20º COLÓQUIO BRASILEIRO DE MATEMÁTICA

DEFORMAÇÃO E METAMORFOSE DE OBJETOS GRÁFICOS

JONAS DE MIRANDA GOMES

BRUNO COSTA

LÚCIA DARSA

LUIZ VELHO

IMPA 24 - 28 JULHO, 1995

impa

JONAS DE MIRANDA GOMES (IMPA, RJ)
BRUNO COSTA (IMPA, RJ)
LUCIA DARSA (IMPA, RJ)
LUIZ VELHO (IMPA, RJ)

ISBN 85-244-0097-8

Conselho Nacional de Desenvolvimento Científico e Tecnológico

INSTITUTO DE MATEMÁTICA PURA E APLICADA

Estrada Dona Castorina, 110 - Jardim Botânico

22460-320 - Rio de Janeiro, RJ, Brasil

Abstract

This course makes a comprehensive study of the problem of warping and metamorphosis of graphical objects. It presents a unified view of the metamorphosis problem involving drawings, surfaces, images, volume data, etc. A detailed discussion of the concept of a graphical object is introduced in the course. This is necessary to create a solid foundation preparing the basis for the introduction of a unified framework involving the problem of warping and morphing of graphical objects. We try to keep the mathematics simple so that the material will be understood by a large audience with different background. At the end of the course we will discuss the use of metamorphosis techniques in the entertainment industry.

Authors Addresses

Bruno Costa

Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
Title: Computer Science Phd Student
Phone: (516) 632-8470
E-mail: bruno@visgraf.impa.br

Lucia Darsa

Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
Title: Computer Science Phd Student
Phone: (516) 632-8470
E-mail: lucia@visgraf.impa.br

Jonas Gomes

IMPA – Instituto de Matemática Pura e Aplicada
Est. Dona Castorina, 110
22460-320, Rio de Janeiro, RJ, Brazil
Phone: (5521) 294.9032
E-mail: `jonas@visgraf.impa.br`

Luiz Velho

IMPA – Instituto de Matemática Pura e Aplicada
Est. Dona Castorina, 110
22460, Rio de Janeiro, RJ, Brazil
Phone: (5521) 294.9032
E-mail: `lvelho@visgraf.impa.br`

Preface

These course notes cover an important area of applications of computer graphics. In fact, the area of object deformation has several applications that range from the entertainment industry, to important problems in engeneering.

The course will not go into too much details of the problem, that is why we classified it as an elementary course. Nonetheless, some of the concepts found in these notes are quite new. The course represents the first attempt on the graphics literature to look at the problem of object metamorphosis from a unified point of view.

In the applications there are different techniques for deforming objects, but these techniques seem unrelated. The main reason for this discrepancy was the lack of the concept of a generic graphical object. This concept is introduced in chapter 2 of these notes. From this concept we build up a conceptualization of the problem that would cover the different morphing applications existing on the literature.

The reader will find some redundancy on the topics covered along the different chapters. This redundancy, although being good for the reader, is not proposital. Its origins come from the fact that these course notes were written by four authors. It is necessary to spend a great amount of time reading the different chapters and devising a compatibilization between them. This is almost impossible to attain when writing course notes for a congress, where we have a very limited period of time between the course acceptance, and the deadline to deliver the course notes.

The course notes were written in English because we will teach this same course in the SIGGRAPH '95 international conference organized by

the Special Interest Group in Graphics of the Asssociation of Computing Machinery. This congress will be held in Los Angeles, in August.

We wish to thank everybody that contributed, direct or indirectly, to the final version of these notes. In particular we wish to thank the organizing committee of the Coloquio Brasileiro de Matemática and our colleagues of the the VISGRAF group at IMPA, which, are part of the stimulating working environment that helped to breed these ideas.

Rio de janeiro, April, 15 1995

Bruno Costa
Lucia Darsa
Jonas Gomes
Luiz Velho

Contents

Chapter 1

Fundamentals

These course notes investigate warping and morphing in computer graphics. Warping and morphing are important graphical operations with several applications in many areas. The subject has received a lot of attention in recent years both from the academia and the industry. On the academic side, research work has already addressed most of the basic problems quite effectively. On the industry side, application software has been developed, incorporating warping and morphing techniques successfully in various areas.

Warping and morphing transform the shape of graphical objects. Warping deforms a single object. Morphing interpolates between two objects. These shape transformations can be applied to the various types of graphical objects, such as 2D drawings, images, surfaces and volumes.

Today, we can say that the field has reached a maturity state and is ready for a broad conceptualization. One of the goals of these notes is to develop such a conceptualization. We hope that our effort will help to consolidate the understanding of the current work and will provide a solid basis for further research.

The conceptual framework presents a unifying view of warping and morphing. It is based on the notions of graphical object and shape transformations. These notions make possible an uniform treatment of the basic problems. In addition, a discussion of the specification and

computation of these transformations is given with practical examples.

1.1 Metamorphosis

1.1.1 Shape Change in Nature

The shape of objects in the physical world undergo constant changes. This is inherent of the dynamic aspect of nature. Living beings are born, grow and die. Their body changes shape as the result of complex genetic mechanisms.

Growth processes generate internal forces that make organisms develop. As a consequence, their shape is modified. The transformations induced by these mechanisms are, in general, very complex. Simple forms gradually evolve into highly intricate configurations. They start as amorphous blobs of matter and reach a definite embodiment that is revealed with the creation of distinctive parts and traits.

A typical example is a plant: a single seed grows into a complete tree, with stems, leaves, etc. Figure 1.1, from (Prusinkiewicz, Hammel and Mjolsness, 1993), shows a schematic illustration of some stages of the evolution of a plant.

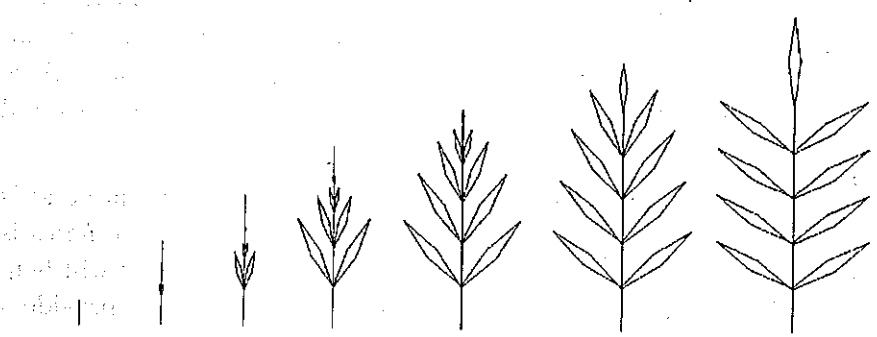


Figure 1.1: Plant growing.

Inanimate things may also have their shape changed under the action

of external forces. These forces include environmental phenomena, such as wind, rain and lightning, as well as, other processes such as mechanical impact, combustion reactions, etc. Figure 1.2 shows the deformation of a tube that was bent over another tube.

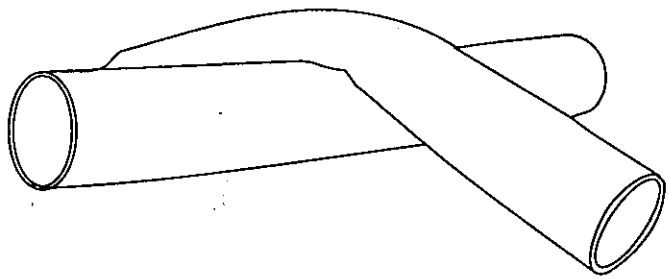


Figure 1.2: Deformation of a tube.

Depending on the material properties of the object, some deformations may be permanent and others temporary. This classifies deformations as *elastic* and *inelastic*.

An important characteristic of the shape transformations discussed above is that, with a very few exceptions, they are *continuous* deformations. However, we should point out that non-continuous deformations also are also present in Nature. Several examples could be given, such as material fracture, explosions etc.

1.1.2 Analysis of Shapes

The theory of transformations can be used in the comparison of related forms and in the study of shape evolution.

A powerful method to analyze the correlation between shapes of the same class is to consider the transformations required to deform one shape into the other. In this way, it is possible to classify the members of a given family based on a few parameters, namely the amount of deformation from a base shape.

Figure 1.3; from (Thompson, 1961), shows leaves of different types, and indicates the transformation that relates them.

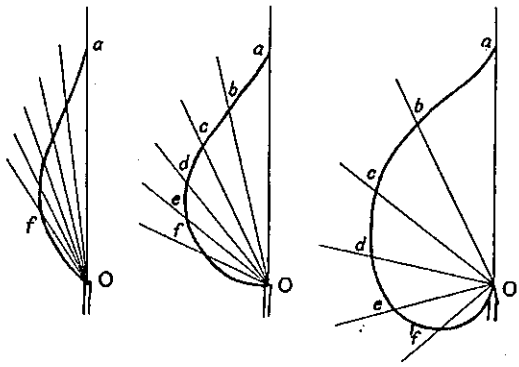


Figure 1.3: Shapes of the same class under a transformation.

Transformations also can be used to study the evolution of forms. For example, as a species develops it goes through a succession of transitional forms.

Figure 1.4, from (Thompson, 1961), shows three stages of the evolution of the crocodile. It compares the skull of a modern crocodile (1.4 a) with those of two other crocodiles from earlier historical periods (Figures 1.4 b and c).

1.1.3 Images and Illusion

Images are pictorial representations over a two-dimensional support. They can depict bidimensional shapes, such as in a drawing, or even the projection of three-dimensional objects, such as in a photography. In both cases, images give the illusion of being the actual objects that they represent.

When a sequence of images is exhibited in rapid succession it produces the illusion of movement, which is called *animation*. This is the basic principle of film and television.

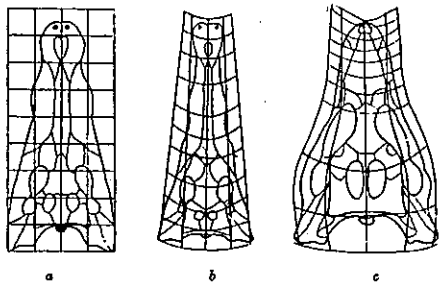


Figure 1.4: Three stages of the evolution of the crocodile.

Photographic images can be distorted in many ways through the use of optical devices and projections. Video images can also be distorted using electronic analog devices. If an increasing amount of distortion is applied to a base image, it is possible to generate animated sequences that produce the impression of continuous deformation.

Figure 1.5 shows the distortion of an image using an analog electronic device.

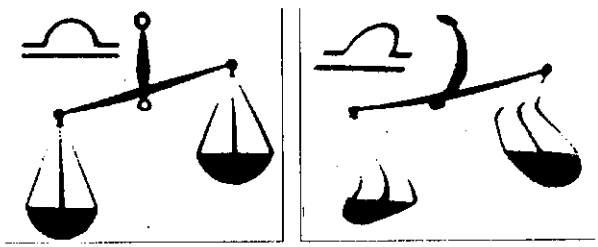


Figure 1.5: Image distortion.

Two images can be combined by mixing color information. In this process, first the images must be superimposed and then their color

values are blended. When the proportions of the mixture between images A and B change continuously from 100% of A and 0% of B to 0% of A and 100% of B, then a smooth transition from A to B is achieved. This visual effect is known as *cross dissolve*. It is an expressive resource that has been incorporated in the film language, and it is often used to indicate the passage of time / space. Figure 1.6 shows a cross dissolve effect from the classic film "Limite" by the brazilian film maker Mario Peixoto.

Figure 1.6: Cross dissolve from the film "Limite".

Cross dissolve techniques have also been used for decades by the film and video industry, to attain metamorphosis effects between two different objects with similar shapes, represented by two different images.

1.2 Uses of Shape Transformations

Shape transformations have many uses in applications that deal with animation, modeling and analysis of forms.

1.2.1 Animation

The fact that shape transformations are continuous in general, makes them ideally suited to applications dealing time-varying phenomena.

We can define a continuous shape deformation and make the parameter of the transformation dependent of a time variable. A simple example is an animation that incorporates stretch and squash using non-uniform

scaling, (Lasseter, 1987). Figure 1.7 shows a few frames of the animation of a ball bouncing on the ground enhanced by a stretch and squash effect.

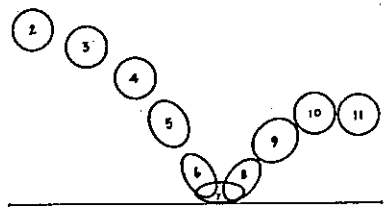


Figure 1.7: Stretch and squash in bouncing ball.

Another way to use shape transformations for animation is to interpolate the shape of two objects. This technique is common in traditional animation. It consists of creating first a sequence of *keyframes* depicting the main events of a scene. These keyframes contain important transition elements of the motion and convey the action that is taking place in the scene. Afterwards, the intermediate frames, or the *inbetweens* are generated by interpolating the keyframes. In a computer animation system, the intermediate frames can be computed using shape transformation techniques. For this, it is necessary to specify the initial and final shapes and a correspondence between them.

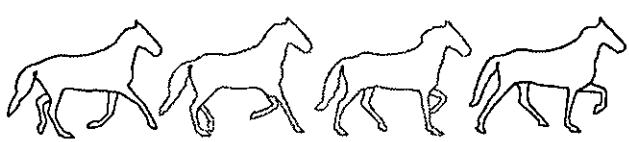


Figure 1.8: Keyframing using shape interpolation.

In Figure 1.8, from (Sederberg et al., 1993), an animated sequence of

a cantering horse is shown. The first and last frames are given, and the intermediate frames are computed using shape interpolation.

1.2.2 Shape Families

Shape transformations can be used in various ways for modeling purposes. In fact, deformations constitute the basis of powerful operators that modify the shape of objects. A pioneering example of the use of deformations as a shape operator appeared in (Barr, 1984). In this work the non-linear transformations of *taper*, *bend* and *twist* are discussed and rather complex objects are modeled by the deformation of simple primitive objects. Figure 1.9 shows the action of the operators *taper*, *twist* and *bend* on a cube.

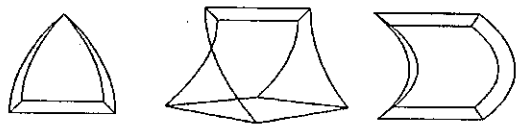


Figure 1.9: A cube deformed by taper, twist and bend.

Figure 1.10, from (Barr, 1984), shows a rather complex chair constructed from simple geometric primitives, superquadrics, using the bend operator.

Shape interpolation makes it possible to create a continuous family of shapes with desired features. If a set of shapes with distinct formal characteristics are defined, we can generate a shape that blends these characteristics in any proportion using interpolation.

Figure 1.11, from (Chen and Parent, 1989), shows an example of a family of shapes generated by interpolating two bottles with different styles.

Another example of the use of interpolation to construct a family of shapes is the *multiple master* font technology developed by Adobe

Figure 1.10: A chair constructed using warping operations.

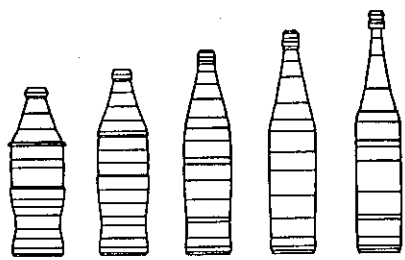


Figure 1.11: Shape family of bottles.

Systems for digital typography. In this case, a new font design is created by interpolating an existing existing master font. This is illustrated in Figure 1.12, The word "Morphing" written in the second and third lines uses a font obtained from the font of the first line (Tekton), by interpolating the parameters of *width* and *weight*.

1.2.3 Fitting and Matching

Shape transformations can be used for analysis and correction of forms. In particular, it may be employed in the problem of form recognition. In this type of application the goal is to match a given shape to a template. One way to do that is by computing the amount of deformation necessary

Morphing

Morphing

Morphing

Figure 1.12: Generation of new fonts by deformation.

to align some features of the shape with each element from a set of templates. The best match is the one that requires the least amount of deformation.

Figure 1.13, from (Sclaroff and Pentland, 1994), shows a template, a set of shapes and the correspondence between the template and each shape under a deformation.

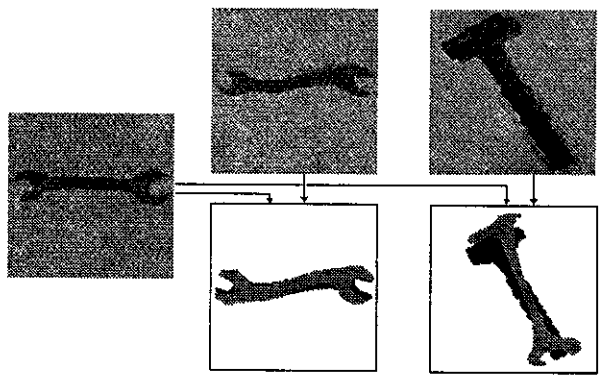


Figure 1.13: Shape recognition using deformations.

Shape interpolation may be employed to fit the shape of one object

to another. In this way, it is possible to establish a connection between two related objects, making their attributes mutually compatible.

One example of shape fitting is the registration of a satellite photograph to a corresponding elevation data set. The image must be distorted to be in perfect alignment with the surface, so that it can be used as a texture map for the terrain model.

Figure 1.14, from (Litwinowicz and Miller, 1994), shows the warping of the picture of a face, so that it fits the corresponding 3D surface. This distorted image is mapped onto a head model as a texture.

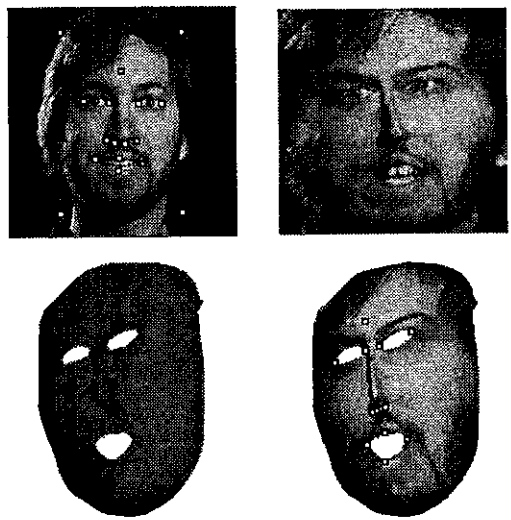


Figure 1.14: Texture registration.

1.3 Conceptual Framework

Until recently, the research of shape deformation and metamorphosis has taken a pragmatic approach that led to a somewhat artificial classification of object types. Moreover, a diverse treatment has been given to the transformations of each kind: images, drawings, surfaces and volumes.

The techniques available in the literature were developed for objects of a certain dimension represented in a particular way. As a consequence, they may reflect a narrow conceptual vision of the problem.

The present work attempts to demonstrate that all these problems share a common conceptualization that will be generalized for abstract n-dimensional graphical objects, regardless of their representation. This integrated view will show that, despite a few inherent idiosyncrasies, some of the existing techniques and solutions to most of these problems can be extended to the others, revealing new connections between them.

The framework is based on two unifying concepts:

- graphical objects;
- transformations.

The concept of a *graphical object* allows one to establish an abstraction that encompasses all types of representations for graphical entities.

The concept of *transformation* allows one to study in a uniform way the different classes of operators that modify the geometry and the properties of graphical objects.

Using this conceptual basis the problem of specifying a shape transformation will be analyzed and the associated computational methods will be studied.

1.3.1 Graphical Objects

There are many types of graphical objects. They have different characteristics, but share some common properties. From a conceptual point of view it is advantageous to create a higher level abstraction that captures the essential features of these objects. This makes possible to define general operations with graphical objects and to investigate more effectively the practical problems involved in the their implementation.

Below we list some types of graphical objects:

Particle Systems;
2D Vector Drawings;

Images;
Surfaces;
Volume Data.

All graphical objects have two basic components:

- shape;
- properties or attributes.

The shape gives the geometry of the object and provides a geometric support where the properties of the object are defined. These attributes are normally used to generate renderings of the object. On an image, the attribute function is responsible for defining the color of the pixels. In a particle system, the attribute function defines the particle color, duration, velocity etc.

1.3.2 Transformations

In order to devise a mathematical model for metamorphosis of graphical objects, it is necessary to understand the mechanisms underlying an object transformation.

1.3.3 Shape Transformation Mechanisms

Object metamorphosis involve two basic mechanisms:

- shape deformation;
- object combination.

A deformation $f: U \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$ can be interpreted either as moving points of the underlying space, or as a change of coordinates of the space. In the first case, a point P is mapped into a new point $P' = f(P)$ of the space. In the second case, if a point $p \in U$ has coordinates

$$P = (x_1, \ldots, x_n),$$

and

$$P' = f(P) = f(x_1, \ldots, x_n) = (y_1, \ldots, y_n),$$

then we have

$$\begin{aligned} y_1 &= f_1(x_1, \ldots, x_n); \\ y_2 &= f_2(x_1, \ldots, x_n); \\ &\vdots \\ y_n &= f_n(x_1, \ldots, x_n). \end{aligned}$$

The vector $(y_1, \ldots, y_n)$ define the new coordinates of the point P. Both interpretations are useful, and in fact they have a great influence in the techniques used to compute the deformation. This will be discussed with more detail in later chapters.

The problem of *shape deformation* can be posed as the change of a coordinate system associated with the object. This formulation gives a mathematical model that allows effective specification as well as efficient computation. Coordinates provide a handle that indicate where deformations should be applied and a framework for numerical implementation.

The change of coordinate systems may be specified either as a global transformation of the space (e.g. an scaling of the whole space), or its specification can done in a piecewise manner. There is a large family of deformations which can be defined globally. We could mention the family of linear, afine, and projective mappings, and the non-linear transformations of taper, twist and bend mentioned earlier in this chapter.

More complicated transformations are usually specified by parts. In such a piecewise transformation the space is decomposed into regions and a simple transformation is applied to each region. It is necessary to maintain some degree of differentiability between the transformations of neighboring regions. Piecewise transformations are much more flexible to be used to attain certain specific results, but they are much more difficult to be specified and implemented.

Figure 1.15 shows a comparison of a global and a piecewise transformations.

Object combination is required in order to compute the interpolation of the object properties. In this process, the two objects must be de-

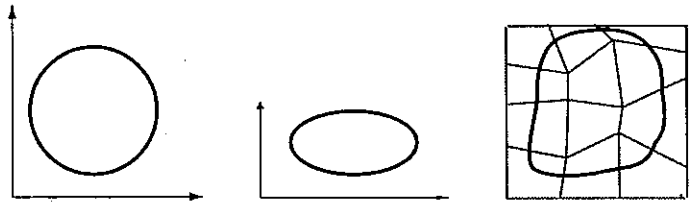


Figure 1.15: Global and Piecewise specifications.

formed to a common frame of reference such that their properties can be mixed. The operation consists of two steps: first the corresponding features in both objects are aligned; second the property information is blended.

1.4 Warping and Morphing of Graphical Objects

Warping and morphing can be analyzed under the same framework. These operations have in common several properties and share the same basic mechanisms. The main difference is that warping is a unary operation, where we deform one object to obtain a new one. On the other hand, morphing is a binary operation: we start from two objects and the goal is to obtain a transition between them. Nonetheless, warping can be interpreted as a morphing between two versions of the same object, one that is deformed and another that is not.

We should observe that both operations must take into account the shape and the properties of the objects. Therefore These two operations use essentially the same processing pipeline that consists of the following stages:

1. transformation of the geometry;
2. generation of attributes.

In the case of warping, the geometric transformation is defined relative

to the shape of the object and the generation of attributes requires only a compatibilization of the attribute values with the deformed geometry.

In the case of morphing, the geometric transformation is based on a correspondence between common features of the two objects and the generation of attributes includes compatibilization and blending (interpolation) of attribute values.

Figure 1.16 shows a diagram of the processing pipeline for warping and morphing. In this illustration, the first column of operations corresponds to the transformation of geometry, and the second column to the generation of attributes.

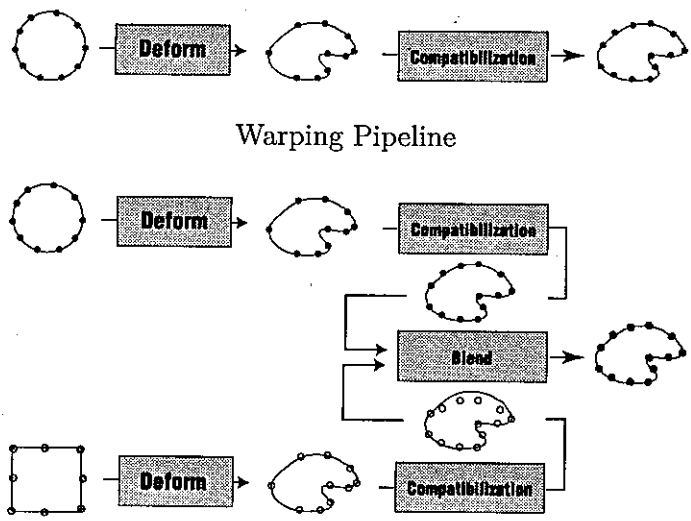


Figure 1.16: Processing pipeline for warping and morphing.

1.4.1 Some Examples

Defining a graphical object warp, or a metamorphosis between two graphical objects in general involves a lot of work related with the de-

scription of each graphical object and with the specification and computation of the desired transformation.

Finding the correct morphing sequence is in general a very difficult task. In general we have infinity possiblities for choosing a morphing sequence between two graphical objects.. The right sequence involves considerations of different nature: perceptual, physical, computational, etc.

Figure 1.17 shows a a morphing transformation between the drawing of a banana, and the drawing of an ice-cream cone.

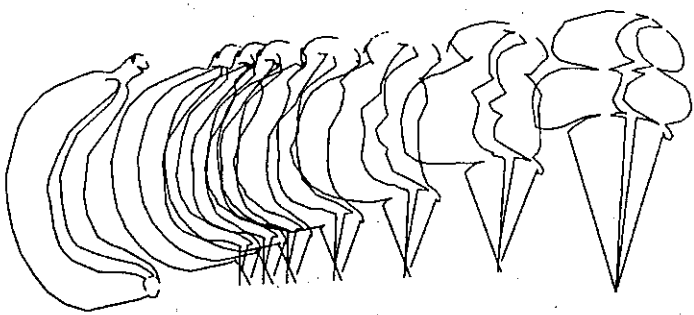


Figure 1.17: Transformation of curves.

Figure 1.18 shows an example of the warping of an image. In this transformation there is a change of the position of each pixel of the image, which amounts to a distortion in the objects present on the image.

1.5 Paradigm of the Universes

In order to develop a conceptualization to study the problems of warping and morphing of graphical objects, we will employ the paradigm of the four universes. This paradigm is a methodology for studying problems in computational applied mathematics. It was first introduced by Ari Requicha, (Requicha, 1980), in the context of geometric modeling, and it was subsequently extended for different areas of computer graphics in

Figure 1.18: Image Warping.

(Gomes and Velho, 1995a).

1.5.1 Understanding an Area

The problems in computer graphics, such as warping and morphing, are conveniently modeled, and solved, using methods from different areas of mathematics. The diversified nature of the models involved calls for some unifying paradigm that enable us to address the problems at the appropriate level of abstraction.

These abstraction levels isolate conceptual characteristics intrinsically attached to the different objects and phenomena. In that way, we can search for the right mathematical tools to tackle problems in each level. Once we know these tools, more specific problems can be posed, probably associated to lower abstraction levels.

1.5.2 Levels of Abstraction

Real world objects are associated to an abstraction level called the *Physical Universe*, denoted by P. The mathematical objects describing elements of the real world belong to the abstraction level called the *Mathematical Universe*, denoted by M. The elements of M are represented in the computer using a finite symbolic description which is associated

to a third abstraction level called *Representation Universe*, denoted by R. The symbolic descriptions in R are implemented in a computational system by mapping the finite descriptions from the representation universe, into specific data structures from another abstraction level, the *Implementation Universe*.

Figure 1.19 shows a diagram illustrating the conceptual framework, with these four abstraction levels.

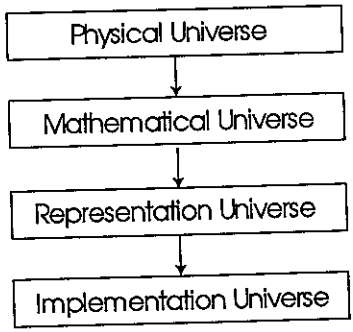


Figure 1.19: Levels of Abstraction.

These four levels of abstraction encapsulate common properties of the objects being studied and allow a global, conceptual view of the methods and techniques involved. For more details, the reader should consult (Gomes and Velho, 1995a).

1.5.3 Conceptual Problems

Once the correct abstraction levels are established for an area, it is possible to have a global conceptual view of the important problems of the area. They can be understood and formulated without taking into account technical details of the specific mathematical models used. The hierarchy of abstractions makes possible to search, in a systematic way, for proper mathematical tools, to pose and solve specific problems. The main problems that can be posed using the universes paradigm are:

1. Define the elements of the mathematical universe M;
2. Define operations on the elements of the mathematical universe M;
3. Construct representation schemes for the elements of M;
4. Devise appropriate data structures and implementation techniques.

In order to use the above paradigm to properly posing the problem of warping and morphing, we will need:

- Define precisely a graphical object in the mathematical universe;
- Define the operations of morphing and warping of graphical objects;
- Devise representation schemes for the specification of graphical objects and their transformations;
- Devise an implementation framework in order to allow a numerical computation of graphical objects transformations.

1.6 Structure of the Notes

The structure of the course notes reflects the conceptual approach described in the previous section.

This first chapter, *Fundamentals*, focuses on metamorphosis in the physical world. It describes shape change in nature and the applications of shape transformations in several areas. The chapter also gives a brief review of the use of warping and morphing in computer graphics and introduces the conceptual approach adopted in this work.

Chapter 2, *Graphical Objects*, presents a unifying model for the description of different types of shapes in computer graphics. It introduces a very broad definition of shape, so as to encompass all of the "objects" used in computer graphics.

Chapter 3, *Representation of Graphical Objects*, discusses the different problems in obtaining a finite description of a graphical object from its continuous description in the mathematical universe.

Chapter 4, *Reconstruction of Graphical Objects*, discusses the problem of recovering a continuous graphical object from its finite description on the representation universe.

Chapter 5, *Transformation of Graphical Objects*, develops a general framework for shape deformation and interpolation. It characterizes the properties of these two operators and how they affect graphical objects.

Chapter 6, *Morphing of Graphical Objects*, uses the concepts introduced in the previous chapters in order to give a precise description of the problem of metamorphosis between two graphical objects.

Chapter 7, *Specification of Transformations*, investigates the various alternatives to define the warping and morphing operations. It analyzes the parameters required to specify these transformations and how to construct operator representations for them.

Chapter 8, *Computation of Transformations*, discusses the implementation of warping and morphing. It presents the pipeline for the computation of discrete shape transformations. It elaborates on the issues of sampling and reconstruction of graphical objects.

Chapter 9, *Warping Techniques*, discusses the different methods that have been used in the implementation of warping and morphing.

Chapter 10, *Blending Techniques*, Discusses some of the most used blending techniques.

Chapter 2

Graphical Objects

At the end of the previous chapter we discussed the use of adequate abstraction paradigms for computer graphics. These abstraction levels allow us to pose the different problems of this area. In these notes we are concerned with the problem of object metamorphosis. Therefore, we need a clear concept of the notion of "object", in order to define precisely what we mean by metamorphosis, and also, to develop different metamorphosis techniques.

In computer graphics, a multitude of different elements constitute the mathematical universe, such as points, curves, surfaces, fractals, volume arrays, images, $3D$-images, data sets, vector graphics, raster graphics, etc. These different objects are manipulated on the computer using similar techniques. However, in general, the relationship between these techniques is hindered because there does not exist a unifying concept that encompasses the different objects, their representation and implementation on the computer.

The objective of this chapter is to introduce the concept of a *graphical object*, so as to include, in a unified way, the many elements we will use to apply the metamorphosis operations. This concept was introduced in (Gomes et al., 1994).

The concept of a graphical object, and its different representations, enable us to relate seemingly different algorithms and techniques existing on the computer graphics literature. Besides the impact on the develop-

ment of new algorithms and on the understanding of existing ones, this concept has also a great influence on systems design and development.

2.1 The Concept of a Graphical Object

Intuitively, a graphical object consists of any of the entities processed in a computer graphics system: points, curves, surfaces, fractals, 2D-vector graphics, 3D-vector graphics, image, 3D-image, volume data, etc. A simple definition, broad enough to encompass all of the above objects, is given below.

A *graphical object* $\mathcal{O}$ consists of a finite collection $\mathcal{U} = \{U_1, \ldots, U_m\}$, of subsets, $U_i \subset \mathbb{R}^n$, of some euclidean space $\mathbb{R}^n$, and a function $f: U_1 \cup \cdots \cup U_m \longrightarrow \mathbb{R}^p$. The family $\mathcal{U}$ is called the *geometric data set* of the object. The union $U = U_1 \cup \cdots \cup U_m$ defines the *shape* of the object, and f is the *attribute function* of the object. The *dimension* of the graphical object is defined to be the dimension of the shape U.

The shape U defines the geometry and the topology of the object, and the function f defines the different properties (attributes) of the object, such as color, texture, scalar fields, vector fields, etc. Each of these attributes is defined by a function $f_j: U \longrightarrow \mathbb{R}^{p_j}$, $j = 1, \ldots, k$, such that $p_1 + p_2 + \cdots + p_k = p$, and the attribute function f,

$$f: U = U_1 \cup \cdots \cup U_m \longrightarrow \mathbb{R}^{p_1} \oplus \cdots \oplus \mathbb{R}^{p_k} = \mathbb{R}^p,$$

has coordinates $f = (f_1, \ldots, f_k)$.

