Tel-Aviv University

The Raymond and Beverly Sackler Faculty of Exact Sciences

School of Computer Science

Example-Based Rendering

Thesis submitted towards a Ph.D. degree

by Iddo Drori

under the supervision of Prof. Yehezkel Yeshurun and Prof. Daniel Cohen-Or

July 2004

Abstract

This thesis focuses on synthesis by example. The key idea is using a given data set to synthesize signals in various modalities and dimensions. The synthesis is based on a set of examples, rather than on a physical model. The main unifying theme throughout this work is decomposing the examples into parts and using the fragments to construct and compose a synthesized result. Such data driven approaches can be classified according to the level of features or parts used, from local features through intermediate level fragments to whole objects. This thesis focuses on the problems of decomposing, recombining, and composing mid level parts, within a data driven approach.

Example-Based Image Synthesis and Shape Analysis: We introduce an example-based synthesis technique that extrapolates novel styles for a given input image. The technique is based on separating the style and content of image fragments. Given an image with a new style and content, it is first adaptively partitioned into fragments. Stitching together novel fragments produces a coherent image in a new style for a given content. The aggregate of synthesized fragments approximates a globally non-linear model with a set of locally bilinear models. We show the result of our method for various artistic, sketch, and texture filters and painterly styles applied to different image content classes. Next, we present a framework for learning, representing and querying large data sets of shapes by multi-linear analysis of level sets. Shape regions are represented by an implicit shape model that does not require dense point correspondence and handles varying topology. We analyze a large training set of shape models by multi-linear factorization of the tensor produced from their level sets. This allows to synthesize canonical samples which are independent of any subset of factors, and to reduce dimensionality while maintaining the variability of known factors. We demonstrate the applicability of our approach by analysis of a multidimensional corpus of ceramics shapes and sketches of archaeological pottery.

Sound and Image Completion: We present a method for automatically filling in gaps of textural sounds. Our approach is to transform the signal to the time-frequency space, fill in the gap, and apply

the inverse transform to reconstruct the result. The complex spectrogram of the signal is partitioned into separate overlapping frequency bands. Each band is fragmented by segmentation of the time-frequency space and a partition of the spectrogram in time, and filled in with complex fragments by example. We demonstrate our method by filling in gaps of various types of textural sounds. We present a new method for completing missing parts caused by the removal of foreground or background elements from an image. Our goal is to synthesize a complete, visually plausible and coherent image. The visible parts of the image serve as a training set to infer the unknown parts. Our method iteratively approximates the unknown regions and composites adaptive image fragments into the image. Values of an inverse matte are used to compute a confidence map and a level set that direct an incremental traversal within the unknown area from high to low confidence. In each step, guided by a fast smooth approximation, an image fragment is selected from the most similar and frequent examples. As the selected fragments are composited, their likelihood increases along with the mean confidence of the image, until reaching a complete image. We demonstrate our method by completion of photographs and paintings.

Video Compositing and Segmentation: Fusion of image sequences is a fundamental operation in numerous video applications and usually consists of segmentation, matting and compositing. We present a unified framework for performing these operations on video in the gradient domain. Our approach consists of 3D graph cut computation followed by reconstruction of a new 3D vector field by solving the Poisson equation. We introduce new methods for fusing video smoothly and separating foreground elements from a video background for compositing by defining smooth and sharp transition constraints on a gradient video compositing equation. We demonstrate the applicability of smooth video transitions by fusing pairs for video mosaics, video folding, and video texture synthesis, and demonstrate the applicability of sharp video transitions by video segmentation, video trimap extraction and 3D compositing into a new sequence. Our results demonstrate that our method maintains coherence of the video matte and composite, and avoids temporal artifacts. Object segmentation in image sequences is one of the fundamental problems in computer vision and graphics. This problem is usually addressed either by discrete representations which are currently manifested by graph partitioning techniques, or by continuous methods typically referred to as active contours. In this work we take a unified approach by fitting splines to graph cuts. The strengths of this approach stem from the dual discrete and continuous representations and from allowing the user to refine the result of the cut by fitting a new spline to it

and modifying its points. Segmentation of an object in video is performed by a series of updates to the control points and computation of a minimum graph cut. Usually the graph cut results in a discrete representation over which the user has no control, and which is not always our desired result. Therefore our approach is to fit a spline to the resulting cut in key-frames. This allows the user to change the control points of the spline and then perform additional iterations of cut computation.

