) add 8-codes to the control chains:

$\bar{\mathbf{c}}:=add\,8(\mathbf{c}, f)$; $\bar{\mathbf{d}}:=add\,8(\mathbf{d}, g)$;

(3) weave the resulting chains:

$\mathbf{e}:=weave\,(\bar{\mathbf{c}}, \bar{\mathbf{d}})$;

(4) compute the pixel set $DC(\mathbf{p}_0, \mathbf{e})$ by traversing the chain $\mathbf{e}$.

Step (1) has time-complexity $O(|\mathbf{c}|+|\mathbf{d}|)$. The steps (2) up to (4) are each of time-complexity $O(|\mathbf{e}|)=O(|\bar{\mathbf{c}}|+|\bar{\mathbf{d}}|)=O(F(|\mathbf{c}|)+G(|\mathbf{d}|))$. Hence, this solution is not efficient if $F(|\mathbf{c}|)$ or $G(|\mathbf{d}|)$ are large compared to $|\mathbf{c}|$ and $|\mathbf{d}|$, respectively.

Hence, in order to obtain an algorithm which is linear in the lengths of its control chains, the 8-codes should not be added explicitly to the chains **c** and **d**. The 8-codes are not needed for the computation of the pixel set. Indeed, in property 3.17 the following formula is stated which expresses **p**(**e**, t) directly in terms of the points of the chains **c** and **d**, instead of $\bar{\mathbf{c}}$ and $\bar{\mathbf{d}}$.

$$\mathbf{p}(\mathbf{e},t)=\mathbf{p}(\mathbf{c},\lceil F^{-1}(t-\lfloor\frac{|\bar{\mathbf{d}}|}{|\bar{\mathbf{c}}|+|\bar{\mathbf{d}}|}t\rfloor)\rceil)+\mathbf{p}(\mathbf{d},\lceil G^{-1}(\lfloor\frac{|\bar{\mathbf{d}}|}{|\bar{\mathbf{c}}|+|\bar{\mathbf{d}}|}t\rfloor)\rceil)$$

This expression gives rise to the following invariants for *compute_w_curve*.

$\mathbf{P_0}$ $\quad t\in[0..|\mathbf{e}|]$
$\mathbf{P_1}$ $\quad S=\{\mathbf{p}_0+\mathbf{p}(\mathbf{e},s)\mid s\in[0..t]\}$
$\mathbf{P_2}$ $\quad \mathbf{p}=\mathbf{p}_0+\mathbf{p}(\mathbf{c},a)+\mathbf{p}(\mathbf{d},b)$
$\mathbf{P_3}$ $\quad a=\lceil F^{-1}(t-\lfloor\alpha t\rfloor)\rceil \wedge b=\lceil G^{-1}(\lfloor\alpha t\rfloor)\rceil$

where $\alpha=\frac{G(|\mathbf{d}|)}{F(|\mathbf{c}|)+G(|\mathbf{d}|)}$. We aim at a solution for *compute_w_curve* with the following structure in which, each time the body of the loop is executed, one element is added to the pixel set S.

```
|[
 chain    c, d;
 int  t, a, b;
 point p;

 c, d :=bresh8(p1-p0), bresh8(p2-p1);
 t, a, b :=0, 0, 0;
 p, S :=p0, {p0}; { P0,1,2,3 }
 do a<|c| v b<|d|      →
     if B0 →
                 p:=p+v(ca);
                 a:=a+1;
                 adapt t such that P0,2,3 hold

     [] B1 →
                 p:=p+v(db);
                 b:=b+1;
                 adapt t such that P0,2,3 hold
     fi
     S:=S ∪ {p} { P0,1,2,3 }
 od
]|
```

The guards $B0$ and $B1$, and the two statement lists concerning t remain to be found. First we examine the effect on **p**(**e**, t) of a statement of the form $t:=\hat{t}$ with $\hat{t}>t$.

$\mathbf{p}(\mathbf{e}, \hat{t})$–$\mathbf{p}(\mathbf{e}, t)$
= { property 3.17, $\mathbf{P}_0$, and definition 1.21 of p}
$(\Sigma i : a \le i < \lceil F^{-1}(\hat{t}-[\alpha\hat{t}])\rceil : v(\mathbf{c}_i))$
+
$(\Sigma i : b \le i < \lceil G^{-1}([\alpha\hat{t}])\rceil : v(\mathbf{d}_i))$
= { calculus }

$$\begin{cases} v(\mathbf{c}_a) & \text{if} \quad \lceil F^{-1}(\hat{t}-[\alpha\hat{t}])\rceil = a+1 \wedge \lceil G^{-1}([\alpha\hat{t}])\rceil = b \\ v(\mathbf{d}_b) & \text{if} \quad \lceil F^{-1}(\hat{t}-[\alpha\hat{t}])\rceil = a \wedge \lceil G^{-1}([\alpha\hat{t}])\rceil = b+1 \end{cases}$$

Hence, if we define $B0$ and $B1$ as follows

$$B0 := (\lceil F^{-1}(\hat{t}-[\alpha\hat{t}])\rceil = a+1 \wedge \lceil G^{-1}([\alpha\hat{t}])\rceil = b)$$
$$B1 := (\lceil F^{-1}(\hat{t}-[\alpha\hat{t}])\rceil = a \wedge \lceil G^{-1}([\alpha\hat{t}])\rceil = b+1)$$

$\mathbf{P}_2 \wedge \mathbf{P}_3$ are left invariant by the following two statement lists, if $B0$ and $B1$, respectively, are preconditions of these lists.

```
{ B0 ∧ P2 ∧ P3 }
p:=p+v(ca);
a:=a+1;
t:=t^
{ P2 ∧ P3 }
```

```
{ B1 ∧ P2 ∧ P3 }
p:=p+v(db);
b:=b+1;
t:=t^
{ P2 ∧ P3 }
```

With some simple computing we can express the Bi's in terms of F and G instead of their inverses.

$\lceil F^{-1}(t-[\alpha t])\rceil = a$
= { definition of ceiling function }
$a-1 < F^{-1}(t-[\alpha t]) \le a$
= { F is a monotonous function }
$F(a-1) < t-[\alpha t] \le F(a)$
= { $t-[\alpha t]$ is a monotonic function in t }
$min\{s \in \mathbb{N}_0 \mid s-[\alpha s] > F(a-1)\} \le t < min\{s \in \mathbb{N}_0 \mid s-[\alpha s] > F(a)\}$

A similar expression can be deduced for b and G, and reads as follows.

$\lceil G^{-1}([\alpha t])\rceil = b$
=
$min\{s \mid [\alpha s] > G(a-1)\} \le t < min\{s \mid [\alpha s] > G(a)\}$

So, with the following two abbreviations

$M.i := min\{s \in \mathbb{N}_0 \mid s-[\alpha s] > i\}$
$N.i := min\{s \in \mathbb{N}_0 \mid [\alpha s] > i\}$,

$B0$ and $B1$ can be given by

$$B0=(M.F(a)\leq \hat{t}<M.F(a+1) \wedge N.G(b-1)\leq \hat{t}<N.G(b))$$
$$B1=(M.F(a-1)\leq \hat{t}<M.F(a) \wedge N.G(b)\leq \hat{t}<N.G(b+1)).$$

Since $B0$ and $B1$ are only evaluated if **P** holds and since we chose $\hat{t}$ to be larger than t, they can be simplified to

$$B0=(M.F(a)\leq \hat{t}<M.F(a+1) \wedge \hat{t}<N.G(b))$$
$$B1=(\hat{t}<M.F(a) \wedge N.G(b)\leq \hat{t}<N.G(b+1)).$$

Choosing $\hat{t}=M.F(a)$ in case of $B0$ and $\hat{t}=N.G(b)$ in case of $B1$ and adding the invariants

$$\mathbf{P_4} \quad Fa=F(a) \wedge Gb=G(b)$$
$$\mathbf{P_5} \quad tc=M.Fa \wedge td=N.Gb$$

reduce the Bi's further to

$$B0=(tc<td)$$
$$B1=(td<tc).$$

For both choices of $\hat{t}$, $\hat{t}>t$ holds. Furthermore, notice that $tc \neq td$ as can be seen from

$tc=td$
$\Rightarrow$ { $\mathbf{P_5}$ }
$tc=M.Fa \wedge tc=N.Gb$
$\Rightarrow$ { definition of M and N }
$tc-1-[\alpha(tc-1)]\leq Fa<tc-[\alpha tc] \wedge [\alpha(tc-1)]\leq Gb<[\alpha tc]$
$\Rightarrow$ { calculus }
$[\alpha tc]<tc-Fa\leq 1+[\alpha(tc-1)] \wedge [\alpha(tc-1)]\leq Gb<[\alpha tc]$
$\Rightarrow$ { $[\alpha tc]$ is an integer }
$[\alpha tc]\leq [\alpha(tc-1)] \wedge [\alpha(tc-1)]<[\alpha tc]$
$\Rightarrow$ { }
$[\alpha tc]<[\alpha tc]$
$\Rightarrow$ { }
false

Hence, $B0 \vee B1=true$ and consequently, the if-statement covers all possible cases. Since we added the invariants $\mathbf{P_4}$ and $\mathbf{P_5}$, we have the concern over their invariance. $\mathbf{P_4}$ is kept invariant, according to the definition of the primitive function F, by the following two statement lists.

$\{ P_4 \}$	$\{ P_4 \}$
$a := a+1;$	$b := b+1;$
$Fa := Fa+f(a)+1$	$Gb := Gb+g(b)+1$
$\{ P_4 \}$	$\{ P_4 \}$

For the invariance of P_5, we allow, for the time being, a statement of the form $t := M.Fa$.
The program *compute_w_curve* then takes the following form.

```
|[
 chain   c, d;
 point   p;
 int  t, tc, td;
 int  a, b, Fa, Gb;

 c, d := bresh8(p1−p0), bresh8(p2−p1);
 a, b, Fa, Gb := 0, 0, f(0), g(0);
 t, tc, td := 0, M.Fa, N.Gb;
 p, S := p0, {p0};
 do a< |c| ∨ b< |d|      →
    if tc<td  →
              p := p+v(ca);
              a := a+1;
              Fa := Fa+f(a)+1;
              t, tc := tc, M.Fa
    [] td<tc  →
              p := p+v(db);
              b := b+1;
              Gb := Gb+g(b)+1;
              t, td := td, N.Gb
    fi;
    S := S ∪ {p}
 od
]|
```

At a first glance the time complexity of the above program appears to be linear in $|c|+|d|$, but it is not, since the computations of *M.Fa* and *N.Gb* can, in general, not be done in constant time; with binary search they can be done in $O(\log F(|c|))$- and $O(\log G(|d|))$-time. In that case the total time complexity is $O(|c|\log F(|c|)+|d|\log G(|d|))$. Compared to the naive solution as sketched at the beginning of this section, the new approach is assymptotically better if the computations of F and G are more than linear in their arguments.

3.6.2.1 A linear algorithm

The algorithm given in the previous section was not linear because of the computational complexity of $M.Fa$ and $N.Gb$. However in some cases these computations are simple. Here we consider the case $\alpha=\frac{1}{2}$. Hence,

$$\alpha=\frac{G(|d|)}{F(|c|)+G(|d|)}=\frac{1}{2}.$$

Hence, $F(|c|)=G(|d|)$. In this case the following computation on M can be made.

$M.i=j$
$=$ { definition of M and $\alpha=\frac{1}{2}$ }
$min\{s \mid s-\lfloor\frac{1}{2}s\rfloor>i\}=j$
$=$ { property of min }
$j-\lfloor\frac{1}{2}j\rfloor=i+1 \wedge j-1-\lfloor\frac{1}{2}(j-1)\rfloor=i$
$=$ { calculus }
$j-\lfloor\frac{1}{2}j\rfloor=i+1 \wedge \lfloor\frac{1}{2}j\rfloor=\lfloor\frac{1}{2}(j-1)\rfloor$
$=$ { property 1.14 of rounding }
$j-\lfloor\frac{1}{2}j\rfloor=i+1 \wedge j \bmod 2=1$
$=$ { property 1.14 of rounding }
$j-\frac{1}{2}j+\frac{1}{2}=i+1 \wedge j \bmod 2=1$
$=$ { calculus }
$j=2i+1$

Hence, $M.i=2i+1$ and one can prove similarly $N.i=2i+2$. In the program of the previous section, consequently, the statements $tc:=M.Fa$ and $td:=N.Gb$ may be replaced by $tc:=2Fa+1$ and $td:=2Gb+2$. In fact, we may completely dismiss tc and td from the program by replacing the guards $B\,0$ and $B\,1$ by

$$B\,0=(Fa\leq Gb)$$
$$B\,1=(Fa>Gb).$$

This follows from substituting the new expressions for tc and td. The resulting program has computational complexity $O(|c|+|d|)$ which is linear in the lengths of the control chains. The program takes the following form. Notice that we also left out the ghost variable t.

```
|[
 point    p0
 chain    c, d;
 int  a, b, Fa, Gb;

 c, d :=bresh8(p1-p0), bresh8(p2-p1);
 a, b, Fa, Gb :=0, 0, f(0), g(0);
 p, S :=p0, {p0};
 do a<|c| ∨ b<|d|      →
     if Fa≤ Gb  →
                p :=p+v(ca);
                a :=a+1;
                Fa :=Fa+f(a)+1
     [] Fa> Gb  →
                p :=p+v(db);
                b :=b+1;
                Gb :=Gb+g(b)+1
     fi;
     S :=S ∪ {p}
 od
]|
```

Remember that this program has been derived under the assumption that $\alpha=\frac{1}{2}$, that is $F(|\mathbf{c}|)=G(|\mathbf{d}|)$. It is, hence, only applicable for restricted combinations of distribution functions and control chains.

With a slight change in the definition of w-curves some subclasses of w-curves may be handled with the above algorithm. The most important of these classes are those featuring symmetrical distribution functions.

For the new curves, called e-curves, the control chains are made of equal length by adding 8-codes in a uniform way to the smallest one using the function *lengthen*.

3.39 **Definition :** *lengthen*

For all chains **c** and all positive integers k the function *lengthen* is defined by

$$lengthen(\mathbf{c}, k):=\begin{cases}\mathbf{c} & |\mathbf{c}|\geq k\\ \mathbf{c}\,\underline{w}\,[8]^{k-|\mathbf{c}|} & |\mathbf{c}|<k\end{cases}$$

□

The following property of *lengthen* follows directly from property 3.6.

3.40 **Property :**

For all chains **c**, all positive integers k, and all $t \in [0..max\{k, |\mathbf{c}|\}]$

$$\mathbf{p}(lengthen(\mathbf{c}, k), t) = \begin{cases} \mathbf{p}(\mathbf{c}, t) & |\mathbf{c}| \geq k \\ \mathbf{p}(\mathbf{c}, t - \lfloor \frac{k}{|\mathbf{c}|+k} t \rfloor) & |\mathbf{c}| < k \end{cases}$$

□

The definition of e-curves can now be given as follows.