Essentially, a graphical object is a function $f: U \longrightarrow \mathbb{R}^p$, where U is the object shape, and f is the attribute function. However the reader should observe the subtle distinction we made between the shape and the geometric data set of a graphical object. The geometric data set is used to in order to characterize some distinguished features of the object shape, which might be relevant for the solution of some specific problems. The examples below will clarify the need for this distinction.

Graphical Object and Decomposition

A subset $U \subset \mathbb{R}^3$ of the euclidean space. U is naturally a graphical object, and its default attribute function is the standard inclusion $i: U$

$\longrightarrow \mathbb{R}^3$, $i(p) = p$.

Consider a subset $U \subset \mathbb{R}^3$, and suppose that we obtain a decomposition of the set U as shown in Figure 2.1.

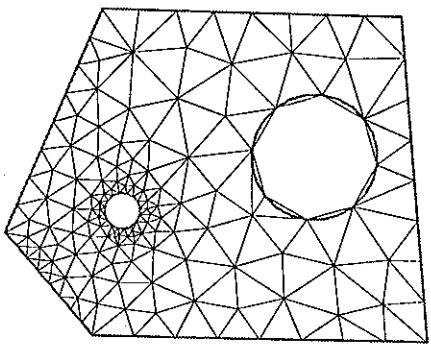


Figure 2.1: Shape decomposition.

The geometry of the inner edges, defining the decomposition, might be of great relevance to solve some problems related with the object U. This is the case in several applications, such as computation of scene illumination using radiosity, finite element analysis computation, etc. These inner edges are part of the geometric data set of the graphical object. Therefore, different decompositions of U define different graphical objects. Nevertheless, all of these graphical objects have the same shape U.

Signals

It is clear from the definition of a graphical object that any function $f: U \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is a graphical object. U is the shape, and the function itself defines the attributes of the graphical object. In engineering a function is called a signal. Therefore our definition of graphical object includes signal as an example. This is very important because the different signals used in computer graphics and its applications, such as the color, audio and the image signal.

Image

An image is a function $f: U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^n$. The shape of the image is the set U. It is, in general, a rectangle of the plane, and coincides with the geometric data set. The function f is the attribute function of the image. To each point $p \in U$, $f(p)$ defines the attributes of p. These attributes are, in general, color, opacity, scene depth etc.

Given two distinct images, as shown in Figure 2.2, the shape is the same: a rectangular region of the plane. The difference between these two graphical objects resides in the attribute function, which is responsible for the grayscale intensities that define the woman's face and the cheetah.

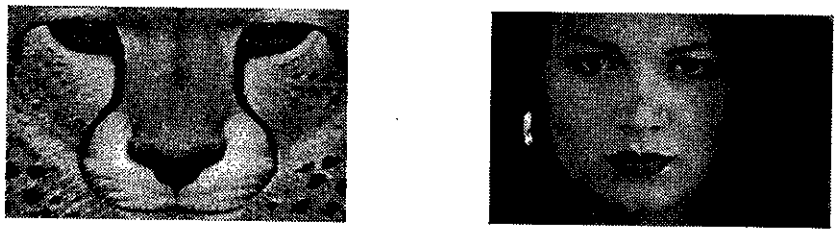


Figure 2.2: Distinct graphical objects with the same shape.

In the examples of figure 2.2 the attribute function specifies the texture that characterizes the geometric information of the woman face and the cheetah. When this geometric information is very important to the solution of some problems, we must incorporate it into the geometric data set of the graphical object. This remark will be very important in later chapters. This is best illustrated with the example of a segmented image discussed below.

Segmented Image

In Figure 2.3(a) we show the image, of an woman's face. In Figure 2.3(b), we show the same image with a polygonal curve defining the face

"boundary". The shape of the image in (a) is the rectangle of its domain, and coincides with its geometric data set. The shape of the segmented image in (b) is same rectangle of the plane, but the geometric data set also contains the curves that define the boundary of the face. Therefore, these two images define two different graphical objects with the same shape, the same attribute function, but with different geometric data set.

Figure 2.3: Two different graphical objects.

Circle and Vector Fields

The shape of the unit circle on the plane is defined by the equation

$$C = \{(x, y) \in \mathbb{R}^2; x^2 + y^2 = 1\}.$$

The map $N: C \longrightarrow \mathbb{R}^2$, $N(x, y) = (x, y)$, defines a unit vector field normal to the circle C. In the same way, the map $T: C \longrightarrow \mathbb{R}^2$, $T(x, y) = (-y, x)$, defines is a unit tangent vector field to C, that points in the clockwise direction (see Figure 2.4). These two vector fields are attributes of the circle. They define the attribute function $f: C \longrightarrow \mathbb{R}^4 = \mathbb{R}^2 \oplus \mathbb{R}^2$, $f(x, y) = (x, y, -y, x)$.

Particle System

A particle system consists of a finite family U_i, $i = 1, \ldots, n$, of subsets of the space. Each U_i, called a *particle*, has different attributes, such as color, weight, external forces, opacity, duration in time etc.

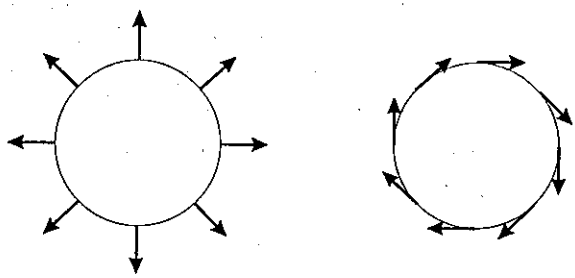


Figure 2.4: Circles with normal and tangent unit vector fields.

Particle systems have been used to simulate a wide variety of natural phenomena in computer graphics, such as fire, rain, etc.

2.1.1 Graphical Objects and Metamorphosis

The importance of the distinction we made between the shape and the geometric data set of a graphical object is greatly dependent on the application. We will give below a preview of the role of the concept of graphical objects in the context of object metamorphosis, using images.

Intuitively, a metamorphosis consists of a continuous transition from some object $\mathcal{O}_1$ to another object $\mathcal{O}_2$ (see Chapter 1). In this process we should take into account both the geometry and the attributes of each graphical object $\mathcal{O}_1$ and $\mathcal{O}_2$. The geometric data set of each object is used to obtain the transition between the geometric features of both objects. Along with the change of object geometry, we should perform a transition from the attributes of object $\mathcal{O}_1$ to those of object $\mathcal{O}_2$. We summarize these two steps by saying that during the metamorphosis operation the objects undergo a *geometry alignment* and an *attribute interpolation*. This two step process that comprises the metamorphosis transformation is illustrated in Figure 2.5.

In brief, when the objects have distinct shape with different features, it is necessary to first merge the different geometric features, before changing the attributes. Since morphing involves two objects, and in

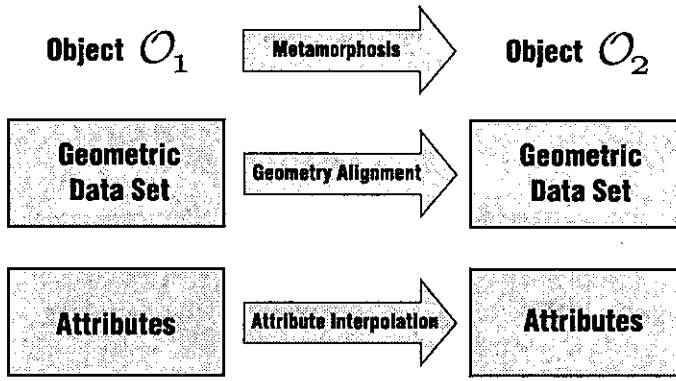


Figure 2.5: Object metamorphosis.

general it is very difficult to obtain a geometric alignment between them, we perform the warping operation to obtain an approximate alignment, and then we blend the two warped object.

In a cross dissolve between two images we are just interpolating the image attributes, without any geometry alignment (see Figure 2.6, from (Gomes and Velho, 1995b)).

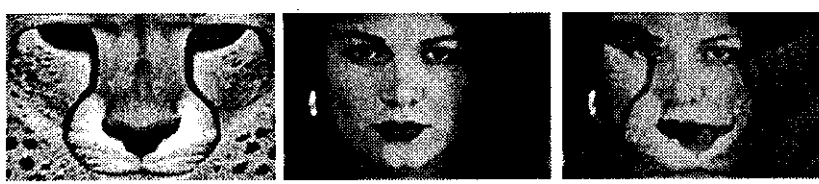


Figure 2.6: Cross dissolve between two images.

In Figure 2.7, from (Gomes and Velho, 1995b), we show a comparison between the simple cross dissolve and a metamorphosis transformation with geometry alignment. Image (c) shows a cross dissolve of the images

in (a) and (b). In this case there is only attribute interpolation, and this causes several alignment problems (eyes, mouth, etc.). The image in (d) shows an attribute interpolation with an alignment of the geometry. The reader should pay special attention to the alignment of the face contour, the mouth, and the eyes.

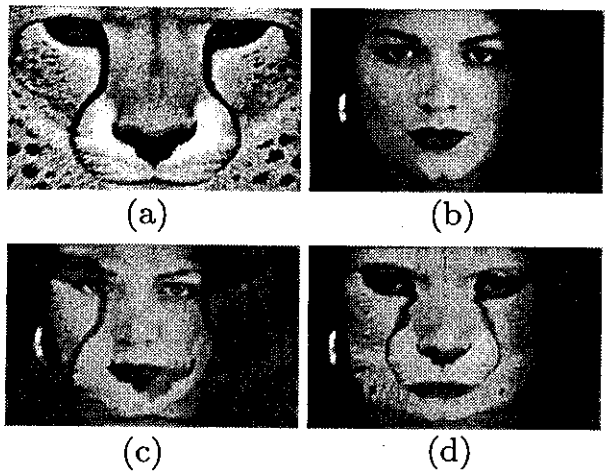


Figure 2.7: Image metamorphosis.

2.2 Shape Description

When defining a graphical object, a very delicate and important point is related to the mathematical description of its shape, that is, how the object shape is specified. In general we use functions to describe shapes, this method is called *functional shape description.* In this case we can devise two different ways to obtain the shape description: *implicit* and *parametric.*

2.2.1 Implicit Shape Description

The implicit description of a shape $U \subset \mathbb{R}^k$ is defined by

$$U = \{p \in \mathbb{R}^k \; ; \; f(p) \in A\},$$

where $f: \mathbb{R}^k \supset V \supset U \longrightarrow \mathbb{R}^m$, is a function, and A is a subset of $\mathbb{R}^m$. This set is denoted by $f^{-1}(A)$. The most common case occurs when the set A is a unit set $\{c\}$, $c \in \mathbb{R}^m$. In this case, we have

$$U = f^{-1}(c) = \{p \in \mathbb{R}^k \; ; \; f(p) = c\}.$$

This is illustrated in Figure 2.8, for $k = 2$ and $m = 1$.

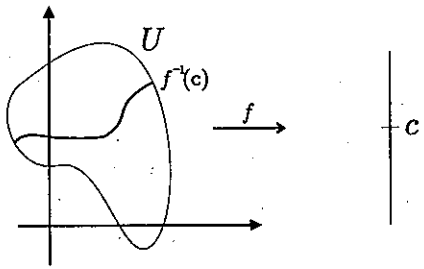


Figure 2.8: Implicit object.

The reader should observe that f is a k-dimensional vector valued function. Therefore it has k coordinates $f(p) = (f_1(p), \ldots, f_k(p))$. The equation $f(p) = c$ in fact constitutes a system of k equations

$$\begin{aligned} f_1(p) &= c_1; \\ f_2(p) &= c_2; \\ &\vdots \\ f_k(p) &= c_k, \end{aligned}$$

where $c = (c_1, c_2, \ldots, c_k)$. Intuitively, this shows that the solution has $m - k$ degrees of freedom, and therefore, it defines a graphical object of dimension $k - m$ in $\mathbb{R}^k$.

When the shape of a graphical object $\mathcal{O}$ is described implicitly, we say that $\mathcal{O}$ is an *implicit object.*

We should observe that if $f: U \longrightarrow \mathbb{R}^p$ is the attribute function of an implicit object $\mathcal{O}$, defined by an implicit function $g: U \longrightarrow \mathbb{R}^m$, then $\mathcal{O}$ is completely described in a functional form by the function $h: U \longrightarrow \mathbb{R}^{p+m}$, defined by $h = (g, f)$, where g defines the shape, and f describes the object attributes.

A very comprehensive survey of the use of implicit graphical objects in computer graphics is given in (Gomes and Velho, 1992).

Implicit objects are very flexible to manipulate on the compute. They can describe a wide variety of object shapes in computer graphics. This is illustrated by the two classes of implicit objects studied below.

Blobs

The pioneering work showing the potential of implicit models was done by J. Blinn, (Blinn, 1982). This work was motivated by the need to display molecular structures more accurately. The model is based on electron density maps. We start from a finite number of points which define the *skeleton* of the shape. For each point p_i of the skeleton we define a density function D_i, using a Gaussian, centered at p_i. That is,

$$D_i(\mathbf{x}) = b_i \exp(-a_i r_i^2),$$

where $r_i = \|\mathbf{x} - \mathbf{p}_i\|$ is the euclidean distance from $\mathbf{x}$ to the skeleton point $\mathbf{p}_i$. The parameters a and b are respectively the standard deviation and the height of the function.

The implicit function f is defined by the sum

$$f(x) = \sum_{i=1}^{n} D_i(x),$$

where n is the number of points on the skeleton.

A more flexible shape specification is given in terms of the radius ρ of an isolated point skeleton, and the *blobbyness* factor β that controls how the point blends with others. The new equation is

$$D_i(\mathbf{x}) = c \exp(\frac{\beta_i}{\rho_i^2} r_i^2 - \beta_i) \tag{2.1}$$

Since c now is included in the contribution of each term, it can be set to a standard value such as 1. The effect of changing the level surface can be achieved through the blobbyness factor.

The implicit function D_i defines an algebraic distance which has spherical symmetry around each point p_i. Therefore, the implicit shapes defined by each D_i is a spherical shape. This function can be generalized to allow for arbitrary quadric shape functions if r_i^2 is substituted in the equation by $\mathbf{x}Q_i\mathbf{x}$.

Figure 2.9, from (Wyvill, 1994), shows a textured blobby object generated using an skeleton consisting of two points. Initially we have two spherical shapes, and as the points get closer together the shape of the implicit object changes because the effect of the density function of each skeleton point overlaps.

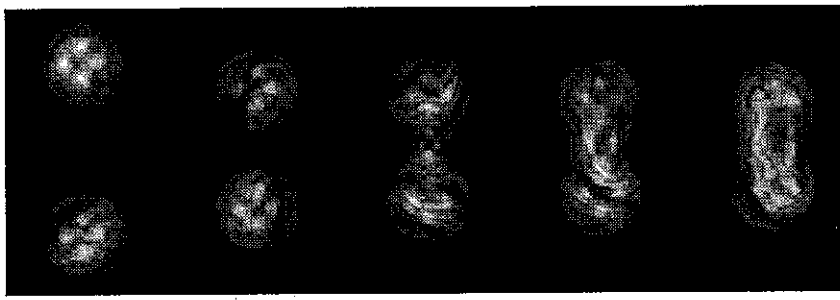


Figure 2.9: Blob models.

The reader should notice that the implicit objects in Figure 2.9 have a texture. This texture is an attribute, defined by some function $g: \mathcal{O} \longrightarrow \mathbb{R}$, which associates to each point of the object shape, the intensity value of the texture.

Hypertexture

These graphical objects were introduced simultaneously by (Perlin and Hoffert, 1989) and (Kajiya and Kay, 1989). A hypertexture is defined

by a modulation of an implicit shape function

$$H(D, x) = f_n(f_{n-1}(\ldots f_1(D(x)))),$$

where D is the implicit shape function, and f_i are density modulation functions.

The equation $H(D, x) = c$ gives an implicit description of a graphical object. The repeated modulation process, with conveniently chosen modulation functions, results in a very effective method to describe objects with a "fuzzy geometry", such as fur and hair. An example of an object generated with a hypertexture object is shown in Figure 2.10, taken from (Perlin and Hoffert, 1989).

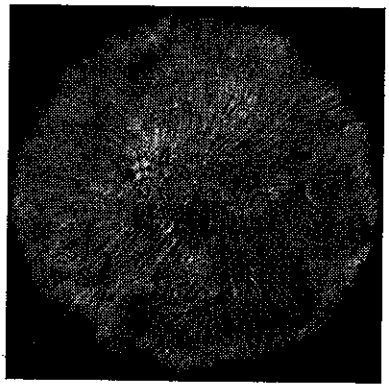


Figure 2.10: Image of a hypertexture object.

2.2.2 Parametric Shape Description

In a parametric description, the shape $U \subset \mathbb{R}^k$ of a graphical object is described by defining a coordinate system on U. This coordinate system is defined by a function $\varphi: V \subset \mathbb{R}^m \longrightarrow U$, with $m \leq k$. V is called the parameter space. Intuitively, a point $p \in U$ is described by $p = \varphi(v)$, $v \in V$, with m degrees of freedom. Therefore the graphical object has dimension m.

Parametric Curves and Surfaces

When $m = 1$ in the above definition, we obtain a *parametric curve* in the k-dimensional space. It is defined by a function $\varphi: I \subset \mathbb{R} \longrightarrow \mathbb{R}^k$, where I is some interval of the real line. When $k = 2$ we have a plane curve, and when $k = 3$ we have an spatial curve.

A *parametric surface* in 3-space is defined by a map $\varphi: U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^3$, from the 2-dimensional plane. This is illustrated in Figure 2.11.

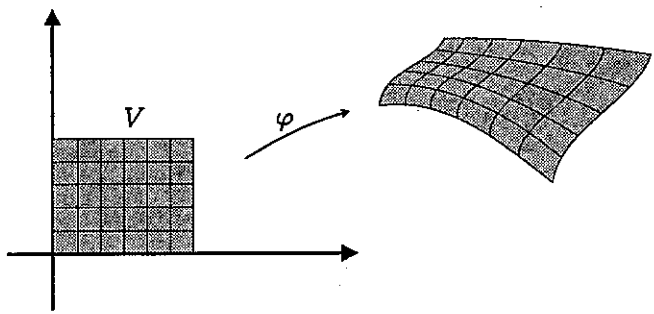


Figure 2.11: Parametric shape description.

Parametric curves and surfaces have its own long chapter in the evolution of shape description techniques in computer graphics. A comprehensive study of this topic is done in (Farin, 1988).

2.2.3 Piecewise Shape Description

It is not always possible to describe the shape of an object globally using an implicit or parametric form (see (Hoffmann, 1989)). Therefore, when describing a complex object, its shape is, in general, decomposed, and each set in the decomposition is described either implicitly or parametrically. Shape descriptions of this type are called *piecewise descriptions*. Figure 2.12 shows an example of a complex shape which has a parametric piecewise description.

It is possible to use a piecewise implicit, piecewise parametric, or even

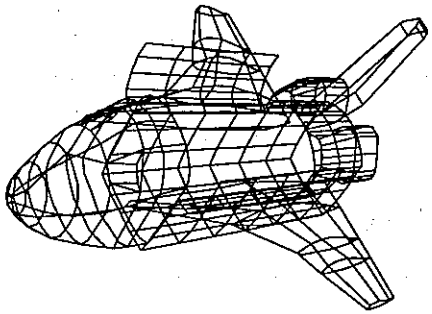


Figure 2.12: Parametric piecewise shape description.

a hybrid piecewise (implicit/parametric) description.

2.2.4 Shape Geometry and Topology

It is very common to impose certain topological or geometrical restrictions when defining a shape. These restrictions are used to guarantee some properties of the object's topology and geometry which are of great importance in the solution of specific problems. A good example, is the use of shapes that are characterized as *manifolds*. Manifold shapes are locally equivalent to euclidean spaces, but globally their topology can be quite complex.

Shape description is a very important, and widely studied, topic in the area of geometric and solid modeling. It would be impractical to go over details here. For more information, the reader should consult (Hoffmann, 1989), (Gomes et al., 1993), or the abundant references therein.

2.2.5 Point Membership Classification

Apart from the shape description technique used to define the object's shape, it is important to point out that, from the description of a shape $\mathcal{O}$, we must be able to compute the *characteristic function* $\chi_{\mathcal{O}}$ of the shape:

$$\chi_{\mathcal{O}} = \begin{cases} 1 & p \in \mathcal{O}; \\ 0 & p \notin \mathcal{O}. \end{cases}$$

The characteristic function allows us to characterize the shape: it distinguishes points that belong to the shape, for which $\xi_{\mathcal{O}} = 1$, from points that do not belong to it, where $\xi_{\mathcal{O}} = 0$. For this reason, this function is well known in the geometric modeling literature by the name of *point membership classification* function. Robust methods to compute this function constitute an important part in the process of shape description and representation.

From a computational point of view an object is well described, when its point membership classification function is defined, and it is computable. Of course we have to precise the meaning of the work computable. The interested reader should consult (Blum, 1991).

Chapter 3

Representation of Graphical Objects

We can use only a finite number of variables and parameters in any computational process. Therefore, in order to manipulate a graphical object in the computer it is necessary to devise a discretization of the object's shape and its attributes. As described before, in Chapter 1, this operation takes us from the, continuous, mathematical universe to the, discrete, representation universe.

In this chapter we will study the problem of representation of a graphical object. That is, our objective is to obtain a reasonable, and correct, answer to que question: how different graphical objects are mapped into the representation universe?

3.1 Object Representation

Intuitively, an object representation is a relation between the mathematical and the representation universe. It associates to each object a discrete description of its geometry, topology and attributes. This is necessarily a two-step process:

- discretization of the object's shape;
- discretization of the object's attribute function.

When we define a representation $\mathcal{O}'$ of some graphical object $\mathcal{O}$, it is very important that we are able to recover $\mathcal{O}$ from its representation $\mathcal{O}'$. This operation is called *reconstruction*. Therefore, the mathematical and representation universes are related by the operations of representation and reconstruction as illustrated in Figure 3.1.

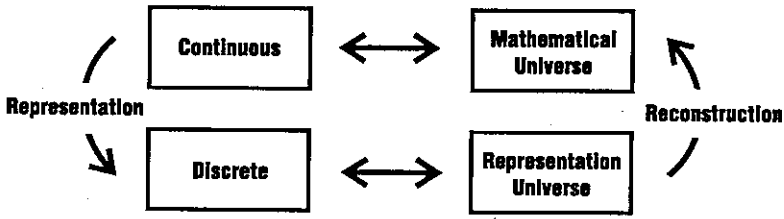


Figure 3.1: Discretization and reconstruction of graphical objects.

In this chapter we will study the representation of graphical objects. The reconstruction operation is also of fundamental importance in computer graphics. It will be discussed with details in next chapter.

We should remark that we will study object representation from a global point of view. In this way, we will not focus on specific representation techniques for different graphical objects. Our main concern is to present a general framework for object representation. In this framework, we will see that the problem of representing an image is exactly the same of representing a geometric model, because both of them are graphical objects.

The reason for the existence of a classical distinction between image and geometric model representation, is related to the fact that images and geometric models have been historically considered as completely different graphical objects. We think that there is a simple explanation for this singular fact: an image has a very well defined and trivial shape (in general, a rectangle in the plane), therefore the discretization process is focused on the attribute function. It seems natural to borrow techniques from the area of signal processing in order to obtain a representation. In geometric modeling the emphasis has been on the object's

shape rather on its attributes. Therefore, objects with complex shapes require more elaborated representation techniques.

It should be clear that by the representation of an object we mean a discretization of its shape and its attributes. The shape of an object can be described in many different ways, and its representation is highly influenced by its description. The attributes of a graphical object are defined by a function whose domain is the object's shape. Therefore, its representation reduces to the classical problem of function representation studied in the area of signal processing.

These two facts split the study of object representation into two different areas:

- function representation;
- shape representation.

We should point out however that these two faces of the representation problem are interrelated. This is clear from the fact that a shape can be defined using a functional description. In fact, the basic ingredients in the definition of a function are its domain U, and the expression f which allows us to compute the image $f(p)$ for each point $p \in U$. Function representations can be obtained by either decomposing the domain U, into simpler subsets, or decomposing the function expression $f(x)$ into simpler ones. As we will see in this chapter, both methods are useful when representing graphical objects.

3.2 Function Representation

The shape of a graphical object can be defined using a functional description. Also, a functional description determines the object attributes. Therefore the study of function representation is of most importance.

Consider some function space Ω, that is, Ω consists of an space whose elements are functions. A representation of a function $f \in \Omega$ is done by decomposing f using some prescribed "dictionary". The dictionary is defined by some family $\{g_\lambda; \lambda \in \Lambda,\ g_\lambda \in \Omega\}$ of functions. For any

function f of the space, we write

$$f = \sum_{\lambda \in \Lambda} c_\lambda g_\lambda. \tag{3.1}$$

where the sum is supposed to exist.

The correspondence

$$f \mapsto (c_\lambda)_{\lambda \in \Lambda} \tag{3.2}$$

constitutes a representation of the function f using the dictionary $\{g_\lambda\}$. This decomposition is called the *analysis* of the function f, and equation (3.1) which allows us to reconstruct the function f from its representation, is called the *syntheses*. Of course, the mathematical details of function representation can get quite complicated. In fact, just to mention some problems, we could list:

- give a clear definition of the space of functions Ω;
- construct the dictionary $\{g_\lambda\}$;
- study the problem of syntheses and analysis associated with some prescribed dictionary;
- devise robust techniques to compute the representation, and to reconstruct a function.

We will not go over these details here. Defining the space of functions needs a substantial knowledge of the theory of measure and integration. Constructing good dictionaries is one of the purposes of the non-elementary theory of functional analysis. Therefore, we will only discuss briefly some representation techniques commonly used in computer graphics. This discussion will lack the mathematical rigor, but it will give the reader the opportunity to devise the beauty and the difficulties of the area of function representation.

From a computational point of view we need a version of the reconstruction equation (3.1) with a finite number of terms on the sum. In general we write

$$f_N = \sum_{k=1}^{N} c_k g_k, \tag{3.3}$$

where we impose the condition

$$||f - f_N|| \longrightarrow 0 \quad \text{as} \quad N \longrightarrow \infty, \tag{3.4}$$

for some prescribed norm $||\ \ ||$ on the space of functions.

There are some cases where the reconstruction of the function f does not use the same family of functions g_λ from the dictionary. These functions are used to obtain another family $\tilde{g}_\lambda$, and coefficients $\tilde{c}_\lambda$ such that

$$f = \sum_\lambda \tilde{c}_\lambda \tilde{g}_\lambda. \tag{3.5}$$

From the point of view of operator theory, a function representation scheme is a non-singular linear operator $R: \Omega_1 \longrightarrow \Omega_2$ between two space of functions. The reconstruction is done by the inverse operator R^{-1}.

Point Sampling

This is the simplest and mostly used function representation scheme. We choose a sequence

$$(\ldots, t_{-1}, t_0, t_1, t_2, \ldots),$$

of points from the domain of the function f, and the representation is the sequence obtained by evaluating f at the points of (t_i):

$$f \mapsto (\ldots, f(t_{-1}), f(t_0), f(t_1) \ldots).$$

This is illustrated in Figure 3.2.

The decomposition dictionary is defined by the family of Dirac delta masses

$$\delta(t - t_k), \quad k \in \mathbb{Z}.$$

In fact a well known property of these masses show that

$$f(t_j) = \int_{\mathbb{R}} f(t)\delta(t - t_j)dt. \tag{3.6}$$

This representation technique is called *point sampling.*

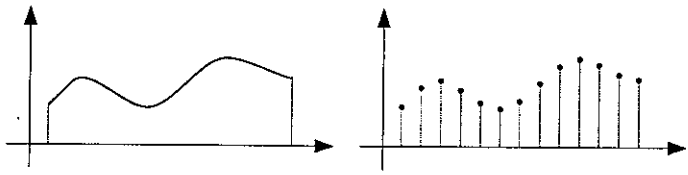


Figure 3.2: Point sampling.

Area Sampling

From the point of view of the general theory of functions, it does not make much sense to evaluate a function at a sequence of points. In fact, if we had defined precisely the concept of a function space, we would have identified functions which differ in sets of measure zero.

Apart from technical considerations, it is easy to convince ourselves that if a function is not continuous, point sampling is prone to severe problems. This is illustrated in Figure 3.3, which shows that samples of $f(t_1)$ and $f(t_2)$ are quite distinct, in spite of the fact that the points t_1 and t_2 are close.

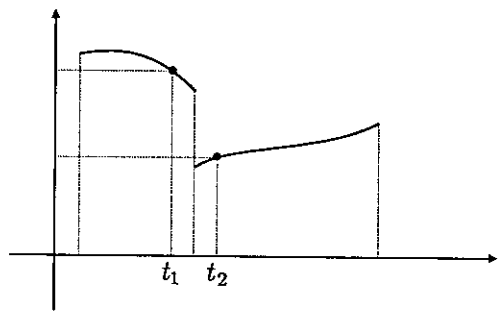


Figure 3.3: Discontinuity and point sampling.

As we will study later, discontinuities introduce arbitrarily high frequencies in the function, and in this case we need to use more robust representation techniques. A very common representation technique in this case consists in substituting point sampling by some average technique. This accounts for the substitution of the Dirac mass δ in equation (3.6) by some function that performs a weighted average.

Average representation techniques are generically known in the computer graphics literature by the name of *area sampling*. Different choices of the dictionary weighting functions, originate different area sampling techniques. Area sampling has been used in the generation of sinthetic images for a long time. They come with different flavors: analytical sampling, α-buffer, supersampling, etc. What distinguishes these different area sampling techniques are essentially the computational method used.

In what follows, we will discuss some of the most used averages for area sampling computation.

Box Average

This is the simplest case. We take a partition $a = t_0 < t_1 < \cdots < t_n = b$ of the interval $[a, b]$ where the function f is defined. The dictionary functions g_k, $k = 0, \ldots, n-1$, are defined by normalizing the characteristic function $\chi_{[t_k, t_{k+1}]}$ of each partition interval $[t_k, t_{k+1}]$. That is,

$$g_k = \frac{1}{t_{k+1} - t_k} \chi_{[t_k, t_{k+1}]}.$$

The graph of this function is illustrated in Figure 3.4. It is called a *box* function.

In this case, in each interval $[t_k, t_{k+1}]$, the function f is represented by the analytical average

$$\int_{\mathbb{R}} g_k(t) f(t) dt = \frac{1}{t_{k+1} - t_k} \int_{t_k}^{t_{k+1}} f(t) dt$$

of the function f on the interval $[t_k, t_{k+1}]$.

The reader should notice that the above dictionary of characteristic functions introduces high frequencies on the reconstructed function, because of the discontinuities.

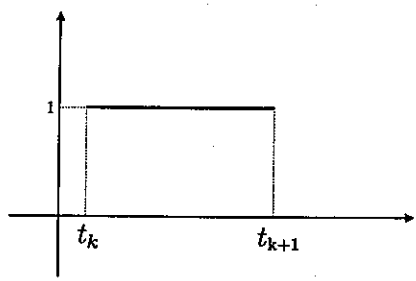


Figure 3.4: Graph of the box function.

Triangle Average

Improved average dictionaries are obtained by successive smoothing of the characteristic function dictionary above. In a first step, by convolving the box function g_k with itself, we obtain the *hat function* $h_k = g_k * g_k$, which on the interval $[t_{k-1}, t_{k+1}]$ is given by

$$h_k(t) = \begin{cases} 0 & t \leq t_{k-1}, \quad \text{or} \quad t \geq t_{k+1} \\ \frac{t-t_{k-1}}{t_k - t_{k-1}} & t \in [t_{k-1}, t_k] \\ \frac{t_{k+1}-t}{t_{k+1}-t_k} & t \in [t_k, t_{k+1}] \end{cases}$$

The graph of this function is shown in Figure 3.5

Higher Degree Averages

Continuing with the smoothing process, will lead us to splines dictionaries of order 2, 3, 4, and so on. Figure 3.6 shows the graph of a basic spline of degree 2, defined by three successive convolutions $s_k = g_k * g_k * g_k$. This family of splines approximate the gaussian function as the degree grows to infinity.

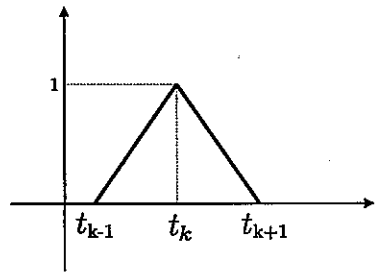


Figure 3.5: Graph of the hat function.

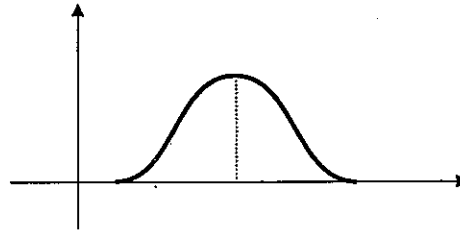


Figure 3.6: Graph of a basic cubic spline function.

Supersampling

This average technique consists in obtaing a partition of the function domain U into a finite number of sets

$$U = \bigcup_i U_i.$$

In order to compute the function average on each partition set U_i, we choose a finite number of points $p_1, p_2, \ldots, p_n$, sample the function f at these points, and take the arithmetic average

$$\frac{1}{n} \sum_{i=1}^{n} f(p_i),$$

as the sample of the function in U_i.

It can be shown that as the number of points increase to infinity, the above average converges to the analytical average of the function on the set U_i, that is

$$\lim_{n \longrightarrow \infty} \frac{1}{n} \sum_{i=1}^{n} f(p_i) = \frac{1}{\text{Area}(U_i)} \int_{U_i} f(u) du.$$

For this reason, we consider supersampling as being an area sampling tecnhique.

3.2.1 Quantization

We discussed in the previous sections the problem of function discretization. We concentrated our discussion in representing the function by simpler functions from a dictionary. It is very important to observe that the representation values must also be discretized in order to obtain a finite representation for the function.

