

A System's Architecture for Warping and Morphing of Graphical Objects

JONAS GOMES,[†] LUCIA DARSA,^{††} BRUNO COSTA,^{†††} LUIZ VELHO[†]

[†]IMPA – Instituto de Matemática Pura e Aplicada,
Estrada Dona Castorina, 110, 22460-320 Rio de Janeiro, RJ, Brazil,

[‡]Equator Technologies Inc.
520 Pike St. Suite 900, Seattle, WA 98101-4001,

^{††}Microsoft Corporation
One Microsoft Way, Redmond, WA 98052-6399,

Abstract. This paper describes a system architecture that enables the use of different techniques of warping and morphing applied to various classes of graphical objects (images, surfaces, plane curves, volumetric data, implicit surfaces etc.) Combinations of warping and morphing techniques can lead to more powerful and expressive results. The possibility of experimenting with alternative techniques for distinct classes of graphical objects creates a very flexible and robust warping and morphing system.

Keywords: Warping, Morphing, Interaction, Graphics System.

1 Introduction

Metamorphosis is a fundamental characteristic of the constant evolution of nature. In computer graphics, the presence of warping and morphing is ubiquitous, ranging from the projective warping of the camera transformation to more complex transformations of shape and motion.

These operations encompass a broad range of applications in different areas, including shape design, rendering, image and volumetric data registration, and special effects for the video, television and the movie industry. In fact, warping and morphing constitute an intrinsic part of a rendering or animation system.

1.1 Previous Work

The importance of warping and morphing is reflected in the large number of publications from different research fields. The literature covers a wide range of warping and morphing techniques for different graphical objects.

Image warping has been addressed in computer graphics since the original work of Ed Catmull on texture mapping. Since then, research in this area has constantly increased. An overview of the field can be found in [4]. Several techniques have appeared in the literature. Also, image warping techniques have been studied separately related with the problem of image registration, in the areas of photogrammetry, and medical images.

Curve warping and morphing have been studied in [3]. This topic has a wide range of applications, specially in the areas of shape design and animation.

Surface warping techniques date back to the beginning of the use of free-form surfaces for surface design. In fact changing the control parameters is a flexible form of defining intrinsic warps of surfaces. There are several papers on warping and morphing of surfaces; for a survey see [1].

Volume metamorphosis has also been studied. This topic has applications in the area of medical images.

There is a number of commercial software available to

compute warpings and morphings of graphical objects. These packages have a high degree of functionality, but they are targeted at specific classes of graphical objects. There exists commercial graphical packages that offer solutions for warping and morphing of images as well as for surfaces, but they are usually not integrated.

1.2 Contribution

Certainly, the overview of the field cited above represents only a percentage of the work that has been published concerning warping and morphing. A few remarks about the existing literature are in order:

1. Morphing and warping research has covered particular classes of graphical objects.
2. There has been no attempt to obtain an integrated approach to warping and morphing;
3. There exists no flexible software architecture that enables us to experiment with warping and morphing of distinct graphical objects, using different techniques.

Certainly, problems 1 and 2 above are fundamental in addressing problem 3. Problems 1 and 2 have been addressed in [5] (see also [4]). This paper focuses on problem 3. More precisely, we describe a software architecture we have developed which has the following characteristics:

1. Allows for warping and morphing of different graphical objects (images, surfaces, plane curves and volume data);
2. Allows the use of different warping and morphing techniques, which are shared among various types of graphical objects;
3. Allows the inclusion of new graphical objects in the system;

4. Allows the development of plug-ins using different warping and morphing techniques;
5. Provides a uniform and coherent interface, across different graphical objects and techniques.

1.3 Structure of the Paper

The structure of the paper is as follows: Section 2 discusses the concept of graphical objects, and warping and morphing operations; Section 3 gives an overall description of the system's architecture; sections 4, 5 and 6 describe in detail the architecture; section 8 concludes the paper summarizing the results and describing future work.

2 Warping and Morphing

In this section we will bring into the stage the two major actors in the study of warping and morphing: graphical objects and transformations.

2.1 Graphical Objects

A *graphical object* [5] is characterized by its shape and attributes. The shape describes the topology and geometry of the graphical object, and the attributes carry information about its different properties: color, material, etc.

Mathematically, the shape of a graphical object is a subset $U \subset \mathbb{R}^n$ of some n -dimensional euclidean space, and the attributes are defined by a function $f : U \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ (we are encapsulating all of the attributes in one function). A graphical object $\mathcal{O}$ is denoted by $\mathcal{O} = \mathcal{O}(U, f)$. The dimension of U defines the *dimension* of the graphical object. $\mathbb{R}^n$ is the space where the object is embedded.