3.41 **Definition :** e-curve

For all control points $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$, and all distribution functions $f_i \in \mathcal{D}_m$ with $F_1(m) = F_0(m)$ and m given by

$$m := max\{|\mathbf{c}_0|, |\mathbf{c}_1|\} \text{ with}$$
$$\mathbf{c}_0 := bresh\,8(\mathbf{p}_{i+1} - \mathbf{p}_i) \text{ and } \mathbf{c}_1 := bresh\,8(\mathbf{p}_{i+1} - \mathbf{p}_i),$$

an e-curve $E(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f, g)$ is defined as the discrete curve $DC(\mathbf{p}, \mathbf{e})$ with

$$\mathbf{p} = \mathbf{p}_0$$
$$\mathbf{e} = add\,8(lengthen(\mathbf{c}_0, m), f)) \underline{w}\, add\,8(lengthen(\mathbf{c}_1, m), g).$$

□

Subclasses of e-curves can now be defined similarly to definition 3.21.

3.42 **Definition :** subclass for e-curves

For all complete sets of functions Sf and Sg with

$$(\forall n : n \in \mathbb{N}_0 : F_n(n) = G_n(n)),$$

the subclass $E(Sf, Sg)$ is defined by

$$E(Sf, Sg)$$
$$:=$$
$$\{E(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f, g) \mid \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \in \mathbb{Z}^2 \wedge f \in Sf \cap \mathcal{D}_n \wedge g \in Sg \cap \mathcal{D}_m\}$$

with the abbreviations n and m given $n = D_8(\mathbf{p}_1 - \mathbf{p}_0)$ and $m = D_8(\mathbf{p}_2 - \mathbf{p}_1)$.

□

Within a subclass $E(Sf, Sg)$ a continuous version of an e-curve can be computed. A property similar to property 3.25 may be given. This results in the continuous curves CE to be defined as follows.

3.43 **Definition : CE**

The function $CE(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg)$ is defined for all points $\mathbf{p}_i$ $(i \in [0..2])$ and all complete sets Sf and Sg with

$$(\forall n : n \in \mathbb{N}_o : F_n(n) = G_n(n)),$$

by

$$CE(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, Sf, Sg) := \mathbf{p}_0 + \mathcal{F}(\tau)(\mathbf{p}_1 - \mathbf{p}_0) + \mathcal{G}(\tau)(\mathbf{p}_2 - \mathbf{p}_1),$$

for all $\tau \in [0, 1]$.

□

Hence, *CE* equals *CW*, for complete sets Sf and Sg suitable for e-curves. The complete sets computed in section 3.5.2 for ellipse and circle segments, consist of symmetrical distribution functions. These circles and ellipse segments, consequently, may be genereted with the integer algorithm for the pixel sets of e-curves and in a time linear to the length of the control chains.

3.A Appendix

Let f be a continuous multiplicative function (see definition 3.32) with domain $\mathbb{R}^+$.

3.44 Property :

(a) $f(0)=0 \vee f(0)=1$
(b) $f(1)=0 \vee f(1)=1$

Proof

Both (a) and (b) follow directly from $a^2=a \Rightarrow f(a)^2=f(a)$.

□

3.45 Property :

(a) $f(0)=1 \Rightarrow (\forall x : x \in \mathbb{R}^+ : f(x)=1)$
(b) $f(1)=0 \Rightarrow (\forall x : x \in \mathbb{R}^+ : f(x)=0)$

Proof

(a) follows from $f(0)=f(x)f(0)$ and (b) from $f(x)=f(1)f(x)$.

□

3.46 Property :

If $f(0)=0$ and $f(1)=1$ then an $r \in \mathbb{R}^+$ exists such that the following holds.

$$(\forall x : x \in \mathbb{R}^+ : f(x)=x^r)$$

Proof

Let $x \in \mathbb{R}^+$ with $x \neq 0 \wedge x \neq 1$. Let $X \in \mathbb{R}^+$ with $X>1$. We write X as $X=x^{n_X+r_X}$ with $n_X \in \mathbb{N}_0$ and $r_X \in [0, 1)$.
Notice that $f(x) \geq 0$, since $f(x)=f(\sqrt{x})f(\sqrt{x})$ and that $f(x) \neq 0$, because $1=f(1)=f(x)f(\frac{1}{x})$. Hence, $f(x)>0$ and similarly, $f(X)>0$. Consequently, we may write

$$\frac{\log f(X)}{\log X} = \frac{n_X \log f(x)+\log f(x^{r_X})}{n_X \log x+\log (x^{r_X})}$$

Consequently,

$$(\forall x : x \in \mathbb{R}^+ - \{0, 1\} : \lim_{X \to \infty} \frac{\log f(X)}{\log X} = \frac{\log f(x)}{\log x}).$$

By defining $r := \frac{\log f(2)}{\log 2}$, and noticing that r is positive, the proof of the property is completed.

□

4

Extensions of w-curves

4.0 Introduction

In conventional curve modeling the shape of a curve may be controlled in two ways.

(a) choosing different curve flavours, e.g. Bezier curves instead of Hermite curves.

(b) introducing more control points and using these control points for fine tuning the shape of the curve.

Both (a) and (b) may be translated into the realm of w-curves. For (a) in section 4.1 parameterised 8-distribution functions are introduced. For (b) we give in section 4.2 two methods for defining w-curves with more than three control points. Both methods come down to defining some weaving scheme for more than two control chains; the schemes given in sections 4.2.1 and 4.2.2 are called consecutive weaving and simultaneous weaving, respectively. Distribution functions for second and third order Bezier curves are given in an example of simultaneous weaving.

Finally, we give in section 4.3 a method for improving the computational effort needed for w-curves. The method is based on precomputing the order in which codes are taken from the control chains. This order is represented by a so-called canonical chain.

4.1 Parameterised w-curves

W-curves are defined by three control points and two distribution functions. The control points are used for controlling the tangents in the end points, whereas the distribution functions can be used to change the flavour of the curves. This last possibility is clearly not available for well-known curves as Bezier and Hermite, since they come just in one flavour: there is exactly one (Bezier) curve on three given control points. So if we can control in an intuitive manner the flavour of the curve by adapting the distribution functions, w-curves have a clear advantage over other curves. Here we give an example for adapting distribution functions by adding a parameter mechanism.

In the algorithms for computing the pixel set of a w-curve, as given in section 3.6.2, a distribution function was represented by an array of integers; there was, however, no mentioning of how this array was filled. The filling of the array should, preferably, be of low cost, e.g. by means of an incremental scheme for computing polynomials. In this section we give another cheap way for incrementally computing a distribution function.

A distribution function is, as we defined in 3.7, a function with an interval $[0..n]$ (for some $n \in \mathbb{N}_0$) as its domain and a subset of $\mathbb{N}_0$ as its reach. Below we generate such a function from a discrete curve, i.e. a w-curve given by 3 control points; one of these control points is used as a parameter to control the shape of the resulting distribution function.

> Let W be a discrete curve with the property that for all $x \in [0..n]$ at least one y exists such that $(x, y) \in W$. For such a discrete curve W a function $f_W: [0..n] \rightarrow \mathbb{N}_0$ is defined for all $x \in [0..n]$ by
>
> $$f_W(x) := max\{y \mid (x,y) \in W\}$$

Hence, if we can define a curve with the proper "domain" and the proper "reach" ($\subseteq \mathbb{N}_0$) the above defined function f_W is a distribution function. As a first example such a discrete curve W is defined as a w-curve.

$$W := W(q_0, q_1, q_2, f, g)$$

where the points q_0 and q_2 are given by

$$q_0 = (0, 0) \wedge q_2 = (n, n)$$

and where f and g are two already known distribution functions, e.g. the identity function and the function symmetrical to the identity.

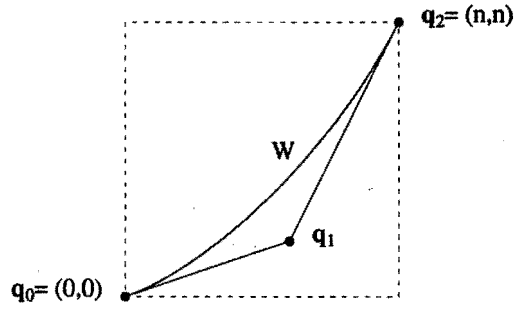
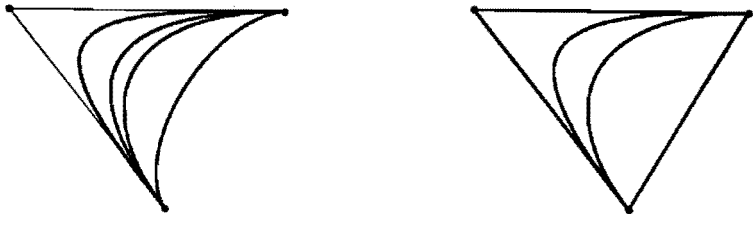


Figure 4.1: parameterised distribution function f_W

The remaining control point q_1 can be used as a parameter (see figure 4.1). The choices for q_0 and q_1 are not arbitrary; they are chosen such that f_W is small at the beginning and relatively large at the end of its domain. Hence, f_W can be used as a

distribution function for a w-curve that is tangent to a control chain at its start point.

The function f_W can be computed incrementally by generating the w-curve W. Similarly a symmetrical version g_W of f_W can be defined. Using these two distribution functions, the w-curve $C=W(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, f_W, g_W)$ can be influenced by the choice of $\mathbf{q}_1$; if $\mathbf{q}_1$ is chosen closer to the lower right corner of the square, the distribution function f_W has relative small values at the beginning, and consequently, C is closer to the control lines. By choosing $\mathbf{q}_1.x=\mathbf{q}_1.y$, the resulting distribution function is the identity function, and hence, according to section 3.5.2, in that case C is an ellipse segment. By moving $\mathbf{q}_1$ away from the diagonal of the square, the deviation from an ellipse segment becomes larger.



(a) (b)

Figure 4.2: parameterisation of w-curves
(a) example 4.1
(b) example 4.2

4.1 **Example :**

Figure 4.2(a) shows 4 curves parameterised with the scheme of figure 4.1 with $n=413$. The values of $\mathbf{q}_1$ are for these curves from left to right given by (413,0), (267,133), (0,0) and (0,413), respectively. Hence, the third curve has the identity function as its distribution function and is, consequently, an ellipse segement.

□

The scheme shown above for defining a distribution function leaves room for all kind of variations.

By defining W as the union of two w-curves W_0 and W_1, the distribution function f_W can vary more. Not only a linear function f_W can now be made but also a constant function ($f_W(i)=n$) and a peak function ($f_W(i)=0$ for $i\neq n$, $f_W(n)=n$). W-curve W_0 is given by $\mathbf{q}_0=(0,0)$, $\mathbf{q}_{01}$, and $\mathbf{q}_1$, and W_1 by $\mathbf{q}_1$, $\mathbf{q}_{12}$, and $\mathbf{q}_2=(n,n)$; $\mathbf{q}_1$ is again the parameter point and $\mathbf{q}_{01}$ and $\mathbf{q}_{12}$ are the leftmost and rightmost intersection point, respectively, of the square with the line through $\mathbf{q}_1$ with slope 1. See example 4.2.

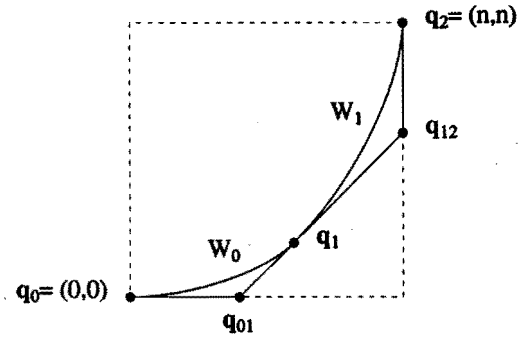


Figure 4.3: parameterised distribution function f_W with $W = W_0 \cup W_1$

4.2 Example :

Figure 4.2(b) shows 4 curves parameterised with the scheme of figure 4.3. The points $\mathbf{q}_i$ are chosen as in example 4.2. Again the third curve is an ellipse segment. The peak function ($\mathbf{q}_1 = (413,0)$) results in a curve that coincides with the control lines of the curve, whereas the constant function ($\mathbf{q}_1 = (0,413)$) results in a line segment joining the two end points.

□

4.2 More control points

In chapter 3 w-curves are defined as curves given by three control points and in their definition the operators *add*8 and $\underline{w}$ are used. In this section w-curves given by more than three control points are defined. Two essentially different extensions of w-curves are defined; both are based on the weave operator for combining two chains. In the first, the chains are weaved in some order using the weave operator; in the second approach, the chains are weaved all at once, using a generalisation of the weave operator $\underline{w}$. These two approaches are called consecutive weaving and simultaneous weaving, respectively.

4.2.1 Consecutive weaving

In case of consecutive weaving the operator $\underline{w}$ is used (repeatedly) for combining chains (initially control chains) in some order. The order of weaving the chains could, for instance, be from left to right: weave the first two control chains, weave the result with the next chain etc. If this scheme is used, curves with the same

control points but with these points in reversed order, may not be exactly the same pixel sets. To avoid this asymmetry we choose a recursive scheme in which in cases of an odd number of chains extra control points are introduced. Figures 4.4 (a), (b), and (c) illustrate the recursive subdivision scheme used in the following definition.

4.3 **Definition : W_n**

For all complete sets of functions Sf and Sg, for all $n \geq 1$, and for all control points $\mathbf{p}_i$ $(i \in [0..n])$, $W_n(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n, Sf, Sg)$ is defined as the discrete curve $DC(\mathbf{p}, \mathbf{e})$ with

$$\mathbf{p} = \mathbf{p}_0$$
$$\mathbf{e} = X(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n)$$

where the chain $X(\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_k)$ is recursively defined, for all $k \in \mathbb{N}$, and all points $\mathbf{q}_i$ $(i \in [0..k])$, by

$$X(\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_k) = \begin{cases} add8(X(\mathbf{q}_0, \ldots, \mathbf{q}_h), f) \ \underline{w} \ add8(X(\mathbf{q}_h, \ldots, \mathbf{q}_k), g) \\ \qquad \text{if } k = 2h \\ \\ add8(X(\mathbf{q}_0, \ldots, \mathbf{q}_h, \mathbf{q}), f) \ \underline{w} \ add8(X(\mathbf{q}, \mathbf{q}_{h+1}, \ldots, \mathbf{q}_k), g) \\ \qquad \text{if } h > 0 \wedge k = 2h+1, \text{ with } \mathbf{q} = [(\mathbf{q}_h + \mathbf{q}_{h+1})/2] \\ \\ bresh8(\mathbf{q}_1 - \mathbf{q}_0) \\ \qquad \text{if } k = 1 \end{cases}$$

with f and g the appropriate functions from Sf and Sg, respectively.

□

Notice that W_1 is equal to a Bresenham line and that W_2 equals a w-curve.