The process of discretizing the representation values is called *quantization.*

3.3 Shape Representation

We have already mentioned before the relationship between functional and shape representation. A function $f: U \longrightarrow \mathbb{R}^m$ is basically characterized by its domain and an equation $y = f(x)$, defining the correspondence rule. In order to represent f we have two options: decompose the equation f or the domain U. Of course there is a close relationship between these two approaches. In the section about function representation we discussed function decomposition, and in this section we discuss domain decomposition.

When we have a very complex shape, in general it cannot be defined by a global functional description. In this case, we must devise more complex representation schemes. These representations subdivide into two main categories:

- Constructive Shape Representation;

- Decomposition Shape Representation.

Constructive Shape Representation;

This representation is based on the existence of some algebra defined over a collection of subsets of the space. We select a finite number of simple shapes $s_1, s_2 \ldots s_n$, and from them we obtain more complex shapes by representing them using an arithmetic expression from the algebra

$$S = f(s_1, s_2, \ldots, s_n).$$

The simpler objects are called *primitive graphical objects*, or simply *primitive shapes*. Primitive objects are chosen based on he fact that they should be easy to describe and represent. In simpler words, in the constructive shape representation, complex objects are constructed from simpler objects using shape operators.

A classical example of constructive shape representation is given by the CSG[1] representation used for solid modeling (Requicha, 1980). This representation uses the Boolean algebra of sets, with the operations of union, intersection, and complementation. This representation is illustrated in Figure 3.7: Figure (a) shows a graphical object, and figure (b) shows how this object is constructed from the primitive shapes of a square with side of length 1, and a circle of radius 1, both centered at the origin.

The object in Figure 3.7 is obtained by conveniently traversing the tree structure on Figure 3.7(b) and performing the indicated operations according to the following semantics

- $-$ is the set subtraction operation (left branch minus right branch);
- $+$ is the usual set union operation;
- $S(s_1, s_2)$ is the scaling transformation of the plane;
- $T(x, y)$ is the translation of the plane by the vector (x, y).

[1] CSG means Constructive Solid Geometry

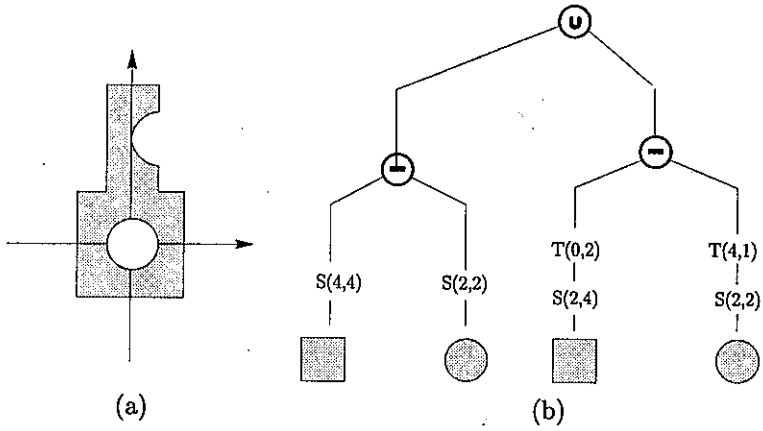


Figure 3.7: CSG constructive representation.

In order to avoid degenerated shapes, the set operators used by the CSG representation need to be combined with a topological operation which performs a regularization. We do will not enter in details here. The interested reader should consult (Hoffmann, 1989).

Another important example of constructive shape representation is given by the use of Binary Space Partition trees (BSP trees). In (Naylor, 1994) the reader can find a good description of the use of BSP trees to represent shapes. In (Radha, 1993) BSP trees have been used to represent images.

Shape Representation by Decomposition

In a decomposition representation, the object shape is subdivided into a family of disjoint subsets

$$U = \bigcup_{\lambda} U_{\lambda}. \tag{3.7}$$

This representation is based on the well known "divide and conquer" paradigm. The geometry of each subset U_{λ} is easier to describe, and

a combinatorial scheme allows us to reconstruct the original object at least with a good degree of approximation. Different decomposition techniques can be used, based on the huge multitude of space decomposition techniques (see (Carvalho, Gomes and Velho, 1993)).

A classical use of decomposition representation is the BRep representation used in geometric modeling[2]. In this representation the object's shape is defined by its boundary, and the boundary is represented by a decomposition into vertices, edges, faces and shells (Baumgart, 1975). A combinatorial scheme is necessary to guarantee the correct topology of the boundary shape of the reconstructed object. This representation is illustrated for a "two dimensional solid" in Figure 3.8

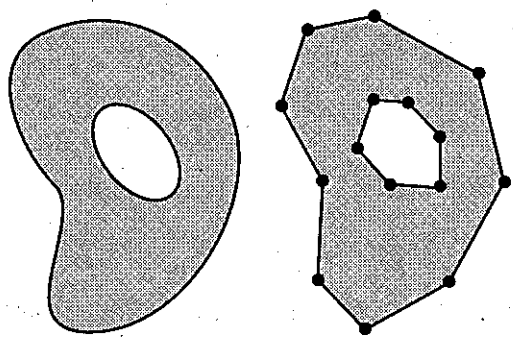


Figure 3.8: Boundary representation by decomposition.

Matrix Representation

The oldest example of a representation by decomposition of a graphical object is certainly the matrix representation used to obtain a digital image from a continuous one. This decomposition has been extended to represent volumes, and has influenced the development of other decomposition representation techniques.

[2]BRep stands for Boundary Representation

Consider an image $f: [a,b] \times [c,d] \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^n$. The matrix decomposition is obtained by making a uniform decomposition of the image's shape

$$U = [a,b] \times [c,d] = \{(x,y) \in \mathbb{R}^2; a \leq x \leq b, \text{ and } c \leq y \leq d\}.$$

We define a grid, Δ_U, on U,

$$\Delta_U = \{(x_j, y_k) \in U; x_j = j \cdot \Delta x, \quad y_k = k \cdot \Delta y \quad j,k \in \mathbb{Z}, \quad \Delta x, \Delta y \in \mathbb{R}\},$$

as illustrated in Figure 3.9.

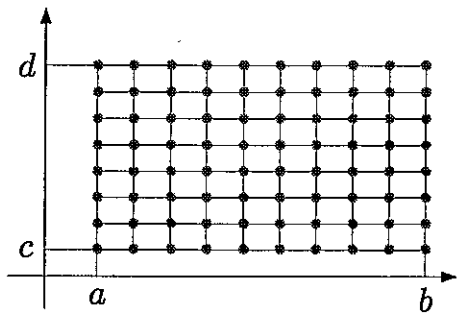


Figure 3.9: Matrix representation of an image.

Each rectangle in the decomposition is completely described by the coordinates (x_j, y_k). The image attribute function is represented (sampled) in each rectangle and the value obtained is associated with the integer coordinates (j,k). The image $f: U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^n$ is conveniently represented by the $m \times n$ matrix $A = (a_{jk}) = (f(x_j, y_k))$. This is the reason for the name *matrix representation.*

In the literature the matrix representation is also called *raster representation*, because the graphical object represented by this method is known by the name of *raster graphics.* We prefer the name matrix representation, because it reflects the generality of the representation and its extensions to higher dimensional spaces. On the contrary, the term raster representation, is closely related with image display on *raster devices* (where the matrix representation came from).

The matrix representation can also be used to represent graphical objects other than images. The bidimensional grid defined on the image domain U is easily extended for n-dimensional euclidean space. By conveniently enumerating the grid cells, we define the object's geometry and topology. In each cell, we define the object's attributes by restricting the attribute function to the cell, and representing this restriction using some function representation technique. In Figure 3.10, from (Gomes and Velho, 1995b), we show matrix representations of $1D$ (circle), $2D$ (disk) and $3D$ (solid torus) graphical objects.

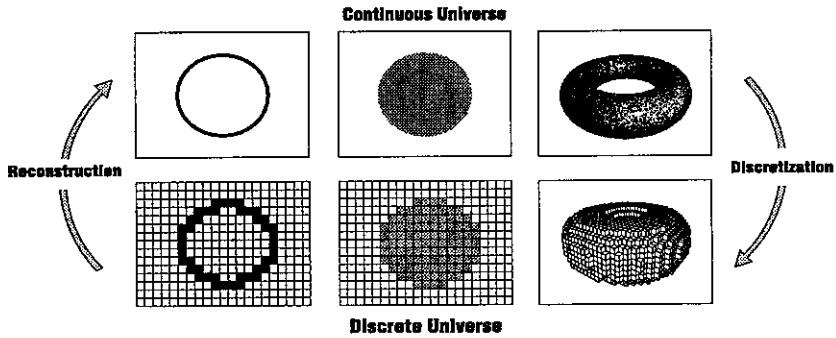


Figure 3.10: Matrix representations of different objects).

In the area of solid modeling the matrix representation is called *spatial enumeration.* It is a very popular representation method for volume data (see (Kaufman, 1994)). In computer graphics in general, it is called a *volume array.* Researchers from image processing prefer to refer to a tridimensional matrix representation of a solid by the name of *3D-image.*

The process of obtaining a matrix representation from a continuous description of a graphical object is called *rasterization.* The continuous description of a graphical object is usually called a *vector description.* For this reason, on the literature the word rasterization appears as the operation that converts from vector to raster graphics.

We should point out that the rasterization operation consists of two steps:

- discretize the object's shape into the matrix blocks;
- compute the object's attributes for each block.

Adaptive Decompositions

The matrix representation uses a uniform spatial grid to obtain a decomposition of the object. More efficient decomposition representations use an adaptive subdivision of the object's shape. The adaptiveness criteria in general exploits some properties of the attribute functions of the object. Well known examples of these representations are the *quadtree*, for bidimensional graphical objects, and the *octree*, for 3-dimensional objects. Figure 3.11(a) shows an adaptive decomposition by triangles; Figure 3.11(b) shows a quadtree decomposition.

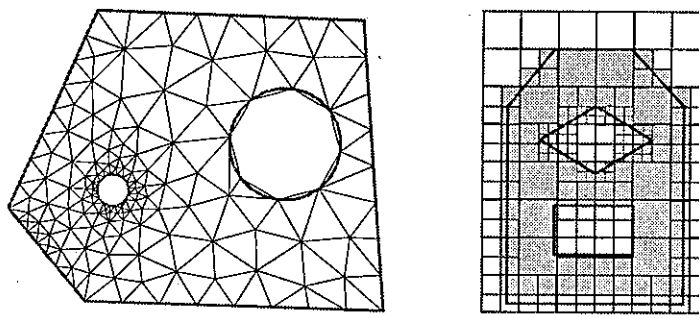


Figure 3.11: Adaptive decompositions.

We should point out here the poor choice of the names used for these adaptive representation by decomposition: these names make a confusion between the representation method, and the subjacent spatial data structures used to associate the representation with some computer language for implementation purposes.

Chapter 4

Reconstruction of Graphical Objects

As we mentioned in the previous chapter, when we define a representation $\mathcal{O}'$ of some graphical object $\mathcal{O}$, it is very important that we are able to recover $\mathcal{O}$ from its representation $\mathcal{O}'$. This operation is called *reconstruction.* The mathematical and representation universes are related by the operations of representation and reconstruction, as illustrated in Figure 4.1

In this chapter we will study the problem of representation and reconstruction of graphical objects. These two problems are related to two important questions:

How is it possible to obtain a continuous description of a graphical object from the discrete version of its representation?

The reconstruction operation is of fundamental importance in computer graphics. An image must be reconstructed by some display device, such as a monitor, before it can be viewed. To perform computer calculations with a graphical object, it must be reconstructed, so that the computations can be done on the continuous domain.

We must observe that the reconstruction process occurs whenever the object must be materialized. The materialization process depends on the application. From the user's point of view the reconstruction process must be performed, so that we can observe the object; from the

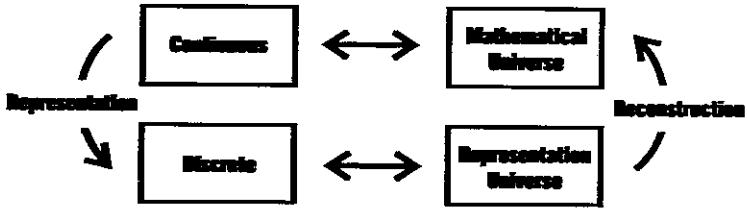


Figure 4.1: Discretization and reconstruction of graphical objects.

system's point of view, correct reconstruction is a guarantee of the correct semantics of the object representation.

4.1 Representation and Reconstruction

The representation of a graphical object is complemented by the reconstruction operation. More precisely, we start with a graphical object $\mathcal{O}$, represent it obtaining another object $\mathcal{O}_d$. In order to perform some operations with the represented object $\mathcal{O}_d$, we need to reconstruct it, obtaining another continuous object $\mathcal{O}_r$. When the original object $\mathcal{O}$ coincides with the reconstructed object $\mathcal{O}_r$, $\mathcal{O} = \mathcal{O}_r$, we say that the reconstruction is *exact*.

When the object shape is very simple (e.g. an image), the problem of representation reduces to the problem of discretization and reconstruction of its attribute function. When the object shape is complex, we have also to cope with shape discretization and reconstruction.

Ideally, both in shape and attribute discretization, we should be able to obtain exact reconstruction from the discrete representation of the object. The experience from image processing shows that this is a very difficult task. If we are not able to perform an exact reconstruction, we must devise a reconstruction methodology to recover important properties of the object on the continuous domain. These properties are related with the object shape, and its attributes:

- recover the topology;

- recover the geometry;
- recover the object attributes.

The object characteristics that we should recover in the reconstruction process is greatly dependent on the applications. In general, a good compromise consists in using representations that allow us to recover the topology, and obtain a good approximation of the object geometry and some of its attributes.

This problem is well illustrated by the simple example of the B-Rep representation shown in Figure 3.8, of Chapter 3 (page 57). We should remark however, that, in some applications, maintaining the object's topology is not the most important focus of the problem.

Representation and reconstruction are the two major problems we face when working with graphical objects. More specifically,

- we must devise robust representation schemes. This means that we must devise discretization methods that carry relevant information about the geometry, topology, and the attributes of the graphical object.
- we must develop reconstruction techniques in order to obtain the original object from its representation. Since the representation is, in general, not unique, reconstruction techniques are closely related to each particular representation.

Therefore, good solutions for the representation/reconstruction problem have a great dependency on each specific problem. When we discretize and reconstruct images, we usually exploit the relationship between resolution and visual acuity. Visual acuity is also exploited when we use polygonal decimation to reduce the number of polygons according to the distance that the object is being observed. On the other hand, when the representation is used as input for a milling machine, the reconstruction process must generate the object's shape within tolerance bounds which are acceptable for the manufacturing process.

4.1.1 Reconstruction and Extension

The problem of function reconstruction is in fact a special case of a very important problem in topology, known by the name of *function extension*. This problem can be pose in the following way:

Consider a domain U of the euclidean space $\mathbb{R}^m$, a subset $A \subset U$, and a map $f: A \longrightarrow V \subset \mathbb{R}^n$. The problem of mapping extension consists in finding a map $h: U \longrightarrow V$, such that the restriction $h|A$ of h to the set A coincides with f, that is $h|A = f$.

In general we impose additional conditions on the extended map h.

It is clear that the reconstruction problem is an special case of the extension problem. In a close inspection of the two problems show us that the main distinction is a matter of point of view: when we use the term reconstruct, it means that the function $f: A \longrightarrow V$ was obtained by somehow restricting a previously known function h. When we use the term extend, it means that we have no previous information about f.

4.2 Function Reconstruction

In this section we will discuss the problem of function reconstruction. We start with a function f, and obtain its representation f_d. When f_d undergoes a reconstruction process, we obtain a function f_r. Ideally we should look for exact reconstruction, that is $f_r = f$, but this is very difficult to obtain. In order to understand why, we should point out that the reconstruction process consists in recovering the function from the dictionary elements

$$f_r = \sum_{\lambda} c_\lambda g_\lambda. \tag{4.1}$$

Several problems may arise which prevent f_r from being equal to f:

- The dictionary might have an infinite number of "basis" functions g_λ. This means that the sum in (4.1) is infinite, and must be truncated during computation;
- The dictionary basis functions g_λ might not have compact support. Therefore, they must be restricted to some bounded domain for computations;

- The representation process, for obtaining the coefficients c_λ, may introduce a loss of information about the function (representation is not exact).

A very important and illustrative case occurs when we use representation by point sampling. A partial, but relevant, answer for the reconstruction from point sampling problem is given by the famous theorem of Shannon (Shannon, 1949). In order to state the theorem, we consider a function f with finite energy

$$\int_{\mathbb{R}} f^2(t)dt < \infty, \tag{4.2}$$

and such that there exists a constant $K > 0$ satisfying

$$\hat{f}(u) = 0, \quad \text{for} \quad |u| > K. \tag{4.3}$$

where $\hat{f}$ is the fourier transform of the function f. The function is said to be *band limited.*

Finite energy is a very plausible condition from a physical point of view. The condition on equation (4.2) states that the function does not have arbitrarily high frequencies. In fact, this condition implies the f derivatives up to a certain order are bounded, and this means that f does not vary too much.

Shannon's theorem states that, under the above conditions, if f is sampled by taking points $1/(2K)$ apart, then the point sampling representation is exact. More precisely, we can recover f from its samples, using the equation

$$f(t) = \sum_{n=-\infty}^{+\infty} f(\frac{n}{2K}) \frac{\sin \pi(2Kt - n)}{\pi(2Kt - n)}. \tag{4.4}$$

The sampling interval in Shannon theorem is $\Delta t = t_{n+1} - t_n = 1/(2\Omega)$. Therefore the sampling rate is $\sigma = 1/\Delta t = 2\Omega$. This is the minimum number of samples we should take per unit interval, if we wish to recover the function f completely from its samples. This sampling rate is called *Nyquist rate.* Of course, higher sampling rate than Nyquist's rate can

be used to obtain exact reconstruction. Using more samples than the Nyquist rate, is called *supersampling*.

The graph of a typical dictionary function used in the Shannon theorem,

$$g_n(t) = \mathrm{sinc}\pi(2Kt - n) = \frac{\sin \pi(2Kt - n)}{\pi(2Kt - n)} \tag{4.5}$$

is shown in Figure 4.2, for $n = 0$.. Notice that these functions do not have compact support. Therefore the reconstruction equation (4.4) does not furnish exact reconstruction in practice.

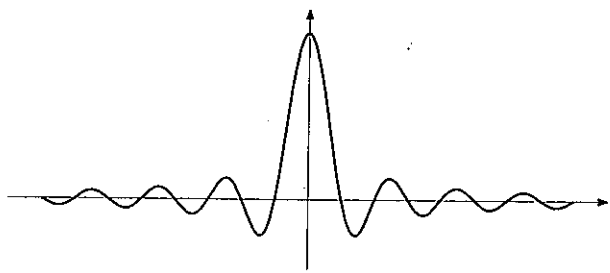


Figure 4.2: Graph of the Shannon basis function.

In general, the dictionary functions in the Shannon reconstruction are substituted by simpler functions with compact support, which are simple to compute. The most used functions are the box, triangle and higher degree splines studied before.

Figure 4.3 illustrates the reconstruction process with these dictionary functions for the one-dimensional case. In (a) we have a box reconstruction, the resulting function is piecewise constant. In (b) we show a reconstruction with a hat function, the resulting function is piecewise linear. In (c) we show the reconstruction with basic splines of degree 3. In this case the resulting function is a piecewise cubical polynomial of differentiability class C^2.

Notice that as we increase the degree of the reconstruction basis, we obtain smoother reconstructions. Lower degree filters introduce high

frequencies in the reconstructed signal. Higher degree filters smooth out high frequencies, and have the effect of blurring the reconstructed image.

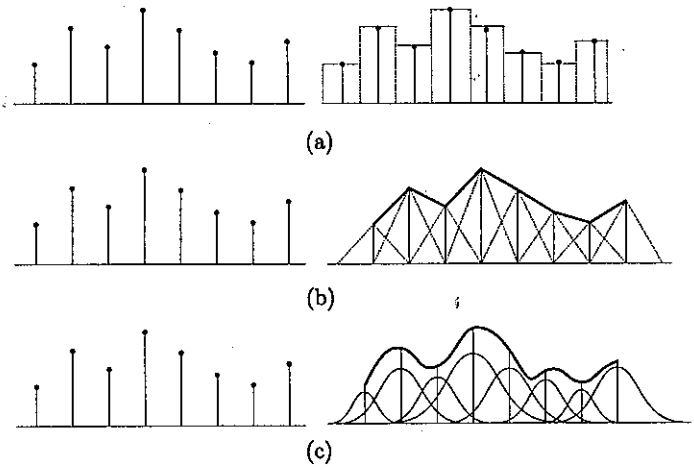


Figure 4.3: Reconstruction with box, hat and spline dictionaries.

4.3 Shape Reconstruction

When we have a functional description of a shape, all of the discussion from the previous section applies to the problem of shape sampling and reconstruction. The additional problem when dealing with shapes, consists in representing and reconstructing the shape topology.

The operation of shape reconstruction is usually called *structuring*. When the object is defined using a parametric description, the structuring is easy, because it is trivially induced by the structuring of its domain. In general the domain is decomposed using a matrix representation. For implicit descriptions of graphical objects, the problem of sampling and reconstruction is much more difficult. This problem has been discussed to some extent in (de Figueiredo, 1992).

A simple, but illustrative, example of shape representation and reconstruction is shown in the following section.

Circle Sampling and Reconstruction

In Figure 4.4(a)-(b) we show the discretization of a circle using 5 samples. In Figure 4.4(c) we make a linear reconstruction of the circle from its samples, obtaining a pentagon. In this reconstruction process, we obtain the exact topology, and a coarse approximation of the circle's geometry. In figure 4.4(d), we make a linear reconstruction of the circle from the same samples, using a wrong topology information.

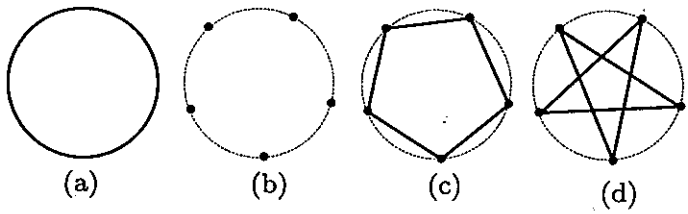


Figure 4.4: Circle representation and reconstruction.

Figure 4.5(c) shows a linear reconstruction of the circle, from four samples. In Figure 4.5(d) we reconstruct the circle from the same samples, using cubical (Bezier) reconstruction.

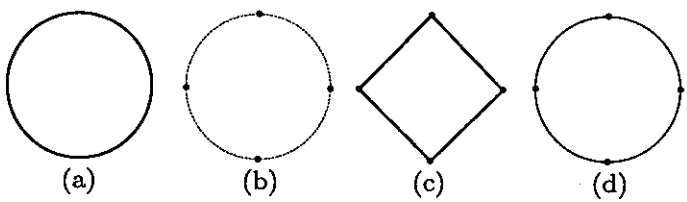


Figure 4.5: Linear and cubical reconstruction.

4.4 Aliasing

In this section we will discuss briefly a very important problem that occurs when we perform point sampling, without taking into account the Nyquist sampling rate.

4.4.1 Aliasing and Sampling

Intuitively, Shannon theorem says that if a function has finite energy, and does not have arbitrarily high frequencies, than it is possible to obtain an exact point sampling representation by sufficiently increasing the sampling rate. This can be seen by looking at the function in the frequency domain, using a Fourier transform to go from the spatial to the frequency domain. This is illustrated in Figure 4.6.

The comb function shown in (b) is a sum of Dirac delta functions spaced according to the sampling rate, in such a way that the samples of the signal f, shown in (c), are obtained by multiplying f by the comb function.

It is well known from Fourier analysis that the Fourier transform of a comb function is another comb function with different spacing between two consecutive impulses. Also, multiplication in the spatial domain correspond to convolutions in the frequency domain (Gonzalez and Wintz, 1987). These classical results are vital to derive (e) and (f): the comb function shown in (e) is the Fourier transform of (b), just like (d) is the Fourier transform of (a); and (f), which is (c) in the frequency domain, is also obtainable as a convolution of (d) and (e).

According to Figure 4.6, sampling in the spatial domain, is equivalent to convolution with a comb function on the frequency domain, and the frequency of this comb function varies inversely with the sampling frequency.

From the above scenario, we conclude that if the sampling rate is sufficient high, and the function is band limited, we do not have overlapping in the spectrum of the sampled function (Figure 4.6(f)). Therefore, it is possible to recover the original spectrum of the function f from the spectrum of the sampled function, using some filtering technique. From the original espectrum, we reconstruct the function f using the inverse

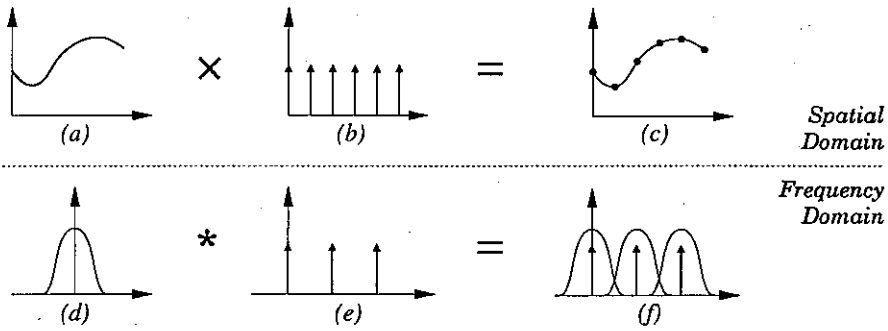


Figure 4.6: Sampling process in spatial and frequency domains.

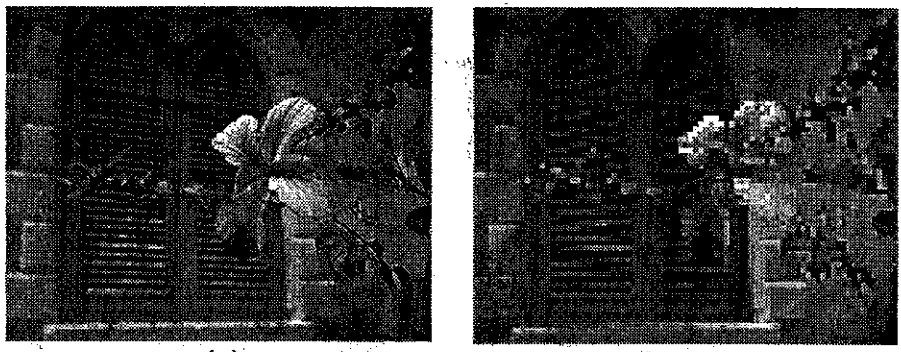
Fourier transform. In this process we obtain the reconstruction equation (4.4).

On the other hand, when the sampling rate is not sufficiently high the function frequencies overlap in the convolution process (see Figure 4.6(f)). When this happens, it is impossible to separate the frequencies of the original function from the frequencies introduced by the sampling process. Therefore, during the reconstruction high frequencies will be introduced. These high frequencies interfere with the original frequencies, causing a distortion of the reconstructed function. This phenomenon is called *aliasing*.

Figure 4.7, from (Gomes and Velho, 1995b), illustrates the distortion caused by aliasing in the sampling process. The image in (a) was sampled using a sampling rate compatible with the image frequencies. Aliasing artifacts are not noticeable. The image in (b) was sampled using a low sampling rate and the aliasing caused a distortion on the window shade.

4.4.2 Aliasing and Reconstruction

In general, the process of sampling and reconstruction is prone to artifacts. Sampling artifacts are known as aliasing, and reconstruction artifacts arise because, in general we do not have exact reconstruction.



(a) (b)

Figure 4.7: Aliasing artifacts

When we sample a function with aliasing, it is impossible to have exact reconstruction. When it is sampled without aliasing, exact reconstruction is still, in general, impossible because of the several reconstruction problems mentioned previously on this chapter. But since the sampling was done according to bounds imposed by the Shannon theorem, we have greater flexibility in choosing reconstruction basis, which seems more adapted to the problem at hand.

Figure 4.8, from (Gomes and Velho, 1995b), shows the same image from Figure 4.7(b) reconstructed using two different reconstruction basis. In figure (a) it was reconstructed with the hat basis. Comparing this image with the image from Figure 4.7(b), we observe that the high frequencies have been smoothed out. This is even more noticeable on the image in Figure 4.8(b), which was reconstructed using the cubic spline basis.

We should point out that the distortions on the windows shade caused by aliasing have not disappeared in Figure 4.8, by the use of better reconstruction basis. Aliasing distortions can only be avoided, or at least minimized, by taking precautions before sampling. According to the statement of Shannon theorem, this can be done in two different ways:

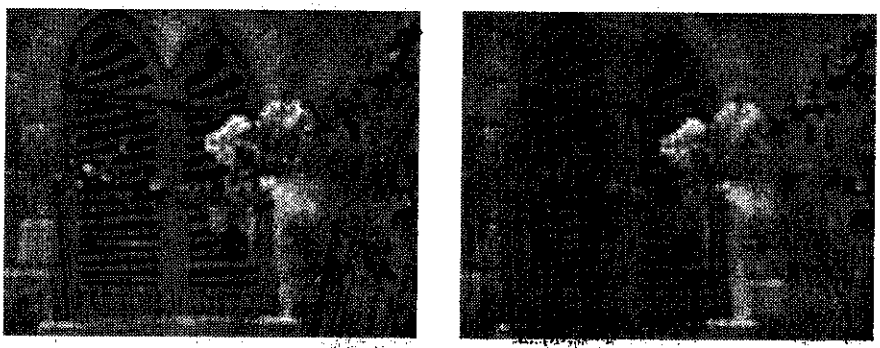


Figure 4.8: Reconstruction with different basis (from (Gomes and Velho, 1995)).

- increasing the sampling rate;
- smoothing out the image high frequencies.

4.5 Resampling

When applying an operation to some graphical object there are several advantages in working on the continuous domain (mathematical universe), rather then on the discrete one (representation universe). Therefore, instead of transforming the discrete representation of the object, we should reconstruct it, apply the transformation to the reconstructed object, and then sample the resulting object. This process, called *resampling*, is illustrated in Figure 4.9, from (Wolberg, 1990), for the one-dimensional case of a function.

In fact, since the object transformation might introduce high frequencies in the transformed object, it is advisable that we use some smoothing process (low-pass filter) in order to decrease the frequencies of the transformed object. This minimizes aliasing problems when sampling the transformed object. Therefore, the resampling operation should be

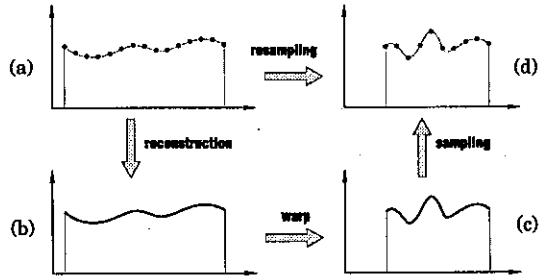


Figure 4.9: Object transformation cycle: reconstruct, transform and sample.

accomplished according to the illustration in Figure 4.10, from (Wolberg, 1990).

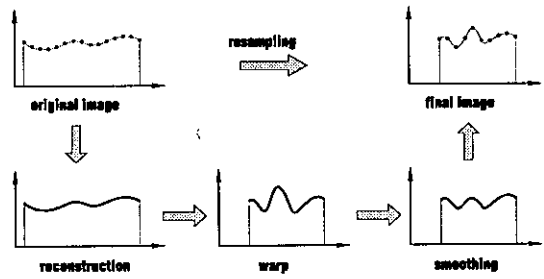


Figure 4.10: Resampling steps.

4.6 Everyday Graphical Objects

In our customary use of objects most emphasis is given on the object's shape, rather then on the attributes (the exception occurs when the object is an image). From a geometric point of view the objects can

be classified according the dimension of its shape, and the dimension of the euclidean space in which they are embedded. We can identify the following types of objects:

- 1 dimensional objects, embedded into 1-dimensional space. This is a one-dimensional signal;
- 1 dimensional objects embedded into k-dimensional space. This is generally called a k-dimensional curve;
- 2 dimensional objects, embedded into 2-dimensional euclidean space (the typical example here is an image);
- 2 dimensional objects, embedded into $\mathbb{R}^k$ dimensional space. There objects are called k-dimensional surfaces;
- 3 dimensional objects embedded into 3-dimensional objects, called solids.
- Objects with fractional dimension, called *fractals.*

When we consider attributes, a lot more objects appear on the scene. We could mention, images, hypertextures, volume arrays ($3D$-images), etc.

In this section we will describe the commonly used graphical objects, based on the conceptual description introduced previously. The main purpose of this section is to fix terminology in order to avoid unnecessary confusion in later chapters.

Color

This is a particular case of the fact that every signal is a graphical object. A color is defined by its spectral distribution which consists of a function $f\colon [\lambda_a, \lambda_b] \longrightarrow \mathbb{R}$, which associates to each visible wavelength λ its energy $f(\lambda)$. It is interesting to observe that this object is in general used as an attribute. The problem of color representation and reconstruction is very important in several applications of computer graphics and image processing.

Images

We use the term *image* when the image $f: U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^n$ is considered in the continuous domain. In general U is a rectangle of the plane, and $\mathbb{R}^n$ is some finite dimensional representation of some color space. This is another example of a graphical object which is a signal.

We should observe that an image can also be represented by the graph $\{(x, y, f(x, y)\ ;\ (x, y) \in U\}$ of its attribute function. This is illustrated in Figure 4.11, from (Gomes and Velho, 1995b). Ths graph is a surface embedded into the $3D$ space, therefore it is another graphical object. This fact shows that the same underlying concept (an image here) can be represented by different graphical objects.