The most common examples are:

- A *curve* is a 1-dimensional graphical object embedded in $\mathbb{R}^n$. For $n = 2$, we have a *plane curve*.
- A *planar region* is a 2-dimensional graphical object embedded in the plane, typically with a constant attribute color;
- An *image* is a 2-dimensional graphical object embedded in the plane. A planar region corresponds to a binary image.
- A *surface* is a 2-dimensional object embedded in $\mathbb{R}^3$;
- A *solid* is a 3D graphical object embedded in $\mathbb{R}^3$.

Some graphical objects (e.g. an image) have a very simple shape and complex attributes, while other objects (e.g. a surface) might have complicated shapes and simple attributes.

2.1.1 Computing with Graphical Objects

The concept of a graphical object allows us to obtain a unified computational pipeline. In fact, when dealing with graphical objects, we are faced with the following problems:

- Description of graphical objects;
- Representation of graphical objects;
- Reconstruction of graphical objects.

The description of graphical objects accounts for the creation of objects, and it is closely related to the user interface. The representation of graphical objects accounts for the discretization of the shape and attributes. We should emphasize the importance of representation and reconstruction techniques. In general, the user describes a discrete object that has to be reconstructed in order to perform operations with them. On the other hand, continuous graphical objects must be discretized to obtain a computational representation. The interplay between reconstruction and discretization techniques constitute a fundamental problem when dealing with graphical objects. The computational framework for graphical objects is shown in Figure 1.

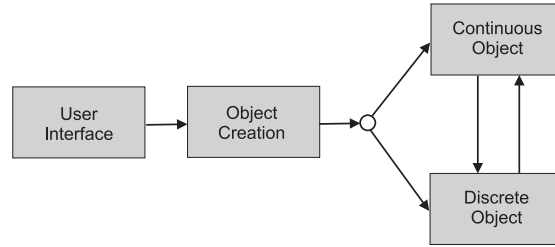


Figure 1: Graphical objects computational pipeline.

In general, a graphical object is described by functions (either parametrically or implicitly). On the other hand, the attribute is also defined by a function. Therefore, in the computational pipeline of Figure 1, we are dealing with functions. That is, we must devise techniques and interfaces to describe functions, sample and reconstruct them. This unified view is one of the contributions of the concept of graphical objects.

2.2 Transformations of Graphical Objects

A transformation of a set $U \subset \mathbb{R}^n$ is a function $T : U \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$. When $T = L|_U$ is the restriction of a function $L : \mathbb{R}^n \rightarrow \mathbb{R}^n$, defined on the whole space, we say that T is a *global* or *extrinsic* transformation. Otherwise, T is called an *intrinsic* transformation. T is called a *local* transformation if for any point $p \in U$, T does not move points outside a small neighborhood of p .

A transformation of a graphical object $\mathcal{O} = \mathcal{O}(U, f_1)$ is achieved by transforming both the shape U and the attribute function f_1 , as illustrated in Figure 2: $T(\mathcal{O}(U, f_1)) = \mathcal{O}(T(U), f_2)$, where $f_2(T(p)) = f_1(p)$.

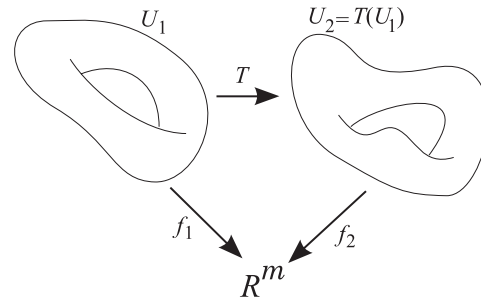


Figure 2: Graphical object transformation.

2.2.1 Computing with Transformations

The computational pipeline for dealing with transformations is shown in Figure 3. This pipeline is very natural:

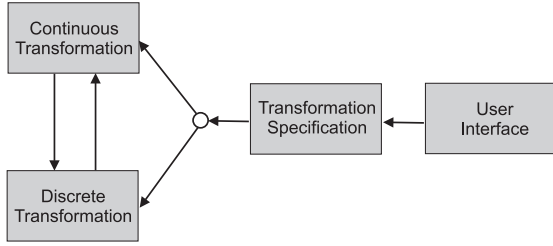


Figure 3: Transformations computational pipeline.

- Devise techniques to specify transformations. This is closely related to the user interface.
- Devise techniques to discretize transformations.
- Devise techniques to reconstruct transformations.