For computing the continuous curve corresponding to a W_n-curve property 3.25 can be used. Here we give an expression for the continuous curve belonging to $W_4(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, Sf, Sg)$. The limit of the sequences $(X(k\mathbf{p}_0, k\mathbf{p}_1, k\mathbf{p}_2))_{k \in \mathbb{N}}$ and $(X(k\mathbf{p}_2, k\mathbf{p}_3, k\mathbf{p}_4))_{k \in \mathbb{N}}$ are called C_0 and C_1, respectively, and are given, according to property 3.25 or definition 3.28, by

$$C_0(\tau) = \mathcal{F}(\tau)(\mathbf{p}_1 - \mathbf{p}_0) + \mathcal{G}(\tau)(\mathbf{p}_2 - \mathbf{p}_1)$$
$$C_1(\tau) = \mathcal{F}(\tau)(\mathbf{p}_3 - \mathbf{p}_2) + \mathcal{G}(\tau)(\mathbf{p}_4 - \mathbf{p}_3)$$

Hence, the limit C of the sequence $(X(k\mathbf{p}_0, k\mathbf{p}_1, k\mathbf{p}_2, k\mathbf{p}_3, k\mathbf{p}_4))_{k \in \mathbb{N}}$ is given for $\tau \in [0, 1]$ by

$$C(\tau)$$
$$= \{ \text{ property 3.25 } \}$$
$$C_0(\mathcal{F}(\tau)) + C_1(\mathcal{G}(\tau))$$

$= \{$ the above expressions for C_0 and C_1 $\}$
$\mathcal{F}(\mathcal{F}(\tau))(\mathbf{p}_1-\mathbf{p}_0)+\mathcal{G}(\mathcal{F}(\tau))(\mathbf{p}_2-\mathbf{p}_1)+\mathcal{F}(\mathcal{G}(\tau))(\mathbf{p}_3-\mathbf{p}_2)+\mathcal{G}(\mathcal{G}(\tau))(\mathbf{p}_4-\mathbf{p}_3)$.

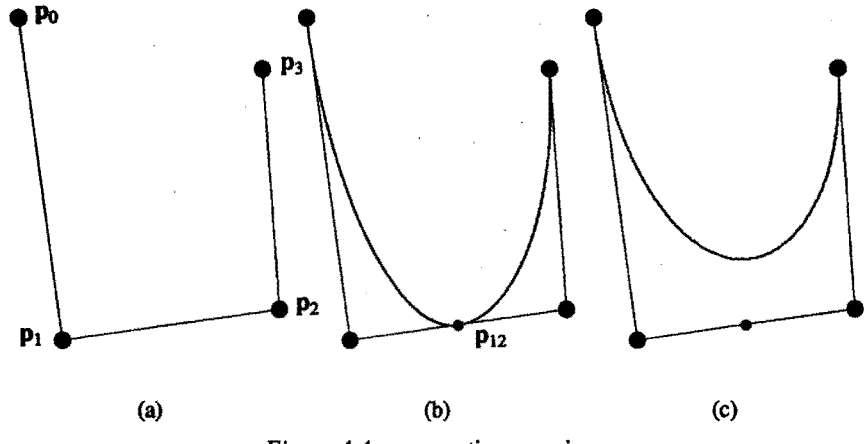


Figure 4.4: consecutive weaving
(a) the control points
(b) the curves $DC(\mathbf{p}_0, X(\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}_{12}))$ and $DC(\mathbf{p}_0, X(\mathbf{p}_{12},\mathbf{p}_2,\mathbf{p}_3))$
(c) $W_3(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, Sf, Sg)$

Hence, applying consecutive weaving results in curves with higher order blending functions. The figures 4.5, 4.6, and 4.7 show the construction of a W_5-curve.

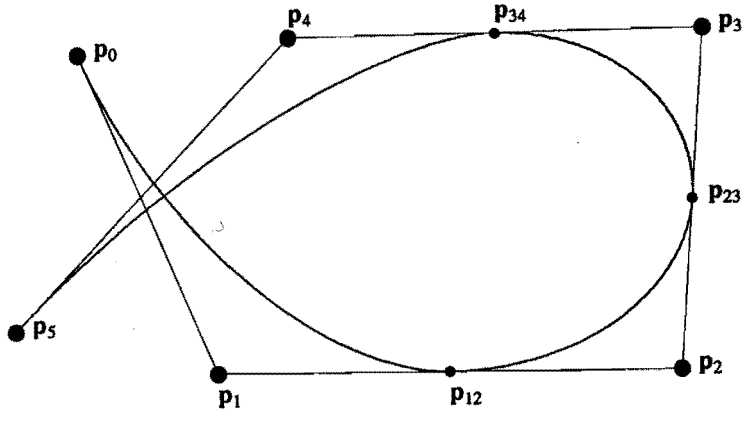


Figure 4.5: construction of a W_5 (1)

Figure 4.8 illustrates the computation of a W_4-curve. The two "weave" subtrees correspond to W_2-curves. The chains at level 0 are the control chains and may be

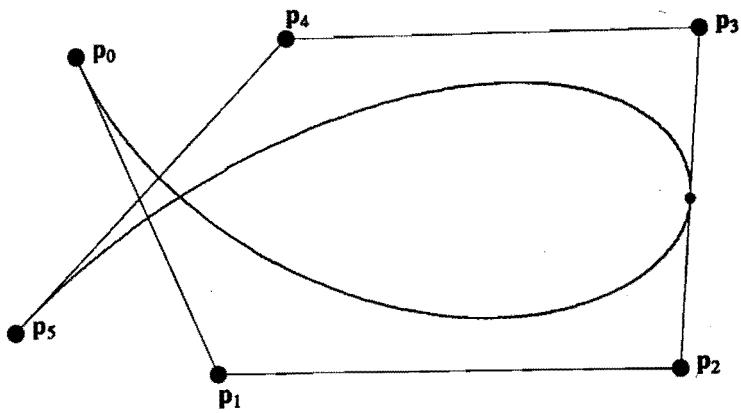


Figure 4.6: construction of a W_5 (2)

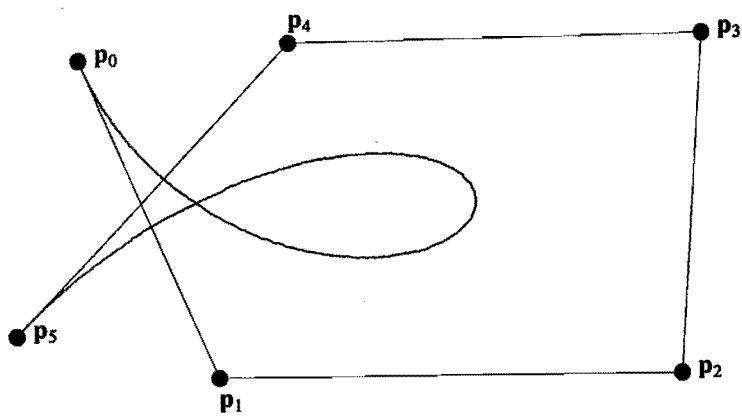


Figure 4.7: construction of a W_5 (3)

computed in $O(m_0)$ time where m_i is the maximum length of the chains at level i of the tree. In section 3.6.2.1 we saw that the computation of the pixel set of a W_2 curve may be done in $O(m_0)$ time if the following two conditions are met:

(a) the control chains have both length m.
(b) the distribution functions are compatible: $F(m)=G(m)$

Here, however, the chain of a W_2-curve, including the 8-codes, must be computed. This computation takes $O(m_1)$ time. Consequently, if $n>1$, a W_n-curve may, in general, not be computed in $O(m_0)$ time. The definition of W_n-curves may be adapted, resulting in different curves, such that the resulting curve is computable in $O(m_0)$

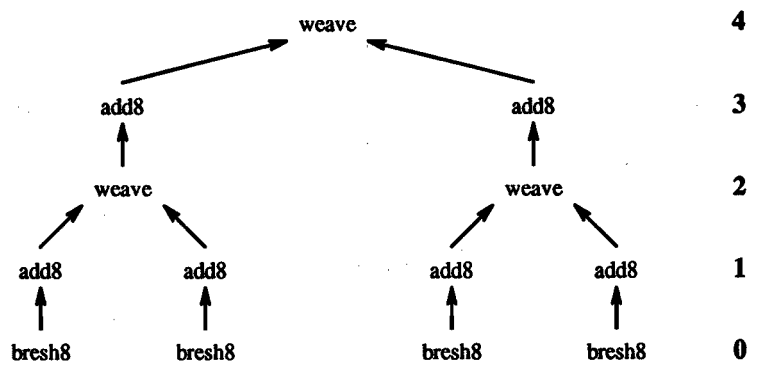


Figure 4.8: computation scheme for W_4-curve

time. Such an adaptation is not given here, since we introduce in section 4.3 a method by which W_n-curves may be computed in time linear to m_0.

4.2.2 Simultaneous weaving

In simultaneous weaving all control chains are weaved at once using the simultaneous weave operator $\underline{W}$, as defined below. In section 3.6.2.1 we used the fact that the chains are (and remain) of equal length to arrive at a linear algorithm for e-curves. For this reason it suffices to define simultaneous weaving for chains of equal length only.

4.4 **Definition** : simultaneous weave operator

For all $n \geq 1$ and chains $\mathbf{c}_i$ $(i \in [0..n))$, all of the same length, the chain $(\underline{W} i : i \in [0..n) : \mathbf{c}_i)$ of length $n \mid \mathbf{c}_0 \mid$ is defined by

$$(\underline{W} i : i \in [0..n) : \mathbf{c}_i)(t) := \mathbf{c}_m(d),$$

for all $t \in [0..n \mid \mathbf{c}_0 \mid)$ with the abbreviations m and d given by $m = t \bmod n$ and $d = t \operatorname{div} n$.

□

Figure 4.9 illustrates the W-operator.

4.5 **Property** :

For all $n \geq 1$ and for the chains $\mathbf{c}_i$ $(i \in [0..n))$, all of the same length,

$$\mathbf{p}((\underline{W} i : i \in [0..n) : \mathbf{c}_i), t) = (\Sigma i : i \in [0..n) : \mathbf{p}(\mathbf{c}_i, \lceil \tfrac{t-i}{n} \rceil))$$

for all $t \in [0..n \mid \mathbf{c}_0 \mid]$.

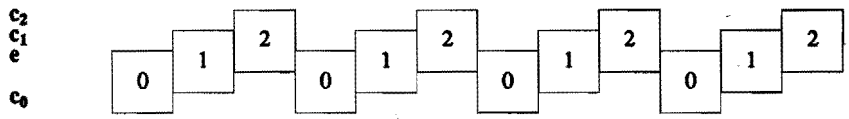


Figure 4.9: the simultaneous weave operator $\underline{W}$
$\mathbf{c}_0=[0000]$, $\mathbf{c}_1=[1111]$, $\mathbf{c}_2=[2222]$
$\mathbf{e}=(\underline{W}i : i\in\{0,1,2\} : \mathbf{c}_i)=[012012012012]$

Proof

We use $\mathbf{c}$ as a shorthand for the chain $(\underline{W}i : i\in[0..n) : \mathbf{c}_i)$. Let $t\in[0..n\,|\mathbf{c}_0|]$.

$\mathbf{p}(\mathbf{c},t)$
= { definition 1.21 of $\mathbf{p}$ }
$(\Sigma j : j\in[0..t) : v(\mathbf{c}(j)))$
= { definition of $\mathbf{c}$ and $\underline{W}$ }
$(\Sigma j, m, d : j\in[0..t) \wedge m=j \bmod n \wedge d=j \operatorname{div} n : v(\mathbf{c}_m(d)))$
= { definition of *mod* and *div* : $j=n(j \operatorname{div} n)+j \bmod n \wedge m\in[0..n)$ }
$(\Sigma m, d : dn+m\in[0..t) \wedge m\in[0..n) : v(\mathbf{c}_m(d)))$
= { calculus }
$(\Sigma m : m\in[0..n) : (\Sigma d : dn+m\in[0..t) : v(\mathbf{c}_m(d))))$
= { calculus }
$(\Sigma m : m\in[0..n) : (\Sigma d : d\in[\frac{-m}{n}\,..\,\frac{t-m}{n}) : v(\mathbf{c}_m(d))))$
= { calculus }
$(\Sigma m : m\in[0..n) : (\Sigma d : d\in[0..\lceil\frac{t-m}{n}\rceil) : v(\mathbf{c}_m(d))))$
= { definition 1.21 of $\mathbf{p}$ }
$(\Sigma m : m\in[0..n) : \mathbf{p}(\mathbf{c}_m, \lceil\frac{t-m}{n}\rceil))$

□

Given this operator $\underline{W}$ for weaving a number of chains definition 3.41 of e-curves can be extended to curves with any number of control points.
Notice that in the definition below, the combination of *add*8 and *lengthen* together with the condition $F_i(m)=F_0(m)$, results in equal-length chains for weaving with the $\underline{W}$ operator.

4.6 **Definition** : nth-order e-curves

For all $n \geq 1$, all control points $\mathbf{p}_i$ ($i \in [0..n)$), and all distribution functions $f_i \in \mathcal{D}_m$ with $F_i(m) = F_0(m)$ with m given by

$$m := max\{\, |\mathbf{c}_i| \mid i \in [0..n) \,\}$$

with the chains $\mathbf{c}_i$ given by

$$\mathbf{c}_i := bresh\, 8(\mathbf{p}_{i+1} - \mathbf{p}_i),$$

the nth-order e-curve $E_n(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n, f_0, f_1, \cdots, f_{n-1})$ is defined as the discrete curve $DC(\mathbf{p}, \mathbf{e})$ with

$$\mathbf{p} = \mathbf{p}_0$$
$$\mathbf{e} = (\underline{W}\, i : i \in [0..n) : add\, 8(lengthen(\mathbf{c}_i, m), f_i)).$$

□

A continuous variant *CE* of the E_n-curves may be defined similar to *CW* in definition 3.28 and *CE* in definition 3.43. In the definition of *CE* we notate by $f_{i,k}$ the distribution function in the complete set Sf_i with domain $[0..k]$. The primitive of $f_{i,k}$ is notated by $F_{i,k}$. The limit of $F_{i,k}$, according to definition 3.25, is notated by $\mathcal{F}_i$.

4.7 **Definition** : CE

For all n, all control points $\mathbf{p}_i$, and all complete sets Sf_i ($i \in [0..n]$) with

$$(\forall i,k : i \in [0..n] \wedge k \in \mathbb{N}_o : F_{i,k}(k) = F_{0,k}(k)),$$

the continuous curve $C = CE(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n, Sf_0, Sf_1, \cdots, Sf_{n-1})$ is defined, for all $\tau \in [0, 1]$ by

$$C(\tau) := \mathbf{p}_0 + (\Sigma i : i \in [0..n) : \mathcal{F}_i(\tau)(\mathbf{p}_{i+1} - \mathbf{p}_i))$$

□

Below an example of simultaneous weaving with 2 and 3 control chains is shown; it results in second and third order Bezier curves. The first part of the example shows an ordinary weave, since it only concerns two control chains; it is given here as an introduction to the second part.