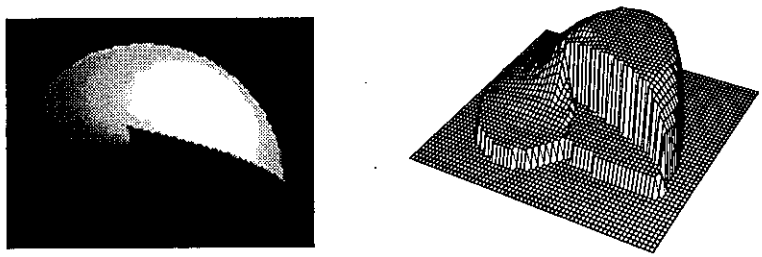


Figure 4.11: Image and graph of its attribute function.

Digital Images

This is an image where the domain and the values of the attribute function have been discretized (sampled and quantized). In general, it is described using a matrix representation as described in Chapter 3

Drawings

Drawings are constituted by some finite collection of curves on the plane. It is very common in a drawing that these curves limit some regions of the plane, and we associate with the drawing, some attributes of these

regions. These objects are sometimes called *two-dimensional solids* (See Figure 4.12).

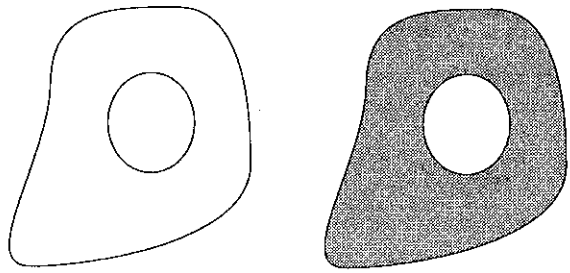


Figure 4.12: Two-dimensional drawing and solid.

The association of spatial attributes to drawings have been widely used in illustration applications under the generic name of *object oriented graphics.*

When the curves in a drawing are represented by polygonal lines, it is usually called a *Polygonal drawing.*

Surfaces and Polyhedral Surfaces

Surfaces are two dimensional objects of the space. A surface can be described in different ways. In general a surface is represented using some polyhedral approximation. When this happens, it is called a *polyhedral surface.*

Solids and Polyhedral Solids

The term solid should be used only for a bounded, tridimensional set of the euclidean space. But the reader will find some exceptions to this rule in the literature. One example is the use of the term two-dimensional solid described before (see Figure 4.12).

When a solid is represented by approximating polyhedrons, it is called

a *polyhedral solid.* In this case, in general, the polyhedra defines the surface which bounds the solid.

Volume arrays and 3D-images

This is a common name used for the matrix representation of a solid. Volume array is the name adopted by most computer graphics researchers. In general $3D$-image is mostly used by researchers from the area image processing.

Scattered data

This is the generic name given for the sampling of some object in the space. Devising reconstruction techniques for scattered data is a very active research area. Difficult problems exist if we do not impose some additional restrictions on the underlying graphical object.

Animation

In general, the word animation is used associated to some finite sequence of images $f_1, f_2, \ldots, f_n$. When this sequence is exhibited with some time frequency, the human eye performs a temporal integration between them, and we perceive motion.

From the point of view of a graphical object, an animation is a map $F: U \times R \longrightarrow \mathbb{R}^n$ where $U \subset \mathbb{R}^2$, and $\mathbb{R}^n$ is some color space. For each $t \in \mathbb{R}$ this map defines an image $F_t: U \longrightarrow \mathbb{R}^n$. The finite sequence which we usually call an animation is in fact a time sampling of the function F. We will discuss this with more details later on in these notes.

Audio

An audio signal is just a function $f: U \subset \mathbb{R} \longrightarrow \mathbb{R}$. This is another example of a very important graphical object which is a signal.

Chapter 5

Transformation of Graphical Objects

The transformation between graphical objects is the basic process in computing with graphics. The main operations in essentially all areas of graphics processing involve some form of transformation between graphical objects. Figure 5.1, from (Wolberg, 1990), illustrates the relationships between the main areas of graphics computation.

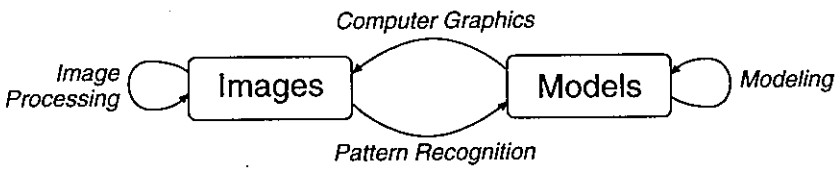


Figure 5.1: Main areas of graphics computation.

This diagram shows two forms of graphical objects, and the main areas of graphics as processes of transformation between them. Apart from distracting details, the main idea behind this diagram is to show the distinction of the transformations between graphical objects of different and same dimensionality. Rendering, for instance, is a process of trans-

formation of 3D graphical objects into a 2D graphical object, usually an image. For an example of a rendering process that outputs drawings rather than images, see (Winkenbach and Salesin, 1994).

Our concern, in the study of transformations of graphical objects is the case where the input and output have the same dimensionality. The applications of such transformations span the areas of image processing and modeling, among others.

The discussion in this chapter is restricted to graphical objects and transformations in the mathematical universe, according to the abstraction paradigms discussed in chapter 1. This more abstract treatment provides the means for a general discussion which is independent of the representation and computational details. Therefore, it is applicable to all kinds of graphical objects. The conceptual study of the general problem will be useful in establishing a broader view of the frequently used 2D and 3D cases, and it will help to improve the study of each case.

5.1 Fundamental Concepts

Most of the examples of metamorphosis mentioned in Chapter 1 occur for real world objects. From the mathematical point of view, this means that they are realized by transformations between real three-dimensional objects. In computer graphics, we are interested in the metamorphosis problems for objects of different dimensions. Several applications involve metamorphosis in dimensions 1, 2 and 3, or even higher dimensional graphical objects:

- one-dimensional metamorphosis could be used to combine different audio signal to obtain some audio effect;
- plane metamorphosis between one dimensional objects (drawings), have been widely used by the desktop publishing industry in order to obtain font warping effects. Also, Disney animators have used several metamorphosis effects in the dozens of movies produced.
- three-dimensional metamorphosis of solids and surfaces can be used to blend different graphical objects, creating new ones. This process was used extensively in the deformation of the T1000 character of

the movie "Terminator 2: The Judgment Day", and on the movie "The Mask".

Objects are visualized by projecting them on the screen using a camera transformation. For real world images we use a photographic, video or movie camera, and for synthetic models we use a virtual camera. Image metamorphosis has been widely used in the movie and video industry. In spite of being a two-dimensional morphing between two-dimensional objects, when we perform an image metamorphosis we are interested in obtaining a transition between the three-dimensional objects depicted on the image. Therefore, it would be fair to say that image metamorphosis is a $2\frac{1}{2}$-D problem.

When the object is synthesized, we have two options to transform an object which appear on the image: either working on the object space, or working on the image space. Working on the object space force us to use $3D$ transformations; working on the image space, means that we should project the objects on the screen, and use $2D$ transformations (see Figure 5.2).

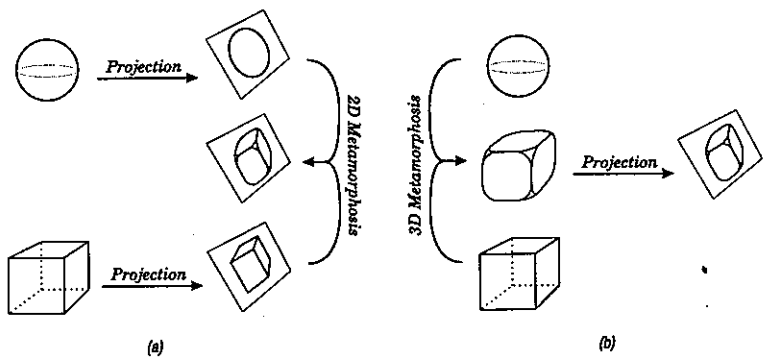


Figure 5.2: Morphing of objects in 3D space: (a) 2D techniques; (b) 3D techniques.

The camera projection reduces a morphing from the three-dimensional space to the plane, decreasing in this way the degrees of freedom we have

to attain some particular effects. Small details of the object shape might not transform well if we work on the image space. For an illustration of this fact, the reader should consult (Lerios, Garfinkle and Levoy, 1995). Also, some of the image characteristics come from the correlation between the objects on the scene (a clear example of this fact are the shadows). Performing the warping on the image space must take into account the warping compatibility between these elements.

Working on the image space may give acceptable results if, among other restrictions, the objects are seen from a similar point of view with very similar lighting conditions. This is illustrated in figure 5.3, which shows a $3D$ and a $2D$ warp of a dice. Although the deformation is just roughly the same, the differences in the transformation of the shadows and highlights are evident. The results of transforming 3D objects in the 2D space can be confusing, as the visual clues given by shadows and highlights will be wrong.

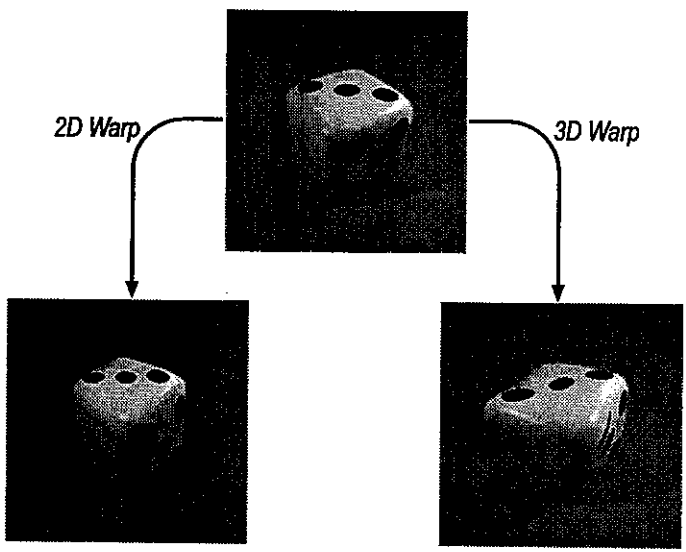


Figure 5.3: Different results transforming in 2D and 3D spaces.

When we must obtain morphing effects between two real world objects (e.g. human beings), in general we have no option besides doing metamorphosis using their images. Nevertheless, when the objects are synthesized we have the choice to perform the metamorphosis either in the object or in the image space. In general, working on the object space gives more flexibility because we have more degrees of freedom, but it is much more difficult to obtain the desired morphing effects.

5.2 Transformations

A graphical object is essentially a function $f: U \subset \mathbb{R}^n \longrightarrow \mathbb{R}^p$, where U is the shape, and f is the attribute function. Therefore, a transformation between graphical objects involves essentially a *functional transformation*, that is, a transformation between two function spaces. A simple and geometric way to achieve function transformations consists in making changes either in the domain U, or in the image set $f(U)$ of the function f. This has the effect of changing either the object shape, or the attribute values of the object. In the first case, we say that we have a *domain transformation*, and in the second case, we say that we have a *range transformation*.

The rotation of an image, is an example of a domain transformation; a change in the color of the pixels of one image is an example of a range transformation.

Now we will give an example to show that working on the function space or making changes in the domain or in the range might give different results.

Procedural Texture

A two-dimensional procedural texture is image $f: U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^k$, where the attribute function f is computed in a procedural way. In general, it depends on a finite number of parameters, and by changing these parameters, we obtain different flavors for the texture attributes. Figure 5.4(a) and (b) shows two different wood textures generated procedurally.

Procedural textures yield an interesting example to illustrate the problem of range and function transformation. Consider the problem of

blending the two wood textures shown in Figure 5.4. Value interpolation here means that we should interpolate the computed texture intensities on a point-by-point basis. Function interpolation, means that we should use a functional interpolation. That is, we should interpolate the parameters that define each of the wood textures. The entirely different results obtained, can be seen by the two images in the middle of figure 5.4.

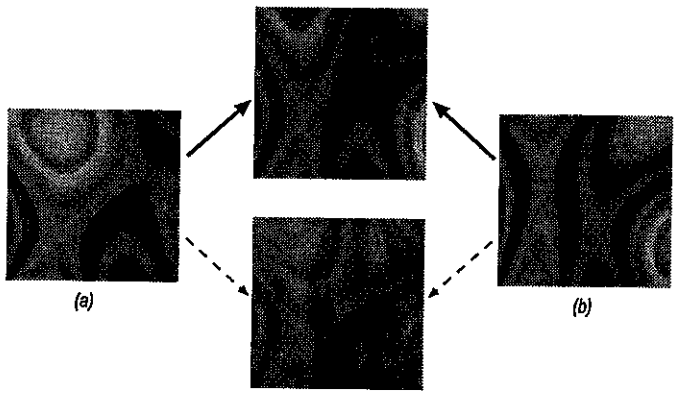


Figure 5.4: Combinations of procedural textures: in color space (lower); in parameter space (upper).

The choice between using a functional, a domain or a range transformation, is dictated by different factors which range from the underlying application to the level of difficulty involved. In our example above, the combination in the parameter space generates much better results, since a new texture function is generated. On the other hand, we should observe that functional interpolation was possible because the two parameter spaces are the same. A transformation between a wood textured object and a marble textured object, for instance, needs to be done on the image space of the two textures.

The idea of interpolating in the parameter space can be taken to extremes where the parameter spaces are very large, and describe most of the object information. Karl Sims (Sims, 1991) describes, among other schemes, how to grow synthetic 3D plants using a "genetic" parameter

space, an application of morphing very close to natural metamorphoses. Interpolations between plants that share a common genetic structure, or a common parameter space, are done to create evolution sequences. Frames from an interpolation in the genetic space are shown in figure 5.5 (from (Sims, 1991)).

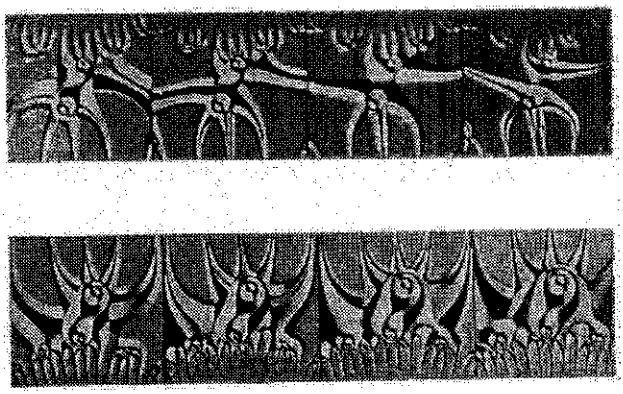


Figure 5.5: Frames from a genetic cross-dissolve.

We will study domain and range transformations of graphical objects with more details in the sections to follow.

5.2.1 Properties of Transformations

In the applications we have to impose some restrictions on the mapping. In general, we need to use mapping of differentiability class C^k, with $k \geq 0$, that is, mappings which are at least continuous. Also in general we require the mapping to be bijective. Therefore the mapping T will always posses an inverse T^{-1}. Also it is also natural to require that both T and T^{-1} have differentiability class C^k, with $k \geq 0$. When $k = 0$ we say that T is a *homeomorphism.* When $k \geq 1$ we say that T is a *diffeomorphism.*

In general, a transformation changes the relation between the points of the space. By studying the effect of a transformation T on points of

the space, we are able to devise three distinct classes of transformation:

- isometry;
- expansion;
- contraction.

An *isometry* T preserves the distance between points, that is

$$||T(X) - T(Y)|| = ||X - Y||.$$

An *expansion* increases the distance between the points of the domain, that is,

$$||T(X) - T(Y)|| \geq C||X - Y||,$$

where, $C > 1$. A *contraction* decreases the distance between points, that is,

$$||T(X) - T(Y)|| \leq C||X - Y||,$$

with $C < 1$. These classes of transformations are illustrated in Figure 5.6. Image (b) shows an isometric transformation (rotation) of (a); image (c) shows a contraction, and image (d) shows an expansion.

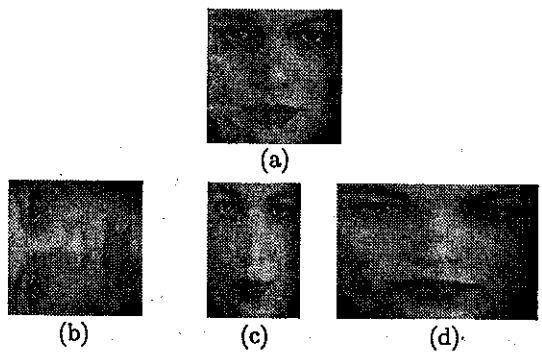


Figure 5.6: Isometry, contraction and expansion transformations.

In general, a transformation changes its behavior from region to region of its domain. This is exemplified by the mapping $f\colon [0,\infty) \longrightarrow \mathbb{R}$, defined by $f(t) = t^2$, which contracts on the interval $[0, 1/2]$, and expands on the interval $[1/2, \infty)$ (see Figure 5.7). We should observe that the inverse mapping $g(x) = \sqrt{x}$, dashed graph in Figure 5.7, performs exactly in the opposite way: it expands on the interval $[0, 1/2]$, and contracts on the interval $[1/2, \infty)$.

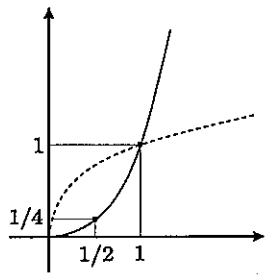


Figure 5.7: Contracting and expanding map.

The projective warp of the image shown in Figure 5.8 is a two-dimensional example of a transformation that contracts in some parts of its domain, and expands on other parts.

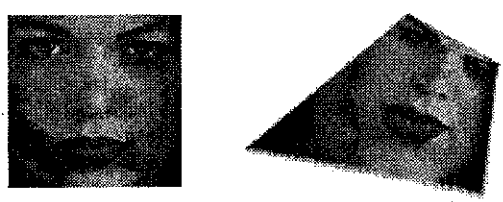


Figure 5.8: Projective warping: contraction and expansion.

The change of characteristics of a mapping over its domain, creates problems when transforming graphical objects, because in the sam-

pling/reconstruction process we must use filters with a kernel of variable size. This has been discussed in the context of texture mapping in (Heckbert, 1986). The reader should also consult (Wolberg, 1990) for a more general discussion in the context of image warping.

5.2.2 Family of Transformations

Graphical object metamorphosis consists of a transition between the shape and the attributes of two graphical objects. From the mathematical point of view, this transition is achieved by a continuum of transformations from one object to the other. This notion of a "continuum of transformations" can be mathematically described using k-parameter family of transformations.

A k-parameter family of transformations of a subset $U \subset \mathbb{R}^n$, is a map $T: U \times \mathbb{R}^k \longrightarrow V$. For each vector $v \in \mathbb{R}^k$, we obtain a transformation $T_v: U \longrightarrow V$ defining the transformed set $T_v(U)$. The space $\mathbb{R}^k$ is called the *parameter space*. It is sometimes convenient to take the parameter space as being a subset of $\mathbb{R}^k$, instead of the whole space.

If $P \in U$ is a point, the set

$$\mathcal{O}(P) = \{T_v(P) \ ; \ v \in \mathbb{R}^k\},$$

is called the *orbit* of the point P generated by the family of transformations. Intuitively, it describes the trajectory of the point as we vary the parameter of the family.

Family of Rotations

A rotation R of an angle θ around the z-axis is defined by the matrix

$$R = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

If the angle θ is allowed to vary on the set of real numbers, we obtain a one-parameter family of rotations $R: \mathbb{R}^3 \times \mathbb{R} \longrightarrow \mathbb{R}^3$. For each $\theta \in \mathbb{R}$, we obtain a rotation R_θ of the family.

It is immediate to verify that the orbit of a point P out of the z-axis is a circle with center at the axis. The action of the family on a line segment parallel to the z-axis, produces a cylinder.

Family of Twists

A *twist* around the z-axis is a transformation obtained by rotating a point around the z-axis by an angle θ, which varies with the z-coordinate. Therefore, points with differing z coordinate will undergo an different amount of rotation. (Barr, 1984).

A twist transformation can be expressed analytically by the matrix

$$R = \begin{pmatrix} \cos f(\theta) & -\sin f(\theta) & 0 \\ \sin f(\theta) & \cos f(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

where the function f is a function of the coordinate z. If the function f is the identity function, $f(z) = z$, the twist transformation reduces to a rotation.

By allowing the angle θ to vary on the set of real numbers, we obtain a 1-parameter family of twists. It is easy to see that under the action of the family of twists, the orbit of a point P out of the z-axis is an helix. Figure 5.9, from (Barr, 1984), shows a sequence of twists of a graphical object under the action of a one-parameter family of twists.

Animation and Family of Transformations

When $k = 1$, we obtain a 1-parameter family, $T: U \times \mathbb{R} \longrightarrow V$. In this case, for each $t \in \mathbb{R}$ we obtain a transformation $T_t: U \longrightarrow V$. By interpreting the parameters as being the time, this family of transformation represents a change of the set U along the time. This is what is usually called an *animation*. Therefore we will interpret a k-parameter family of transformation as a k-parameter animation.

The reader should observe that a k-parameter animation is itself a graphical object. Its shape is the set $U \times \mathbb{R}^k$, and T is the attribute function. This is an important point to remind: a k-parameter family of graphical object is itself a graphical object.

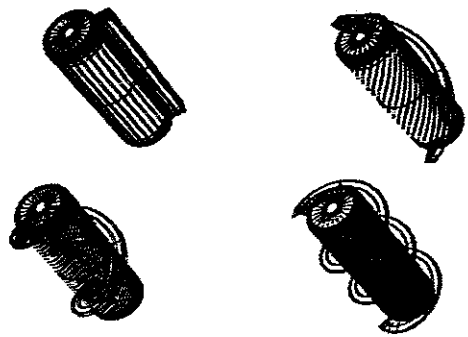
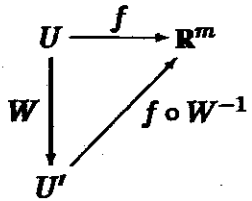


Figure 5.9: Action of the twist family.

An animation is represented by discretizing the parameter space into a finite number of points. In the 1-parameter case, we obtain this discretization by taking a finite sequence $t_1 < t_2 < \cdots < t_n$. In this case we obtain a finite sequence of transformations $T_{t_1}, \ldots, T_{t_n}$. Each T_{t_i} is called a *frame* of the animation. In the general case, the space of parameters in discretized by using a uniform k-dimensional grid.

5.3 Domain Transformations: Warping

Consider a graphical object with shape $U \subset \mathbf{R}^n$, and attribute function $f: U \to \mathbf{R}^m$. A domain transformation is a mapping $W : U \to U' \subset \mathbf{R}^n$ that produces a change of coordinates in the shape of the object. In general, the spatial relationship between points is not maintained by this mapping function. A domain transformation can be seen as a tool for the creation of new deformed graphical objects, as shown in the diagram below



An example of a domain transformation is shown in Figure 5.10. In this example a shear transformation was applied to the domain of an image.

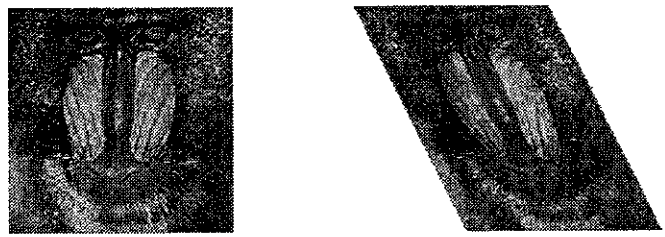


Figure 5.10: Domain transformation of an image.

A more interesting example is shown in Figure 5.11. A tridimensional non-linear warp was applied to the image domain shown in (a), in order to obtain the image in (b). This example illustrates well the fact that $2D$-texture mapping, (Catmull, 1974), is a classical example of image warping.

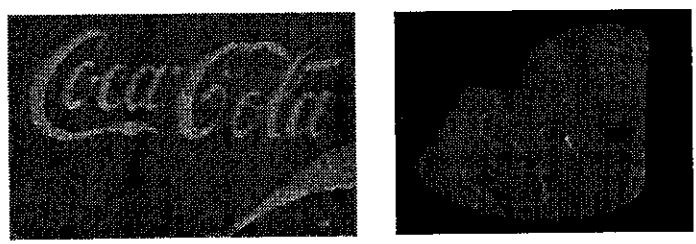


Figure 5.11: Texture mapping and warping.

It is helpful to analyze examples of domain transformations of one dimensional objects. Such one dimensional examples will be extensively used in this chapter, as they are simpler to analyze and understand,

possess the same basic properties, and are easily extended to higher dimensions.

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be a graphical object defined by $f(x) = \sin x$ The shape of the object is $\mathbb{R}$, and f is its attribute function. This object has a graphical representation shown in Figure 5.12(a). Consider the domain transformation defined by

$$W(x) = \sqrt{x}, \qquad x \geq 0,$$

and the resulting transformed object g, shown in (b), is obtained as

$$\left.\begin{array}{l} W^{-1}(x) = x^2 \\ g(x) = f \circ W^{-1}(x) \end{array}\right\} \Rightarrow g(x) = f(x^2) = \sin x^2.$$

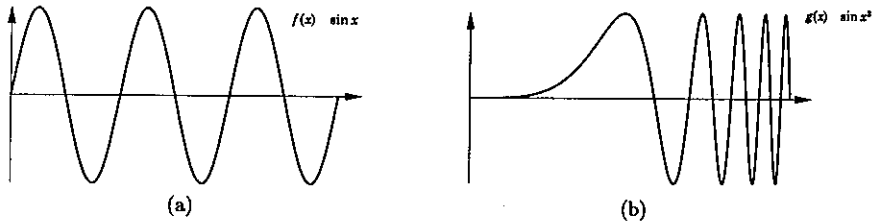


Figure 5.12: Domain transformation: expansion and contraction

Note that the domain is contracted for values of x in $[0, \frac{1}{2})$, and expanded in $(\frac{1}{2}, +\infty)$. This means that

$$|b^2 - a^2| > |b - a|.$$

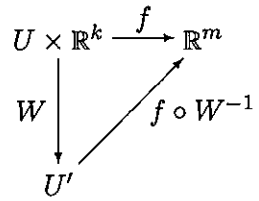
Since

$$|b^2 - a^2| = |b + a|.|b - a|,$$

W is an expansion if $|b + a| > 1$, and a contraction otherwise. For a vicinity of a, if $|(a + \epsilon) + a| > 1$, W is an expansion. If $\epsilon \rightarrow 0$, then for $|a| > \frac{1}{2}$, W is an expansion in the vicinity of a.

5.3.1 Family of Domain Transformations

The deformation of an object, through a transformation of the domain, involves two entities: an input object and a domain transformation. Suppose now that we substitute the transformation of the domain by some k-parameter family of domain transformation $W: U \times \mathbb{R}^k \longrightarrow \mathbb{R}^m$. We would obtain the diagram of transformation



It is interesting to observe that the diagram above allows us different interpretations:

- Parameterized input object: the object $f: U \longrightarrow \mathbb{R}^m$ could be considered as a k-parameter family of objects $f_v: U \longrightarrow \mathbb{R}^m$, $v \in \mathbb{R}^k$;
- Parameterized input transformation: the domain transformation W could be considered as a k-parameter family of transformations $W_v: U \longrightarrow U'$, $v \in \mathbb{R}^k$.
- parameterized input object and transformation: in this case we split the parameter space $R^k = \mathbb{R}^q \oplus \mathbb{R}^p$, and consider a q-parameter family of objects and a p-parameter family of transformations $f_u: U \longrightarrow \mathbb{R}^m$, $u \in \mathbb{R}^q$, $W_v: U \longrightarrow U'$, $v \in \mathbb{R}^p$.

We will discuss each of the above cases with more details.

Parameterized input object

In this case, our interpretation of the diagram results in a k-parameter family of objects which is transformed by the same mapping W of the domain. Intuitively this means that we have an animation of a graphical object, and for each frame of this animation its domain is being transformed by the same transformation.

From the above diagram, the original parametric object is given by $f : U \times \mathbb{R}^k \rightarrow \mathbb{R}^m$. For each vector v in $\mathbb{R}^k$ an object f_v, which is transformed according to

$$g_v : \mathbb{R}^n \rightarrow \mathbb{R}^m \quad g_v = f_v \circ W^{-1}.$$

A one-dimensional example is shown in Figure 5.13, where the parameter t is also one-dimensional. In that case, the input object is a 1-parameter family of lines of the plane $\mathbb{R}^2$ defined by $f_t(x) = x.t$. The transformation W is a translation given by $W(x) = x + a$. From the composition of f_t and W^{-1}, the transformed object g_t is obtained as

$$\left.\begin{array}{l} W^{-1}(x) = x - a \\ g_t(x) = f_t \circ W^{-1}(x) \end{array}\right\} \Rightarrow g_t(x) = f_t(x - a) = t(x - a),$$

and the result, a translated family of lines, is shown in Figure 5.13(c).

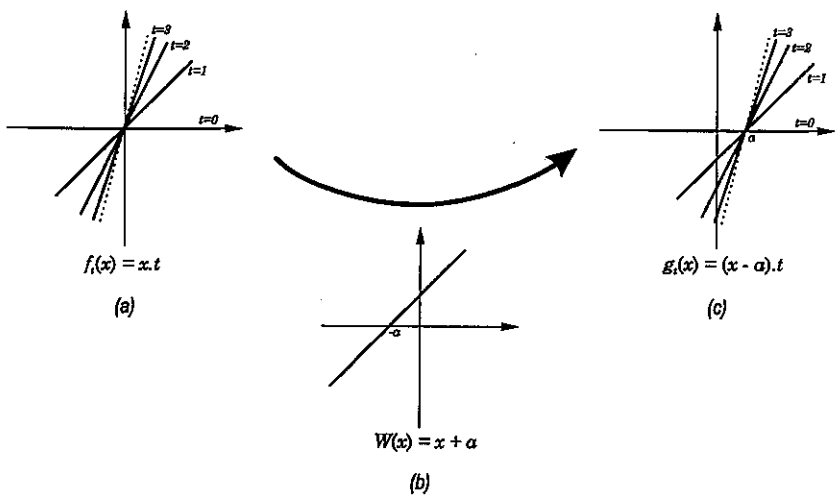


Figure 5.13: Parameterized output object: a translation of a parameterized input.

A two-dimensional example is shown in Figure 5.14 for the case of images. Here, the function used as input to the transformation is an

animation, i.e., it has the form $f : U \times \mathbb{R} \to C$. A warping transformation W is then applied to every frame f_t of f to produce a transformed image g_t. In this example, a 60 degrees rotation was used:

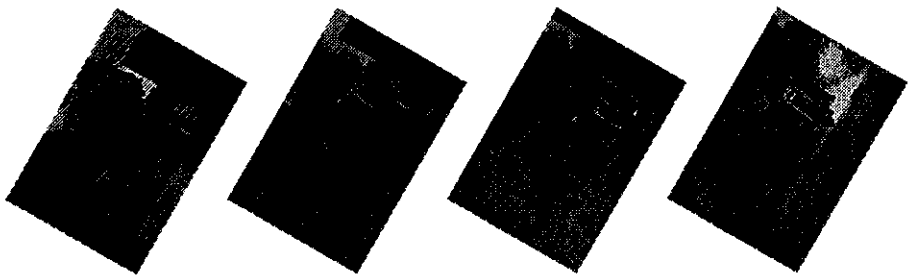


Figure 5 1: Frames of an animation: a fixed rotation of an input animation

$$W(x,y) = \begin{pmatrix} \cos\frac{\pi}{3} & -\sin\frac{\pi}{3} \\ \sin\frac{\pi}{3} & \cos\frac{\pi}{3} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \left(\frac{x - y\sqrt{3}}{2}, \frac{x\sqrt{3} + y}{2} \right).$$

The computation of the transformed animation frames will be done compositing the inverse warping

$$W^{-1}(x,y) = \begin{pmatrix} \cos\frac{\pi}{3} & \sin\frac{\pi}{3} \\ -\sin\frac{\pi}{3} & \cos\frac{\pi}{3} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \left(\frac{x + y\sqrt{3}}{2}, \frac{-x\sqrt{3} + y}{2} \right).$$

and the input frames, resulting in

$$g_t(x,y) = f_t \circ W^{-1}(x,y) = f_t \left(\frac{x + y\sqrt{3}}{2}, \frac{-x\sqrt{3} + y}{2} \right).$$

An interesting application of transformation of k-parameter family of graphical objects is found in the correction of distortions in head-mounted displays (Watson and Hodges, 1995). Here, an inverse warping

is used to compensate the distracting distortions caused by the optical systems in head-mounted displays; the deformation is a function of the lenses, and therefore the same warping is applied to every frame of the computer generated animations.

Parameterized Input Transformations

In this case we have a k-parameter family of transformations $W: U \times \mathbb{R}^k \longrightarrow \mathbb{R}^n$, and one object $f: U \longrightarrow \mathbb{R}^m$. For each vector v in $\mathbb{R}^k$ there is a transformation $W_v : U \rightarrow \mathbb{R}^n$, and, consequently,

$$g_v : U \rightarrow \mathbb{R}^m \quad g_v = f \circ W_v^{-1}.$$

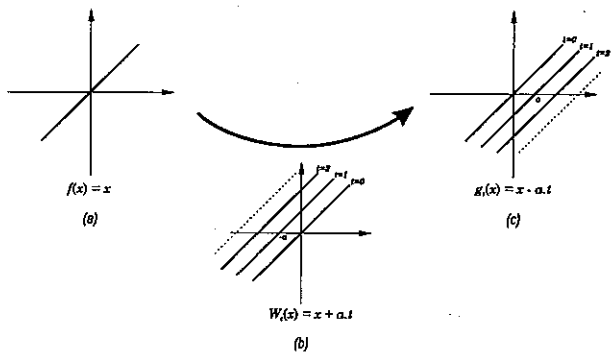


Figure 5.15: Parameterized output object: a variable translation of an input.