Acknowledgements

I wish to thank my advisors Hezy Yeshurun and Daniel Cohen-Or for their wisdom and guidance in this work. I wish to thank Michal Irani for inviting me to give talks on this work at the Weizmann Institute. I wish to thank my wife Sharon for her support during the past four years that led to this dissertation.

This work was supported in part by grants from the Israeli Ministry of Science and the Israeli Academy of Sciences.

Contents

List of Figures

Chapter 1

Introduction

This thesis focuses on synthesis by example. In order to synthesize images, shapes, sounds, and video we begin with a given data set. The synthesis is then based on a set of examples rather than on a physical model. A main unifying theme throughout this work is that we decompose the examples into parts and use the fragments to construct and compose a synthesized result. Such data driven approaches can be classified according to the level of features or parts used, from local features through intermediate level fragments [115] to whole objects. In this work we focus on the problems of decomposing, recombining, and composing mid level parts.

Our motivation for using a data driven approach is that the underlying physical function is too complex to model, whereas rich training data is readily available. The example based approach manifests itself in the various methods. For example, images in novel styles are generated by using a training set of images arranged as a matrix of style and content factors. We analyze shape factors by considering a large organized training set of level set functions. Another example is image completion and sound gap filling which reuse the known portions of the image or audio signal to fill in the missing regions. Finally, the work on video compositing and segmentation provides the necessary building blocks for the operations of recombining pieces of images and video volumes and extracting foreground parts.

1.1 Unifying theme and main contributions

The unifying theme in this work is the decomposition and recombination of signals by example. We focus on signals of one, two and three dimensions and of various modalities. In synthesizing images in new styles by example, images are broken into overlapping patches. The idea is to approximate a globally non-linear model by multiple locally linear models. The input image is decomposed into a set of overlapping fragments of various sizes, and fragments of the training images are considered at multiple positions, scales, and orientations. At the fragment level, similar fragments are approximately locally linear. Composing the local fragments back into a full image approximates the globally non-linear model by locally linear models. Our approach is to adaptively decompose the input image into fragments, and then synthesize a coherent image by stitching together synthesized fragments that follow multiple constraints. Taking an adaptive approach captures features at multiple scales and avoids drastic blocking artifacts. We proceed in a multi-scale top-down fashion maintaining a quad-tree for the input image as shown in Figure 1.1. The leaves of the tree correspond to elements that together cover the image. Each element is extended to include overlapping boundaries. These boundaries form the constraints among adjacent elements which are necessary to generate a coherent image. During image generation, elements are traversed in an order that preserves coherence with previously synthesized regions. The key idea is to apply the synthesis at the tile level, and such that the recomposition of their fragments is coherent. The use of a multi-linear model is extended to handle shapes rather than images by representing collections of shapes by their level sets.

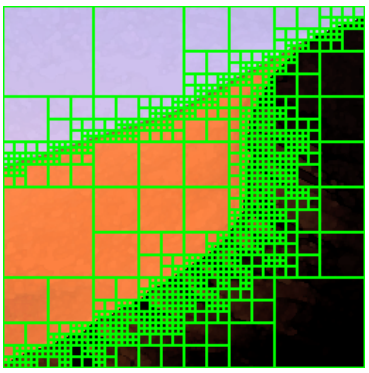


Figure 1.1: Image decomposition.

Figure 1.2: Visualization of adaptive neighborhoods overlayed completion result.

Interpolation methods can be used to fill in missing regions, relying on certain smoothness assumptions to estimate a function from samples. In this work we complete regions by compositing intermediate scale image fragments. To capture structures of various sizes we take an adaptive approach, where fragments have different sizes based on the underlying structure. The size of the neighborhood is determined by the image statistics and spatial frequency as shown in Figure 1.2.

Filling gaps of sound is performed by segmenting the complex spectrogram into parts. We then apply techniques used in texture and image synthesis for context-based sound synthesis. We adopt direct image space methods for synthesizing a time varying audio signal by using the complex spectrogram. The complex spectrogram of the signal is partitioned into separate overlapping frequency bands as shown on the vertical axis of the left panel of Figure 1.3. Each band is fragmented by segmentation of the time-frequency space and a partition of the spectrogram in time as shown on the horizontal axis of the right panel of Figure 1.3, and filled in with complex fragments by example. At each step of filling in a frequency band, a target fragment is defined with a causal region of overlap with the input and previously synthesized regions. The spectral boundaries in the time-frequency space are irregular and based on a local segmentation. To maintain a coherent sound stream, and motivated by our auditory working memory, the time extents of each fragment are determined by the nearest times in the partition of all previously (lower) synthesized frequencies with the greatest response inside the gap.