4.8 **Example** :

An nth-order Bezier curve (see for instance [Boe84]) is denoted by $B_n(\tau)$ and given, for all $\tau \in [0, 1]$ by

$$B_n(\tau) = (\Sigma i : i \in [0..n] : \binom{n}{i} \tau^i (1-\tau)^{n-i} \mathbf{p}_i),$$

where $\mathbf{p}_0$ up to $\mathbf{p}_n$ are its control points. First we consider $B_2(\tau)$. In order to

find suitable 8-distribution functions for this curve, its parameter function is rewritten in the affine invariant form of property 1.4.

$$
\begin{aligned}
& B_2(\tau) \\
& = \{ \text{ definition of 2nd order Bezier curve } \} \\
& (1-\tau)^2\mathbf{p}_0+2\tau(1-\tau)\mathbf{p}_1+\tau^2\mathbf{p}_2 \\
& = \{ \text{ calculus } \} \\
& \mathbf{p}_0+(1-(1-\tau)^2)(\mathbf{p}_1-\mathbf{p}_0)+\tau^2(\mathbf{p}_2-\mathbf{p}_1)
\end{aligned}
$$

The blending functions in the above expression are symmetrical (see property 3.36) and hence it suffices to find an 8-distribution function for the blending function $\mathcal{F}_1(\tau)=\tau^2, \tau\in[0..1]$. $\mathcal{F}_1$ is multiplicative (definition 3.32); multiplicative 8-distribution functions $F^{-1}_{1,k}$, however, with $\mathcal{F}_1(\tau)=\lim_{k\to\infty} F^{-1}_{1,k}(\tau)$ do not exist. This follows from the fact that $F^{-1}_{1,k}(\tau)=\tau^2$ implies $F_{1,k}(\tau)=\sqrt{\tau}$ and consequently, $F_{1,k}(k)=\sqrt{k}$. This last fact contradicts with property 3.11, since distribution functions are non-negative on their domain. Fortunately, by reparameterising the curve, a way around may be found.

$$
\begin{aligned}
& B_2(\tau) \\
& = \{ \text{ reparameterisation with } \sigma=\tau^2 \} \\
& \mathbf{p}_0+(1-(1-\sqrt{\sigma})^2)(\mathbf{p}_1-\mathbf{p}_0)+\sigma(\mathbf{p}_2-\mathbf{p}_1)
\end{aligned}
$$

The new blending functions, however, are neither symmetrical nor multiplicative. A second attempt to cast the Bezier form in a more manageable affine form is done by choosing new control points $\mathbf{r}_i$; this results in different blending functions. If the new control points $\mathbf{r}_i$ are chosen as follows,

$$
\begin{aligned}
\mathbf{r}_0 &= \mathbf{p}_0 \\
\mathbf{r}_1 &= \mathbf{p}_0+2(\mathbf{p}_1-\mathbf{p}_0) \\
\mathbf{r}_2 &= \mathbf{p}_2,
\end{aligned}
$$

B_2 is given by

$$
\begin{aligned}
& B_2(\tau) \\
& = \{ \text{ calculus } \} \\
& \mathbf{r}_0+\sqrt{\sigma}(\mathbf{r}_1-\mathbf{r}_0)+\sigma(\mathbf{r}_2-\mathbf{r}_1)
\end{aligned}
$$

These new blending functions are multiplicative but not symmetrical. Nevertheless distribution functions for these blending functions can be found (see property 3.37) and are, for instance, given, for all $i\in[0..k]$ and $k\in\mathbb{N}_0$ by

$$
\begin{aligned}
f_{0,k}(i) &= i \\
f_{1,k}(i) &= 0
\end{aligned}
$$

These functions, however, do not fulfill the requirement for consecutive weaving, $F_{0,k}(k)=F_{1,k}(k)$. They can be changed according to the same property 3.37 to

$$f_{0,k}(i)=2i$$
$$f_{1,k}(i)=k$$

and these functions are such that, according to property 3.11(a), $F_{0,k}(k) = F_{1,k}(k) = k(k+2)$.
A similar deduction for a third order Bezier curve results in

$$\begin{aligned}
&B_3(\tau)\\
&= \{ \text{ definition of 3rd order Bezier curve } \}\\
&(1-\tau)^3\mathbf{p}_0+3\tau(1-\tau)^2\mathbf{p}_1+3\tau^2(1-\tau)\mathbf{p}_2+\tau^3\mathbf{p}_3\\
&= \{ \text{ calculus } \}\\
&\mathbf{r}_0+\tau(\mathbf{r}_1-\mathbf{r}_0)+\tau^2(\mathbf{r}_2-\mathbf{r}_1)+\tau^3(\mathbf{r}_3-\mathbf{r}_2)\\
&= \{ \text{ reparameterisation with } \sigma=\tau^3 \}\\
&\mathbf{r}_0+\sigma^{\frac{1}{3}}(\mathbf{r}_1-\mathbf{r}_0)+\sigma^{\frac{2}{3}}(\mathbf{r}_2-\mathbf{r}_1)+\sigma(\mathbf{r}_3-\mathbf{r}_2)
\end{aligned}$$

with

$$\begin{aligned}
\mathbf{r}_0&=\mathbf{p}_0\\
\mathbf{r}_1&=\mathbf{p}_0+3(\mathbf{p}_1-\mathbf{p}_0)\\
\mathbf{r}_2&=\mathbf{p}_0+3(\mathbf{p}_2-\mathbf{p}_1)\\
\mathbf{r}_3&=\mathbf{p}_3.
\end{aligned}$$

From property 3.37 we can see that these functions can be generated by the following primitive functions.

$$F_{0,k}(x):=\lfloor C_{0,k}x^3 \rfloor$$
$$F_{1,k}(x):=\lfloor C_{1,k}x\sqrt{x} \rfloor$$
$$F_{2,k}(x):=\lfloor C_{2,k}x \rfloor$$

for appropriate values of $C_{i,k}$. $F_{0,k}(k)=F_{1,k}(k)=F_{2,k}(k)$ can now be established by choosing:

$$C_{0,k}=\frac{1}{k\sqrt{k}}$$
$$C_{1,k}=1$$
$$C_{2,k}=k\sqrt{k}$$

Given a primitive function the corresponding distribution function can be readily computed (see definition 3.10 of primitive functions) by

$$f_{0,k}(0)=F_{i,k}(0)$$
$$f_{i,k}(i)=F_{i,k}(i)-F_{i,k}(i-1)-1, \text{ for } i \in [1..k]$$

In figure 4.10 the results of using these distribution functions are shown.

□

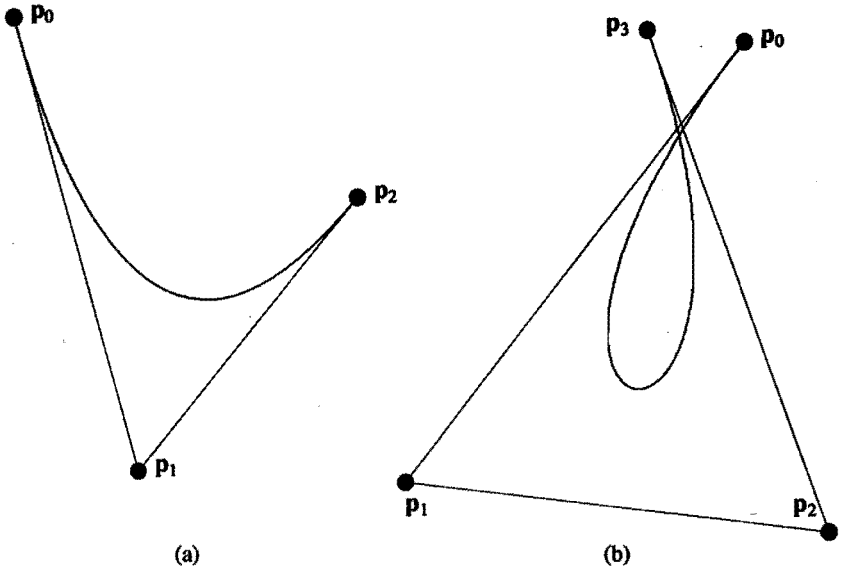


Figure 4.10: Example 4.8
Bezier curves obtained by simultaneous weaving.

4.3 Canonical w-curves

In the previous sections the weave operator is applied to compute a discrete curve from a set of control chains. Distribution functions are used to control the final result of weaving. Independent of what distribution functions are chosen, some relations between the control chains and the resulting chain after weaving can be stated.

Consider the following situation: let $\mathbf{c}_i$ be control chains, for $i \in [0..n)$; we say the codes of chain $\mathbf{c}_i$ have colour i. Let $\mathbf{r}_c$ be the result of weaving the control chains $\mathbf{c}_i$ using a given set of distribution functions. The chain $\mathbf{r}_c$ has the following two properties.

(a) The chain $\mathbf{r}_c$ equals $\mathbf{c}_i$ after all codes of colours different from i are removed.

(b) The colouring of $\mathbf{r}_c$ is independent of the actual codes in the control chains. It is, however, influenced by the length of the control chains.

Hence, the chain $\mathbf{r}_c$ can be constructed from the colouring s of $\mathbf{r}_c$ and the control chains $\mathbf{c}_i$, by replacing, according to (a), all codes of colour i by the codes of chain $\mathbf{c}_i$. Note that according to (b) the chain $\mathbf{r}_d$, for control chains $\mathbf{d}_i$ with $|\mathbf{d}_i| = |\mathbf{c}_i|$, can also be constructed from the colouring s of $\mathbf{r}_c$. For control chains $\mathbf{d}_i$ with $|\mathbf{d}_i| \leq |\mathbf{c}_i|$, s can also be used to compute $\mathbf{r}_d$; this is simply accomplished by lengthening the chains $\mathbf{d}_i$ to the length of chain $|\mathbf{c}_i|$ by means of the lengthen operator. This does in general not result in exactly the same chain $\mathbf{r}_d$ (not even after removing the added 8-codes); the corresponding continuous curves, however are the same.
Hence, the chain s may be used to compute (an approximation of) $\mathbf{r}_d$ for all control

chains $\mathbf{d}_i$ with $|\mathbf{d}_i| \leq |\mathbf{c}_i|$.
Below a definition of a discrete curve based on the colouring of a chain is given; such a colouring is called a canonical chain.

4.9 **Definition** : canonical chain

A canonical chain s (of order n) is a finite sequence of colours, elements of $[0..n]$. An element $\mathbf{s}_i$ of a canonical chain is called a colour code.

□

4.10 **Definition** : canonical weave curves

Given a canonical chain s containing n colours and N codes of each colour; given the control points $\mathbf{p}_i$ $(i \in [0..n])$ with $|bresh\,8(\mathbf{p}_{i+1}-\mathbf{p}_i)| \leq N$ $(i \in [0..n))$, the canonical weave curve $CA(\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_n, \mathbf{s})$ of order n is defined as the discrete curve $DC(\mathbf{p}, \mathbf{e})$ with the point p and the chain e given by

$$\mathbf{p}=\mathbf{p}_0$$
$$(\forall k, j, i : k \in [0..|s|) \wedge \mathbf{s}_k = i \wedge \#_i(\mathbf{s}, 0, k) = j : \mathbf{e}(k) = \mathbf{c}_i(j))$$

with $\mathbf{c}_i = lengthen(bresh\,8(\mathbf{p}_{i+1}-\mathbf{p}_i))$ $(i \in [0..n))$.

□

Computation of a canonical weave curve of order n takes the following steps:

(0) colour mix selection
computation of a canonical chain s with n different colours and $|\mathbf{s}|=nN$. This may be done using a w-curve algorithm but other methods are also allowed. N should at least be the size of the largest possible control chain. This may be considered as a preprocessing step; it needs to be done only once.

(1) geometry computation
Compute the control chains and make them of length N with the lengthen operator.

(2) chain construction
Replace every colour code in s with the next code of the (lengthened) control chain of the same colour.

The preprocessing may be relatively expensive, the steps (1) and (2), however, have a computational complexity linear to N.

We come to the following conclusions.

(1) Canonical curves make the inefficient weave algorithms worthwhile.

(2) In fact canonical curves are an abstraction from weave curves, since all kind of algorithms can be used to generate a canonical chain of colours.

(3) The use of the lengthen operator to obtain a chain of length N seems inefficient in memory usage and seems to introduce overhead in computing the chain. However, the memory usage is relatively small compared to that of the frame buffer. The overhead introduced by lengthening the curve to length N may be reduced by storing also canonical chains of length $N/2$, $N/4$ etc. This solution only doubles the memory usage but limits this overhead to the length of the control chain.

5

Filling of closed discrete curves

5.0 Introduction

In this chapter an algorithm for drawing the interior and the boundary of a closed discrete curve is given. For this purpose a well-known scan-conversion algorithm for polygons is adapted. (See for instance, [Fol90]). Its general principles are as follows.

(a) compute all intersections of the polygon with all (relevant) scan lines and sort these intersections for every scan line from left to right.

(b) fill the scan lines between the found intersection points according to some filling rule. Several of these rules exist; the non-zero winding rule is used here. Another well-known rule is the even-odd rule.

Different from normal scan line filling routines is that there is no need for computations of intersection points; hence, no problem with inaccurate computations. It is suitable for every discrete curve. It resembles more polygon filling using edge coherence, in that it efficiently computes the intersection point with the next scan line. Moreover this algorithm is suitable for processing chains since it uses all codes once and in the sequence as they occur in the chain.

5.1 Filling a closed discrete curve

First of all we define the notion of incidence of a discrete curve and a horizontal scan line. A discrete curve is incident with a scan line if it has points on the scan line. An incidence may be an intersection. A curve intersects a scan line if it has points both below and above the scan line. An intersection may be described with a 6-tuple containing the following items:
- i, j: the indices of the first and last point on the scan line
- *min*, *max*: the x-coordinate of the leftmost and rightmost point on the scan line
- y: the y-value of the scan line
- d: the direction of intersection: $d \in \{up, down\}$

An incidence is also described by such a 6-tuple. It can take one other form, namely a local maximum or minimum, in these cases dir is *hor*. Figure 5.1 shows all the three

possible cases for an incidence with respect to its direction.
The formal definition of incidence given below uses the expression $dir(\mathbf{c}, j)$ to indicate the direction of the code $\mathbf{c}(j \bmod |\mathbf{c}|)$; *dir* is defined as follows.

5.1 Definition : $dir(\mathbf{c}, j)$

For all chains $\mathbf{c}$ and all $j \in \mathbb{Z}$ $dir(\mathbf{c}, j)$ is defined as follows.

$$dir(\mathbf{c}, j) := \begin{cases} up & \text{if } v_y(\mathbf{c}(j \bmod |\mathbf{c}|)) = 1 \\ hor & \text{if } v_y(\mathbf{c}(j \bmod |\mathbf{c}|)) = 0 \\ down & \text{if } v_y(\mathbf{c}(j \bmod |\mathbf{c}|)) = -1 \end{cases}$$

□

5.2 Definition : incidence

An incidence of a discrete closed curve $DC(\mathbf{p}, \mathbf{c})$ with scan line y is a a 6-tuple (i, j, min, max, y, d) with

$$i \in [0 .. |\mathbf{c}|) \wedge j \in (i .. \infty)$$
$$min, max \in \mathbb{Z}$$
$$d \in \{up, hor, down\}$$

and

$$dir(\mathbf{c}, i) \neq hor \wedge (\forall k : k \in (i .. j) : dir(\mathbf{c}, k) = hor) \wedge dir(\mathbf{c}, j) \neq hor$$

$$min = min\{\mathbf{p}_x + \mathbf{p}_x(\mathbf{c}, k) \mid k \in (i .. j]\}$$
$$max = max\{\mathbf{p}_x + \mathbf{p}_x(\mathbf{c}, k) \mid k \in (i .. j]\}$$

$$d = \begin{cases} dir(\mathbf{c}, i) & \text{if } dir(\mathbf{c}, i) = dir(\mathbf{c}, j) \\ hor & \text{if } dir(\mathbf{c}, i) \neq dir(\mathbf{c}, j) \end{cases}$$

□

Horizontal closed discrete curves, curves with points on only one scan line, have according to the above definitions no incidences. Computing of the interior and the boundary pixels of a horizontal curve is trivial. In the algorithms below only non-horizontal curves are considered.