We should notice that the specification of a transformation by the user either produces a continuous transformation or a discretized one. The techniques of discretization and reconstruction play a very important role when computing with transformations.

The reader may have noticed the similarity between the computational pipeline in Figures 1 and 3. The reason is that the natural habitat of both graphical objects and transformations is a space of functions. Therefore, those computational pipelines are indeed computational pipelines for functions.

2.3 Warping and Morphing Transformations

In order to define warping and morphing of a graphical objects, we need to introduce the concept of family of transformations.

A continuous k -parameter family of transformations of a graphical object $\mathcal{O} = \mathcal{O}(U, f)$, $U \subset \mathbb{R}^n$, is a continuous transformation $T : \mathbb{R}^k \times \mathcal{O} \rightarrow \Omega$, where Ω is a space of graphical objects. Thus, for each $v \in \mathbb{R}^k$, we obtain a transformation $T_v(\mathcal{O}) = T(v, \mathcal{O})$ of the graphical object $\mathcal{O}$. As v varies on the parameter space $\mathbb{R}^k$, the object changes continuously.

Suppose that $T_{v_0}(\mathcal{O}) = \mathcal{O}$ for some $v_0 \in \mathbb{R}^k$, and take a continuous path $\alpha : [0, 1] \rightarrow \mathbb{R}^k$, such that $\alpha(0) = v_0$. We obtain a 1-parameter family of transformations $\bar{T} : [0, 1] \times \mathcal{O} \rightarrow \Omega$, defined by $\bar{T}(t, \mathcal{O}) = T(\alpha(t), \mathcal{O})$. By interpreting t as time, this family represents an animation that describes a continuous deformation of the object $\mathcal{O}$. This is called a *warping* of the object $\mathcal{O}$.

Consider a continuous k -parameter family of transformation $T : \mathbb{R}^k \times \Omega \rightarrow \Omega$, defined on some space of graphical objects Ω . Let $v_0, v_1 \in \mathbb{R}^k$ and $\mathcal{O}_1, \mathcal{O}_2 \in \Omega$ such that $T(v_0, \mathcal{O}_1) = \mathcal{O}_1$ and $T(v_1, \mathcal{O}_1) = \mathcal{O}_2$. If $\alpha : [0, 1] \rightarrow \mathbb{R}^k$ is a continuous curve satisfying $\alpha(0) = v_0$ and $\alpha(1) = v_1$, the 1-parameter family $\bar{T} : [0, 1] \times \mathcal{O}_1 \rightarrow \Omega$ defined by $\bar{T}(t, \mathcal{O}_1) = T(\alpha(t), \mathcal{O}_1)$ is a *metamorphosis* (or a *morphing*) from $\mathcal{O}_1$ to $\mathcal{O}_2$. This metamorphosis is an animation that produces a continuous deformation from object $\mathcal{O}_1$ to object $\mathcal{O}_2$. $\mathcal{O}_1$ is called the *source object* and $\mathcal{O}_2$ is called the *target object*.

For each point $p \in \mathcal{O}_1$, the curve $\gamma(t) = \bar{T}(t, p)$, $t \in [0, 1]$ describes a path in $\mathbb{R}^n$ from p to $q = T(1, p) \in \mathcal{O}_2$.

This is called an *animation path* of the point p . Notice that the knowledge of the animation path for all points of the source graphical object characterizes the morphing completely.

The operations of warping and morphing result from a clever coupling of the concepts and techniques used to study graphical objects and their transformations. The computational framework for the operations of warping and morphing is shown in the diagram of Figure 4. This diagram is a merging of the diagrams of the computational pipeline for graphical objects (Fig. 1) with the computational pipeline for transformations (Fig. 3). The symmetry of the diagram well illustrates the interplay between graphical objects and transformations to accomplish the operations of warping and morphing.

According to the diagram in Figure 4, a warping of a graphical object is obtained in three major steps:

- Creation of the graphical objects;
- Specification of the transformation family;
- Computation of the transformation applied to the graphical object.

In the next section, we will see how this conceptual pipeline provides the elements for a computational framework.

2.4 Computational Elements

The design of a computational framework for the warping and morphing system can be based on six fundamental elements:

- Graphical object creation/representation;
- Transformation specification/representation;
- Warping reconstruction;
- Mapped object computation;
- Shape combination;
- Attribute combination.