This is illustrated in the example of Figure 5.15, for one-dimensional objects. The input object is simply $f(x) = x$, and the 1-parameter family of transformations $W : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is given by $W_t(x) = x + at$, where a is a fixed real number. The transformed object, shown in (c), is then

$$\left.\begin{array}{l} W_t^{-1}(x) = x - at \\ g_t(x) = f \circ W_t^{-1}(x) \end{array}\right\} \Rightarrow g_t(x) = f(x - at) = x - at.$$

Note that the output function in this case is also a k-parameter family, just as in the previous situation.

A two-dimensional example is shown in Figure 5.16, where an animation is created from a static image and a parameterized family of transformations.

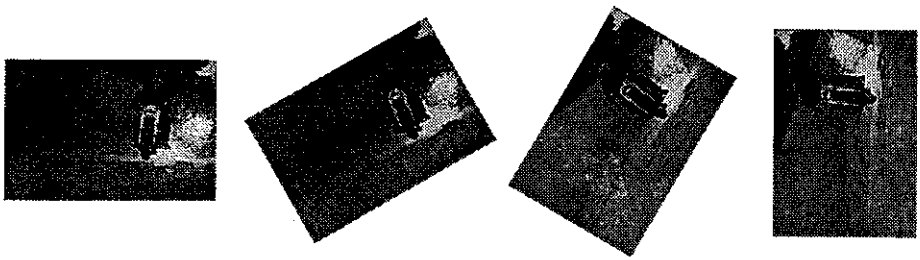


Figure 5.16: Frames of an animation: rotating an image

In this example, the transformation is a 1-parameter family of rotations, where the parameter is the angle of rotation θ:

$$W_\theta(x,y) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{5.1}$$

When each transformation is applied to the domain of an image f, produces an image g which is a version of f rotated by an angle θ, as follows:

$$W_\theta^{-1}(x,y) = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

and therefore

$$g_\theta(x,y) = f(x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta).$$

As observed before, g_θ can be interpreted as an animation if the angle

θ is increased with time. The resulting animation depicts a rotating image, as shown by the frames in Figure 5.16.

Parameterized Input Object and Transformation

In this last case, both the input object and the transformation are parametric, i.e., the input graphical object is given by $f : U \times \mathbb{R}^k \to \mathbb{R}^m$, or $f_u : U \to \mathbb{R}^m$, and the transformation is given by $W : U \times \mathbb{R}^l \to U'$, or, as before, $W_v : U \to U'$. The transformed object is therefore a $(k+l)$-parameter family of transformations,

$$g_{uv} : U \to \mathbb{R}^m \quad g_{uv} = f_u \circ W_v^{-1}.$$

Again, this is easily illustrated using a one-dimensional example, shown in Figure 5.17. The input 1-parameter family of graphical object is a family of rotating lines, defined by $f_u(x) = x.u$ and the one-parameter family of transformations is defined by $W_v(x) = x+a.v$, where a is a constant (this represents a family of translations). Therefore, the output g_{uv} is given by

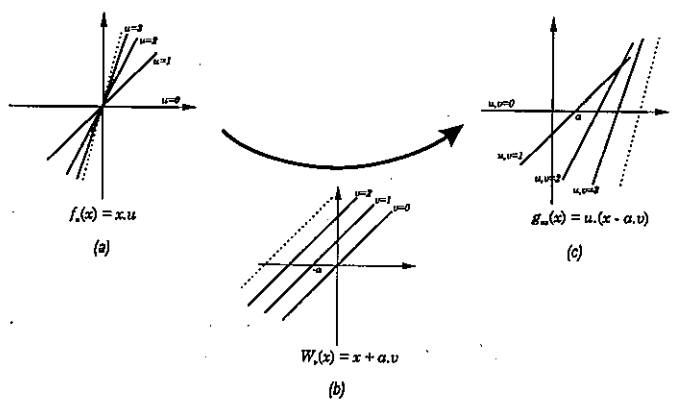


Figure 5.17: A parametric output object: input object and transformation are variable.

$$\left.\begin{array}{l} W_v^{-1}(x) = x - av \\ g_{uv}(x) = f_u \circ W_v^{-1}(x) \end{array}\right\} \Rightarrow g_{uv}(x) = f_u(x - av) = u(x - av),$$

The uv parameter space of the resulting graphical object has dimension 2 (see Figure 5.18). By choosing any curve in this space we are able to obtain an animation. The sequence of frames shown in Figure 5.17(c) was obtained using the curve with equation $u = v$. Note how (c) is a natural combination of the "movements" in (a) and (b).

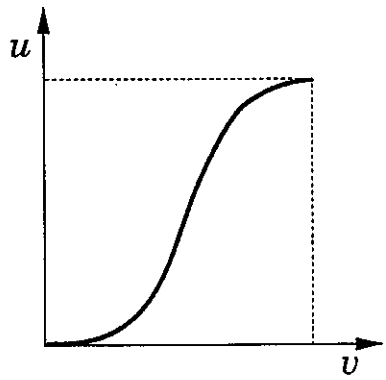


Figure 5.18: Sample path in $2D$ parameter space

The selection of values for u and v allows precise control over the result of the transformation. In general, a suitable path in the parameter space must be chosen to control how the final animation will appear. Paths with at least C^1 continuity, such as the one shown in Figure 5.18, are used to produce smooth animations. Intuitively, the graph indicates the relationship between the "playback speed" of the input animation (v axis) and the rate at which the transformation occurs (u axis).

An experienced animator, is able to use such parameterized families of transformations to produce surprising animations where a subject is transformed while moving. An example of this kind, using images and a simplistic rigid warping—rotation—is shown in Figure 5.19.

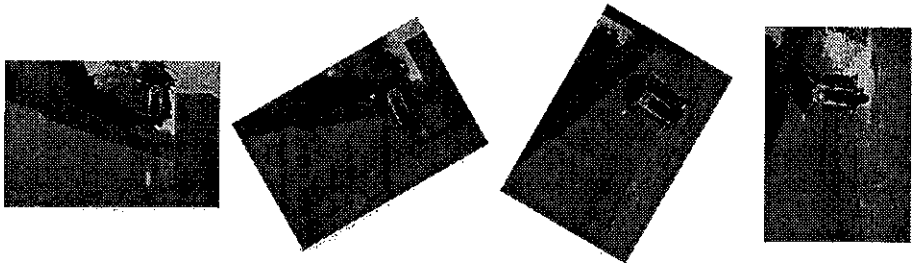
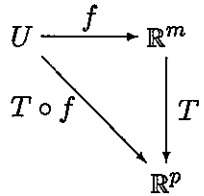


Figure 5.19: Frames of an animation: progressive rotation of another animation

5.4 Range Transformations

In the previous section we studied object transformations by changing the object shape. An object can also have its attributes transformed to create a new object. The attributes are defined by the values of the attribute function, and for this reason these transformations are called *range transformations*. These transformations do not modify the spatial relationship of the points, they alter just the values of the object attributes.

If the object is defined by $f: U \longrightarrow \mathbb{R}^m$, and $T: \mathbb{R}^m \longrightarrow \mathbb{R}^p$ is a transformation, the new graphical object is defined by $T: \circ fU \longrightarrow \mathbb{R}^p$. This is illustrated by the diagram below



Recall that a graphical object is defined as a subset U and a collection of its properties f_i as

$$f : U \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^{r_1} \times \mathbb{R}^{r_2} \times \cdots \times \mathbb{R}^{r_k},\ r_1 + r_2 + \cdots + r_k = m,$$

where $f^j : U \subseteq \mathbb{R}^n \to \mathbb{R}^{r_j}$ are the coordinate functions of the attribute function $f = (f^1, f^2, \ldots, f^k)$.

Therefore, we have

$$T : \mathbb{R}^{r_1} \times \mathbb{R}^{r_2} \times \cdots \times \mathbb{R}^{r_k} \to \mathbb{R}^{r_1} \times \mathbb{R}^{r_2} \times \cdots \times \mathbb{R}^{r_k},$$

or simply $T : \mathbb{R}^m \to \mathbb{R}^p$, where T has coordinates $T = (T^1, T^2, ..., T^k)$.

The reader should notice that in general we might not have uncoupled transformations $T^i : \mathbb{R}^{r_i} \to \mathbb{R}^p$, because the transformation of some of the object attribute values are not, in general, independent of the other object attributes.

The simple one dimensional example in Figure 5.20 summarizes the above discussion, evidencing the amplitude modification character of range transformations. In this example we take the object $f(x) = \sin x$ and a transformation $T(x) = x/2$, the output $g(x)$, shown in (b), is given by

$$g(x) = T \circ f(x)) = \frac{\sin x}{2}.$$

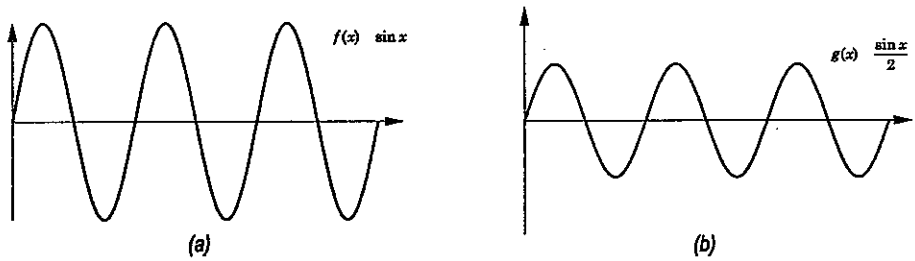


Figure 5.20: Range transformation of an object.

Two-dimensional range transformations are also easy to obtain. Recall that in the simplest case, an image is expressed by a function $f : U \to C$, where the range C is a color space. Therefore, modifying image attributes amounts to making changes in the image color space. An example is the gamma correction transformation used before displaying

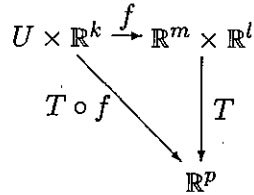
an image. Another widely used example is the luminance mapping which allows us to obtain a monochrome image form a colored one.

5.4.1 Family of Range Transformations

Similarly to domain transformations, we should study the use of range transformation and k-parameter families. The combination of parameterized object and transformations result into three distinct situations, analogous to those discussed for domain transformations:

- Range transformation of k-parameter families of objects;
- k-parameter family of range transformations of one graphical object;
- A combination of both of the above situations.

These three situations result in parametric objects that can be represented by families of transformations, as has been done before for domain transformations. We will not go over the details of each case here, because they are analogous to those related with domain transformation. The three cases are summarized by the diagram below



In the first case we have $l = 0$, in the second case we have $k = 0$, and in the last case, both k and l assume non-zero values.

An illustrative example of parameterized range transformation is given by the operation of cross dissolve between two images. In this case we have two images $f, g: U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^3$. We define a 1-parameter family of transformation $T: U \times [0,1] \longrightarrow \mathbb{R}^3$, by

$$T(u_1, u_2, t) = T_t(u_1, u_2) = (1-t)f(u_1, u_2) + tg(u_1, u_2).$$

Figure 5.21: Cross dissolve between two images.

T_t is a one-parameter family of images such that when the parameter t varies from 0 to 1, the images in the family vary from f to g. In Figure 5.21 we illustrate the operation of cross dissolving between the face of a woman (a), and a cheetah (b). Figure (c) shows an intermediate frame, obtained for the value $t = 1/2$ of the parameter.

Chapter 6

Morphing of Graphical Objects

In the previous chapter we studied the transformations of graphical objects. In this chapter we will use the concepts introduced before to define in a precise way the concept of metamorphosis between two graphical objects.

6.1 Morphing: Shape and Attribute Combination

Suppose we have two graphical objects $\mathcal{O}_1$ and $\mathcal{O}_2$. Intuitively, a metamorphosis operation between these two objects is attainable by devising a continuous transition from object $\mathcal{O}_1$ to object $\mathcal{O}_2$. In order to attain this, we must use a k-parameter family of transformations to obtain both a shape transition, and a transition between the attributes of the two graphical objects.

Denote by Ω some space of objects $\{f: U \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m\}$. If $\mathcal{O}_1$ and $\mathcal{O}_2$ are objects in Ω, a *morphing* or *metamorphosis* between $\mathcal{O}_1$ and $\mathcal{O}_2$ is a k-parameter continuous family of transformation

$$\varphi: \mathcal{O}_1 \times \mathbb{R}^k \longrightarrow \mathcal{O}_2.$$

Intuitively, for each parameter $v \in \mathbb{R}^k$ from the parameter space, we obtain a new graphical object $\varphi_v(\mathcal{O}_1)$, and this family φ_v performs the

transition from one object to the other, as v varies on the parametric space.

The reader should notice that the parameter space has dimension k, this means that we have k degrees of freedom in choosing an animation of the morphing transformation between $\mathcal{O}_1$ and $\mathcal{O}_2$. Mathematically this is done by defining a curve $c\colon [0,1] \longrightarrow \mathbb{R}^k$ on the parameter space. In this case, we obtain a 1-parameter family

$$\varphi \circ c\colon \mathcal{O}_1 \longrightarrow \mathcal{O}_2,$$

such that for exact $t \in [0,1]$, $\varphi(c(t)).\mathcal{O}_1$ is a graphical object of the morphing transformation. The curve c is chosen so that when $t = 0$ we obtain $c(0) = \mathcal{O}_1$, and when $t = 1$ we obtain $c(1) = \mathcal{O}_2$.

In general the k-parameter family of transformation that performs the metamorphosis between object $\mathcal{O}_1$ and $\mathcal{O}_2$ splits into two parameterized families of domain and range transformations. This is illustrated by the diagram below

$$\begin{array}{ccc} U \times \mathbb{R}^p & \xrightarrow{f} & C \times \mathbb{R}^q \\ \Big\downarrow W & & \Big\downarrow T \\ U & \xrightarrow[g]{} & C \end{array}$$

the morphing transformation is given by $g = T \circ f \circ W^{-1}$. If W is the identity transformation, g is simply a range transformation of f; on the other hand, if T is the identity transformation, g turns out to be a domain transformation of f.

In general, the morphing transformation g is obtained in a two step process:an intermediate object h is created in a first pass from a domain transformation of f performs a warp of the object shape, and then a range transformation T is applied to h in order to change the attributes, and obtain g. From the above diagram the transformations involved into this two step process are given by: $h = f \circ W^{-1}$, and $g = T \circ h$.

Figure 6.1 illustrates a morphing transformation between two static images, with all the intermediate steps. Horizontal arrows depict warping transformations, and the vertical additions represent attribute

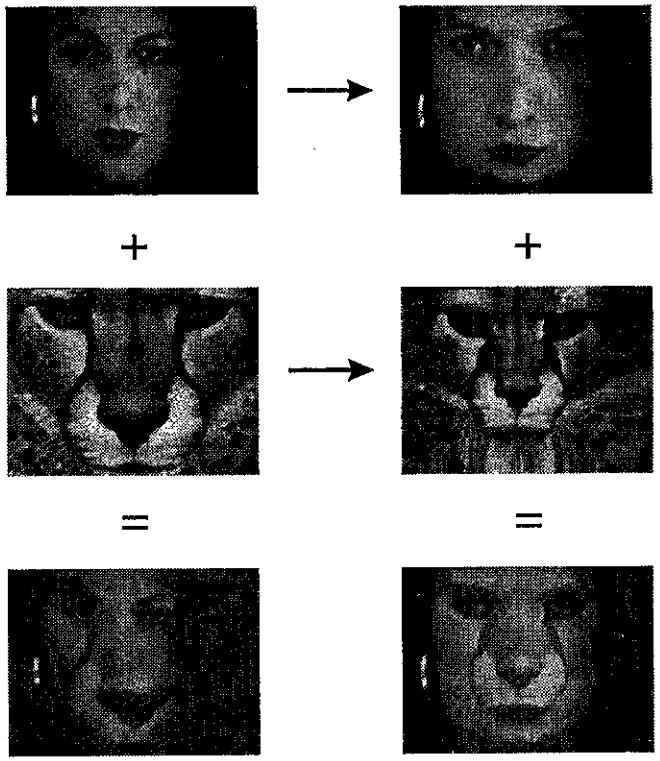


Figure 6.1: A complete image morphing and its steps.

change by cross dissolving. The inputs are the image with the woman face, in the upper-left part, and the image of the cheetah, just below it.

These two images are warped so that their main features, such as eyes, mouth and nose are aligned, as shown in the upper right part of the figure. The cross-dissolve performed between the two original images and the two warped images are shown in the bottom part of the figure. The morphing, on the right, is much more appealing than a pure cross-dissolve, and effectively represents a new, perceptually feasible, object created as a combination of the two inputs.

Figure 6.2, from (Gomes and Velho, 1995b), shows 8 frames of the morphing sequence between the two images discussed in the previous paragraph.

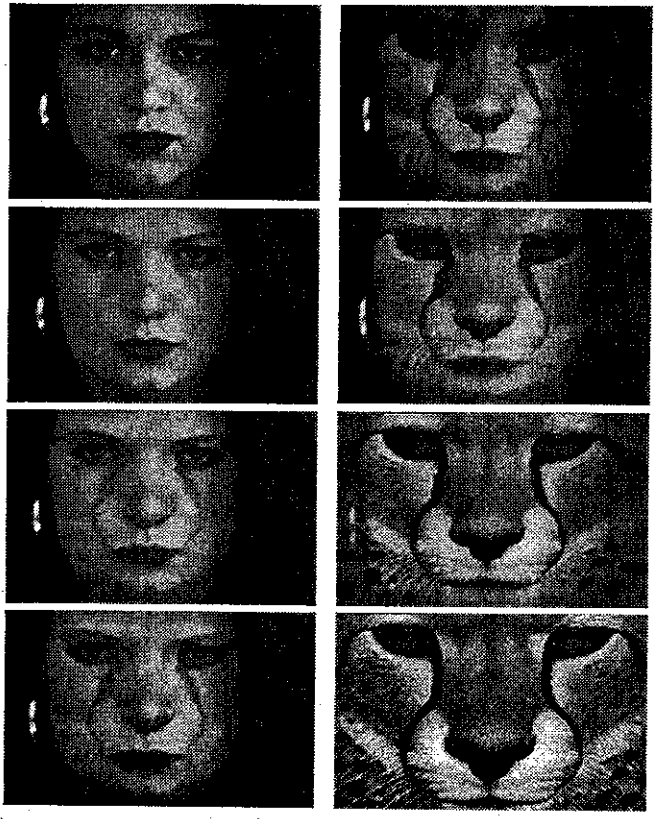


Figure 6.2: Frames from a morphing sequence.

We should point out that the above description holds for the morphing between arbitrary graphical objects. In fact since two images always have the same shape, there is one important point in the morphing process which is hindered by the above example. In fact, the geometry alignment phase in general can be obtained only up to a certain approximation of the geometry. Therefore, we must perform a blending

operation in order to get a new object which represent a perfect alignment of the geometry of each object.

This blending step is well illustrated in Figure 6.3, from (Lerios, Garfinkle and Levoy, 1995), for the case of a morphing between two objects in $3D$-space. It shows the warping of each object, to obtain an approximate allignment of the geometry, and the blending of these two warps.

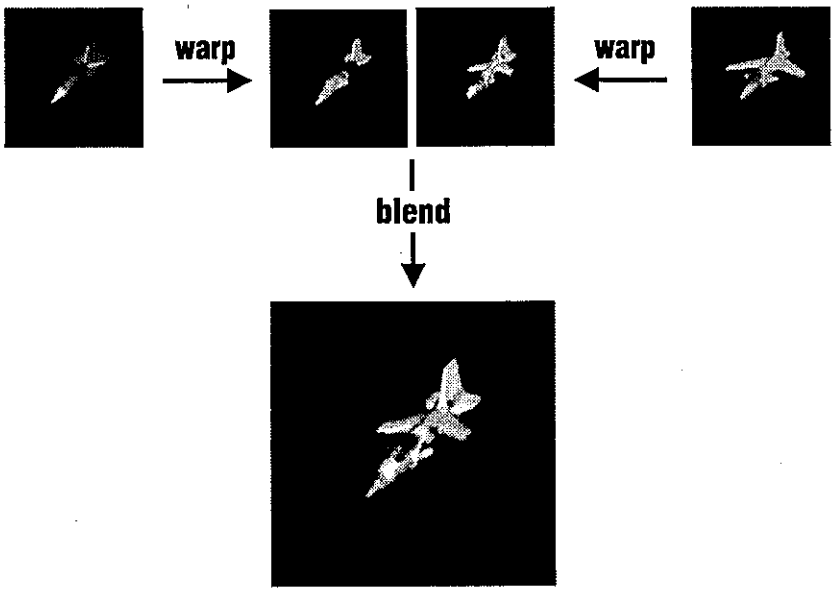


Figure 6.3: Warp and blending in 3D space.

Blending techniques will be discussed in Chapter 10.

6.1.1 Parameters and Morphing Transformation

From the diagram on page 106, a morphing transformation between two graphical objects depends on $p + q$ parameters. Therefore, metamorphosis is a $(p + q)$-parameter animation. Several combinations of these

parameters are possible, also, the reader should notice that the objects themselves could be parameterized. Several particular situations are possible in the applications. Therefore, we are left with several degrees of freedom in order to choose a morphing animation.

In the example shown in Figure 6.2, we have two non-parameterized objects (the woman face and the cheetah), and we define a morphing transformation between them. This morphing transformation performs both domain and range transformations. The domain transformation aligns come features of the image shape, and the range transformation performs grayscale interpolation.

An important example occurs when we have a morphing transformation between two parameterized graphical objects. As we well know, each of these graphical objects can be interpreted as an animation, and for this reason, a morphing transformation between them is called *animation morphing.*

In this case, the parameter space has at least dimension 3:

- one parameter for the transformation;
- one parameter for changing attributes;
- one parameter for the animation of each graphical object.

By carefully manipulating the values of the parameters on the parameter space, the user can obtain a "parameter path" which performs nice transitions between the two animations.

In the example of a metamorphosis between the woman face and the cheetah, the domain warping was attained by using a large number of parameters to specify the warping family. Even the color attribute interpolation in this example was note done by a simple 1-parameter, linear, cross dissolve. Some experimentation showed us that it was much more effective to blend the face attributes faster than the hair attributes. Therefore, a non-linear cross-dissolve was used instead.

In a non-linear cross-dissolve technique the color change depends on the position of the pixel. More precisely, we have the one-parameter family

$$T(u_1, u_2, t) = h_1(u_1, u_2, t)f(u_1, u_2) + h_2(u_1, u_2, t)g(u_1, u_2),$$

where

$$h_1(u_1, u_2) + h_2(u_1, u_2) = 1$$

This reduces to the linear cross dissolve for $h_1(u_1, u_2, t) = 1 - t$, and $h_2(u_1, u_2, t) = t$.

By observing the morphing sequence shown in image 6.2 the reader can notice the change of grayscale values in the face and in the hair: it takes much longer for the hair attributes to be surpassed by those from the cheetah. The possibility of controlling the progress of warping and attribute transformations independently is effectively used in practice to help the creation of more convincing effects.

Figure 6.4 depicts a path on the parameter space in the case of dimension 3. The figure also shows the path projection on the *ct*-plane, and on the *tw*-plane.

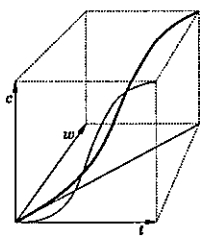


Figure 6.4: A particular choice of parameters for animation morphing

We can observe in this example that the warp evolves linearly with the input animation (projection on the *wt*-plane), but the attribute transformation does not (projection on the *ct*-plane). It starts and ends slowly and accelerates at the middle, so that no attribute combination is noticeable at the beginning or the ending of the animation and it takes place quickly (compared to warping) during the middle of the animation. This particular progression is normally used with few modifications as a practical template that produces perceptually better results.

We should point out that there is a trade off between the user control

over the morphing transformation, and the number of parameters: the smaller the number of parameters the easier it will be to control the animation of the morphing sequence. As the number of parameters increase we have to work harder to control the animation, but we have greater flexibility in choosing the best result.

In general, specifying families of transformation requires a huge number of parameters. This problem will be discussed in next chapter.

6.2 Some Examples

The same techniques and remarks discussed above apply for morphing between objects other than images. Image morphing is an example of a two-dimensional morphing between two-dimensional graphical objects. We show some other morphing examples, of different dimensionality, below:

Drawing Morphing

Figure 6.5 shows a morphing sequence between two drawings. This is an example of a two-dimensional morphing between one-dimensional graphical objects.

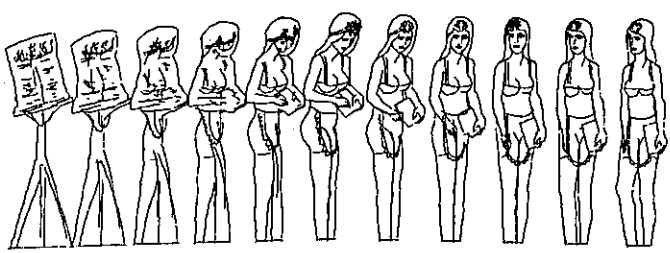


Figure 6.5: Drawing morphing.

Surface Morphing

Figure 6.6, from (Kent, Carlson and Parent, 1992), illustrates a morphing between polyhedral surface. This is an example of three-dimensional morphing between two-dimensional graphical objects.

Figure 6.6: Polyhedral surface morphing.

Volume Morphing

Figure 6.7, from (Lerios, Garfinkle and Levoy, 1995), shows some frames of a metamorphosis sequence between an orangutan head, and a human head. These two graphical objects are represented using volume arrays. This is an example of a three-dimensional morphing between three-dimensional objects.

Audio Morphing

The examples above don't include a case of a one-dimensional morphing between two one-dimensional graphical objects. An example of this nature could be obtained using audio signals.

A good example of audio morphing can be found in the movie "Farinelli, il castrato", a France/Belgian/Italian production of 1994. This movie tells the story of the famous opera singer Farinelli, from

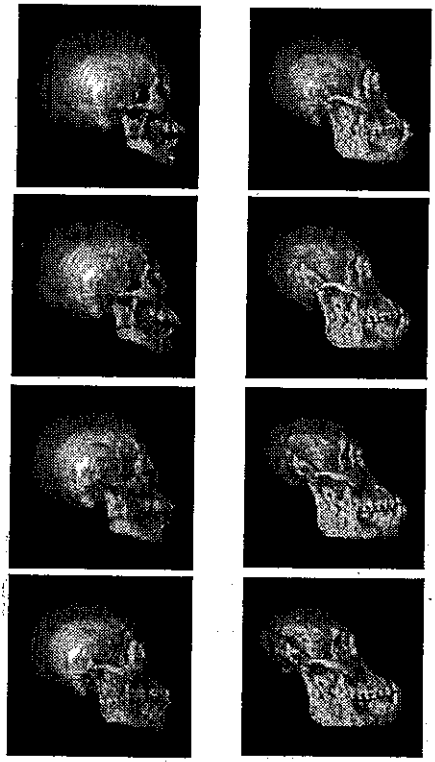


Figure 6.7: Volumetric morphing.

XVIII century. The unique voice of the character Farinelli was obtained using a very accurate morphing between the voice of the polish soprano Ewa Mallas Godlewska, and the american counter-tenor Derek Lee Ragin. This process took two years to accomplish.

Chapter 7

Specification of Transformations

The previous chapter discussed several forms of transformations of graphical objects in the mathematical universe. To that end, mathematical abstractions for both graphical objects and their transformations were introduced. The computational use of those abstractions requires a suitable representation model, which involves some form of discretization of the elements of the mathematical universe. Representation schemes for graphical objects were discussed in chapter 3; the problem of representing and implementing transformations of graphical objects will be the subject of this chapter and the next.

7.1 Specification

The manipulation of transformations in the computer requires the description of a transformation using some finite, representation scheme. Besides the representation problem, we are faced with the specification of the transformation by the user. Some examples will help to clarify the problem.

Linear maps on the space $\mathbb{R}^n$ are completely characterized by knowing its values at n linearly independent points; affine mappings need $n+1$ points. Therefore, in order to specify the transformation from the

rectangle to the parallelogram in Figure 7.1, the user only needs to specify the image of the points A, B and C. This is equivalent to specifying the image of the oriented line segments $\vec{BA}$ and $\vec{BC}$.

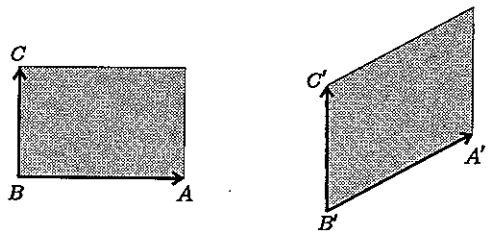


Figure 7.1: Affine warping specification.

Projective mappings of the space, need $n+2$ points to be completely characterized. Therefore, to obtain the projective warping shown in Figure 7.2 the user needs only to specify the correspondence between the vertices A, B, C, D in the original image, and the corresponding vertices A', B', C' and D', on the warped image.

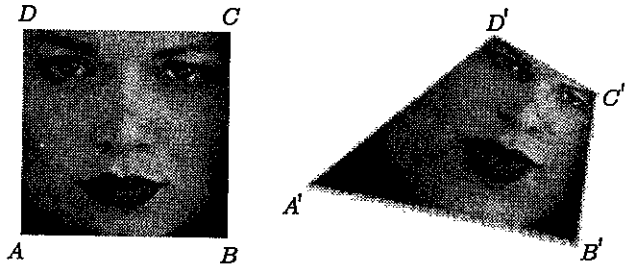


Figure 7.2: Projective warp with two vanishing points.

From this finite specification of the affine and projective mappings in the examples above, the computer should be able to reconstruct the pro-

jective mapping and apply it to any point of the image domain. This is not a difficult task, since these mappings possess an exact representation by matrix using homogeneous coordinates.

The above examples, although simple, exemplify our goal when dealing with transformation on the computer:

- the user should specify a finite, and hopefully small, number of parameters;
- the program should provide a good user interface;
- the computer should provide a robust and simple representation from the user specification.

This is illustrated by the simple diagram in Figure 7.3.

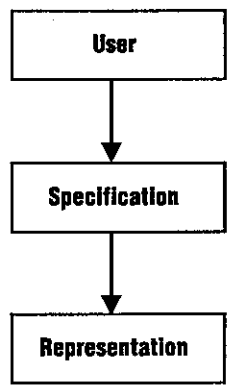


Figure 7.3: User interface, specification and representation.

This is our goal when dealing with transformations on the computer. However, we should observe that non-linear transformations in general are very difficult to be specified and represented by using a finite number of parameters.

Specification and User Interface

In the metamorphosis problem, we need to specify families of transformations, and in general, the parameter space has an infinite number of parameters. Therefore, we need a very flexible way to specify these parameters.

Take a simple warping transformation, e.g., a rotation around the origin by an arbitrary angle. One way of representing this transformation, consists of storing a single parameter that represents the amount of rotation to be performed, and an equation that describes the calculation to be done with that number. Another, more pictorial, representation is to store a vector that indicates the rotation. This form of representation may be easier to be used as an interface, because it is possible to manipulate it directly, and it provides a natural graphical interface for the user.

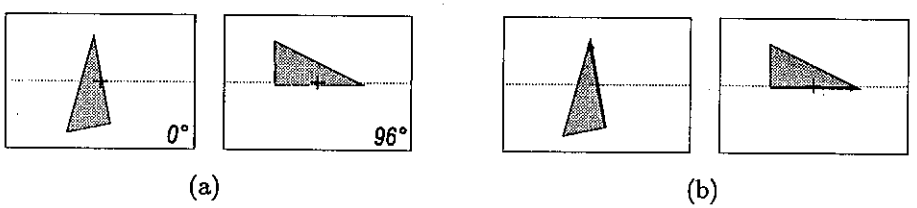


Figure 7.4: Rotation representation: (a) by angle; (b) by vector.

To achieve certain specific results, as in aligning two graphical objects through rotation, an angle representation would require some guess work to accomplish the desired result, while a vector representation allows an interactive specification of the desired alignment.

The reader should observe that there is a close relationship between the specification and the representation of a transformation. As we mentioned above, in general, projective mappings, and this includes linear and affine mappings, are represented by matrices, independent of the specification method used on the interface.

In the rotation example mentioned above, the conversion between an-

gles and vectors is trivial, which makes the uncoupling of specification and representation easy. In general, the two concepts are tightly connected, however, and exact conversions between forms of representation are rare. This leads to the interchangeable use of the terms transformation specification and transformation representation.

7.2 Specification × Computation

After the user specification, the transformation is represented on the computer, and the computation of transformation takes place. This pipeline is easier to be explained using a client-server model.

In all client-server relationships, some form of specification of the desired results must be given by the client, so that the server is able to accomplish what has been specified. As the server works based only on the specification, and not on the desires, the looser the specification is, the widest range of conforming results can be achieved. In this way, there are two distinguished phases in such relationships:

- **Specification**, an expression of the *desired* results;
- **Computation**, a way to obtain the *specified* goals.

These two phases, shown schematically in figure 7.5, are related, although not intrinsically. The importance of the distinction between these phases in morphing is evidenced in the detection of new combinations of existing techniques.

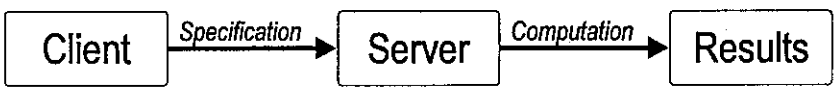


Figure 7.5: Specification and computation.

Although in the literature the form of specification was generally associated with the computation technique being used, it is possible to

dissociate the two, in order to obtain different forms of specification with the same underlying technique used to compute the transformation (Costa, Darsa and Gomes, 1992), as exemplified in section 9.4.