First an algorithm is given for computing all incidences of a discrete curve $DC(\mathbf{p}, \mathbf{c})$ with the scan lines with numbers y in a given integer interval, say $[0..Y]$. This algorithm also sorts these incidences per scan line y on their x-coordinates in a list $S[y]$ of incidences.

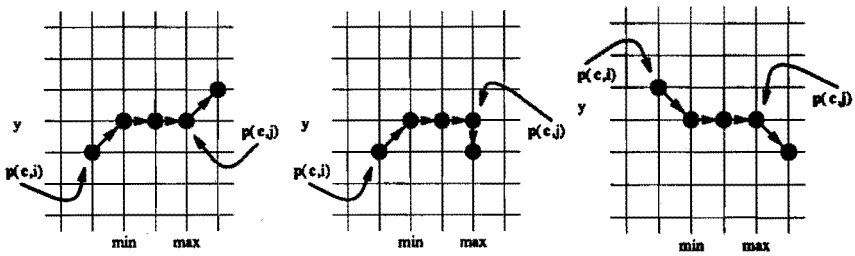


Figure 5.1: three types of incidences
(a) intersection with $d = up$
(b) incidence with $d = hor$
(c) intersection with $d = down$

Hence, with predicate **R** given by

R All incidences I of $DC(\mathbf{p}, \mathbf{c})$ with $I.i \in [0\,..\,|\mathbf{c}|)$ and $I.y \in [0\,..\,Y]$, are added to S.

we can write down the following invariants.

P All incidences I of $DC(\mathbf{p}, \mathbf{c})$ with $I.i \in [0\,..\,i)$ and $I.y \in [0\,..\,Y]$, are added to S

$\mathbf{P_0}$ $dir(\mathbf{c}, i) \neq hor \wedge (\forall k : k \in (i\,..\,j) : dir(\mathbf{c}, k) = hor)$

$\mathbf{P_1}$ $min = min\{\mathbf{p}_x + \mathbf{p}_x(\mathbf{c}, k) \mid k \in (i\,..\,j]\} \wedge$
$max = max\{\mathbf{p}_x + \mathbf{p}_x(\mathbf{c}, k) \mid k \in (i\,..\,j]\}$

$\mathbf{P_2}$ $(x, y) = \mathbf{p} + \mathbf{p}(\mathbf{c}, j)$

The assumption that the curve $DC(\mathbf{p}, \mathbf{c})$ is not horizontal implies that at least two codes in **c** have a non-horizontal basic vector. Using the invariants $\mathbf{P}_i$ we come to the following program for the calculation of the array S. In it we use the procedure *Add_incidence_to_S* that is informally specified by

```
add_incidence_to_S(i, j, min, max, y, d)
```

```
i, (x, y) := 0, p;
do dir(c, i)=hor   →
     (x, y) := (x, y)+v(c_i);
     i:=i+1
od;
(x, y) := (x, y)+v(c_i);
j, min, max:=i+1, x, x; { P ∧ P_0-2 }
do i< | c | →
     if dir(c, j)=hor→
          (x, y) := (x, y)+v(c_j mod n);
          if min>x→min:=x [] max≤x→max:=x fi;
          j:=j+1
     [] dir(c, j)≠hor→
          if dir(c, i)≠dir(c, j) → d:=hor
          [] dir(c, i)=dir(c, j) → d:=dir(c, i)
          fi;                                   is an incidence }
          { P_0-2 ∧ dir(c, j)≠hor => (i, j, min, max, y, d)
          add_incidence_to_S(i, j, min, max, y, d);
          (x, y) := (x, y)+v(c_j mod n);
          i, j, min, max:=j, j+1, x, x
     fi { P ∧ P_0-2 }
od { R }
```

After computing the incidences we can start filling the curve. As mentioned before the non-zero winding rule is used for determining which points are inside the curve and which are not. The rule can roughly be stated as follows:

> a point **p** is inside a closed curve if the number of intersections of the curve with a half line starting at **p**, differs from zero. The number of intersections is counted by adding 1 if the half line is intersected in one direction (*up*) and -1 if it is intersected in the other direction (*down*).

In our case this half line is chosen to be a horizontal half line starting at the point **p** and going to the left. Our definition of winding number can, hence, be given as follows.

5.3 **Definition** : winding number for points

A winding number $wnr(x,y)$ for a point (x, y) is defined as

$$wnr(x,y) :=$$
$$(\mathbf{N}\, I : I \text{ is an incidence in } S[y] \wedge I.min < x \wedge I.dir = up)$$
$$-$$
$$(\mathbf{N}\, I : I \text{ is an incidence in } S[y] \wedge I.min < x \wedge I.dir = down)$$

□

Figure 5.2 illustrates the notion of winding number. Notice that in the point set $DC(\mathbf{p}, \mathbf{c})$ both points with a non-zero winding number and points with a zero winding number can exist. Using winding numbers the region of a curve $DC(\mathbf{p}, \mathbf{c})$ is defined as follows.

5.4 **Definition :** region

$$region(\mathbf{p}, \mathbf{c}) := \{(x,y) \in \mathbb{Z}^2 \mid wnr(x,y) \neq 0\} \cup DC(\mathbf{p}, \mathbf{c})$$

□

It is sufficient to know winding numbers at the leftmost points of incidences, since the points inbetween two succesive leftmost points have the same winding numbers. Instead of using winding numbers as defined above, we therefore define winding numbers for incidences and use these new numbers for computing a region. A winding number for incidences is defined as the winding number of its leftmost point (min, y). If two incidences on the same scan line, however, have the same *min*, the value of their winding number depends on their order in the list $S[y]$; remember that the list $S[y]$ of incidences is sorted on min. In the next definition we denote With $\#L$ the number of elements of the list L. For convenience sake we define a winding number for a non-existing incidence $L[\#L]$; for (closed) curves this winding number is always 0.

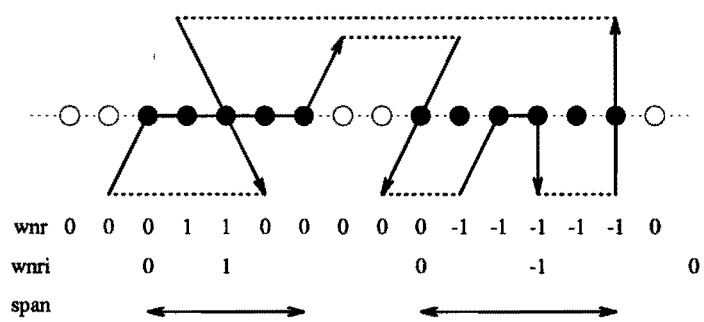


Figure 5.2:

5.5 **Definition :** winding number for incidences

For the j-th incidence ($j \in [0..\#L]$) in a sorted list L of incidences the winding number $wnri(L, j)$ is defined by

$$wnri(L, j) :=$$
$$(\mathbf{N}\, k : k \in [0..j) \wedge L[k].dir = up)$$
$$-$$
$$(\mathbf{N}\, k : k \in [0..j) \wedge L[k].dir = down)$$

□

The winding numbers of incidences are such that if (x, y) lies between two begin points of consecutive incidences with the scan line y, the winding number of (x, y) is the winding number of the rightmost incidence. Using this property of *wnri* a span is defined as an interval of points on a scan line. A span is a 4-tuple (i, j, min, max). It starts at incidence $S[y][i]$ and ends at an incidence $S[y][j]$. The maximum and minimum x-coordinates for the spans $S[y][i]$ and $S[y][j]$ are *min* and *max*, respectively. The indices i and j are such that the scan line y lies for the x-coordinates in $[min..max]$ in the *region* (p, c).

5.6 **Definition :** span

A span of scan line y is a tupel (i, j, min, max) with

$$0 \leq i \leq j < \#S[y]$$
$$min, max \in \mathbb{Z}$$

and

$$wnri(S[y], i) = 0 \wedge$$
$$(\forall k : k \in (i..j] : wnri(S[y], l) \neq 0) \wedge$$
$$wnri(S[y], j+1) = 0$$

$$min = S[y][i].min$$
$$max = max\{S[y][k].max \mid k \in [i..j]\}$$

□

The following property indicates that it suffices to compute and draw all spans for all scan lines $y \in [0..Y]$.

5.7 **Property :**

$$region(\mathbf{p}, \mathbf{c}) \cap \{(x, y) \mid x \in \mathbb{Z} \wedge y \in [0..Y]\}$$
$$=$$
$$\{(x, y) \mid y \in [0..Y] \wedge (\exists s : s \text{ is a span of scan line } y : x \in [s.min..s.max])\}$$

□

An algorithm for computing spans and drawing them using the *drawspan* function is given below. It consists of two nested loops one over all scan lines and one over all incidences of the scan line with the curve. The invariant for the outer loop is given by

Q The spans of the scan line with numbers in $[0..y)$ are drawn
$\wedge\ wnri = 0$

The invariants of the inner loop are:

$\mathbf{P}$ All spans s of scan line y with $s.j \in [0..j)$ have been drawn.
$\mathbf{P_0}$ $i \in [0..j]$
$\mathbf{P_1}$ $wnri(S[y],i)=0 \wedge (\forall k : k \in (i..j] : wnri(S[y],j) \neq 0)$
$\mathbf{P_2}$ $wnri = wnri(S[y],j)$
$\mathbf{P_3}$ $max = max\{S[y][k].max \mid k \in [i..j)\}$

```
y, wnri := 0, 0; { Q }
do y<Y →
    i, j, max := 0, 0, -inf;    { P ∧ P0-3 }
    do j<>#S[y] →
        s := S[y][j];
        if s.max>max → max := s.max fi;
        if s.dir=up  → wnri := wnri+1
        [] s.dir=hor → skip
        [] s.dir=down → wnri := wnri-1
        fi;   { P0 ∧ P1 ∧ P2-3 (j, j+1) }
        if wnri=0 → { (i, j, S[y][i].min, max) is a span }
            drawspan(S[y][i].min, max, y);
            i, max := j+1, -inf
        fi;
        j := j+1    { P ∧ P0-4 }
    od;
    y := y+1 { Q }
od
```

The above algorithm can be improved to draw less points by adding a variable which keeps track of which part of the current scan line has already been drawn. This prevents that a pixel is drawn twice by *drawspan*. The time complexity of the total algorithm is dominated by the number of pixels which must be drawn by the *drawspan* functions. The number of times a *drawspan* function is called is majored by the number of incidences.

6

Thick Curves

6.0 Introduction

Page description languages, e.g. PostScript [Adobe] , use drawing primitives like lines, circles, Beziers etc. These languages often not only support the rendering of these primitives but also the rendering of thick versions of these primitives; that is the same primitive, but drawn as if a brush of some form and a given thickness is used instead of a pen; a pen is a brush with thickness one pixel. Two standard forms for brushes are a circle and a rectangle. If a circular brush is used, a mathematical description of a thick curve of thickness d reads: the set of points with distance at most d to a point on the original curve. The original curve is called the *center curve.* If a rectangular brush is used a mathematical description of such a thick curve is the set of all points within a perpendicular distance d of the center curve. These two descriptions are not equivalent as can be seen in figure 6.1.

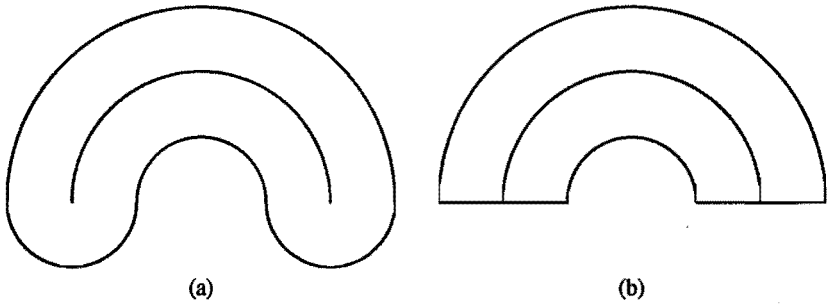


Figure 6.1: border of a thick circle segment
(a) circular brushed (b) rectangular brushed

Several techniques exist for drawing thick primitives. Three are discussed shortly below. A more detailed summary of these and other methods is given in [Fol90] .

(0) For some primitives like circles and lines a thick curve can be drawn by filling between an inner and outer primitive, which both might happen to be a drawing primitive. However, for some primitives, like ellipses, the inner and outer curve are hard to compute and are certainly not drawing primitives.

(1) Another technique often used is based on transforming the primitive to a polyline representation and subsequently drawing the thick edges of the polyline as filled rectangles. Several ways of filling the cracks between the rectangles exist.

(2) An elegant method for circular brushing *discrete* curves is given in [Pos89]. It is based on drawing a filled circle for each pixel on its center curve and is suitable for all curves given by a 4- or 8-connected chain.

Notice that a circular-brushed curve is a rectangular-brushed curve with at the endpoints, if any, a filled circle. For this reason, only rectangular brushing is considered in the sequel. In this chapter an algorithm for drawing a thick representation of a center curve, using a rectangular brush, is given. The center curve may be any discrete curve and in contrast with the method in [Pos89] the set of pixels forming the thick curve will be generated in horizontal spans instead of pixel for pixel. The algorithm computes the two curves at distance d of the center curve and uses the fill algorithm as given in the previous chapter for filling a region between these curves, called *offset curves*. Offset curves are not only of interest as an intermediate result in computing thick curves but also play a role in design of e.g. fonts. In [Far89] a method is given for approximating the offset curve of a conic curve by another conic curve.

In section 6.1 we give an algorithm for producing the offset curves. In section 6.2 this chapter is concluded with an algorithm for computing thick curves.

6.1 Offset curves

Let C be a continuous curve with $\mathbf{n}(x, y)$ for every point $(x,y) \in C$ a normal vector of C with given handiness, assuming that these normal vectors exist. An offset curve OC at distance d of C can now be defined by

$$OC := \{ (x,y) + d\,\mathbf{n}(x, y) \mid (x,y) \in C \} \qquad \text{(OC)}$$

Another offset curve can be obtained by subtracting $d\,\mathbf{n}(x, y)$ rather than adding it to (x,y). For some continuous curves offset curves can be readily computed; for others, they are rather complex. Circles, for instance, have circles as offset curves; the offset curves of ellipses (see figure 6.2), on the other hand, can be shown to be 8-th order polynomials ([Fol90] attributes this result to [Sal96])

Our concern is not with continuous curves but with discrete curves (given by chains). The above definition of offset curves cannot immediately be extended to discrete curves since the notion of normal vectors is not defined for discrete curves. There are only 4 basic vectors of Euclidean length 1. In order to have more possible directions we ignore the length requirement for discrete normal vectors and only require that they are perpendicular to the discrete curve. In section 6.1.2 we define what we consider to be perpendicular to a discrete curve.