Graphical Object Creation/Representation. The representation of a graphical object associates a data structure to a discretization of the object, encapsulating the description of its shape and attributes. A typical application may handle only a specific type of graphical object, for example, an image represented by a matrix of pixel values. The representation is closely related with the specification: in general the user specifies a representation of the object.

Transformation Specification/Representation. The specification of the transformation consists in a discrete representation of the transformation, that is usually manipulated by the user. In a typical situation, the user specifies values of the warp at a discrete set of points at initial and final states. In some key-frame systems, these values are specified at different intermediate key states.

Warping Reconstruction. The warping reconstruction uses the discrete representation of the transformation, computing the transformation values at any point of the spatial and time domains of interest. This includes interpolating the key states in time, and extending the transformation values to all points of the object shape. This is equivalent to computing a state of the warp along the animation paths for different time values.

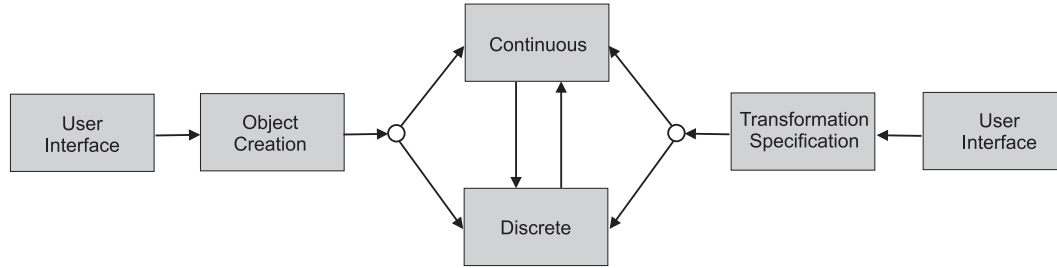


Figure 4: Computational pipeline for warping and morphing.

Mapped Object Computation. This step of the technique enumerates the elements of the representation of the graphical object, traversing its structure. This is necessary in order to apply the warping to the object. The computation of the mapped object is closely related to the reconstruction of the transformation, and in some cases both elements are intermixed in actual implementations.

Shape Combination. This operation is, in general, necessary because we do not have a perfect alignment between the geometry of the two objects. When computing an image metamorphosis the shape combination in general is not present because we have perfect alignment of the image boundaries (also all of the emphasis on an image morphing is in the pixel colors). Nevertheless, shape combination is of fundamental importance when computing morphing of other graphical objects such as curves and surfaces. When the objects have different topologies, it is necessary to use topological surgery in order to proceed the shape combination operation.

Attribute Combination. The operation of combining attributes is used in the morphing transformation to compute the resulting attribute function from the attributes of the source and target graphical objects. The method of combination will vary according to the attribute nature. A typical example is a linear cross-dissolve performed on a pixel-by-pixel basis between two images.

A *morphing technique* is a combination of the six above elements, but the essence of the methods is contained in the specification and reconstruction of the transformation and the graphical objects.

The most common specification techniques are based on points, oriented line segments or meshes. In general, the problem of reconstructing the transformation reduces to the computation using scattered data interpolation techniques. Some of them are affine interpolation, inverse distance weighted functions, radial basis functions and thin-plates splines.

Two well-known image morphing methods are [2] and [7]. In [2], the described morphing method uses oriented line segment specification, field-based reconstruction (inverse distance weighted interpolation) and cross-dissolve for color combination. In [7], a two-pass warping is specified with spline meshes. The warping reconstruction and the mapped object computation in this case are not completely separated. The technique reconstructs the transformation for each scan-line and scan-column, in two consecutive passes. For a description of different warping and morphing techniques the reader should consult [4].

3 System's Architecture

We will now give a description of an architecture of a general morphing system, which addresses the problems stated in section 1.2.

A morphing system has to manage the computational elements of warping and morphing described in section 2.4, and orchestrate their interactions. Although these elements are not necessarily clearly identified and isolated in a given implementation, this separation can bring several benefits to the design and functionality of a system. As an example, the separation of the elements in a system makes it easy to experiment with hybrid techniques. Nevertheless, not all combinations are possible, and the set of specification techniques that suits a particular warping reconstruction technique has to be limited by the system.

One of the primary objectives of the morphing system we developed is to serve as a testbed for experimentation. To attain this, the system design tried to accomplish for the following features:

- extensibility;
- portability;
- flexibility.

Extensibility provides entry points to make the addition of new implementations of computational elements, such as graphical objects and warping reconstruction techniques, an easy task. *Portability* allows the system to run on various platforms with little implementation effort. *Flexibility* has the purpose of making it easy to use the system with different types of graphical objects and warping and morphing techniques. Accomplishing these goals is a very difficult task, specially the last one, because of the rich diversity of existing graphical objects (image, curves, surfaces, volumetric objects etc.) and techniques to warp and morph them.