Morphing in practice requires a good level of user specification to approach the desired results. The specification of a generic warping can be done in several ways and the choice of a particular method will influence both the user interface and the computation.

The example of the projective warping described in the beginning of this chapter is not really the common rule of the game. Usually, we need to manipulate a huge number of parameters to specify some warping in order to achieve certain prescribed goals. These goals could be of a perceptual nature, or could be related with some other criteria determined by the underlying application.

7.3 Defining a Specification

All forms of specification are based on the idea of defining the transformation by manipulating only a finite, and hopefully small, number of parameters. From these specification, mathematical techniques should be used to compute the transformation for any point on the warping domain.

In order to specify a transformation $f: U \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$, we must define its domain U, and for each point $p \in U$, we must find the rule to obtain the transformed point $p' = f(p)$. When the set U has only a finite, and small, number of points, $p_1, \ldots, p_n$, (n small), it is very easy to specify $f(p_i)$ for each i. On the other hand, if U has a huge number of points, possibly infinitely many, specifying the transformation at each point is, in general, an impossible task, unless we are restricted to some specific class of transformation (e.g. projective mappings).

Therefore, we should devise techniques that allow us to specify the transformation only at a finite number of elements. There exists different methods for finding the best specification for each problem, and this constitutes a whole chapter that pervades different areas of mathematics. In fact, most of the mathematical problems consist in finding some function with certain prescribed properties.

Here we are concerned with graphical object metamorphosis. The most suited function specification for our problem, consists in using function extension: we specify the transformation only at some finite set of elements of the domain. From these elements, the function is reconstructed (extended) to the whole domain.

In order to give a precise definition of an specification, we need to use the concept of geometric data set of a graphical object (see the definition of a graphical object in chapter 2).

Consider two graphical objects $\mathcal{O}_1$ and $\mathcal{O}_2$, with geometric data sets $U_1 = U_1^1 \cup \cdots U_n^1$, and $U_2 = U_1^2 \cup \cdots U_m^2$. A specification of a transformation between $\mathcal{O}_1$ and $\mathcal{O}_2$ consists of a set of ordered pairs

$$\mathcal{P} = \{(s_i, d_i) \ ; \ s_i \subset U_i^1, \ d_i \subset U_i^2\}, \tag{7.1}$$

and a family of transformations

$$W^i: s_i \subset U_i^1 \longrightarrow d_i \subset U_i^2. \tag{7.2}$$

This family of transformations $\{W^i\}$ defines a transformation

$$\overline{W}: \bigcup s_i \subset U^1 \longrightarrow \cup d_i \subset U^2.$$

The required transformation $W: \mathcal{O}_1 \longrightarrow \mathcal{O}_2$ is obtained by extending $\overline{W}$ to the whole object shape U^1. In other words, the transformation W is defined so that its restriction to each set s_i, $W|s_i$, coincides with the transformation $W^i: s_i \longrightarrow d_i$.

We should observe that the elements s_i of the geometric data set of the object $\mathcal{O}_1$, can have any dimension ranging from 0 (discrete set) to the dimension of U^1. In a similar fashion, the dimension of the elements d_i range from 0 to the dimension of the graphical object $\mathcal{O}_2$. Some examples of elements of 0, 1 and 2 dimensions will be seen in the next sections. We should observe that nothing prevents the use of different dimension subsets at the same time. One such example is the use of points, polylines and curves together to establish the correspondences between two images in (Litwinowicz and Williams, 1994).

An interesting point to note is that such forms of specification are equivalent to sampling the transformation function, and then representing the transformation by this set of samples. During the computation

phase, the warping is extended to the whole domain in order to be applied to the graphical object. This is an instance of the problem of structured and unstructured sparse data interpolation, which has many well known solutions.

7.3.1 Classification of Specifications

With the objective of devising a classification for the existing specification of warping techniques, it is convenient to introduce the concept of the *source set* S and *target set* D as follows:

$$S = \{s_i \subseteq U | \text{ there exists } d_i \subseteq U' \text{ such that } (s_i, d_i) \in \mathcal{P}\}$$

and

$$D = \{d_i \subseteq U' | \text{ there exists } s_i \subseteq U \text{ such that } (s_i, d_i) \in \mathcal{P}\}$$

Different configurations of S and D result in different specifications, and therefore different transformations. Based on the above definition, we can devise three specification techniques that have been largely used in the literature for obtaining object metamorphosis:

- specification by partition;
- specification by features;
- automatic specification.

We will study each of these methods in the following sections.

7.3.2 Specification by Partition

In a specification by partition, we obtain a decomposition of the shape of both objects that constitutes a partition. That is, the elements U_i^j, $j = 1, 2$, $i = 1, 2, \ldots, n$, satisfy

- $\bigcup_{i=1}^{n} U_i^j = U \quad j = 1, 2;$
- $U_i^j \bigcap U_k^j = \varnothing$, for $i \neq k$, $\quad j = 1, 2.$

More precisely,

- $(s_0, d_0), (s_1, d_1) \in P \Rightarrow s_0 \cap s_1 = \varnothing$ and $d_0 \cap d_1 = \varnothing$
- $U_s = \bigcup_{s_i \in S} s_i, U_d = \bigcup_{d_i \in D} d_i \Rightarrow U_s = U$ and $U_d = U'$.

This is illustrated in Figure 7.6. The object shape and its underlying space are decomposed, and the transformation must be specified for each element of the decomposition.

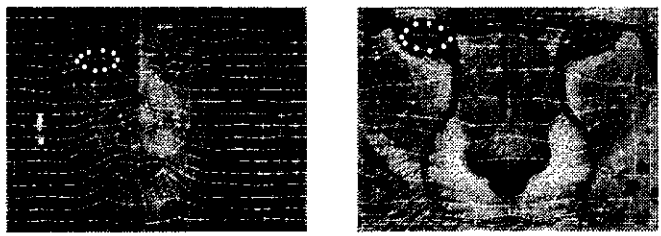


Figure 7.6: Morphing specification by partition.

Specification by partition has an straight dependence on the structure of the domain decomposition: well structured decompositions are easy to work with, while arbitrary decompositions are more difficult to represent and to maintain.

In general, we use well behaved partitions such as triangulation, or cellular decompositions. In general, hese partitions define an unstructured grid over the object shape, and for this reason they are generically called *mesh*. For this reason, specification by partition could also be called *specification by meshes*. In the mesh based approach, two meshes, with equivalent combinatorial topologies, are created as part of the geometric data set of the graphical object. Each mesh defines a partition of the object domain.

Regular meshes, besides greatly simplifying data manipulation in the implementation, also facilitate user comprehension of the process. In any case, the user is responsible for specifying the transformation between two corresponding meshes to attain the desired results.

There are some particular cases of meshes that are specially relevant. Polygonal or polyhedral meshes, for instance, are a natural choice that can give good results when coupled with robust reconstruction techniques in the computation phase. However, the direct specification of a transformation through manipulation of meshes is a laborious task in $2D$, and it is almost impracticable to be used in $3D$.

The use of meshes of spline curves (Smythe, 1990) or surfaces suggests a naturally smooth transformation, and was used in image morphing applications for the first time for the special effects of the movie "Willow" in 1988 (see (Wolberg, 1990)). Although spline meshes permit an extremely efficient computation for sampled implicit objects (discussed in the next chapter), this kind of specification is restrictive sometimes. As noted before, positioning the points of the mesh over the interesting features is a difficult task, and the regularity of the mesh does not necessarily match the "natural" structure of the underlying object. We should point out that because of continuity properties of the warping, the values of the transformation in a mesh greatly influences the results of neighboring meshes.

By definition, specification by partitions try specify the transformation for the entire object domain. This specification enables the user to define more precisely what takes place in all parts of the domain. Although, because of continuity restrictions changes in one set of the partition induces changes in neighboring sets. This forces the user to work on partition domains which do not directly influence the desired results.

7.3.3 Feature Based Specification

In a specification by features, the source and target sets, S and D, do not define partitions. Note that a partition specification we try to specify the transformation for the entire domain, while a feature specification leaves the mapping function in $(U - U_s)$ undefined. In general, some useful conditions are required from the specification to yield reasonable warps. Adjacency relationships in a partition specification, for instance, must be the same in S and D for the warping transformation to be continuous. Also, feature specifications usually satisfy a non-overlapping condition between each feature, so as to avoid problems of multiple definition at feature intersections when defining the warping W. Additional

constraints to the specification of the features is, in general, imposed when defining some warping techniques.

In feature based specification only distinguished features and their transformations are specified by the user. The warp will be computed in such a way to map each feature of the graphical object to the corresponding transformed state. This corresponds to reconstructing the warping transformation from its specification on just a few selected feature positions.

Zero-dimensional Features

An important case of feature based warping is the point based specification. In this case, each feature is described by a point which belongs to the geometric data set of the graphical object. Point based morphing specification is largely used in commercial image morphing.

In some particular cases, it is possible to reconstruct the warping exactly, from a point based specification. This is the case of the affine and projective warpings, as shown in the beginning of this chapter.

One-dimensional Features

This is an example where the features are specified by curves defined on the shape of the graphical object. Therefore, the specification uses 1-dimensional features.

Figure 7.7 illustrates a warping specification by one-dimensional features for the case of images. Several oriented line segments are added to the source image of the woman. These segments become part of the geometric data set of the woman image. They are positioned in such a way to mark disntingueshed features of the face. These segment features are conveniently mapped onto similar features of the target image, in order to specify the transformation.

An implementation of an image morphing system where the features are described by oriented line segments, was done at Pacific Data Images (Beier and Neely, 1992). The system was used to create the classical morphing sequence in Michael Jackson's video clip "Black or White".

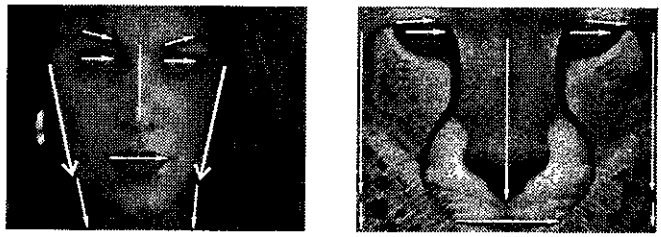


Figure 7.7: Feature specification.

Features of Higher Dimension

The example of one dimensional feature shown above illustrates the case when we have 1-dimensional features (segments), 2-dimensional graphical objects (images) in 2-dimensional space (plane). This specification technique described above of one-dimensional features has been extended recently by (Lerios, Garfinkle and Levoy, 1995), to specify morphings of objects in 3-dimensional space. This extension uses features of different dimensions: points, segments, rectangles or a $3D$-box. Figure 7.8, from (Lerios, Garfinkle and Levoy, 1995), illustrates the use of different features to specify a morphing between a dart and a plane. Figure (a) shows the dart and its features; figure (b) shows the plane and the corresponding features.

Another interesting use of one-dimensional features has been proposed by George Wolberg (Wolberg, 1989). He addresses the problem of warping an arbitrary shaped subset of an image which is specified just by two corresponding outlines. A thinning operation is used to determine the mapping for the interior region of the outlines.

Physics Based Features

This is an area of warping techniques where a lot of activities is going on. In this case, the features have a physical meaning, such as point masses, forces, force fields, velocity and so on.

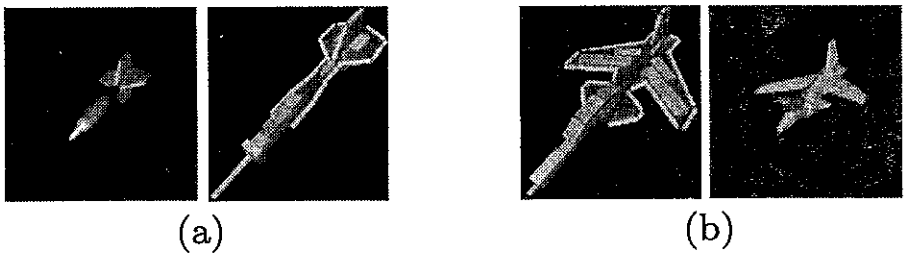


Figure 7.8: Segment specification in 3D space.

By specifying these features the warping computation is done by the convenient interpretation of physical meaning of the features in a convenient physical system. The dynamical or kinematic equations of the physical system are solved in order to compute the desired warping.

In general, the obvious advantage of feature specification is that the user only needs to specify the transformation at relevant features. In practice, however, users are frequently forced to add secondary features in order to obtain a better control over the warping reconstruction process.

7.3.4 Automatic Specification

A warping is defined by a mapping function that establishes a spatial correspondence between all points from the shape of the input object, to the points in the shape of the output object. The warping specification techniques described above enable the user to specify the transformation only at some finite number of elements belonging to the geometric data set of the graphical object.

When the source and the target sets S and D are empty, from the user point of view, we have an *automatic specification*, also called *specification by parameters*. These techniques are not based in a specification by the user of spatial relationships between domains, these specifications are detected by some automatic algorithm and the warping sequence is

generated. Ideally, just the two input graphical objects should be given, and the computation phase would be responsible for determining the suitable mapping function.

In Figure 7.9, there should be no user interference to obtain a warping family that transforms an arbitrary triangle into an arbitrary quadrilateral.

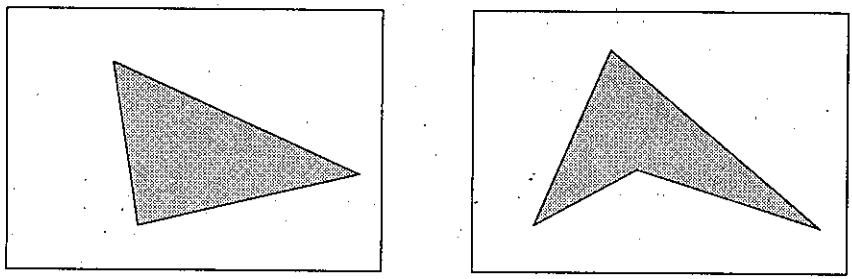


Figure 7.9: Automatic specification.

For automatic and semi-automatic techniques, the correspondence of characteristics is not present in the specification, but is automatically determined by the computation phase instead. It is still necessary, though, to specify other parameters that help to control the progress of the transformation. Ideally, users should be required to provide just the minimum data that is sufficient to obtain the desired results. This leads us toward the development of semi-automatic warping techniques.

One approach is to use some pre-processing step to find the relevant features of the objects. The algorithm establishes a correspondence between marked features which can finally be used to compute the transformation. Another possibility is to determine the appropriate mapping directly from the analysis of the objects, without explicitly manipulating features.

Partial automation can also be obtained by some form of feature digitalization, where the specification is captured from the real world. Rotoscoping of actors with their desired features physically marked (painted

on them, for example) is an effective alternative that can reduce specification time dramatically. Figure 7.10 shows a sequence and the controlling features, which were rotoscoped from an actor's face (from (Litwinowicz and Williams, 1994)).

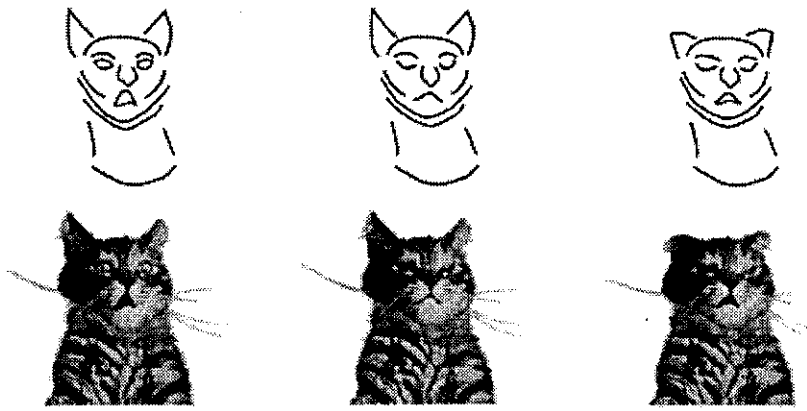


Figure 7.10: Rotoscoped features and resulting animation.

Totally automatic warping techniques have been used for very specific applications. A relevant and quite successful example is found on the multiple master font technology developed by Adobe Systems. This is a system which enables the automatic creation of new fonts by specifying the parameters of font width and weight. This is illustrated in Figure 7.11.

A similar technique has been used by Adobe in the product Super ATM. In this application, when there is a need for some screen font which is not available, the program creates the font by interpolating parameters from existing loaded fonts.

Nonetheless, use of automatic warping specifications are still rare—although some efforts in this direction have been made—and are loose by definition, leaving little control to the user. Despite these automation attempts, the form of specification and the user interface are essential issues in current morphing techniques.

Figure 7.11: Generation of fonts by automatic warping.

7.4 Warping, Attribute Combination and Morphing

We have seen that a general morphing transformation is in fact a composition of various other transformations. The specification of a morph, therefore, amounts to specifying each of its sub-transformations. The separate specifications are not clearly distinguished in many approaches, where from one main specification all the transformations are inferred, avoiding excessive user input. The analysis of different types of specifications separately, though, proves to be helpful, as different types of information have different specification requirements.

Most of the discussion in this chapter is biased towards warping transformations, reflecting the importance given to it in the literature and the greater complexity of warping specifications. There are a few differences, however, when warping transformations are being used to create morphs. First of all, warping is a unary operation, i.e., it takes *one* object and an specification as input producing one warped object as output; on the other hand, the morphing transformation is a binary, operation. It takes *two* objects and a specification and produces one morphed object.

In a plain warping transformation, the specification is totally arbitrary, with practically no restrictions imposed on the user. Basically, there is no such thing as a "natural" deformation, which could be inferred directly from the object; a user specification is an intrinsic necessity.

For morphing transformations, the two input objects can be seen as an expression of what has to be done, as usually there is a natural

association between features of the objects that must be matched in the transformation. In fact, human beings can easily and instantaneously associate features of two input objects if they are structurally similar. This can be seen in figure 7.12, where there is an obvious correspondence between the characteristics of the two images.

The morphing techniques, though, still do not take advantage of most of the information that is present in the input objects, relying in the specification of the transformation to generate the morph between the objects. Ideally, just the two input objects would be enough to create a "standard" transformation that maps one object into the other. One such attempt, based on neural networks, is described in section 9.5. Another reasonable approach would be based on pattern recognition techniques, but this is still an underexplored area.

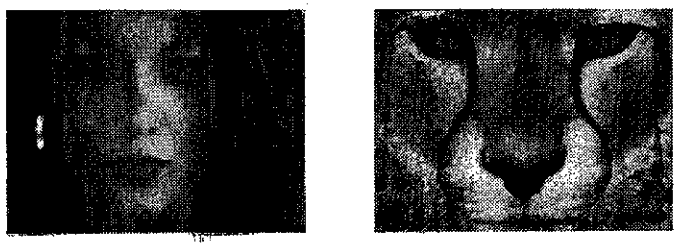


Figure 7.12: Two structurally similar graphical objects.

7.4.1 Attribute Combination Specification

As we have already seen, the transformation of the shape of the objects is not sufficient to generate a complete transformation of two objects. The attributes of the objects at each point of their domain also have to be transformed so that a complete metamorphosis transformation is accomplished. This corresponds to generating intermediate values for the attributes for each step of the transformation.

Take, for instance, the color attribute: a transformation between a green object and a red object would show an object that gradually

changes its color from green to red. The way the color of the intermediate object is computed is dependent on the scheme used to represent the colors and on the intrinsic characteristics of colors. In this simple example, suppose that each color is represented as a red, green, blue vector; then, a linear interpolation of two colors produces RGB values representing colors that lie perceptually between the two original colors. Other, more complex attributes, may require more complex combination transformations to result in values which are able to furnish reasonable transitions between the two original inputs.

Global and Piecewise Transformations

An important tool that helps achieving particular effects is the specification of the rate of transformation in a piecewise and localized way. This results in objects whose parts are transformed at different times. In figure 7.13, from the morphing sequence shown in Figure 6.2, page 109, the change of the grayscale values of the woman's face was done in a localized way. Grayscale intensity of the face predominate over those of the woman hair.

Figure 7.13: A local attribute transformation: face transformed before hair.

Naturally, as in the warping case, a transformation could be given explicitly, through the use of a mathematical formula or a surface describing the function. A more practical interface is to allow the values of the function to be specified at selected places, and let the computation phase interpolate the values in between. Research in this direction has appeared in the literature. The interested reader should consult (Lee et al., 1994), (Lee et al., 1995a), and (Lee et al., 1995b).

The specification of the sparse values of the function plays a very important role, affecting the result of the morphing transformation and the ease-of-use of the user interface. There are two ways to specify a local attribute transformation:

- **Independent of the warping** – in this case, the function takes an absolute position (x, y) as parameter, and consequently the attribute combination is dissociated from the underlying warping transformation. The same results would be obtained if the two transformations were used in a totally independent way.

- **Dependent of the warping** – the function takes as parameter a warped position $W(x, y)$, and thus the attribute combination transformation accompanies the warping transformation, making the resulting morph consistent and harmonious. Naturally, this is the case more applicable to morphing.

In this way, the specification of a local attribute combination that will be used for morphing must be done in conjunction with the warping specification. In a feature based specification, for instance, the values would be associated with each of the marked features, emphasizing the idea that the specification will "move" as the features are transformed. In a spline mesh specification, the local attribute combination values would be associated to each control point of the mesh.

Note that what is specified at each node or feature is in fact a function relating attribute combination amounts to time, so that the attribute combination changes locally as time passes. The techniques for combining the values, however, can be applied in the same way, as these functions are all evaluated at a particular moment when generating a frame of the animation.

Chapter 8

Computation of Transformations

In the previous chapters, object manipulations were performed in the continuous domain, an ideal setting that allows perfect, infinitely precise, transformations. However, the representation of continuous objects presents several problems, specially in the intrinsically digital world of finite memory computers.

Representing continuous objects is impractical, unless in some very specific cases where the object is defined by some mathemtical equation. Therefore, various schemes are employed to represent approximations of continuous objects so as to be able to reconstruct them as precisely as possible. The variety of object representations is the main factor that affects the computation of transformations. This topic was discussed in chapter 3.

This chapter starts by analyzing some of the problems that arise when domain and range transformations are applied to digitally represented objects. This is a prelude to a deeper, closer to implementation, analysis of the various techniques used to compute warping and morphing transformations.

An attempt is made to isolate the idiosyncrasies of different object representation techniques, and extract the essence of each technique. This broader view enables those techniques to be applied to a more

general class of graphical objects than initially imagined.

8.1 Representation of Graphical Objects

Practical object transformations have to deal with objects that are stored under schemes appropriated for computation. The specification, representation and computation of domain and range transformations involve various levels of manipulation of the data structures representing the object under transformation.

Transforming the object's attributes can be easier or harder than transforming its shape, depending on the representation type. Moreover, some of the techniques are unequivocally bound to a certain form of representation, depending on some of its particularities to achieve the desired results and efficiency.

Previous work on $2D$ morphing has concentrated on images and drawings. For images, the range transformation, i.e., the alteration of the values of the pixels, is usually very simple, with no complications other than relatively standard averaging involved (see (Wolberg, 1990) or (Costa, 1994)). When domain transformations are involved, however, several complication arise related to sampling and reconstruction problems. For this reason a wide variety of specification and implementation techniques have been developed.

In image morphing, the warping technique usually determines the possible forms of specification, the efficiency of the computation, the type of the user interface and even the techniques used for the range transformation. Naturally, most work in image morphing has been directed to warping techniques, and therefore this is the area from where other forms of morphing have the most to learn. It will be shown that some of the image morphing techniques can be successfully extended to handle other forms of graphical objects and higher dimensional cases.

The same is true for techniques used to transform $3D$ objects. In fact, the classification of techniques in terms of dimensionality seems to be inappropriate, as the applicability of techniques is much more closely related to the scheme of representation than to the dimension of the object.

8.2 Discrete Graphical Objects

A graphical object was defined in chapter 2 as a function $f : U \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$, where U is the shape, f is the attribute function, and $\mathbb{R}^m$ is the attribute space. Representing such objects generally involves some form of finite sampling. In the case of images, for instance, which are defined in the continuous domain as $f : U \subset \mathbb{R}^2 \rightarrow C$ (C is a color space), the usual representation is through the use of a set of samples of f taken at a finite number of locations p_i.

$$f^* = \{(p_i, s_i) | s_i = f(p_i), p_i \subset U\}.$$

Note that both the domain and the range of the image are digitally represented. The sample positions usually possess some form of structuring that makes a consistent representation of the samples easier. A natural structure is the matrix representation which uses a a rectangular grid, aligned with the major axes of the domain of the image, in such a way that a digital image can be described in a tabular form by a matrix of sample values (see chapter 3.

Analogously, a *digital animation* is a representation of a continuous animation through samples taken at a finite number of positions and instants of time. Since an animation is a function $f : \mathbb{R} \times U \subset \mathbb{R}^2 \rightarrow C$, the positions at which the samples are taken involve time as well. Sampling an animation in time produces animation frames, each of which is a digital image. Similar problems of structuring the samples occur with animations, so that frames are usually—but not necessarily—taken at regular time intervals.

Other forms of representation present similar situations, involving samplings of the domain and range of the functions. As usual, regularly structured samples are easier to work with, as the position of the samples is not stored but inferred, and different objects can have their sampling structure made compatible in a simpler fashion.

8.2.1 Sampling and Reconstruction

As seen in chapters 2 and 3, there are two main operations involved in the use of digital and continuous objects: sampling and reconstruction.

Their relationship is depicted in the diagram in Figure 8.1. Sampling is the process of transforming a continuous object into a digital object, and reconstruction is the process of transforming a digital object into a continuous object through some form of interpolation.

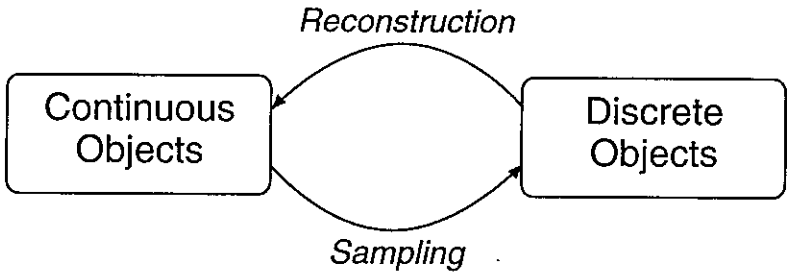


Figure 8.1: Sampling/Reconstruction relationship.

Most object transformations—specially the ones considered here—would benefit from the use of continuous objects instead of discrete ones, both in terms of quality and ease of implementation. Therefore, for the best results, object transformations would be ideally computed on the continuous objects of the mathematical universe. Since objects are really stored as sampled digital objects, the conversion between the two forms of representation is an integral part of the implementation of object transformations. This relationship is more evident in the diagram in figure 8.2, from (Wolberg, 1990).

The reconstruction process is essentially an interpolation process, and thus any interpolation technique can be used. Ideal interpolations can be derived from sampling theory, but other types of interpolation are more feasible, natural and do produce good approximations of the original signal in practice.

The reconstruction and sampling processes are closely related and cannot be treated separately, as a badly sampled object cannot be properly reconstructed, as discussed with details in chapter 4. Intuitively, if a complex image is sampled at a single position, there is no way to reconstruct the original image from the samples, as almost all information

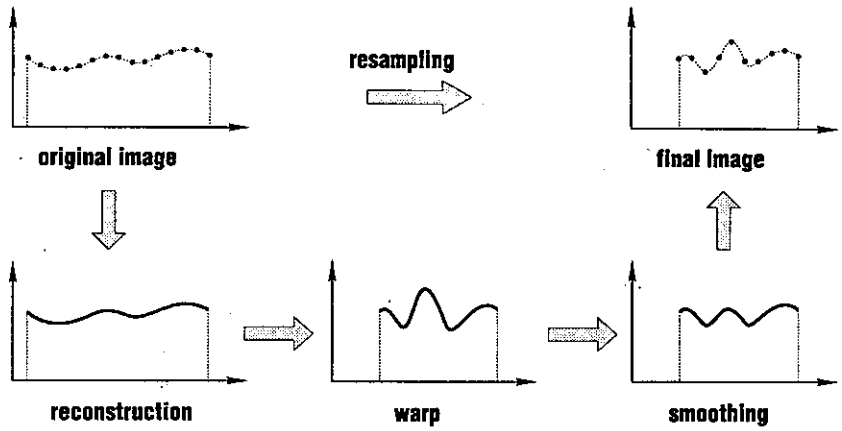


Figure 8.2: Digital object transformation cycle: reconstruct, transform and sample.

has been lost.

On the other hand, if a single sample of a flat color image is taken, the reconstruction of the original object is easy and precise. This trivial image example indicates that the required sampling rate is dependent on the nature of the object being sampled: more "detailed" objects require a higher number of samples.

8.3 Inverse and Forward Methods

This section is based on the contents of chapter 7 from (Gomes and Velho, 1995b). For additional material, the reader should also consult (Wolberg, 1990).

When we have a transformation $T: U \longrightarrow U'$, of the space, it is possible to devise two different interpretations for it:

- T is considered as transforming points of the space U, taking each point p, and moving it to a new position $p' = T(p)$ on the space;

- T can be considered as a change of coordinates of the space U. To each point $p = (x_1, \ldots, x_n)$, T associates new coordinates $Y = (y_1, \ldots, y_n) = T(x_1, \ldots, x_n)$. The coordinates of p in the new system are defined by $T^{-1}(y_1, \ldots, y_n)$.

Considering the above "dual" interpretations for a transformation, results into two distinct ways to proceed for its computation: forward and inverse methods.

For continuous objects, the choice of applying a forward or inverse method is immaterial, as the mapping was done on a point-by-point basis. For discrete objects, however, these two situations are significantly distinct.

We will exemplify this using digital images. Of course, it generalizes to matrix representations of graphical objects of different dimensions. Some care must be taken, however, when applying this technique to some other representation technique of a graphical objects.

8.3.1 Forward Method

Forward mapping processes points in the input with T, determining their mapped position in the output. Since we are working with a discrete object, mapping any set of regular point samples does not necessarily cover all pixels in the output image, leaving holes in it.

On the other hand, since the output image is discretized, many point samples may "overlap", being mapped to the same output pixel. Moreover, the output pixels can be produced in any order, regardless of the traversal of the input. In general, forward mapping may require excessive computation in point sampling unimportant areas, and leave other areas unsampled. Costly computations are required to properly and evenly sample the input image. Taking the pixel geometry as an area instead of a point avoids most of the problems just mentioned (see Figure 8.3). The mapped pixel p_o can is approximated by a quadrilateral, which, after intersected with the regular output grid can be used to determine the influences of p_i in the output pixels.

In this technique, the values of the mapping on the regular grid of the output image, are computed from its values at the unstructured grid

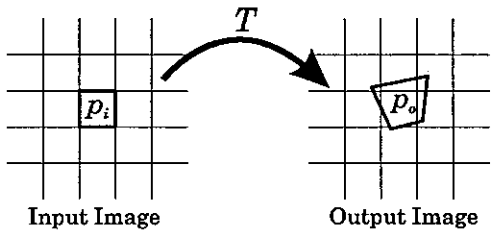


Figure 8.3: Forward mapping of an input pixel p_i.

obtaining by transforming the regular grid of the input image. This is illustrated by Figure 8.4.

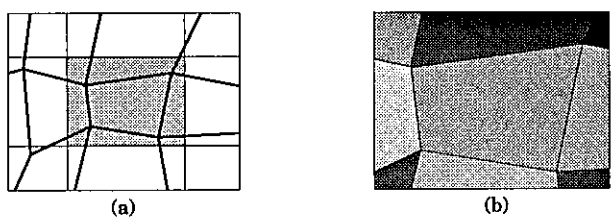


Figure 8.4: Reconstruction of regular grid values from irregular one.

8.3.2 Inverse Methods

Inverse mapping works on the output image, transforming each point with T^{-1} to determine its originating position in the input image. The coverage of the input image is analogous to that of the output image in forward mapping point sampling, i.e., the input image may not be fully covered, samples can overlap in the input and be produced in any order, etc. In this case, however, overlaps in the input image indicate important areas to be sampled with more accuracy, and holes indicate

areas that do not affect significantly the output image. The output image is sampled evenly, and in the order output pixels should be produced. In spite of these properties, artifacts common to all sampling processes can appear. It is not surprising that almost all implementations of digital image transformations use inverse mapping. Naturally, by considering pixels as approximating quadrilaterals the input image pixels influences can be more precisely computed, as they would if a high number of point samples were used (see Figure 8.5).

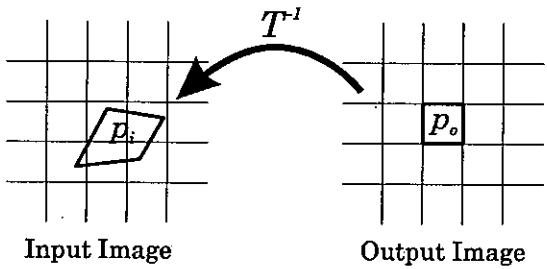


Figure 8.5: Inverse mapping of an output pixel p_o.

In this method, the values of the waping on the unstructured grid obtained from the grid of the target image, by the inverse mapping T^{-1} are computed from the image values on the regular grid of the source image. This is illustrated in Figure 8.6.

8.4 Multiple-pass Warping

A transformation T is *separable* if it can be expressed by a composition of a finite number of transformations f_i, $i = 1, \ldots, n$. That is,

$$T = f_n \circ f_{n-1} \circ \ldots \circ f_2 \circ f_1.$$

The reader should not confuse the concept of separability above with the similar concept used in image processing related to separable filter kernels. There, the mapping composition operation is substituted by the multiplication operation.