Let $\mathbf{n}_d(\mathbf{c}, i)$ be a discrete vector along a normal vector of a chain $\mathbf{c}$ and let it be a

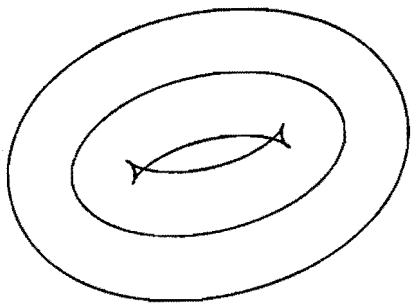


Figure 6.2: an ellipse and its two offset curves

vector of approximated length d. A definition for a set of points OD, similar to the definition of OC may now be given by

$$OD := \{ \mathbf{p}(\mathbf{c}, i) + \mathbf{n}_d(\mathbf{c}, i) \mid i \in [0 .. |\mathbf{c}|] \}. \qquad \text{(OD)}$$

Notice that OD is in general not a connected set and hence, not a discrete curve. In section 6.1.3 interpolation schemes are given to make the set OD connected.

6.1.2 Discrete normal vectors

In this section a definition is given for normal vectors of points on a discrete curve. Considering the continuous curve obtained by connecting the points of a discrete curve by straight line segments, it is obvious that in most of these points a (continuous) normal vector does not exist.
Supplying a normal vector for every edge of a discrete curve can easily be done by rotating the basic vector $\pi/2$ radians. Using these normal vectors for the computation of offset curves, however, does not give the desired result since we are interested in a discrete representation of a continuous thick curve. Hence, the normal vectors have to be defined in such a way that they resemble the vectors $d\mathbf{n}(x, y)$ of equation (OC). This can be done in several ways, since a discrete curve is a discretisation of an infinite number of continuous curves.
In the definition below a discrete normal vector for a point on a chain is defined by rotating an approximation of a difference vector, which plays the role of a tangent vector, by $\pi/2$ radians. This rotation is accomplished by adding 2 to every code involved (see example 2.2). For most points $\mathbf{p}(\mathbf{c}, i)$ the difference vector is simply given by $\mathbf{p}(\mathbf{c}, i+n) - \mathbf{p}(\mathbf{c}, i-n)$, for some appropriate n. For points near the beginning or end of the curve this definition must be slightly adapted. In the definition below a closed curve is also considered as a special case.

6.1 **Definition** : (discrete) normal vector

For all chains $\mathbf{c}$, $i \in [0 .. |\mathbf{c}|]$, and $n \in \mathbb{N}_1$, a normal vector $\mathbf{n}(\mathbf{c}, i)$ of order n at the i-th point $\mathbf{p}(\mathbf{c}, i)$ of chain $\mathbf{c}$ is defined by

$$\mathbf{n}(\mathbf{c}, i) := (\Sigma k : k \in [max\{0, i-n\} .. min\{|\mathbf{c}|, i+n\}) : R(v(\mathbf{c}_k)))$$

where R is a rotation over $\pi/2$ radians.
For closed chains $\mathbf{c}$ the definition reads

$$\mathbf{n}(\mathbf{c}, i) := (\Sigma k : k \in [i-n .. i+n) : R(v(\mathbf{c}_{k \bmod |\mathbf{c}|})))$$

□

The choice of the order n of a normal vector is influenced by the following two arguments.

(1) If we choose to use normal vectors of low order, the number of distinct vectors is limited and so the number of directions these vectors can point in, is limited (see figure 6.3).

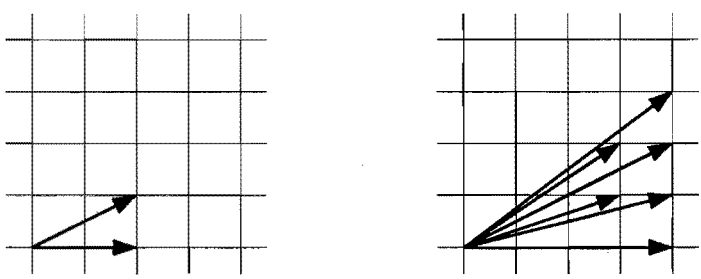


Figure 6.3 : possible directions for normal vectors in octant 0
(a) normal vectors of order 1
(b) normal vectors of order 2

Let m be this number of directions in octant O_0. An angle of at least $2\pi/(8m)$ radians between adjacent directions exist. Notice that the largest gap between two adjacent directions is directly related to n and given by $arctan(\frac{1}{2n})$, the angle between $(1, 0)$ and $(2n, 1)$. Hence, the Euclidean distance of these vectors after scaling them to length d, is majored by the arc length $d\,arctan(\frac{1}{2n}) = \frac{d}{2n} + O(\frac{1}{n^3})$. Hence, the maximum distance of adjacent points on the offset curves becomes smaller if n becomes larger.
The behaviour of m for $n \to \infty$ is given by the following number theoretic result (see for instance [Bak84]) ($gcd(i, j)$ denotes the greatest common divisor of i and j)

$$m = (\mathrm{N}(i, j) : (i, j) \in O_0 \wedge i \leq 2n \wedge gcd(i, j) = 1) = \frac{3}{\pi^2}(2n)^2 + O(n\log(n))$$

(2) By choosing a large order the discrete normal vector may differ more from the expected normal vector in that point. In case of a circle this effect cannot be observed, since the difference between every two points on this curve is a "good" approximation for normal vectors in the point in the middle. In figure 6.4 the effect is demonstrated for a curve other than a circle. This effect is most annoying if it occurs at the beginning of a (not-closed) chain.

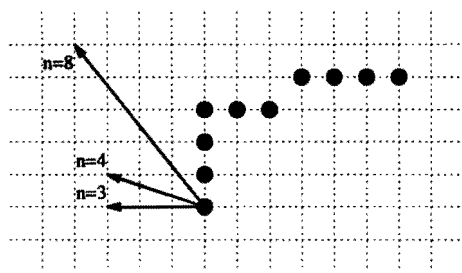


Figure 6.4 : normal vectors of different order in start point of the curve

Argument (1) and (2) contradict each other and in practice a compromise must be made between resolution measured in the number of possible normal vector directions and accuracy in approximating the 'true' normal. For most cases values of n near 10 turn out to be adequate.

A normal vector $\mathbf{n}(\mathbf{c}, i)$ of order n may well be (0, 0) according to its definition. To prevent this we require that the chain $\mathbf{c}$ is a smoothed chain (see section 3.4.) and adapt the definition of a normal vector such that if the normal vector is (0, 0) according to the definition 6.1, the largest order normal vector of order smaller than n, and different from (0,0), is chosen. Such a vector clearly exists if the chain is smooth.

6.1.2.1 Scaling of discrete normal vectors

Scaling a discrete normal vector $\mathbf{n}(\mathbf{c}, i)$ to the vector $\mathbf{n}_d(\mathbf{c}, i)$ of approximately length d may be done by simple floating point arithmetic. Using integer arithmetic it may be done by computing a table of scaled vectors as a preprocess and, subsequently, applying a simple table lookup. This table should contain an entry for all $(4n+1)^2-1$ possible normal vectors of order n. Using symmetry the size of the table can be reduced to $(2n+3)n$, the number of points in $\{(i, j) \mid i \in (0..n] \wedge j \in [0..i]\}$. This preprocessing can simply take place by comparing all possible normal vectors to points on a discrete circle with radius d and storing the closest point in the table.

6.1.3 Interpolation

The set of points *OD* of an offset curve, which can be generated given the scaled normal vectors $\mathbf{n}_d(\mathbf{c}, i)$, is in general not a connected set. Hence, we need to add points to it to make it a discrete curve. These points are added by interpolating the points of *OD*. Several interpolation schemes can be used, the most obvious of which are: interpolation with line segments and interpolation with circle segments. Both schemes can be implemented easily.

6.1.3.1 Line segments

Interpolating the points of *OD* with line segments is done by computing the chains *bresh*$8(\mathbf{q}_1-\mathbf{q}_0)$, where $\mathbf{q}_1$ and $\mathbf{q}_0$ are two points in *OD* belonging to two consecutive points of the curve. The concatenation of all these chains forms a discrete curve for an offset curve at distance *d*. If the gap between $\mathbf{q}_0$ and $\mathbf{q}_1$ is big, interpolation with line segments does not look nice. Moreover linear interpolation does not accord with the fact that we compute normal vectors in the end points of the edges and not on the edges. Indeed the offset curve of an edge is a straight line while, on the other hand an offset curve for a point is a circle.

6.1.3.2 Circle segments

Interpolating two points $\mathbf{q}_0$ and $\mathbf{q}_1$ on an offset curve with a circle segment is ambiguous, since an infinite number of interpolating circle segments exist. In our case we can simply choose the following solution (see Figure 6.5): let $\mathbf{p}_0=\mathbf{p}(\mathbf{c}, i)$ and $\mathbf{p}_1=\mathbf{p}(\mathbf{c}, i+1)$ be the points on the center curve corresponding to $\mathbf{q}_0$ and $\mathbf{q}_1$, respectively, on the offset curve. From the (integer) scaling method as described above, we know that $\mathbf{q}_i-\mathbf{p}_i$ is a point on the circle with center (0, 0) and radius *d*. Let $\bar{\mathbf{c}}$ be the chain segment representing the circle arc between $\mathbf{q}_0-\mathbf{p}_0$ and $\mathbf{q}_1-\mathbf{p}_1$. We can use the chain $\bar{\mathbf{c}}\underline{w}[\mathbf{c}_i]$, or any other combination of the same codes for interpolating $\mathbf{q}_0$ and $\mathbf{q}_1$. By choosing this specific combination $\mathbf{c}_i$ is inserted somewhere in the middle of $\bar{\mathbf{c}}$ and so a situation is obtained in which the offset curve in $\mathbf{q}_0$ is a circle segment of a circle with center $\mathbf{p}_0$ and radius *d*, in $\mathbf{q}_1$ it is a circle segment with center $\mathbf{p}_1$. This situation fits in perfectly with the fact that we compute the normal vectors on the points of the center curve and not on the edges of its chain.

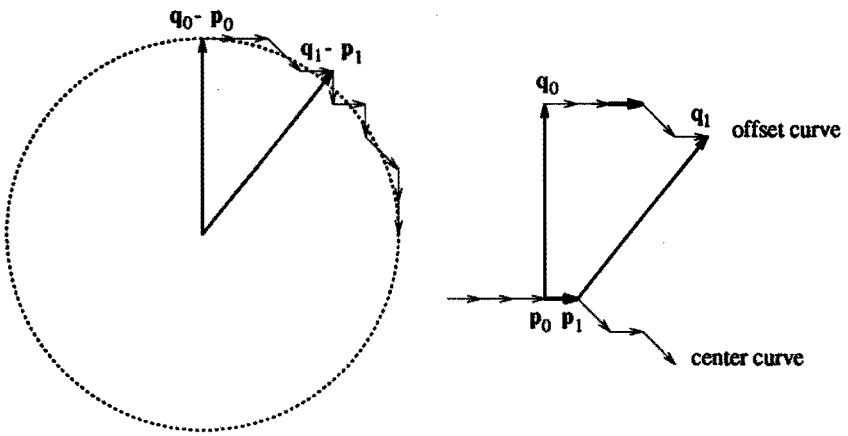


Figure 6.5 : interpolation in an offset curve using circle segments

6.2 Thick curves

A seemingly obvious algorithm for drawing thick curves takes approximately the following steps:

algorithm I :

(a) compute the offset curves

(b) produce a closed curve by concatenating the offset curves; that is, add the lines joining the begin and end points of the offset curves to the begin and end point of the center curve, respectively.

(c) fill this closed curve.

At first glance, this algorithm seems to work properly. Indeed, in many cases this method yields a correctly drawn thick curve. And by using the scan conversion algorithm of section 5.1 for filling a closed discrete curve, it is even a fast method. As we show in some examples below, it is, however, not a correct solution in general.

Firstly we show an example which demonstrates that only knowing the offset curves is not enough for correctly producing a thick curve by filling the area between the offset curves.

Consider a circle with radius r. The inner offset curve belonging to the thicknesses $d<r$ equals the inner offset curve for thickness $2r-d$ (see figure 6.6). Giving just these curves to a filling algorithm would yield at least for one of these two cases an incorrect solution. This clearly shows that offset curves contain not enough information.

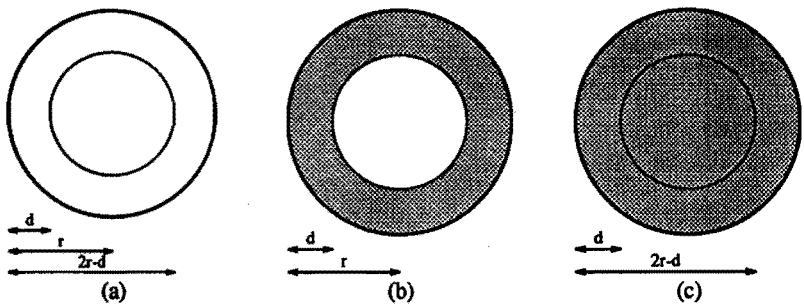


Figure 6.6:

(a) circle with radius r and the same inner offset curves for the thicknesses d and $2r-d$.
(b) filled between center curve and offset curve for thickness d
(c) as (b) for thickness $2r-d$

As a second example showing the failure of filling between offset curves, we use a center curve which is a concatenation of two quarter circles with radius r, as shown in figure 6.7(a). Computing its offset curves results in the curves of figure 6.7(b).

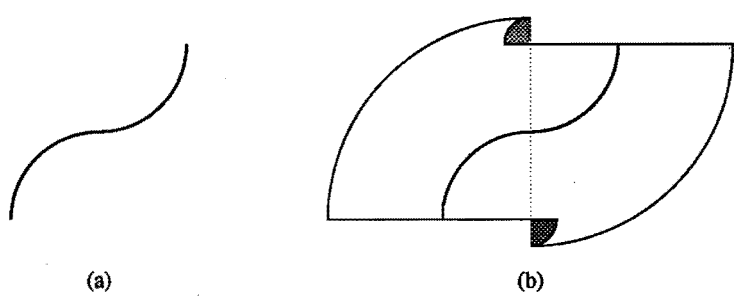


Figure 6.7:

(a) center curve
(b) the corresponding offset curves; the gray areas should be brushed but are not in the area bounded by the offset curves.

Filling these offset curves again produces the wrong result. The two gray areas in the figure should have been drawn, but clearly are not.
From this example we conclude that offset-curves do not (always) determine the complete boundary of the corresponding thick curve.

If in the above examples the thickness for the curves is chosen smaller than the radius of curvature for every point on the center curve, algorithm I works fine.

6.2.1 Computing thick curve

The basic flaw in algorithm I is a result of the fact that it does not use information on which points on the center curve and offset curve are associated. This had for instance the effect that the gray areas in figure 6.7 were, incorrectly, not drawn. Algorithm II circumvents this flaw by filling two quadrangles for each pair of consecutive points on the center curve: one on each side of the center curve. If the center curve points are $\mathbf{p}_0$ and $\mathbf{p}_1$, and their respective points on one of the offset curves are $\mathbf{q}_0$ and $\mathbf{q}_1$, the quadrangle is formed by the curve segments $\mathbf{p}_0\mathbf{p}_1$, $\mathbf{p}_1\mathbf{q}_1$, the interpolation of $\mathbf{q}_1$ and $\mathbf{q}_0$, and $\mathbf{q}_0\mathbf{p}_0$.