3.1 System Components

The architecture of the system is based on a separation into three different levels: *kernel*, *platform* and *support* levels, as shown in the diagram in Figure 5.

Kernel level. This constitutes the core of the warping and morphing system. It is responsible for managing all the computational elements from section 2.4. Each type of element will be implemented in several different ways, corresponding to different techniques for performing the computation relative to that element. This level should be independent of platform.

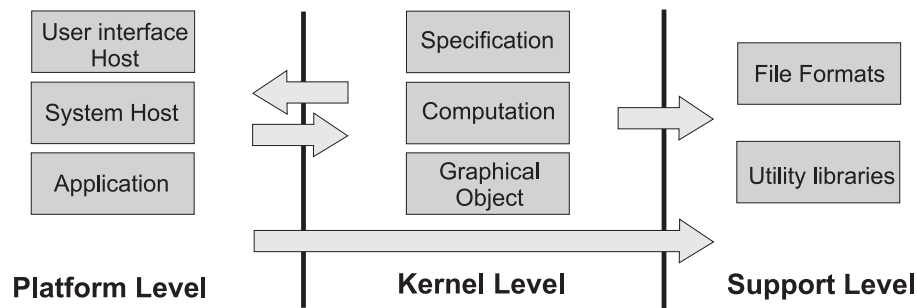


Figure 5: Morphing system architecture

Support level. This module is responsible for the utility libraries, such as container classes, safe object sharing mechanisms, geometry and math libraries. It is also responsible for the input and output of graphical objects, supporting various standard and custom file formats for each type of graphical object. Portability of this level is one of the goals in the implementation of the system.

Platform level. This module is responsible for the entire interface with the operating system. This includes in particular user interface and memory management. This is the level where all the platform dependencies are concentrated.

Figure 5 also shows the internal structure of each block of the system architecture. Each of these blocks will ultimately correspond to classes. In the following sections, we will give more details about each of the system levels, and their components.

4 Kernel Level

The kernel level performs the operations necessary to compute warplings and morphings of graphical objects. In this level, the implementation notably benefits from the warping and morphing concepts introduced in section 2.

The kernel level is basically divided in three parts, which encompass all the computational elements:

- **graphical objects:** graphical object representation;
- **specification of transformations:** transformation specification and representation;
- **computation of transformations:** warping reconstruction, mapped object computation, shape combination and attribute combination.

Each of these parts will be controlled by a *manager*, which is responsible for keeping track of all the available computational element implementations, or *techniques*. It also defines the basic interface for each element, which provides all the operations and information needed for using those elements in computations, so that they are managed independently of their implementation.

4.1 Graphical Objects

The implementation of graphical objects naturally follows object oriented concepts, with each type of graphical object being derived from a generic graphical object. The object interface

will have to adapt to the graphical object class, and to the dimension of the space the graphical object is embedded.

The graphical objects classes are organized in a hierarchy of abstract data types, starting from a very generic root level. A simplified example of such a hierarchy is represented in Figure 6. Each technique that manipulates graphical objects will attempt to plug into this hierarchy at the highest level possible, so that it encompasses the largest group of graphical object types. For instance, the inverse mapping object computation technique can be applied to any implicit discrete graphical object, so this technique should be implemented at this level, and not for some specific class of graphical objects, such as images.

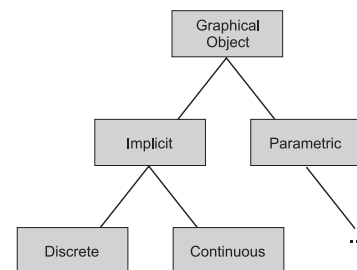


Figure 6: Simplified graphical object hierarchy.

4.2 Specification of Transformations

The specification of a warping or morphing has a strong relationship with the user interface, as the transformations are usually specified graphically through the use of geometric primitives in two states, resulting in a discrete representation of the transformation.