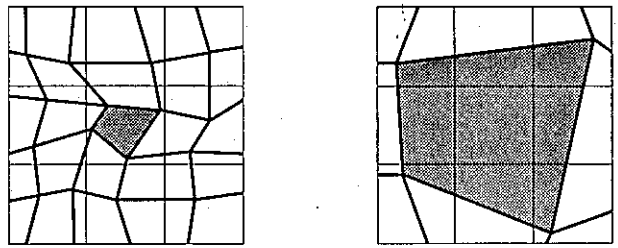


Figure 8.6: Reconstruction of irrregular grid values from the regular one.

If each of the functions f_i is simpler to compute than T, it may be more efficient to implement T indirectly as a composition of these simpler functions. A simple example of this fact is found in the implementation of projective warps. A plane projective warp with two arbitrary vanishing points can be computed by defining projective warps with vanishing point over the coordinate axis. By combining these two classes of projective warps with arbitrary rotations of the plane, we are able to obtain projective warpings with arbitrary vanishing points. This is illustrated by the sequence of images in Figure 8.7: from left to right, we have the original image; a projective warp with vanishing point at the x-axis; a rotation of 45 degrees in the clockwise direction; another projective warp with vanishing point at the y-axis.

The above example shows that a generic projective warp can be realized as the composition of two simpler projective warps and a rotation. This fact has a great influence both on the warping computation and the user specification. It has been used by some popular image manipulation and illustration programs.

When an n-dimensional graphical object is given by its matrix representation, it seems natural to try a decomposition of a warping T of the object into n simpler warps

$$T = s_1 \circ s_2 \circ \ldots, \circ s_n, \tag{8.1}$$

where each s_i affects only the i-coordinates of the graphical object shape.

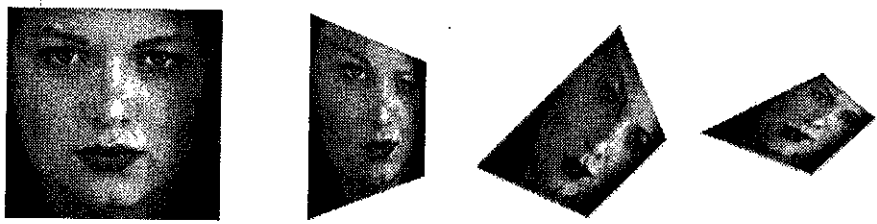


Figure 8.7: Projective warp by sucessive composition.

That is,

$$s_i(x_1, \ldots, x_j, \ldots, x_n) = (x_1, \ldots, s_i^j(x_1, \ldots, x_n), \ldots, x_n).$$

When this decomposition is possible, it simplifies the warping computation a lot, since it reduces an n-dimensional warp to n sucessive 1-dimensional warps. In the literature the term *separable* is used only when we have the decomposition in equation (8.1).

In the case of images, when the warping transformation h is separable, it is possible to compute it by applying separately the horizontal

$$g(x, y) = (g_1(x, y), y),$$

and vertical

$$f(x, y) = (x, f_2(x, y))$$

warps. Therefore, we have a *two-pass warping*. In brief, horizontal lines are invariant by the horizontal warp, and vertical lines are invariante by the vertical warp.

We will perform the computations to find the 1-dimensional warps f and g from h. These computations were taken from (Gomes and Velho, 1995b). Suppose that h has coordinates h_1, and h_2, that is,

$$h(x, y) = (h_1(x, y), h_2(x, y)). \tag{8.2}$$

We must find f and g in such a way that $h = g \circ f$, f is a horizontal warp, and g is a vertical warp. Since we have a two-step process, we have 3 coordinate systems: the system (x, y) of the original image; the system (u, v) obtained from (x, y) by the horizontal warp f; and the sistem (r, s) obtained from the system (u, v) by the vertical warp g. These three systems are illustrated in Figure 8.8.

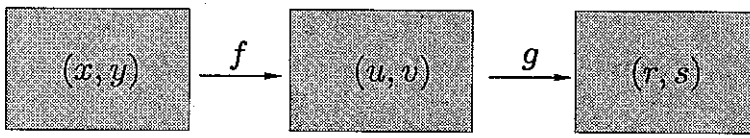


Figure 8.8: Horizontal and vertical warps.

Therefore, we can write

$$(r, s) = h(x, y) = (h_1(x, y), h_2(x, y)). \tag{8.3}$$

Our purpose is to determine the transformations $f(x, y)$ and $g(u, v)$. Since f is a vertical warp, and g is a horizontal warp, we have

$$f(x, y) = (x, f_1(x, y)), \tag{8.4}$$

and

$$g(u, v) = (g_1(u, v), v). \tag{8.5}$$

Therefore

$$(r, s) = g(u, v) = g(f(x, y)) = g(x, f_1(x, y)) = (g_1(x, f_1(x, y)), f_1(x, y)). \tag{8.6}$$

Comparing equations (8.6) and (8.3) we obtain

$$f_1(x, y) = h_2(x, y), \tag{8.7}$$

therefore $f(x, y) = (x, h_2(x, y))$.

Now, we will compute $g(u, v)$. We know that

$$g(u, v) = h(x, y) = (h_1(x, y), h_2(x, y)). \tag{8.8}$$

By comparing this equation with the equation (8.5), and using the fact that $x = u$, we obtain

$$g_1(u, v) = h_1(x, y) = h_1(u, y). \tag{8.9}$$

Therefore, we need to determine a function φ which relates the coordinates y and (u, v), that is, $y = \varphi(u, v)$. We will have

$$g_1(u, v) = h_1(u, \varphi(u, v)), \tag{8.10}$$

that is, $g(u, v) = (h_1(u, \varphi(u, v)), v)$.

The existence of the function φ is granted if g is a 1-to-1 map. But we should observe that even in this case, we face the problem of computing φ, and in general this is not an easy task. We will give an example when the warping h is a rotation of the plane.

8.4.1 Two-step Rotation

If the warping transformation h is a rotation of the plane by an angle θ, we have

$$h(x, y) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{8.11}$$

From this equation we obtain

$$h_1(x, y) = x\cos\theta - y\sin\theta, \tag{8.12}$$

and

$$h_2(x, y) = x\sin\theta + y\cos\theta \tag{8.13}$$

Using the expression of $h_2(x, y)$ in the equation (8.12), along with equation (8.7), we obtain

$$f_1(x, y) = x\sin\theta + y\cos\theta, \tag{8.14}$$

and this determines the transformation f which accomplishes the vertical warp.

Now we will proceed to obtain the horizontal warp. Using equation (8.9) with the expression of $h_1(x, y)$, in equation (8.12), we obtain

$$g_1(u, v) = u \cos\theta - y \sin\theta. \tag{8.15}$$

Now, we need to compute y as a function of the coordinates u and v (that is, we need to compute the function φ which appears in equation (8.10)). We have

$$v = f_1(x, y) = x \sin\theta + y \cos\theta, \tag{8.16}$$

therefore

$$y = \frac{v - u \sin\theta}{\cos\theta}, \quad \text{se} \quad \cos\theta \neq 0. \tag{8.17}$$

Substituting the value of y from above, in equation (8.15), we obtain the equation

$$g_1(u, v) = u \cos\theta - \frac{v - u \sin\theta}{\cos\theta} \sin\theta, \tag{8.18}$$

which determines the horizontal warp of the decomposition. Note that the horizontal warping is not defined for values $\theta = k\pi + \pi/2$. Moreover, it is clear that when θ approaches one of these values, there is a greta distortion of the horizontal warp. We conclude that in order to perform a 87° rotation in an imagem it is advisable to make a 90° rotation, followed by a rotation of 3° in the opposite direction.

Figure 8.9, from (Wolberg, 1990), shows a 30° rotation of an image obtained in two steps: a horizontal followed by a vertical warp.

The reader should notice that when we implement multiple-step warpings we might face problems with the intermediate one-dimensional warpings. A fairly common case, known by the name of "bottleneck problem" in the literature, occurrs when one of the intermediate transformations is not injective. In this case, some points collapse.

Additional material about multiple pass warp of images can be found in (Wolberg, 1990). In particular, this reference describes some three pass warping alternatives for images, which do not present the bottleneck problem.

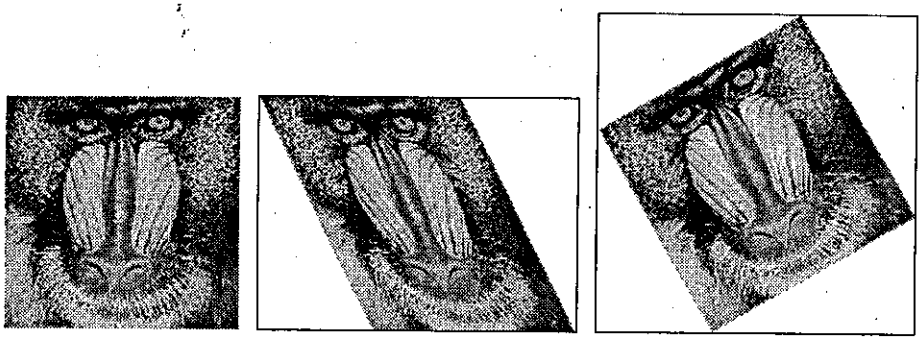


Figure 8.9: Two-step image rotation.

8.5 The Blending Step

We have seen in chapter 5 that a morphing transformation is an attribute combination preceded by warping transformations that align the domain of definition of the graphical objects involved. For the attribute combination—and the combined attributes—to make sense, the warped objects must be coincident. Since an object is defined by a subset of the space and an attribute function, this is equivalent to saying that the two objects are defined over the same subset of the space, or that we have one object with two attribute functions. These two attribute functions are then combined resulting in a single object which is a transition between the two original ones.

This ideal scenario is not easy to achieve in practical situations. Usually, it is not possible to obtain a perfect alignment of the objects with two generic warping transformations, using warping techniques such as the ones described later. This creates the need for specific techniques that make the two domains compatible, performing a *blending* step that generates a single object.

For certain types of object representation, such as boundary represented objects with differing discretizations, this is a specially significant problem. In the simple example in figure 8.10, it is shown that

two such objects may have incompatible combinatorial topologies, even though the geometries are reasonably aligned. A blending step is necessary to obtain a single object, basically reconstructing the warped input objects from their sampled boundary, and resampling the reconstructed boundary under a common sampling structure. In this case, the simplest blending step would be just picking up one of the objects, assuming the geometry output from the warps is close enough.

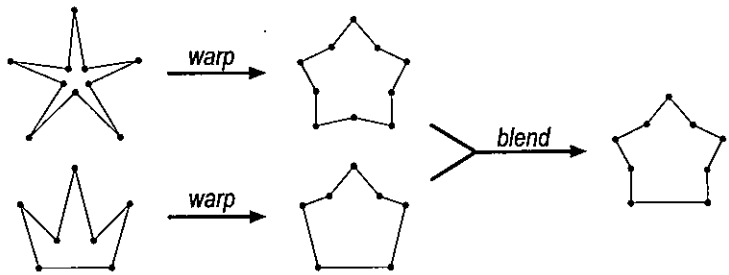


Figure 8.10: Blending in a morphing between two B-Rep objects.

In general, blendings produce small alterations in the shape of the objects, requiring them to be relatively similar. More advanced techniques may even eliminate the need of a warping phase in some circumstances, transforming moderately differing objects in one step with excellent results (Sederberg and Greenwood, 1992), but a warping phase will always help to achieve specific results by giving additional user control.

These advanced techniques are frequently automatic, guided by some form of heuristics, since transformations are an inherently ambiguous problem: a blending between two objects is a multiple solution problem , with no single "correct" answer; the choice of the best solution depends on the application. Sometimes, it reduces to a perceptual problem, and user interaction is required to indicate the better alternative for a particular result. Attempts to define a metric of the quality of a blending allow an arguably best solution to be picked up by an optimization algorithm, as shown in description of the physically based blending in section 10.3.

Chapter 9

Warping Techniques

Given a certain specification, there are various choices in implementing the transformation, not only in selecting the type of reconstruction, but also in deciding how the object data will be warped. Some of the algorithms and forms of specification are intrinsically associated, although different combinations are possible, and in fact, can originate improved warping techniques. In this section, various algorithms which appeared in earlier work will be discussed, and in particular a technique merging previously dissociated specification and algorithms.

9.1 Triangle Mesh

In this algorithm, the deformation is described by two triangular meshes, one defines the undeformed coordinate system, and the other defines a deformation of that system (see Figure 9.1). The two meshes must have the same combinatorial structure, so that the correspondence between pairs of triangles is well defined.

In general, when using a triangle mesh warping, we know the warping transformation w at the vertices of the triangulation, and we must extend it to a transformation W defined on the whole domain. This reconstruction process is attained by extending the transformation to the edges and the interior of each triangle imposing some smoothness at the common edges of neighbor triangles.

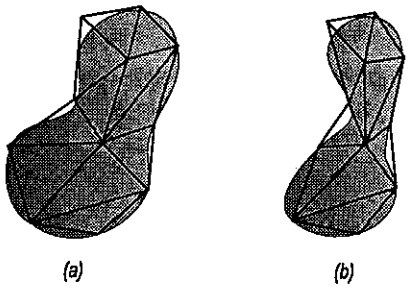


Figure 9.1: Triangle mesh deformation: (a) original; (b) deformed.

A simple way to obtain a continuous extension is possible by the use of barycentric coordinates (see (Birkhoff and Maclane, 1960)). Consider a a triangle Δ with vertices p_0, p_1 and p_2. For each $p \in \Delta$ there exists real numbers λ_0, λ_1, and λ_2, with $\lambda_i \geq 0$ and $\lambda_0 + \lambda_1 + \lambda_2 = 1$, such that

$$p = \lambda_0 p_0 + \lambda_1 p_1 + \lambda_2 p_2.$$

Therefore, we define

$$W(p) = \lambda_0 w(p_0) + \lambda_1 w(p_1) + \lambda_2 w(p_2).$$

This is very effective, and easy to implement, but the resulting warping does not have good smoothness properties.

In fact the method of barycentric coordinates constructs a piecewise linear map, which is linear inside each triangle but has just C^0 continuity at common edges. Therefore, the appearance of some artifacts during the warping will be unavoidable. This is illustrated in Figure 9.2.

Practical tests have shown that the artifacts are more noticeable when the mesh has very irregular triangles. Besides using a "regular" triangulation, a more expensive solution consists in the use of a higher degree interpolating function which has at least C^1 continuous at the common edges.

The triangle meshes used as input to this algorithm can be specified in various ways. The crudest form is to directly construct both triangle

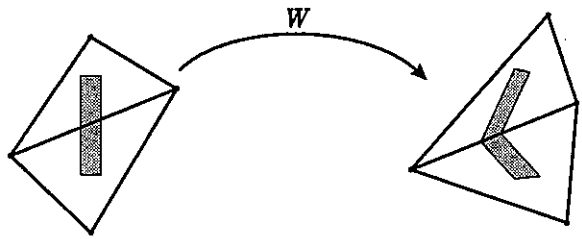


Figure 9.2: Discontinuity of barycentric coordinates mapping.

meshes, ensuring their cross consistency. Obviously this is a tedious and error prone work, and the required data structures are inconvenient to maintain. A much better alternative is to use a form of feature specification, where the features are marked as points or line segments, and apply an automatic triangulation algorithm to construct the mesh. An automatic triangulation that produces regular triangles, such as a Delaunay triangulation (Preparata and Shamos, 1985), which maximizes the smallest of the internal angles of each triangle, is a natural choice. A triangulation with restrictions can be used if line segments are also used as features.

This algorithm has been used to deform digital images by scan converting (Foley et al., 1990) each triangle in the deformed state, and applying an inverse mapping to each of the pixels thus obtained to determine the originating area. Obviously, since it defines a coordinate system mapping independently of the object representation, this algorithm is applicable to other forms of representation. For instance, drawings can be easily transformed by mapping just their vertices according to the global mapping described by the meshes. We should observe that mapping just the vertices is incorrect if the edges are not invariant under this particular change of coordinates, although the results are perfectly acceptable, and even attenuate discontinuities for barycentric triangle mappings.

Special care must be taken with automatically generated meshes to handle a problem that could be avoided in the manual construction of

the meshes: the *foldover*, where the mesh "folds" upon itself. As the association of the features in the source and destination sets is arbitrary, the mesh in the source corresponding to the automatic triangulation in the destination is not necessarily a valid partition (see Figure 9.3). The net effect is that the mapping relation will not be one-to- one any more: it may not be an injective function, or it may not be a function at all.

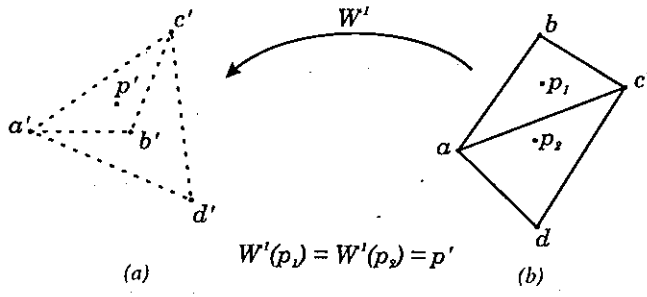


Figure 9.3: Triangle mesh foldover: (a) source; (b) destination.

9.1.1 3D Triangle Mesh Warping

This $2D$ triangle mesh technique can be naturally extended to 3D, and even to dimension n, using simplexes (see (Birkhoff and Maclane, 1960)): a mesh of tetrahedrons is given in two different states, one defining a coordinate system and the other a deformation of it. For corresponding tetrahedrons, interior points can be easily mapped using barycentric coordinates. In this form, this algorithm is suitable for any object representation. In particular, for voxel based volumetric objects, a 3D rasterization of the interior of the tetrahedrons is required.

Describing the deformation directly with the tetrahedron mesh is not viable unless it is obtained algorithmically somehow. Just like the 2D case, an automatic triangulation of points used to indicate features is a feasible and easy to use alternative, and an automatic triangulation can be used to generate the meshes. A regular triangulation, such as a 3D

Delaunay triangulation, will minimize the edge and face discontinuities introduced by the linear mapping inside each tetrahedron.

9.2 Free-form Warpings

This is a generic technique for computing a warping specification by the use of free-form coordinate systems. By changing the control parameters of the free-form coordinate curves, we obtain a change of coordinates which performs the desired warping.

9.2.1 Two-pass Spline Mesh

This algorithm was developed specifically for use with digital images. It takes advantage of the matrix representation of images. Instead of transforming an image directly, an intermediate image is created, which is further deformed to achieve the final result. This corresponds to a composition of two simpler transformations—one containing just the horizontal displacements and the other just the vertical—that is equivalent to the desired deformation. Purely horizontal or vertical deformations are much easier to implement for digital images, specially in avoiding sampling and reconstruction problems. Such decomposable transformations were introduced in section 8.4.

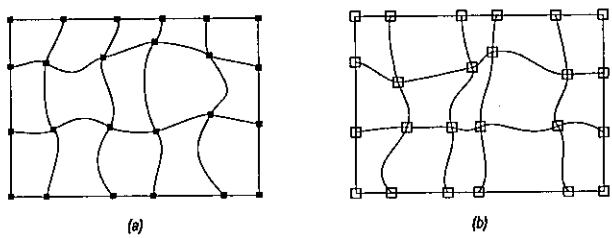


Figure 9.4: Spline meshes.

The two-pass spline mesh warping uses as input two regular spline meshes, such as those in Figure 9.4, restricted to have rectilinear splines

at the edges of the image, and non-crossing splines elsewhere. Once again, the first mesh defines the undeformed coordinate system and the other describes the deformation of that system.

The two passes are analogous, and can be performed in any order. In the horizontal pass, each horizontal scanline is intercepted with the vertical splines in the undeformed and deformed states, as depicted in Figure 9.5(a). The deformed state is determined in this pass by considering just the horizontal displacements. These interceptions are then interpolated to obtain the scanline mapping function, as shown in Figure 9.5(b), by placing interceptions with the deformed splines in one axis and interceptions with undeformed splines in the other. This mapping function relates all points of the scanline in the undeformed coordinate system to their deformed state. This process is repeated for each scanline to obtain a horizontally deformed image.

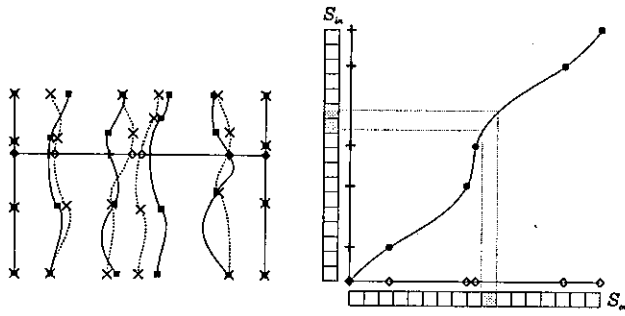


Figure 9.5: Scanline warping: (a) interception; (b) mapping function.

An example can be seen in figure 9.6. The input image on the top left is deformed according to the mesh shown. First, just the horizontal displacements are considered to produce the intermediate image on the top right. Then, this horizontally warped image is used in the second pass, where the vertical displacements are considered, yielding the completely deformed image on the bottom right.

Although this technique proves to be quite versatile and efficient, the fact that it is based on a mesh specification requires the user to enter

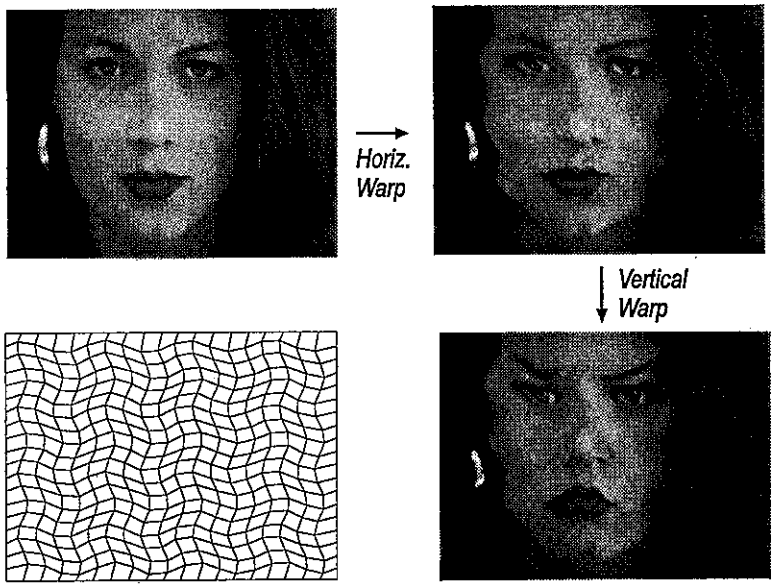


Figure 9.6: Horizontal and vertical passes, with intermediate image.

too much information while defining the deformation. This is specially true in less important areas of the domain, where a precise control is not needed and some kind of automatic process could be used, as discussed previously.

The spline meshes may be directly created and modified, or indirectly obtained in an attempt to reduce unnecessary user input. It is easy to see that, despite its efficiency for digital images, the two-pass spline mesh warping technique is of limited use for other representations. It is possible to use spline meshes to define coordinate systems and deform drawings, for instance, but the two-pass approach is useless in such cases.

This section was strongly based on (Wolberg, 1990), where the reader can find a very detailed description of the algorithm of two-pass spline mesh warping.

9.2.2 3D Three-Pass Spline Mesh Warping

The original 2D algorithm is applicable to digital images; its extension is correspondingly applicable to voxel based volumetric models. It is useful not only for morphing of volumetric data, but also as a tool for modeling volumetric objects through deformation, according to the "lump of clay" modeling paradigm (Sederberg and Parry, 1986). A major advantage of volumetric models is that there is no special treatment given to objects with different topologies, and therefore, the topology of an object can easily change during the transformation.

The 3D algorithm, three-pass spline surface mesh warping, performs three passes on the volumetric data, each of them computing the deformation in just one direction. The composition of these passes results in a complete deformation of the object, as shown in the pipeline in Figure 9.7. This effectively reduces the problem of warping the 3D solid to that of warping several 1D voxel strips, or scanlines. The passes of the algorithm are performed sequentially, but the deformation of each scanline in each pass is independent of the other scanlines, and therefore is suitable for parallelization, as is the 2D algorithm.

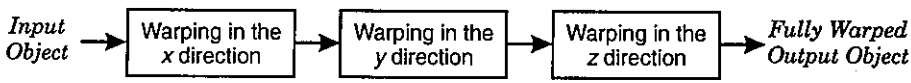


Figure 9.7: Three-pass warping pipeline.

Each pass of the algorithm performs the deformation in the direction of one of the Cartesian axis x, y or z. A mesh is composed of a lattice of control points C_{ijk}, with m points in the x direction, n in the y direction and q in the z direction, totaling mnp control points, in such a way that each control point can be easily referred to by integer indices. The pass for the x axis is as follows, and the others are performed analogously:

- An intermediate mesh I is created by considering just the displacements of the control points in the x direction from the original mesh S to the deformed mesh D. In this way, the control points of I will

have the same y and z coordinates of the corresponding point in S and the x coordinate of the corresponding point in D.

- A spline surface S_i is fitted to each set of control points of with a constant i index, resulting in surfaces that are roughly perpendicular to the direction. An identical process is performed to obtain the I_i spline surfaces passing through the control points of I.

- Each voxel strip in the x direction is intercepted with the S_i and I_i surfaces, resulting in two sets of collinear interceptions.

- For each voxel strip, the interceptions with S_i and I_i are fitted with a spline curve, resulting in a scanline mapping function which is used to map voxels in the source object to their x-direction deformed positions.

The minimization of sampling and reconstruction problems is now a simpler one-dimensional problem, that is performed on a scanline basis. This simplification helps making this algorithm a very efficient procedure to warp volumetric data with good quality results.

An implementation of this algorithm (Darsa and Costa, 1995) has been used to produce the results in figure 9.8. In this case, the specification of the warping was not done directly with 3D meshes, as will be shown in section 9.4, but the computation of the warping was done in three passes using spline meshes just as described above. These images were rendered with the Volvis system (Avila et al., 1994).

In the 2D case, the specification of the warping using such partition based techniques was a difficult work, which is worse in three dimensions. The use of alternative specifications, such as the one discussed in section 9.4, is a much better option. It has the advantage of an easier interface and an efficient, viable for parallelization, implementation for warping volumetric data.

9.2.3 Bezier Warping

In this technique, proposed in (Sederberg and Parry, 1986), objects are enclosed by a bounding parallelepiped which is split by a regular lattice

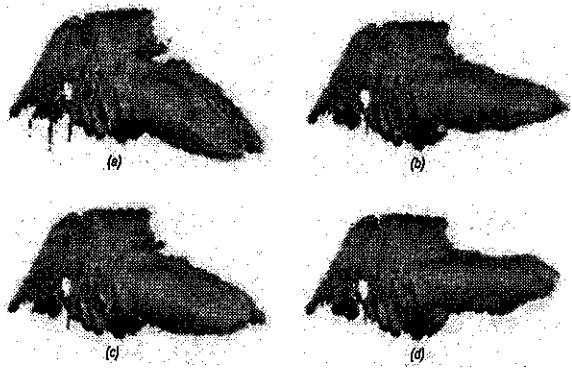


Figure 9.8: Volumetric lobster, warped to raise its claws.

of control points. By altering the position of these control points, a deformed coordinate system is defined, as shown in Figure 9.9.

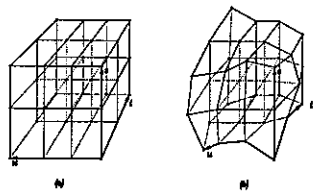


Figure 9.9: Free-form deformation coordinate system: (a) relaxed; (b) deformed.

The mapping from the relaxed system to the deformed one is relatively simple: each point X is decomposed in three coordinates (s, t, u) by projecting it onto the three axis of the parallelepiped, and the deformed point X' is obtained by transforming (s, t, u) with a Bernstein polynomial where the coefficients are the deformed control points. This effectively defines a map $W : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, which is a warp of the three-

dimensional Euclidean space as described in chapter 5.

For morphing applications, however, there would be no relaxed system, but two systems following the features of the underlying objects, as shown in the example in Figure 9.10(a). The mapping must be done from a deformed coordinate system to another deformed system, and this can be done with an intermediate step in a relaxed system, shown in Figure 9.10(b).

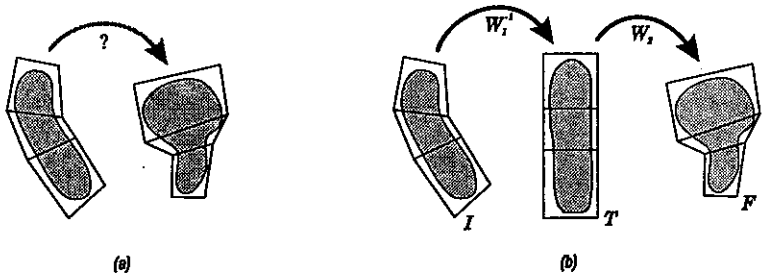


Figure 9.10: Free-form morphing: (a) direct; (b) with an intermediate step.

The polynomial transformation from the relaxed system T to the final system F is the same as described above, but the map from the initial system I to the intermediate system T requires the inversion of the Bernstein polynomial. This inversion involves finding the roots of a trivariate polynomial with a high degree, which increases with the number of control points in the lattice. The interested reader should consult (Nishita, Fujii and Nakamae, 1993).

Since this technique defines in fact a deformation of the ambient space, it can be easily applied to various different object representations. Also, the modifications required to apply this technique to 2D are minimal: a bivariate Bernstein polynomial is used instead, with two-dimensional control points. It is clear that this deformation technique is very similar to the spline mesh deformation described in the previous chapter, but using Bézier curves instead—other polynomial bases generate corre-

sponding results. The main difference is that the two-pass spline mesh warping algorithm was described specifically for digital images, while this technique simply establishes a deformation of the domain of the objects, with no reliance on a particular form of object representation.

9.3 Field Based Warping

This technique is used in general connected with the field based warping specification studied in chapter 7.

9.3.1 2D Field Based

This is the case when the technique is used for graphical objects which are embedded into the 2-dimensional euclidean plane. Therefore, it is quite suitable as a technique for image warping. It was in this context which it first appeared in (Beier and Neely, 1992). It was implemented for the production environment of Pacific Data Images.

The warping mapping is specified among features, and the technique extends the mapping to the whole object shape, by defining "influence fields" around each of the features of the image to be warped. As we discussed on the chapter about warping specification, chapter 7, the features are marked with pairs of oriented line segments (vectors) of which one indicates the original undeformed state and the other the deformation of that feature. Each feature has its own field of influence, defined as shown in Figure 9.11. The v coordinate is the distance perpendicular to the feature, while u is the distance along the feature. Note that u is normalized according to the segment length but v is an absolute distance, so that by stretching the vector in a direction, the neighborhood of that feature is also stretched along that direction, but not perpendicularly to it. It is possible to define u and v differently, but Beier and Neely conducted experiments that indicated the convenience of such definition in terms of user interface.

Each feature pair therefore defines a change of coordinates for the entire domain, and these conflicting transformations must be combined in a smooth way to produce a global change of coordinates. This combination is defined for each point as a weighted average of the positions of

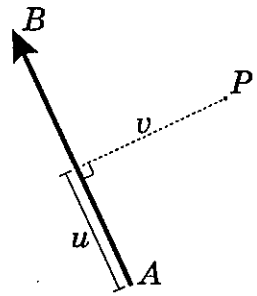


Figure 9.11: Feature defined coordinate system.

this point if mapped according to each feature, as shown in Figure 9.12. The point P is mapped by $F_1 \rightarrow F_1'$ to P_1' and by $F_2 \rightarrow F_2'$ to P_2'; the final mapped point P' is a weighted average of P_1' and P_2'.

The weight of each feature is inversely proportional to the distance of the point to that feature, being defined as

$$weight = \left(\frac{l^p}{a+d}\right)^b.$$

In the equation, l is the length of a feature, d is the distance from the point to that feature, and a, b and p are constants used to adjust the average. The constant a can be seen as the adherence of the feature: for values close to zero, points close to the feature will be mapped exactly as the feature determines; for greater values, points are more loose and free to move even if they are over the feature. The importance of the length of the feature is controlled by p: if it is zero, the length is ignored; otherwise, longer features are more important. The interpretation of b can be seen as the concentration of strength of the feature: large values make the strength of the feature large near the feature and small away from it, with a very quick decrease; small values make this decay slower, reducing the locality of the feature. Note that different values of constants can be associated to each feature, although it may be more convenient to use a global value of b.

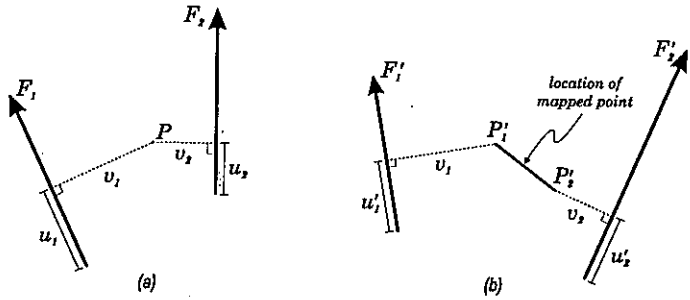


Figure 9.12: Combination of multiple changes of coordinates: (a) original; (b) deformed.

Note that, since the mapping of each point depends on all line segments, the addition of a single line influences the whole deformation, at least in theory. A careful choice of the constants can restrict the field of influence of lines, thus practically making the transformation local. Also, the weight calculation that must be repeated for each point, relative to each one of the lines, is quite expensive. As the interpolation is mostly automatic, sometimes the algorithm generates unpredictable results which can be avoided by tricky manipulations of the line segments. On the other hand, since this algorithm is executed in one single pass, there is no bottleneck problem.

This definition of the coordinate mapping, based on fields of influence around main features of the image, does not depend in any way on the structure or particularities of digital images. It is simply a clever and parameterized form of combination of different transformations, resulting in a global change of coordinates that can be easily used to transform objects in any representation.