In algorithm II we denote with *inpol*$(\mathbf{q}_0,\mathbf{q}_1)$ the interpolation of two points on the offset curve as given in either section 6.1.3.1 or 6.1.3.2. The code $\mathbf{c}_{i-1}+4$ is the code with as basic vector the vector $v(\mathbf{c}_{i-1})$ rotated over π radians (see example 2.2). With *fill*$(\mathbf{p}, \mathbf{c})$ a fill procedure is denoted that fills the closed discrete curve $DC(\mathbf{p}, \mathbf{c})$, e.g. the algorithm given in chapter 5.

Below algorithm II is stated preceded by its postcondition **R** and two invariants $\mathbf{P}_0$ and $\mathbf{P}_1$.

R the center curve $DC(\mathbf{p}, \mathbf{c})$ is drawn with thickness d

$\mathbf{P}_0$ the center curve $DC(\mathbf{p}, [\mathbf{c}(0) \cdots \mathbf{c}(i-1)])$ is drawn with thickness d
$\wedge\, i \in [0..|\mathbf{c}|]$

$\mathbf{P}_1$ $\mathbf{p}_0=\mathbf{p}(\mathbf{c}, i) \wedge \mathbf{q}_0=\mathbf{p}_0+\mathbf{n}_d(\mathbf{c}, i) \wedge \mathbf{r}_0=\mathbf{p}_0-\mathbf{n}_d(\mathbf{c}, i)$

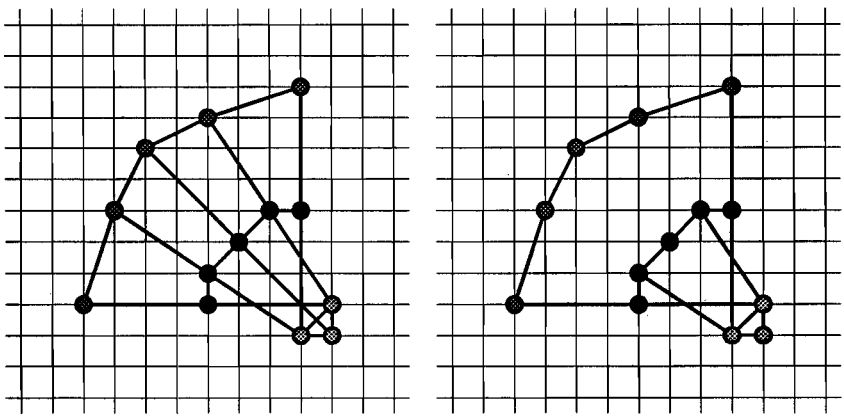
algorithm II :

```
i, p0 := 0, p;
q0, r0 := p0+nd(c, 0), p0−nd(c, 0); { P0 ∧ P1 }
do i< |c| →
      p1, q1, r1 := p0+v(c(i)), p1+nd(c, i), p1−nd(c, i);
      c0, c1 := bresh8(p0, q0), bresh8(q1, p1);
      fill(p0, c0 ⊗ inpol(q0, q1) ⊗ c1 ⊗ [ci−1+4]);
      fill(p1, c1 ⊗ inpol(r0, r1) ⊗ c0 ⊗ [ci−1]);
      i, p0, q0, r0 := i+1, p1, q1, r1 { P0 ∧ P1 }
od { P0 ∧ i= |c| ; hence, R }
```

This algorithm lacks the flaw of the algorithm I, it has, however, the disadvantage that it contains numerous calls to the fill routine, whereas the first algorithm contains only one such call. Thus algorithm I is efficient but wrong, where algorithm II is inefficient but correct. We combine the two algorithms in order to obtain an efficient and correct algorithm.

Algorithm I is more efficient since it fills larger regions; a way for making algorithm II more efficient is by diminishing the number of areas to be filled while still the same points are drawn. For a small curve the quadrangles that are filled by algorithm II are drawn in figure 6.8(a). In 6.8(b) as many as possible consecutive quadrangles

are joined into larger closed curves without loss of interior points.



(a) (b)

Figure 6.8:

(a) the quadrangles between the center curve (●) and its offset curves.

(b) as (a) but with as much as possible consecutive quadrangles joined.

This reduces, in this example, the number of areas from 8 to 4. In the sequel we give some rules for indicating when two quadrangles may be joined.

Let *region*(D) with D a discrete curve $DC(\mathbf{p}, \mathbf{c})$ denote the set *region*$(\mathbf{p}, \mathbf{c})$. We define the sum of two closed discrete curves C and D by given its *region* as follows. (With *wnr*(C, x, y) we denote the winding number *wnr*(x, y) with respect to the discrete curve C)

$$region(C+D) := D \cup C \cup \{(x,y) \in \mathbb{Z}^2 \mid wnr(D, x, y) + wnr(C, x, y) \neq 0\}.$$

Notice, that the region of $C+D$ is the same set as obtained by the algorithm in the previous section if the lists of incidences of C and D are merged (per scan line). Stated otherwise,

$$wnr(C+D, x, y) = wnr(C, x, y) + wnr(D, x, y).$$

We distinguish 3 types of curves. A curve of type 1 is a curve with all the winding numbers of its interior points larger than 0. For type -1 all the winding numbers of the interior points are smaller than 0. Finally, for type 0 both points with winding numbers larger than 0 and points with winding numbers smaller than zero, exist in the interior. In figure 6.9 quadrangles of these 3 types are shown.
The following property holds for curves of the same non-zero type.

$$region(C+D) = region(C) \cup region(D).$$

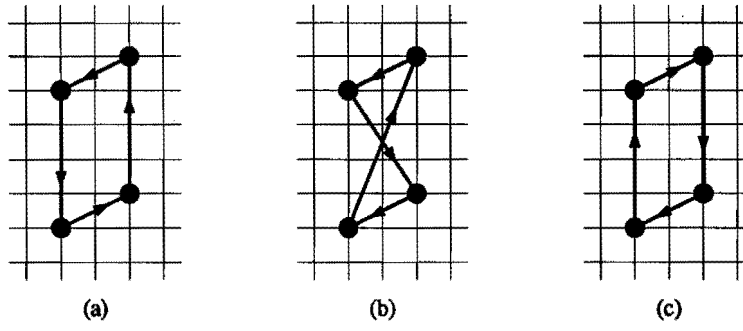


Figure 6.9: quadrangles of type -1 (a), 0 (b), and 1 (c)

Hence, two curves may be added without loss of interior points if they are of the same non-zero type.

We use this property for joining consecutive quadrangles in the thick curve algorithm.

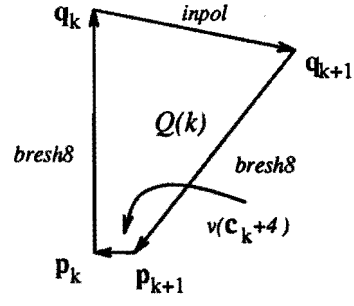


Figure 6.10: quadrangle $Q(k)$

First we define the discrete curve $Q(k)$ as the kth quadrangle in the thick curve algorithm II (see figure 6.10):

$$Q(k) := DC(\mathbf{p}_{k+1}, \mathbf{d}_k \otimes bresh\,8(\mathbf{q}_{k+1}, \mathbf{p}_{k+1})) \text{ with}$$
$$\mathbf{d}_k := [\mathbf{c}_k+4] \otimes bresh\,8(\mathbf{p}_k, \mathbf{q}_k) \otimes inpol(\mathbf{q}_k, \mathbf{q}_{k+1})$$

for $k \in [0\,..\,|\mathbf{c}|]$. Notice that if $Q(k)$ and $Q(k+1)$ are of the same type and the chain $\mathbf{e}_k$ is given by

$$\mathbf{e}_k := inpol(\mathbf{q}_{k+1}, \mathbf{q}_{k+2}) \otimes bresh\,8(\mathbf{q}_{k+2}, \mathbf{p}_{k+2}) \otimes [\mathbf{c}_{k+1}+4],$$

the following holds.

$$
\begin{aligned}
&region(Q(k)+Q(k+1)) \\
&= \\
&region(Q(k)) \cup region(Q(k+1)) \\
&= \\
&region(DC(\mathbf{p}_{k+1}, \mathbf{d}_k \otimes \mathbf{e}_k)),
\end{aligned}
$$

The last equal-sign is based on the fact that the chains $bresh\,8(\mathbf{q}_{k+1}, \mathbf{p}_{k+1})$ and $bresh\,8(\mathbf{p}_{k+1}, \mathbf{q}_{k+1})$ cancel each other out in the computation of *wnr*.
Hence we can compute a discrete curve with the same region as the sum of consecutive quadrangles of the same non-zero type. We use this in algorithm III by filling this discrete curve instead of the individual quadrangles.

6.2.2 Determining the type of a quadrangle

Determining the type of a quadrangle may be done by checking for possible intersecting sides of the quadrangle. We ease this task by dividing the quadrangle along a diagonal in two triangles. From the types of these triangles we compute the type of the quadrangle. The types of the triangles may be simply determined since no intersections occur. In contrast with the definition of type of a curve, as given above, we say that a triangle has type 0 if its three vertices are collinear.

Let t_1 and t_2 be the types of the so obtained triangles. The following algorithm now determines the type t of the original quadrangle.

```
if t1=t2              →  t:=t1
[] t1≠t2 ∧ t2=0       →  t:=t1
[] t1≠t2 ∧ t1=0       →  t:=t2
[] t1=-t2             →  t:=0
fi
```

6.2.3 The algorithm

In this section we give an algorithm for computing thick curves, by filling sets of consecutive quadrangles. The algorithm is simplified in two ways;

(1) only one half of the thick curve is drawn;

(2) the computations of the center curve points $\mathbf{p}_i = \mathbf{p}(\mathbf{c}, i)$ and the offset curve points $\mathbf{q}_i = \mathbf{p}_i + \mathbf{n}_d(\mathbf{c}, i)$ are left out.

Algorithm II does not contain these simplifications and may be used to complete algorithm III.
Invariant $\mathbf{P}_0$ indicates that the first j quadrangles are already filled. $\mathbf{P}_1$ asserts that the quadrangles $Q(j)$ upto $Q(i-1)$ are bounded by the chain $\mathbf{c}$ and $bresh\,8(\mathbf{q}_i, \mathbf{p}_i)$.

$\mathbf{P_0}$ filled area=$(\cup k : k \in [0..j) : region(Q(k)))$
$\wedge j \in [0..|\mathbf{c}|] \wedge i \in [0..|\mathbf{c}|]$

$\mathbf{P_1}$ $region(\mathbf{p}_i, \mathbf{c} \otimes bresh8(\mathbf{q}_i, \mathbf{p}_i)) = (\cup k : k \in [j..i) : region(Q(k)))$

$\mathbf{P_2}$ t=type of $region(\mathbf{p}_i, \mathbf{c} \otimes bresh8(\mathbf{q}_i, \mathbf{p}_i))$

algorithm III :

```
i, j:=1, 0;
c:= [c0+4] ⊗bresh8 (p0, q0) ⊗inpol (q0, q1);
t:=type of Q(0);
do i<|c| →
    tq:=type of Q(i);
    if tq≠t ∨ tq=0→
        fill(pi, c⊗bresh8 (qi, pi));
        c:= [ci+4] ⊗bresh8 (pi, qi) ⊗inpol (qi, qi+1);
        t, j:=tq, i
    [] tq=t ∧ tq≠0→
        c:= [ci+4] ⊗c⊗inpol (qi, qi+1)
    fi;
    i:=i+1
od; { P0-1 ∧ i=|c| }
fill (pi, c⊗bresh8 (qi, pi));
j:=i { P0 ∧ j=|c| }
```

$\mathbf{P_1}$ is kept invariant as can be seen by checking that the following is also an invariant of the loop.

$$\mathbf{c} = rev((\prod k : k \in [j..i) : [\mathbf{c}_k+4])) \otimes bresh8(\mathbf{p}_j, \mathbf{q}_j) \otimes (\prod k : k \in [j..i) : inpol(\mathbf{q}_k, \mathbf{q}_{k+1}))$$

The results of the above algorithm are ambiguous; on the one hand the algorithm produces correctly drawn thick curves, on the other hand it is not as effective as one may hope for. Typically the performance improvement over algorithm II is only 25%; that is 25% less fill operations are performed. However, both discrete curves with a 100% improvement and discrete curves with 0% improvement exist. The lack of performace improvement can be explained by

(1) if for strongly-bended curves one quadrangle is of a non-zero type, its neighbour on the other side is most likely a zero-type quadrangle.

(2) The discrete normal vectors form, because of their inaccuracy a quadrangle of type 0 where their continuous counterparts would not.

Hence, part of the solution lies in improved computations of the normal vectors. This can for instance be obtained by using a curve at a higher resolution for computing the normals of a curve at normal resolution. Another possible improvement lies in smoothing the offset curve. Note that in smoothing always the relation between a

point on the center curve and its offset points must be known in order to guarantee a correct thick curve. The latter method has two advantages firstly the number of zero-type quadrangles becomes less and secondly the borders of the filled area become smoother.

7

Final remarks

With chain coding all discrete curves can be described. This enables one to treat discrete curves, independent of the flavour of the curve ('Bezier', 'Hermite', circle, etc.). This is illustrated in the algorithms for linear transformation, computation of offset curves, computation of thick curves, and filling a closed curve; all of these are presented in this thesis. Where these algorithms merely serve to process existing discrete curves, the main part of this thesis is concerned with the formal specification of discrete curves called w-curves and with algorithms for rendering these curves. The curves fulfill a mild condition concerning tangency with its defining control lines (spanned by its control points). The main difference with ordinary curve discretisation algorithms is that the latter are based upon a continuous curve definition. (Although, indirectly, w-curves are also based on such a definition, since its definition is based on Bresenham's line discretisation algorithm.) Despite their discrete nature, there exist w-curves that are discretisations of, for instance, second order Bezier curves or ellipses.

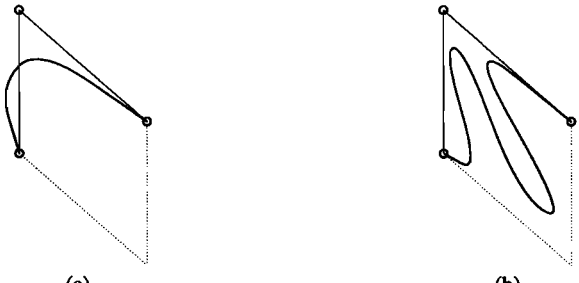


Figure 7.1: two curves not representable by w-curves

Not every continuous function has a w-curve as its discretisation; see figure 7.1(a) for a curve that violates the condition that the w-curve should lie within the parallelogram spanned by the control points. Which curves within the bounds of a parallelogram are w-curves is to be investigated, but clearly curves as in figure 7.1(b) are not among them.

As we saw in chapter 3, the rendering of w-curves can only under certain conditions be done in a time linear to the length of the control chains. These conditions were met in the definitions of e-curves. For simultaneous weaving a similar definition as for e-curves accomplishes linear complexity. For consecutive weaving, however, we

did not give such a solution, but instead referred to the notion of canonical chains, which is a general solution for keeping the time complexity linear for discrete curves of a certain flavour. Another way to achieve this linear time complexity is to eliminate the 8-codes each time the weaving operator is applied. The resulting curves can be computed in linear time but do not have the nice property that a closed formula for their continuous counterparts is known. As an alternative so-called index chains are introduced in [Nie91] ; index chains use run length encoding on the 8-codes in a chain. The computations for consecutive weaving of index chains can also be done in linear time.