4.2.1 Manipulators

All the specification techniques in our system are built upon basic interface building blocks, which we call *manipulators*. They are high-level geometric and interactive entities, similar in essence to *widgets*. Manipulators provide both the semantics and the appearance of the components that allow the handling and modification of the specification of a transformation. Manipulators can be either 2D or 3D, depending on the dimension

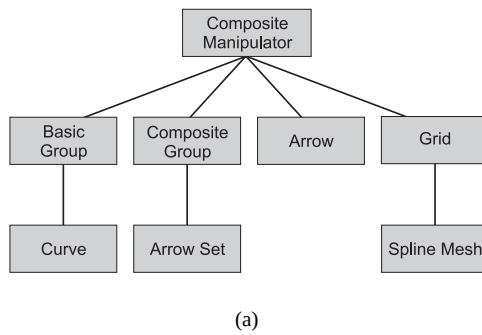


Figure 7: Composite manipulator hierarchy.

of the space where the graphical object is embedded. The implementation of the manipulators is portable, using the high-level input and output graphics primitives from the platform level.

Manipulators can be of two types: basic and composite. Basic manipulators are very simple geometric primitives, including points, squares, lines etc., with some basic interaction capabilities, such as hit testing and dragging. The composite manipulators are more complex, and are constructed from basic manipulators. The composite manipulators are container classes that have to process messages, and appropriately propagate them to their children. Examples of messages include drawing, dragging, selection and deselection.

There are several types of composite manipulators, including basic and composite groups, which are ordered collections of basic and composite manipulators, respectively. Figure 7 illustrates a composite manipulator class hierarchy.

As composite manipulators are collections of other manipulators, manipulators have a hierarchical instantiation structure.

4.2.2 Specification Techniques

The specification manager offers a set of manipulators for the specification techniques. The specification techniques can choose manipulators from this set, or can derive even more complex manipulators from the existing ones, with modified appearance, or imposing technique specific constraints, priorities and relationships among manipulators. In this way, the specification techniques can share and reuse the provided graphical elements.

The specification manager defines the interface of a generic specification technique, i.e., the operations that a given specification technique will have to perform. It also requires that the specification techniques provide methods to extract the geometric and topological parameters from the manipulators which will be needed by the warping reconstruction techniques.

The specification manager defines the syntax of the specification and the implementation of the technique defines its semantics. In this way, the addition of a new specification technique does not involve tracking of user state or any system-dependent graphics, just the implementation of the meaning of the graphical manipulators.

4.3 Computation of Transformations

The separation between the computation manager and the computation techniques makes the process of adding a new computation technique straightforward. The parts that are common to

any computation technique, such as the generation of a series of frames of a transformation by varying the time parameter, are factored out to the computation manager level. The computation manager is also able to handle families of transformations and generate animations from them. The implementation of the computation techniques have to deal with just the computation, at a given point in time, of the specific operation: mapping object computation, warping reconstruction, attribute combination or shape combination.

4.3.1 Transformations Schedulers

The computation manager is also responsible for controlling the flow of the transformations, i.e., controlling the rate of change for different parts and attributes of the graphical objects. This control is offered by the *schedulers*, that determine the schedule according to which, different parts of the graphical object will be transformed. The schedulers can be interpreted as uni-dimensional functions that warp the time parameter, and have the identity transformation as their default state.

Schedulers can be applied to modify the effect of a transformation globally, or modify some elements of the morphing technique (warping reconstruction, attribute or shape combination). The use of different functions for different parts of the domain of the graphical object undergoing the transformation has the effect of localizing the transformation. As an example, consider the image in Figure 8(a). The attribute combination schedulers for the nose and hair specification elements are shown in Figure 8(b). The warping is controlled simply by a global scheduler with ease-in and ease-out. It is interesting to note that the schedulers themselves are specified using a derivation of the Curve composite manipulator, constrained to be a function in the domain of interest.

5 Support Level

This module of the system (Figure 5) manages input and output of graphical objects, and provides several utility libraries.

Input and output of graphical objects is performed by filters managed by an external object format manager, in much the same way as the warping techniques are managed by the computation manager. The addition of new filters is also a well defined process, limited to implementing a module that performs the operations defined by the file format manager, and adding the new module into the system. The file format manager provides mechanisms for the user to select preferred formats (according to the graphical object type), configure format specific options, and for automatically identifying the format of input files so that the appropriate graphical objects can be created and loaded.

The support level is also responsible for providing all the functionality of commonly used classes, such as vectors, matrices, hash tables, lists etc., which will facilitate the sharing of code throughout the system.

6 Platform Level

The main goal of the platform level (Figure 5) is to concentrate all the information related to the operating system and user interface. Besides allowing an easier port of the morphing system, the separation between the platform level and the rest of the software allows the implementation of different front ends for the same kernel level, in the same platform. For instance, it is possible to have two different applications, one using a graphical interface and the other using a command line or scripting interface, using the same kernel and support levels.