9.3.2 3D Field Based

The extension of the field based $2D$ warping technique to compute warpings of graphical objects which are embedded in the 3-dimensional eu-

clidean space is is relatively straightforward. This was done in (Darsa, 1994). Another work on the same direction is described in (Lerios, Garfinkle and Levoy, 1995). The technique is suited to compute warps of 2-dimensional graphical object in 3-space, such as polyhedral surfaces, or to obtain deformations of solids, defined, for instance, as volumetric arrays.

We will describe below the technique introduced in (Darsa, 1994). The basic difference between this technique and the one described in (Lerios, Garfinkle and Levoy, 1995), is that the latter uses features of different dimensions, while fhe former uses 1-dimensional skeletons for feature specification.

As in the two-dimensional case, important features of the object are marked with oriented line segments, and a different position of each segment defines a deformation of the characteristic. In $3D$ space, there is an additional degree of freedom which results in an additional segment in the feature defined coordinate system, as shown in Figure 9.13.

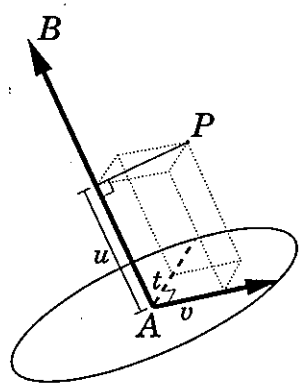


Figure 9.13: 3D feature defined coordinate system.

The two vectors shown in bold, user specified and constrained to be perpendicular, form an abstraction of a *bone*. A set of such bones, which mark all the relevant features in an object, is called an *skeleton*. To

obtain a Cartesian coordinate system, an additional axis is obtained from the cross product between the two vectors in a bone. Each point can then be projected onto each of these axes to determine its coordinates in this system. The u coordinate is normalized according to the segment length, while v and t are absolute distances.

The global change of coordinates defined by the two skeletons is obtained from a combination of the individual changes of coordinates defined by each pair of bones. The combination is a weighted average of the mappings of a point according to each pair of bones, just like the scheme used in the $2D$ case. The weight of each bone is computed as

$$weight = \left(\frac{l^p}{a+d}\right)^b.$$

The interpretation of the constants a, b and p is identical to that in the 2D case. The resulting technique is expressive and provides user control over the transformation, through the manipulation of the features and the subtle adjustment of the constants. There are several ways to improve the control and locality of this form of mapping, as shown by (Litwinowicz and Williams, 1994) and (Arad and Reisfeld, 1995).

Figure 9.14 shows a skeleton composed of 3D features, used to deform the lobster. The results of this deformation can be seen in figure 9.8. Note that the field based warping was used to deform just a spline mesh that envolved the lobster, as described in the next section. Equivalent results were obtained from a direct application of this mapping.

Since this technique simply defines a coordinate mapping, it is independent of the object representation. For B-Rep and implicit volumetric objects, for instance, its application is easy and straightforward. For discrete volumetric objects, on the other hand, besides the required resampling, this is a computationally expensive technique, since it must be applied at least to each voxel.

9.4 Field Controlled Spline Mesh Warping

This is a technique that combines the efficiency of the two pass spline mesh algorithm with the ease of use of feature based specification (Costa, Darsa and Gomes, 1992). After specifying the features of the image with

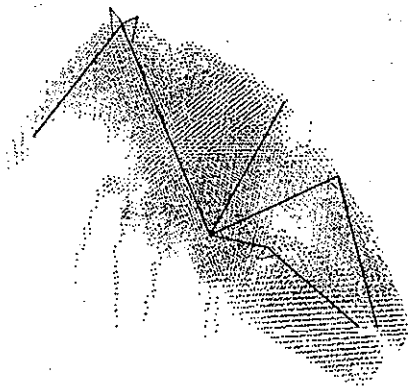


Figure 9.14: Skeleton used to warp a lobster.

oriented line segments, the algorithm uses these features as constraints to deform a relaxed spline mesh— that can be seen as a lower resolution image—which is subsequently used as input to the two-pass spline mesh warping algorithm, or the three-pass algorithm for volumetric data. More precisely, the spline control points are moved according to the field of influence—described in the previous algorithm—of each feature.

An example is shown in figure 9.15. The undeformed image is shown in (a), with the features shown as dark lines in the original state, and as lighter lines in the deformed state. Figure 9.15(b) shows the features superimposed on the deformed spline mesh, with the final deformed image shown in (c).

A disadvantage of this approach is that many of the problems of both algorithms are still present, namely bottleneck and foldover. However, the unpredictable effects of field warping can be detected beforehand, by inspecting the automatically deformed mesh for any apparent folding. Also, it is possible to control the precision of this approximation by increasing the mesh resolution globally and concentrating control points in the interesting areas.

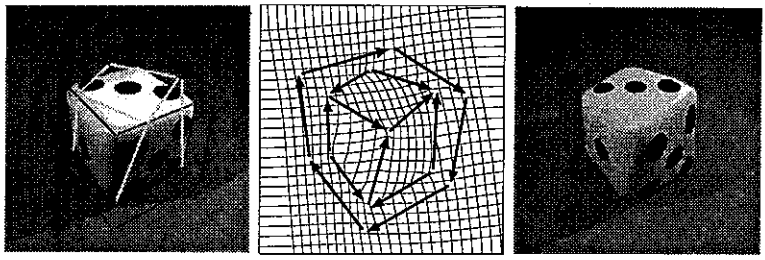


Figure 9.15: Field/mesh warping: (a) features; (b) mesh warped by features; (c) image warped by mesh.

9.5 Example Based Synthesis

Of the image warping techniques described here, this is the only one that fits in the automatic specification category. It is based in an analysis/synthesis paradigm, where example images are fed into a neural network that "learns" their meaning, and associates them to a set of user defined parameters (Beymer, Shashua and Poggio, 1993). The correspondence between images is determined by a regularization network from the optical flow of the images. Afterwards, given any particular value for the set of parameters, the authors claim that the network is able to automatically synthesize an image that is an appropriate combination of the many input images.

An interesting point is that the parameters are completely user specified, in the sense that there is no specific meaning associated to them by the algorithm besides the image that is used as an example of a particular parameter value. Suppose the user provides an image of what he/she considers to be a sad expression nd associates it to a parameter value of 0, and similarly a "happy" expression which is associated to 1. The desired semantics is therefore that a parameter of 0.5 should result in a relaxed expression. Although this is an interesting paradigm, and some examples are presented, it still remains to be seen whether this type of automatic interpolation based on AI techniques is sufficiently predictable and reliable.

Chapter 10

Blending Techniques

In the previous chapters, and specially in chapter 6, about morphing of graphical objects, we have already discussed the importance of blending techniques when defining a metamorphosis transformation between graphical objects. In this chapter we will discuss some of the most common blending techniques.

10.1 Linear Interpolation

When two graphical objects are embedded into some euclidean space $\mathbb{R}^n$, it is possible to obtain a blend of them by taking linear interpolation between the points of each object. The linear blending of two points P and Q in $\mathbb{R}^n$ is defined by the equation

$$(1-t)P + tQ = P + t(Q-P).$$

The reader should notive in particular, that when we have two functions $f, g: U \longrightarrow \mathbb{R}^n$, which take values in $\mathbb{R}^n$, their values can linearly interpolated, originating another function $h: U \longrightarrow \mathbb{R}^n$,

$$h(P) += (1-t)f(P) + tg(P).$$

This is very useful when we need to blend object attributes. When the functions f and g represent images, this linear interpolation is classicaly called *cross-dissolve.*

Linear interpolation is a very common blending technique for triangular polyhedral objects. In this case, we just have to interpolate the corresponding vertices of their combinatorial structure, since the geometry of each piece is invariant by the interpolation process.

Consider, for example, two drawings with the same combinatorial structures, and assuming that the intermediate steps will also have these same structures, a blending of their shapes is trivial. Given a reference vertex in each of the drawings, the remaining vertices are sequentially associated, and a linear interpolation is performed between each of these vertex pairs. The choice of the reference vertices is an important factor in producing good results, as shown by the two possibilities in Figure 10.1.

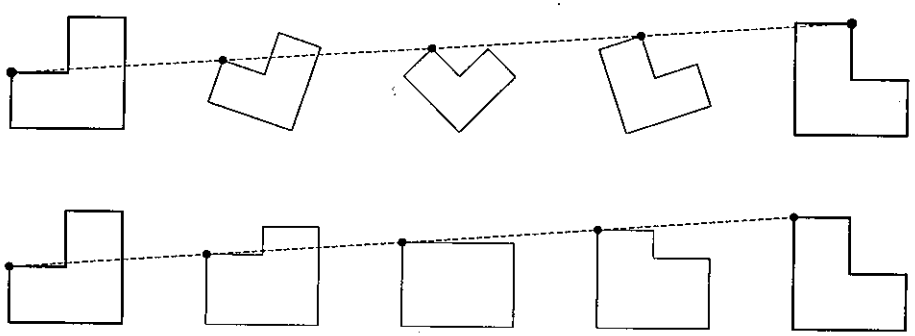


Figure 10.1: Two possible reference vertex associations and the respective blendings.

Naturally, the usual case is to have two input drawings with differing combinatorial structures, and in this case, to create a linear blend, it is necessary to make the structures compatible by inserting new vertices in both structures. As pointed out in (Sederberg and Greenwood, 1992), the main task in 2D shape blending is that of adding such vertices so that the resulting sequential vertex correspondence produces the desired blend. Various heuristics are possible, including a direct homogeneous distribution of vertices proportionally to the number of vertices, used by popular commercial drawing packages (Corel, 1994), or to the perimeter of the outlines.

10.2 Merge of Combinatorial Structures

This technique, developed for use with B-Rep objects, is based on the compatibilization of the structure of combinatorially different objects (Kent, Carlson and Parent, 1992). The idea is to alter the two objects, resulting in two new objects that have exactly the same number and adjacencies of vertices, edges and faces. Once this correspondence is established, the interpolation between the two new objects is straightforward, requiring just the interpolation of geometry between pairs of corresponding vertices. Note that the two input objects must be topologically equivalent and of genus 0, i.e. with no passages through them.

The correspondence algorithm consists in projecting the structure of both models onto the same sphere. Next, the two structures are merged by clipping the projected faces of one model to the projected faces of the other. The merged structure is then mapped onto the surface of both original models. This generates two new models, that have exactly the same shape as the original two models, but that share a common combinatorial structure. These new objects will usually have several coplanar faces. Note that the way the projection is done is fundamental for the success of this technique and can significantly influence the results.

This technique alone, as described in (Kent, Carlson and Parent, 1992), simply attempts to make the structures compatible, with no intent of mapping parts of one object into similar parts of the other. This is useful for objects that do not have such similar parts, or if the similar parts are geometrically aligned beforehand by a warping phase. In fact, the combination of this blending technique with a warping step results in an efficient and viable technique for morphing B-Rep objects.

This technique can be naturally extended to 2D approximate boundary models, or drawings. The 2D polygonal shapes are projected onto a circle, the two structures are merged and subsequently mapped back onto the original shapes. This results in two objects with equivalent structures, but maintaining the geometry of the originals. A 2D example, that is also helpful in understanding the 3D case, is shown in Figure 10.2. This technique is compared with other 2D shape blending methods in the next section.

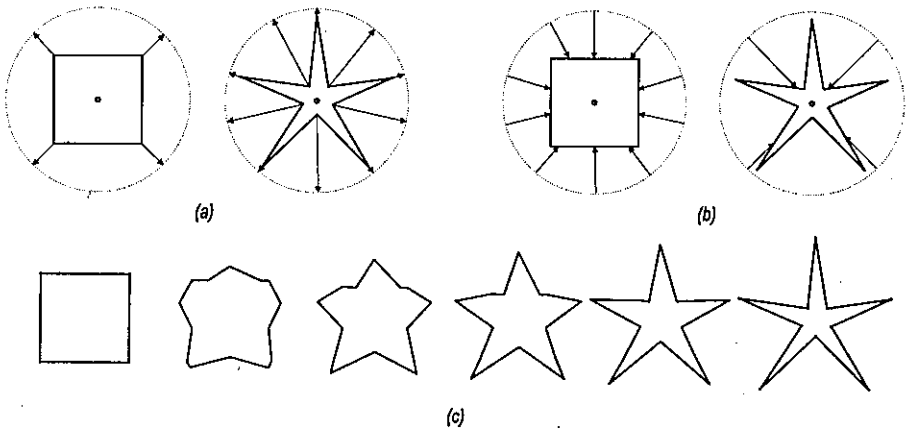


Figure 10.2: A merge of two combinatorial structures: (a) projection; (b) merge; (c) interpolation.

10.3 Physically Based Blending

This technique is in fact an elaborate heuristic that selects the positions for inserting vertices so that the resulting vertex correspondence produces a blend that minimizes work, according to an appropriate energy model (Sederberg and Greenwood, 1992). The 2D polygonal drawings are considered to be constructed of an ideal metal wire, so that the work for bending and stretching this wire can be computed. The best blend is considered to be the one which requires the minimum work to transform one shape into another, by bending and stretching the wire.

Given this work metric, the problem is now that of optimizing the positions of the inserted points so that a minimum work is executed. Obviously, this problem as posed is of a high complexity, but the restriction to insert new vertices only at the position of existing vertices, proposed in (Sederberg and Greenwood, 1992), reduces the complexity of the problem to $\emptyset(mn)$, where m and n are the number of vertices of each drawing.

An interesting consequence of the minimization of work performed is

that matching features in the two drawings tend to be automatically associated. If there is a feature of one object that matches a feature of the other object, and these features are positioned relatively close, the transformation of one feature into the other requires a very small amount of work. In particular, features that do not move do not require any work to be transformed, and therefore do not move during the blending. This is shown in the example in Figure 10.3, where the feet remain static, while the rest is transformed. For features that are relatively distant, however, the algorithm tends to confuse characteristics, producing unnatural results. In this case, a warping phase would be extremely favorable, by bringing the matching features to a closer position.

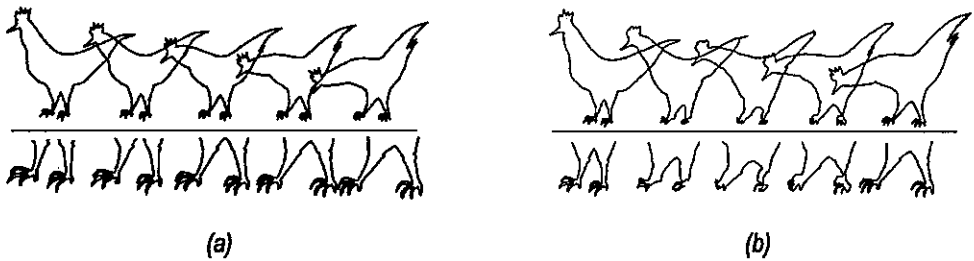


Figure 10.3: (a) Physically based blending; (b) Commercial Software (Sederberg and Greenwood, 1992).

Note that, although the metric proposed by Sederberg and Greenwood can give excellent results, the fact that work is a signed quantity, that can be canceled during the transformation, may lead to peculiar globally optimal solutions. Consider, for example, the top transformation in Figure 10.4, where rotation, shrinking and stretching are performed and, paradoxically, no work is done. With this metric, this transformation is obviously optimal, since it performs the blending with the least work possible. This is due to the cancellation of the shrinking with the stretching, and the rotational work that is not considered in the metric.

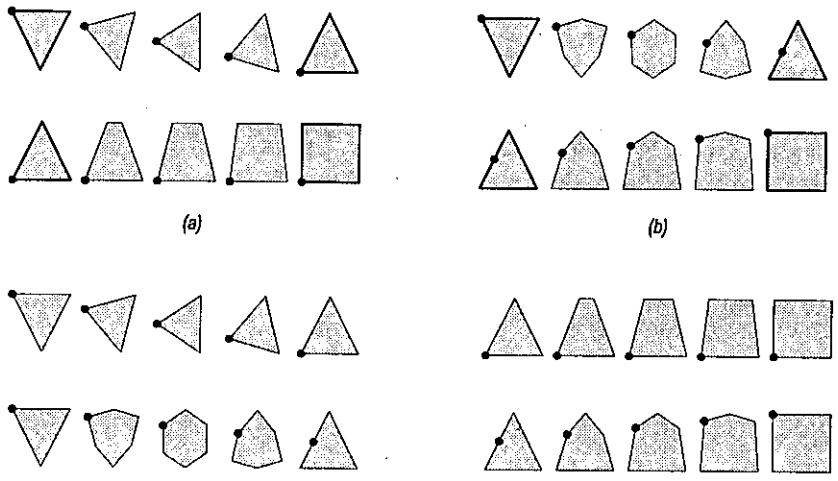


Figure 10.4: Comparison of 2D shape blendings: (a) physically based; (b) structure merging.

The search for better metrics is an important goal in physically based blending, since the technique for the solution of the optimization problem remains unchanged. Among the possible improvements of this heuristic, are the consideration of the total sum of the absolute amount of work required, the peak of work performed during the transformation and the total rotational work.

This technique is obviously directed to objects represented with polygonal boundaries, although an extension to deal with digital images is conceivable. This can be accomplished by first obtaining a boundary description of the regions of the underlying image, either directly user specified or through the use of an edge detection algorithm. These outlines are then transformed performing the least work possible. What remains is the choice of an appropriate mapping for the inside region of the outlines.

The 2D merging of combinatorial structures is in fact a technique with essentially the same approach as the 2D physically based blending: the

aim is to choose where new vertices should be added to each object so that the structures are equivalent and a linear blend produces good results. Of course, the physically based blending is a much more sophisticated method, with excellent results that include the matching of corresponding features if their geometries are relatively similar.

The 2D merging technique, on the other hand, does not manage to match the features, producing good results only if the input objects do not have a natural feature correspondence, as the examples given in (Kent, Carlson and Parent, 1992) and in Figure 10.4.

For morphing applications, feature matching is an essential perceptual goal, making physically based blending a much more advantageous alternative, although a warping step can minimize the importance of feature matching in blending.

10.4 Implicit Objects Blending

Blending operations are very well suited for implicitly defined objects, producing a third implicit object that is a smooth transition between two intersecting surfaces. The study of the blending functions used to combine the characteristic functions describing the objects is a relevant area of research (Rockwood and Owen, 1987). A blending function is a function $B(f1, f2, ..., fn)$ that takes two or more characteristic functions as parameters and is itself described implicitly.

For application in morphing, two forms of blending are specially useful: *exponential* (Blinn, 1982) and *super-elliptic* (Rockwood, 1989). The exponetial blend produces excellent results when the characteristic functions are algebraic, and is described, for two inputs, by a one-parameter family of functions

$$B_t(x, y) = -\log[(1-t)e^{-x} + te^{-y}].$$

The blended object is described by a characteristic function that is obtained simply by the composition of B with f and g, i.e., $B(f, g)$.

Similarly, the super-elliptic blend, for two implicit objects f and g, is

given by the function

$$B(f,g) = 1 - ((\max\{0, 1 - \alpha_1 f\})^\beta + (\max\{0, 1 - \alpha_2 g\})^\beta)^{\frac{1}{\beta}}.$$

The parameters α and β are used to change the shape of the blend: α_i bias the blend towards the primitive i, and β controls the overall tightness of the blend.

A general principle is that the shapes of the input objects—given by the zero level surface of implicit point classification functions—should be blended so that the result is a function where the zero level surface is a smooth transition between the zero level surfaces of the input functions, regardless of the particular function values. Figure 10.5 makes this distinction more clear, showing different blends of two functions that could be interpreted as point classification functions.

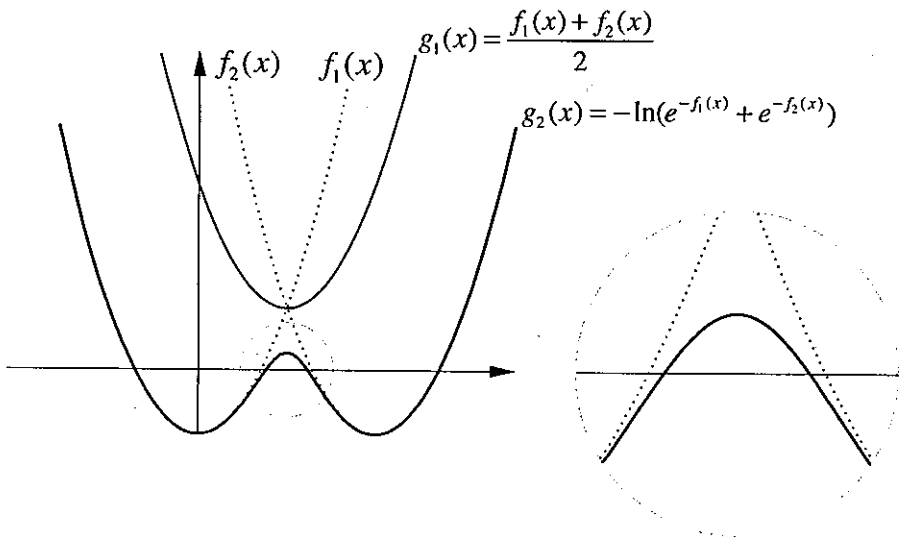


Figure 10.5: Different blendings between two parabolas.

We should remark that while the exponential blend results in a natural smooth combination of the level surfaces of the two parabolas, the

average blend generates a function with a null zero level surface. This result clearly shows that the suitability of a blending technique depends on the interpretation of the blended data.

10.5 Frequency Domain Blending

This general technique consists applying some transform T to obtain a frequency domain description of the graphical object, perform the blending between the objects in the frequency domain, and reconstruct the blended object using the inverse transform T^{-1}.

10.5.1 Scheduled Fourier Volume Blending

Two implicit objects, described by point membership classification functions, can be easily combined into a new implicit object given by an average of the two input functions. Such simple combination is a form of linear blending, with results that are generally not suitable for good quality combination of solids.

An interesting alternative is the blending of the objects in the frequency rather than the spatial domain (Hughes, 1992). This is easily accomplished by applying a Fourier transform to the objects, combining their transformed versions with a linear blend, and then bringing them back to the spatial domain with an inverse Fourier transform.

Once in the frequency domain, it is possible to extract more easily other forms of information on the shape of the objects. High frequency components are responsible for small details on the object surface, while low frequencies describe the overall shape of the object. In this way, a *scheduled* transformation, where the different frequency bands are transformed at different rates, would result in a better transformation: the higher frequencies of the source object are gradually removed, then the low frequencies are interpolated, and finally the high frequencies of the destination object are introduced.

Note that this technique can be easily used on sampled volumetric data with a Fast Fourier Transform, and its use with implicitly defined objects is also possible. However, there is a very restricted control over the transformation in the form of a schedule specification, which is not

sufficient to yield suitable quality results.

10.5.2 Wavelet Volume Blending

Other approaches in the frequency domain are possible. The use of wavelets is the most noticeable improvement in this area (He, Wang and Kaufman, 1994). Volumetrically represented objects are decomposed into sets of frequency bands, which are then smoothly interpolated. The reconstruction of the interpolated bands yields the final morphed object. Figure 10.6, from (He, Wang and Kaufman, 1994), shows a morphing sequence created by this algorithm.

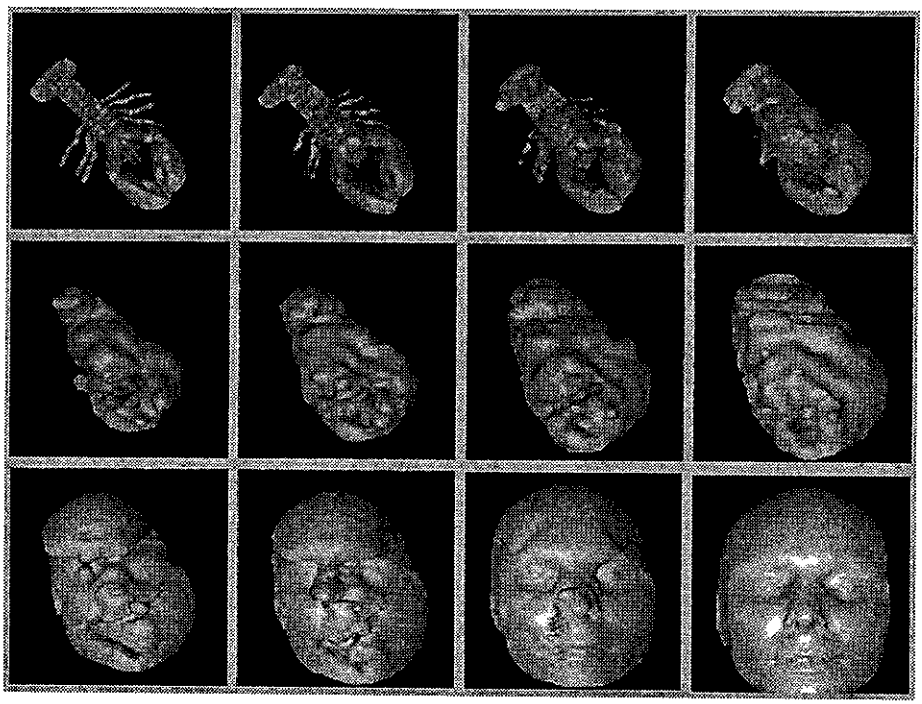


Figure 10.6: Volumetric wavelet morphing.

The main advantage of this approach is that wavelets possess good

localization properties on the frequency-scale space. Also, the decomposition and reconstruction processes in this case can be performed in a multiresolution form, which allows the control of the level of high frequency distortion.

References

Arad, N. and Reisfeld, D. (1995). Image warping using few anchor points and radial functions. *Computer Graphics Forum*, 14(1):35–46.

Avila, R., He, T., Hong, L., Kaufman, A., anspeter Pfister, H., Silva, C., Sobierajski, L., and Wang, S. (1994). Volvis: A diversified volume visualization system. In Bergeron, D. and Kaufman, A., editors, *Proceedings of Visualization '94*.

Barr, A. H. (1984). Global and local deformations of solid primitives. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):21–30.

Baumgart, B. (1975). A polyhedron representation for computer vision. *AFIPS Conf. Proc.*, pages 589–596.

Beier, T. and Neely, S. (1992). Feature-based image metamorphosis. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):35–42.

Beymer, D., Shashua, A., and Poggio, T. (1993). Example based image analysis and synthesis. *M.I.T. Artificial Intelligence Laboratory, A.I. Memo 1431.*

Birkhoff, G. and Maclane, S. (1960). *A Survey of Modern Algebra.* Macmillan Company, New York.

Blinn, J. F. (1982). A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256.

Blum, L. (1991). *Lectures on a Theory of Computation over the Reals.* Instituto de Matemática Pura e Aplicada - IMPA, Rio de Janeiro.

Carvalho, P. C. P., Gomes, J., and Velho, L. (1993). Space decompositions: Theory and techniques. Preprint, Instituto de Matemática Pura e Aplicada - IMPA.

Catmull, E. (1974). Subdivision algorithm for the display of curves surfaces. Phd thesis, University of Utah.

Chen, S. E. and Parent, R. E. (1989). Shape averaging and its applications to industrial design. *IEEE Computer Graphics and Applications*, 9(1):47–54.

Corel (1994). *CorelDraw! 4.0.* Corel Systems Corporation.

Costa, B. (1994). *Image Deformation and Metamorphosis. Master's Thesis.* Computer Science Department. PUC-Rio, Rio de Janeiro.

Costa, B., Darsa, L., and Gomes, J. (1992). Image metamorphosis. In Gomes, J. and Câmara, G., editors, *SIBGRAPI '92 Proceedings*, pages 19–27.

Darsa, L. (1994). *Graphical Objects Metamorphosis. Master's Thesis.* Computer Science Department. PUC-Rio, Rio de Janeiro.

Darsa, L. and Costa, B. (1995). Field-based separable volumetric warping (preprint). Technical report, Department of Computer Science, SUNY at Stony Brook.

de Figueiredo, L. H. (1992). *Computational Morphology of Implicit Curves.* PhD thesis, impa - Instituto de Matemática Pura e Aplicada, Brasil.

Farin, G. (1988). *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, San Diego, CA.

Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1990). *Computer Graphics: Principles and Practices (2nd Edition).* Addison Wesley.

Gomes, J., Costa, B., Darsa, L., and Velho, L. (1994). Graphical objects. Preprint, Instituto de Matemática Pura e Aplicada - IMPA.

Gomes, J., Hoffmann, C., Shapiro, V., and Velho, L. (1993). Modeling in graphics. SIGGRAPH '93 Course Notes.

Gomes, J. and Velho, L. (1992). *Implicit Objects in Computer Graphics.* Instituto de Matemática Pura e Aplicada, IMPA, Rio de Janeiro, Brazil.

Gomes, J. and Velho, L. (1995a). Abstraction paradigms for computer graphics. *The Visual Computer*, 11(5):227–239.

Gomes, J. and Velho, L. (1995b). *Computação Gráfica: Imagem.* Série Computação e Matemáatica. Publicação IMPA-SBM (in portuguese).

Gonzalez, R. C. and Wintz, P. (1987). *Digital Image Processing (2nd Edition).* Addison-Wesley, Reading, MA.

He, T., Wang, S., and Kaufman, A. (1994). Wavelet-based volume morphing. In Bergeron, D. and Kaufman, A., editors, *Proceedings of Visualization '94*, pages 85–91.

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

Hoffmann, C. (1989). *Geometric and Solid Modeling: an Introduction.* Morgan Kaufmann.

Hughes, J. F. (1992). Scheduled fourier volume morphing. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):43–46.

Kajiya, J. T. and Kay, T. L. (1989). Rendering fur with three dimensional textures. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):271–280.

Kaufman, A. (1994). Voxels as a computational representation of geometry. in *The Computational Representation of Geometry.* SIGGRAPH '94 Course Notes.

Kent, J. R., Carlson, W. E., and Parent, R. E. (1992). Shape transformation for polyhedral objects. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):47–54.

Lasseter, J. (1987). Principles of traditional animation applied to 3d computer animation. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):35–44.

Lee, S.-Y., Chwa, K.-Y., Hahn, . J., and Shin, S. Y. (1994). Image morphing using deformable surfaces. *Proceedings of Computer Animation '94 (Geneva, Switzerland).*

Lee, S.-Y., Chwa, K.-Y., Hahn, J., and Shin, S. Y. (1995a). Image morphing using deformation techniques. *Journal of Visualization and Computer Animation*, 6(3).

Lee, S.-Y., Chwa, K.-Y., Shin, S. Y., and Wolberg, G. (1995b). Image metamorphosis using snakes and free-form deformations. *Computer Graphics (Proceedings Siggraph '95).*

Lerios, A., Garfinkle, C. D., and Levoy, M. (1995). Feature-based volume metamorphosis. *Computer Graphics (Proceedings of SIGGRAPH '95).*

Litwinowicz, P. and Miller, G. (1994). Efficient techniques for interactive texture placement. *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 119–122.

Litwinowicz, P. and Williams, L. (1994). Animating images with drawings. *Computer Graphics (Proceedings of SIGGRAPH '94)*, 24:409–412.

Naylor, B. (1994). Computational representations of geometry. in *The Computational Representation of Geometry.* SIGGRAPH '94 Course Notes.

Nishita, T., Fujii, T., and Nakamae, E. (1993). Metamorphosis using bezier clipping. In *Proc. of First Pacific Conference on Computer Graphics and Applications (Seoul, Korea).* World Scientific Publishing Co.

Perlin, K. and Hoffert, E. M. (1989). Hypertexture. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):253–262.

Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY.

Prusinkiewicz, P., Hammel, M. S., and Mjolsness, E. (1993). Animation of plant development. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27:351–360.

Radha, H. S. (1993). Efficient image representation using binary space partitioning trees. Phd dissertation, cu/ctr/tr 343-93-23, Columbia University.

Requicha, A. A. G. (1980). Representations for rigid solids: Theory methods, and systems. *ACM Computing Surveys*, 12:437–464.

Rockwood, A. and Owen, J. (1987). Blending surfaces in solid modeling. In Farin, G., editor, *Geometric Modeling: Algorithms and New Trends*, pages 367–383. SIAM, Philadelphia.

Rockwood, A. P. (1989). The displacement method for implicit blending surfaces in solid models. *ACM Transactions on Graphics*, 8(4):279–297.

Sclaroff, S. and Pentland, A. (1994). Object recognition and categorization using modal matching. *Proceedings of 2nd IEEE CAD-Based Vision Workshop*, pages 258–265.

Sederberg, T. and Parry, S. (1986). Free-form deformation of solid geometric models. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):151–160.

Sederberg, T. W., Gao, P., Wang, G., and Mu, H. (1993). 2D shape blending: An intrinsic solution to the vertex path problem. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27:15–18.

Sederberg, T. W. and Greenwood, E. (1992). A physically based approach to 2-d shape blending. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):25–34.

Shannon, C. E. (1949). *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, Illinois.

Sims, K. (1991). Artificial evolution for computer graphics. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):319–328.

Smythe, D. B. (1990). A two-pass mesh warping algorithm for object transformation and image interpolation. Technical memo #1030, Industrial Light and Magic.

Thompson, D. W. (1961). *On Growth and Form.* Cambridge University Press.

Watson, B. A. and Hodges, L. F. (1995). Using texture maps to correct for optical distortion in head-mounted displays. In Bryson, S. and Feiner, S., editors, *Proceedings of IEEE Virtual Reality Annual International Symposi um '95*, pages 172–178.

Winkenbach, G. and Salesin, D. H. (1994). Computer-generated pen-and-ink illustration. *Computer Graphics (Proceedings of SIGGRAPH '94)*, pages 91–100.

Wolberg, G. (1989). Skeleton-based image warping. *The Visual Computer*, 5(1/2):95–108.

Wolberg, G. (1990). *Digital Image Warping.* IEEE Computer Society Press, Los Alamitos, CA.

Wyvill, B. (1994). *Building and Animating implicit surface models.* in SIGGRAPH '93 Course Notes.