The algorithms concerning chain coding as given in this thesis have three properties which facilitate implementation in hardware.

(a) They use only simple integer arithmetic, i.e. addition, subtraction, and shifting.

(b) They have simple structures; typically one loop in which per step of the iteration one code of the input or output chain is handled.

(c) They can easily be partitioned in building blocks. These building blocks are used, for instance, for generating a chain for a line or the weaving of two input chains.

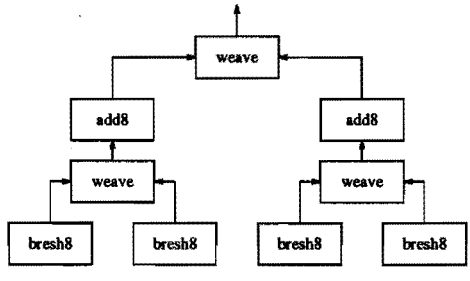


Figure 7.2: building block structure for consecutive weaving

Figure 7.2 shows a possible configuration of such building blocks for the computation of consecutive weave curves. However, considering the fact that all codes generated by the lower blocks in the tree arrive at the root block, a pipeline structure would be more appropriate with respect to load balancing. In [Nie91] these and other considerations are elaborated for consecutive weaving on a transputer network.

7.0 Current and future research

In this thesis we only consider chains in a two dimensional rectangular grid which is either 4-connected or 8-connected. For other regular grids, be it a higher dimensional grid or one with another connectedness, the curve algorithms as given in this thesis can be applied. In all these cases discrete line algorithms can be used for producing a chain representation of a control line; to these chains the same weave and add8 operations can be applied. Hence, extensions, for instance curve algorithms on a 3 dimensional grid, are straightforward. Extending to surfaces, however, is not so straightforward, since chains essentially can represent only 1-dimensional objects. In [Ove90] and [Bri90] a bilinear surface definition is given based on four boundary curves given by chains. Since this surface definition depends on boundary chains, all methods for chain modelling can be applied to model the boundary chains and, indirectly, the surface. A surface modeller, called *pret*, based on these principles is discussed in [Ove91] . The usefulness of *pret* as a surface modeller remains to be investigated.

In [Maa90] an integer algorithm is given for computing a chain for a Bezier curve of arbitrary order. The algorithm is based on computing a linear combination of chains representing Bezier curves; so a chain of a Bezier curve of higher order is obtained.

In [Lie87] the notion of closeness is introduced; closeness is a measure for the accuracy by which a discrete curve approximates a continuous curve. The closeness of w-curves with respect to their corresponding continuous curves remains to be investigated.

In chapter 3 and 4 distribution functions are given for circle and ellipse segments, second order Bezier curves, and third order Bezier curves. For some curves it would be nice to compute distribution functions; for others applying canonical chains seems to be a much more realistic approach. The computation of these canonical chains is in general far from obvious; some further research is required here.

Summary

In this thesis applications of chain coding in the realm of computer graphics are discussed. Chain coding is a technique developed in image processing for representing a discrete curve, a connected set of pixels, by a chain of codes representing the sequence of difference vectors between consecutive pixels. In this thesis chain coding is not only used in processing discrete curves but also in defining discrete curves.

In design continuous curves are often modelled by means of control points indicating the global shape of the curve. The shape of the curve depends also on the flavour of the curves, e.g. Bezier curves, Hermite curves, B-splines etc. The curves, defined in this thesis and called w-curves, are also modelled by means of control points. The flavour of a w-curve is determined by, so-called, distribution functions: a discrete parameterisation scheme. The chain of a w-curve consists precisely of the codes of the chains belonging to the, so called, control lines: line segments connecting consecutive control points. The order of these codes in the chain for a w-curve is determined by the distribution functions. The w-curves are tangent to the first and last control line. The usefulness of w-curves is readily indicated by the fact that they can be used to represent, among others, (discretisations of) ellipses, and second and third order Bezier curves.
Algorithms for rendering w-curves are given. These algorithms use only simple integer arithmetic. For the efficient computation of higher order curves the notion of canonic chain is introduced. In a canonic chain the order of the codes of the control lines is stored; the actual generating of the chain for the w-curve is reduced to producing the codes in that given order. This is a general approach, which may be used outside the realm of w-curves.

The use of chains for the representation of discrete curves allows for algorithms that are applicable for all discrete curves and not only for curves of a specific flavour. In this thesis algorithms for discrete curves are given for the following.

- linear transformation
- computation of offset curves
- computation of thick curves
- filling a closed curve

These algorithms use only simple integer arithmetic and typically have one loop in which per step of the iteration one code of the input or output chain is handled.

Samenvatting

In dit proefschrift worden applicaties van chain coding in het vakgebied van de computergrafiek besproken. Chain coding is een techniek voor het representeren van een discrete curve, een samenhangende verzameling pixels; deze techniek komt oorspronkelijk uit de beeldverwerking. Een chain is een rijtje codes; ieder van deze codes representeert een richting in het discrete vlak. In dit proefschrift wordt chain coding niet alleen gebruikt voor het verwerken van discrete curven maar ook voor het definiëren van discrete curven.

Voor het modelleren van discrete curven worden vaak de zogenaamde stuur- of controlepunten gebruikt; deze punten geven de globale vorm van de kromme weer. De vorm van de curve wordt ook bepaald door de soort curve; bijvoorbeeld Bezier curven, Hermite, B-splines etc. De curven, genaamd w-curven, die in dit proefschrift worden gedefiniëerd worden ook gemodelleerd door controlepunten. De soort curve wordt echter bepaald door een discreet parameterisatieschema met behulp van de zogenaamde distributiefuncties. The chain van een w-curve bestaat precies uit de codes van de chains van de bijbehorende controlelijnen: lijnsegmenten die de controlepunten verbinden. De volgorde van deze codes wordt bepaald door de distributiefuncties. W-curven raken aan hun eerste en aan hun laatste controlelijn. Het nut van w-curven laat zich afmeten aan het feit dat met w-curven onder andere (discretisaties) van ellipsen, en tweede en derde order Bezier curven gerepresenteerd kunnen worden.

Algoritmes voor w-curven worden gegeven. Deze algoritmen gebruiken slechts eenvoudige integer aritmetiek. Voor het efficiënt berekenen van hogere orde curven wordt het begrip canonieke chain ingevoerd. In een canonieke chain wordt de volgorde van de codes van de controle lijnen vastgelegd; het uiteindelijke renderen van de chain is dan gereduceerd to het produceren van codes in deze gegeven volgorde. Deze aanpak is ook algemener dan louter voor w-curven toepasbaar.

Het gebruik van chains voor het representeren van discrete curven staat een algemene algoritmische aanpak toe; algoritmen zijn niet langer slechts voor een soort curven toepasbaar maar voor alle door chains gerepresenteerde discrete curven. In dit proefschrift worden in dit kader algoritmen voor discrete curven gegeven die het volgende bewerkstelligen: lineaire transformaties, berekenen van offset-curven, berekenen van dikke curven, het vullen van gesloten curven. Al deze algoritmen gebruiken slechts eenvoudige aritmetische bewerkingen en bestaan typisch uit een lus waarin iedere keer een code van de invoer of uitvoer chain wordt afgehandeld.

Dankwoord

In het bijzonder een dankwoord voor Kees van Overveld met wiens steun en inspirerende ideeën dit proefschrift zijn huidige vorm en inhoud heeft gekregen.

Verder ook een speciaal dankwoord voor Marloes van Lierop voor het nauwkeurig en gezet lezen van eerdere versies en voor Rens Kessener de projectleider van het STW project "Datastructuren voor rastergrafiek" waarbinnen dit proefschrift zijn aanvang vond.

Dank aan mijn promotoren Dieter Hammer en Frans Peters voor hun opmerkingen die dit proefschrift door het laatste stadium hebben geholpen.

Dank aan mijn ouders aan wie ik dit proefschrift opdraag en aan wie ik alles te danken heb.

Dank aan Marja Nuys voor haar organisatietalent en haar aangename gezelschap.

Dank ook aan al mijn collega's voor het scheppen van een fijne werksfeer en aan mijn vrienden voor de aangename tijd die ik buiten de universiteit heb doorgebracht.

Dank aan allen die zoveel geduld met mij hebben gehad.

Curriculum vitae

11-08-61	geboren te Olland.
1973-1979	Gymnasium-β aan het Jacob Roeland Lyceum te Boxtel.
1979-1985	Wiskunde studie aan de Technische Universiteit Eindhoven.
1985-1989	Project medewerker van het STW-project "Datastructuren voor rastergrafiek".
1989 -	Universitair docent aan de Technische Universiteit Eindhoven.

References

Adobe Adobe Systems Incorporated, *PostScript language reference manual*, Addison-Wesley, June 1990. 16th printing

Bak84 Alan Baker, *A concise introduction to the theory of numbers*, Cambridge university press, Cambridge, 1984.

Boe84 Wolfgang Boehm, Gerald Farin, and Jurgen Kahmann, "A survey of curve and surface methods in CAGD," *Computer Aided Geometric Design*, vol. 1, pp. 1-60, 1984.

Bre65 J.E. Bresenham, "Algorithms for Computer Control of a Digital Plotter," *IBM Systems Journal*, vol. 4, pp. 25-30, 1965.

Bri90 van den Brink and Timmermans, *Discrete bilinear blending (revisited)*, 1990. Internal report, Eindhoven University of Technology

Cou89 Courant, Richard and John, Fritz, *Introduction to calculus and analysis*, Springer, Berlin, 1989.

Far89 Gerald Farin, "Curvature Continuity and Offsets for Piecewise Conics," *ACM Transactions on Graphics*, vol. 8, no. 2, pp. 89-99, April 1989.

Fol90 James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice*, The systems programming series, Addison-Wesley, 1990. 2nd edition

Fre61 Herbert Freeman, "On the Encoding of Arbitrary Geometric Configurations," *IRE Transactions on Electronic Computers*, pp. 260-268, June 1961.

Fre61 H. Freeman, "Techniques for the digital computer analysis of chain-encoded arbitrary plane curves," in *Proceedings of the National Electronics Conference*, vol. 17, pp. 421-432, Chicago, 1961.

Fre69 Herbert Freeman, "A scheme for the efficient encoding of graphical data for communication and information processing," in *Advance in Electronics, proceedings of the 16th electronics congres*, pp. 340-348, Rome, 24-27 March 1969.

Fre74 Herbert Freeman, "Computer Processing of Line-Drawing Images," *Computing Surveys*, vol. 6, pp. 57-97, March 1974.

Lie87 Marloes van Lierop, *Digitisation Functions in Computer Graphics*, Eindhoven, 1987. Ph.D. Thesis, Eindhoven University of Technology

Maa90 C.A. van der Maas, C.W.A.M. van Overveld, and H.M.M. van de Wetering, *An integer algorithm for rendering Bezier curves*, 1990. Submitted for publication in CAD

Mor85 Michael E. Mortenson, *Geometric Modeling*, John Wiley & Sons, New York, 1985.

Nie91 A.J. Niessen, *Consecutive weaving on a transputer network*, 1991. master thesis, Eindhoven University of Technology

Ove90 C.W.A.M. van Overveld, "Discrete bilinear blending and its application in rendering curved surfaces," *Computer Aided Design*, vol. 22, no. 6, pp. 332-343, August 1990.

Ove91 C.W.A.M. van Overveld, *Analysis of the pret system*, Februari 1991. Internal Report, Eindhoven University of Technology

Pos89 K.C. Posch and W.D. Fellner, "The Circle-Brush Algorithm," *ACM Transactions on Graphics*, vol. 8, no. 1, pp. 1-24, January 1989.

Rud76 Walter Rudin, *Principles of mathematical analysis*, Mathematical series, McGraw-Hill, 1976.

Sal96 Salmon, G, *A Treatise on Conic Sections*, Longmans, Green, & Co., London, 1896. 10th edition

Wu82 Li-De Wu, "On the Chain Code of a Line," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 3, pp. 347-353, May 1982.

Index

Am Grunde der Moldau wandern die Steine
Es liegen drei Kaiser begraben in Prag
Das Grosse bleibt gross nicht,
Und klein nicht das Kleine.
Die Nacht hat zwölf Stunden dann kommt schon der Tag
Dann kommt schon der Tag.

Bertold Brecht, Das Lied von der Moldau / Happy End (1929)

1. Algoritmen voor het genereren van w-curven op een regelmatig rooster zijn onafhankelijk van de connectedness van dat rooster.

   (dit proefschrift - paragraaf 3.6.2)

2. De continue varianten van w-curven zijn affien invariante curven.

   (dit proefschrift - paragraaf 3.5)

3. Het gebruik van een canonieke chain als representant van een klasse discrete curven biedt een flexibel alternatief voor het implementeren van een curve-discretisatie algoritme.

   (dit proefschrift - paragraaf 4.3)

4. Het in dit proefschrift veelvuldig gebruikte lijnalgoritme van Bresenham is beter dan "close": het wordt gekarakteriseerd door de constanten 1/2 en 3/4 in formules cd0 en cd1 in (1) in plaats van 1 en 1.

   (1) *Digitisation functions in computer graphics*, Marloes van Lierop, Proefschrift Technische Universiteit Eindhoven, pagina 12.

5. Door het toevoegen van boolean labels aan de zijden van de opgedeelde driehoeken is het algoritme in (2) aanzienlijk efficiënter te maken.

   (2) *A consistent algorithm to fill triangles and triangular patches*, C.W.A.M. van Overveld & M.L.P van Lierop, Proceedings of the European Computer Graphics Conference and Exhibition, 1986.

6. Voorzover de problemen zoals die beschreven zijn in (3) ten aanzien van consistentie van operaties op een geometrisch model van stochastische aard zijn zullen zij zich in versterkte mate voordoen bij het animeren van zo'n model.

   (3) *Computational Geometry and Software Engineering: Towards a Geometric Computing Environment*, A.R. Forrest in Techniques for Computer Graphics edited by David F. Rogers and Rae A. Earnshaw.

7. Een mensenmaatschappij gebaseerd op schaarste is inherent hiërarchisch.

8. De cijfers voor het aantal dierslachtingen in Nederland ten behoeve van de vleesproduktie (in 1988: 6.000 paarden, 455.000 schapen, 1.100.000 runderen, 1.100.000 kalveren, 20.800.000 varkens en daarnaast nog 485.000.000 kilo pluimvee) maken vegetarisme eenvoudig verdedigbaar.

   (4) *Statistisch Jaarboek 1990*, Centraal Bureau voor de Statistiek.

9. Een semantische beschouwing doet vermoeden dat "overheid" en "overhead" etymologisch verwant zijn.

10. De verslaving van de moderne westerse mens aan visuele prikkels heeft onnodige milieuvervuiling tot gevolg.

11. Het steeds vaker gebruiken van 's' als meervoudsuitgang zal er voor zorgen dat pils in het ziekenfondspakket opgenomen wordt.

12. Er is geen reden om naast timmerman woorden als timmervrouw of timmermens te bezigen daar man ook mens betekent; het lijkt echter toch nuttig om ook vrouw in de betekenis van mens te gebruiken.

    (5) *van Dale, Groot Woordenboek der Nederlandse Taal.*