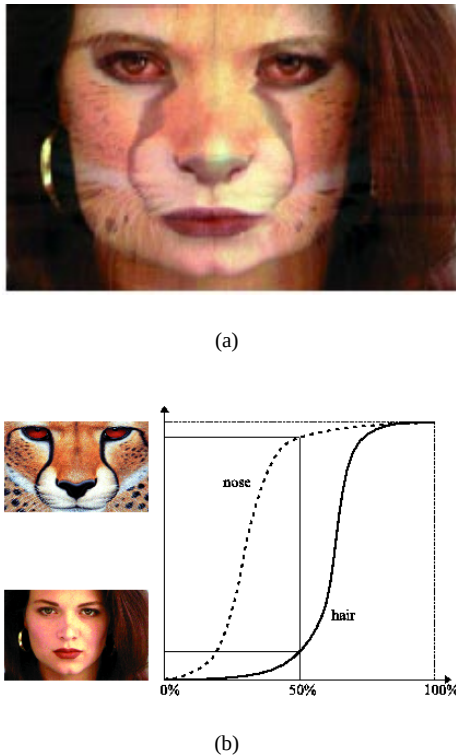


Figure 8: (a) Scheduled morph; (b) Two schedulers for the localized morphing in (a).

6.1 User Interface Host

The simple mapping of calls to the operating system does not solve completely the problem of portability of the morphing system. The *user interface host*, besides providing this “driver” functionality, is responsible for making the morphing application seem native to the system it is being ported to. This obviously implies that semantics of the graphical user interface of the operating system has to leak into the morphing system at some stage. The goal here is to confine that leak to the system dependent part of the code, so that it can be properly adjusted when porting the system.

In this way, the user interface host provides both event handling and output drawing methods at a high level used in the implementation of the manipulators (section 4.2.1). If the interface functionality were designed at a lower level, this could imply in code redundancy: for example, both point and line manipulators would receive mouse events, and might have to duplicate the implementation of rubber banding.

The color and pattern management is an integrated part of the user interface host. Rather than directly requesting a particular color or pattern to be used in a drawing operation, the manipulators select *representation types* according to the idea conveyed and to their states. Examples include hot spot and selected manipulator representation. These can be mapped to monochrome patterns or to the user’s color scheme.

6.2 System Host

The system host manages the platform-dependent resources used by the application, such as memory, interprocess com-

munication and error handling. As the usage of the standard libraries is recommended to maximize portability, the importance of the system host tends to be reduced. For instance, the use of C/C++ standard I/O library is usually enough to process file I/O.

The system host is also responsible for providing simple user interfaces for asking questions and notifying the user of errors, and also for exceptional situation handling to be used in the kernel and support levels. It also provides services for saving and retrieving persistent configuration options in the system, that can be implemented, for instance, through environment variables.

6.3 Application

Given the kernel level, the support level and the core modules of the platform level, an application can be built by making use of the parts of the system that are of interest.

In a command-line application, the specification of the warping or morphing transformation would not be given interactively, but by files. These specification files could be generated by using a separate specification-only software. This kind of separation would enable the user to compute transformations previously specified in an off-line fashion, to split up the work in a farm of workstations. This kind of application can be made portable by providing appropriate user interface and system hosts.

More commonly, the application will use system dependent features, and graphical interfaces. The use of application frameworks to automate the construction of most of the user interface of the software is a relatively simple approach.

7 The System

The *Morphos* warping and morphing system has been implemented in C++. The application and the platform level are currently implemented for the Windows family of operating systems, using OpenGL for the 3D interface and for real-time previews of image warping.

As stated before, the fundamental operations of the system are concentrated in the kernel level. An user-friendly interface, project management, intelligent file format usage and configurability are important aspects of an actual system that must be taken into consideration when designing the architecture. We have concentrated most of the system development effort in designing the kernel level, the support level, and the relation of these parts with the user interface.

7.0.1 Objects and Transformations

Morphos currently supports three types of graphical objects: images, polygonal curves, and surfaces. It has several techniques implemented, including: mesh, features and point specifications; linear and exponential dissolve attribute combination; field-based, radial basis functions, two-pass spline mesh warping, projective mapping etc.

Note that, as the managers are responsible for keeping track of the techniques and graphical objects under their control, none of the dialogs showing plug-ins has to be modified when adding new modules. Also, all the plug-ins can be configured regardless of their type.

7.0.2 User Interface

For metamorphosis the system uses a “side-by-side” interface: the user interacts with the source and the target objects in or-

der to specify the animation path for corresponding points. The transformation is specified by giving elements of the specification in two states (constraints). As our system allows the use of different specification types, several different specifications can be placed over the same graphical object, as illustrated in Figure 9 for images.

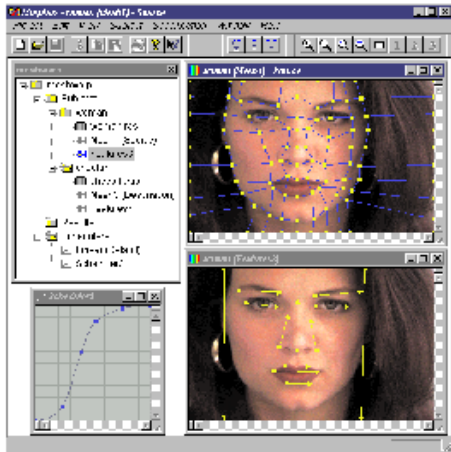


Figure 9: Two specifications for a subject.

Side-by-side interface can also be used for specifying warping operations, specially for graphical objects embedded on the plane (curves and images). This interface is useful when we need to specify precisely the handlers constraints, referencing them to the graphical object shape. Another warping specification is achieved by letting the user specify values for the constrained handlers (points, vectors, etc.). Morphos has an interface for displacement vectors in tridimensional space which uses constrained arcball. This interface is very effective for interactive warping of surfaces and volumes using point or vector specification.

8 Conclusion and Future Work

We have presented a new architecture for a warping and morphing system. The architecture is based on the fundamental unifying concepts of graphical objects and transformations of graphical objects. These two concepts allow us to create an integrated framework that encompasses all warping and morphing techniques, as well as all types of graphical objects, such as curves, images, surfaces and volumes.

The framework clearly separates the representation from the computational aspects, allowing any transformation to be applied to any object in the system.

Also, a warping/morphing scheme is decomposed into several computational elements. This decomposition is very important because it greatly enhances the expressive power of the system and allows the definition of customized warping and morphing schemes that best match the problem at hand.

The system architecture is structured in three levels: kernel, support and platform level. The kernel level encapsulates the operations of warping and morphing. The support level encompasses the file I/O and utilities and the platform level provides services to implement the user interface.

The above structure makes the system highly portable and, at the same time, makes the task of incorporating new technologies very easy.

We demonstrated the capabilities of the system with some examples that include:

- The use of different warping and specification techniques for the same type of graphical object and computation method.
- The integrated use of warping and morphing in the different aspects of a graphical object, such as its shape and color attributes.
- The design of a new warping technique using the basic functionality of the system.
- The application of the system as a test bed comparison of various warping and morphing techniques.

Future work in the system includes: Addition of volumetric data into the system, which can be implemented in a straightforward manner. The main difficulty is the development of efficient methods for visualization and feedback. Use of side-by-side interface for surface metamorphosis. Include some automatic feature detection techniques as an alternative form of warping specification. Hierarchical correspondence schemes for schedule control of the transformations. These mechanisms will allow a more precise specification of the evolution of the morphing transformation. Capability to handle time-varying graphical objects (motion, video and audio warping and morphing). This requires the tracking of the evolution through time of features associated with graphical objects. Also we intend to port the system the UNIX operating system.

Acknowledgements

The morphing technique for plane curves, based on [3], was implemented in the system by Adailson da Silva. The implementation of surface warp uses direct specification and the computation of the warp is done using coordinate systems. The implementation, based on [6], was done by Sergio Pinheiro. Also, Sergio implemented the 3D interface for displacement vectors.

References

- [1] D. Bechmann. Space deformation models survey. *Computers & Graphics*, 18(4):571–586, 1994.
- [2] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 35–42, July 1992.
- [3] E. Carmel and D. Cohen-Or. Warp-guided object-space morphing. *School of Mathematics, Tel-Aviv University*, 1997.
- [4] J. Gomes, B. Costa, L. Darsa, L. Velho, and T. Beier. *Warping and Morphing of Graphical Objects*. SIGGRAPH '97 Course Notes #34, 1997.
- [5] Jonas Gomes, Bruno Costa, Lucia Darsa, and Luiz Velho. Graphical objects. *The Visual Computer*, 12:269–282, 1996.
- [6] W. M. Hsu, J. F. Hughes, and H. Kaufman. Direct manipulation on free-form deformation. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26:177–184, 1992.
- [7] D. B. Smithe. A two-pass mesh warping algorithm for object transformation and image interpolation. Technical memo #1030, Industrial Light and Magic, 1990.