Video compositing focuses on recombining pieces of video volumes to create a whole coherent video. It refines the compositing operation used in image and texture synthesis. In the case of multiple image or video regions, compositing consists of sequentially partitioning the overlapping input by finding a cut and then merging across the overlapping region. Image and video editing operations include

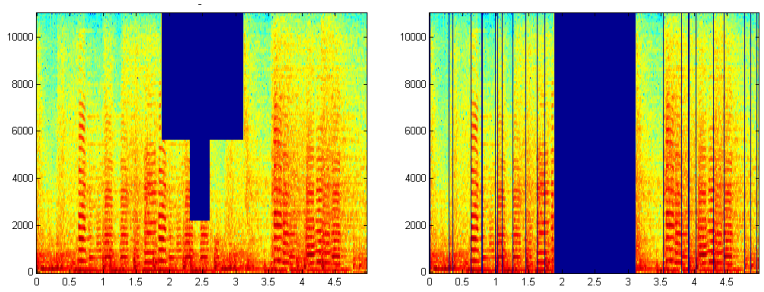


Figure 1.3: Spectrogram frequency and time partition.

temporal transitions, placing a figure over a background, and seamlessly combining different image or video regions. Texture synthesis commonly consists of searching for similar patches followed by merging them together. All these applications involve compositing as an essential step of the process. Image and video composition focuses on recombining parts into a spatially and temporally coherent result. Figure 1.4 demonstrates our method by spatially folding an image shown in (a), placing the left and right portions of the image one over the other, computing a 2D minimum graph cut in the overlapping region (b), and the resulting composite (c). Figure 1.5 shows a frame from the result of video trimap extraction for a time-lapse sequence of a flower opening. The figure demonstrates the various steps of our method: (a) keyframe spline interpolation, (b-c) iterative 3D graph cuts, and a frame of the resulting trimap (d). We introduce operations for smooth and sharp transitions between videos and provide a 3D formulation which reconstructs a temporally coherent output. It avoids temporal artifacts by spreading the residual of a least squares solution over the entire video volume.

(a) (b) (c)

Figure 1.4: Image folding: (a) Input image. (b) Cut. (d) Result of gradient image compositing.

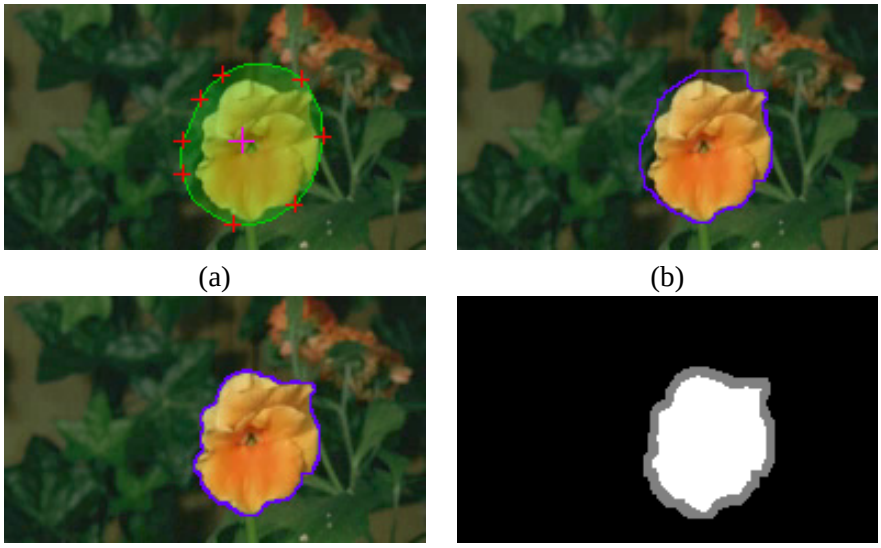


Figure 1.5: Video trimap extraction: (a) Keyframe spline interpolation. (b) Initial slice of 3D graph cut. (c) Result of iterative 3D graph cut. (d) Graph cut trimap.

Graph partitioning techniques are commonly used for object segmentation. We take a unified approach to object segmentation in video by fitting splines to graph cuts. The strengths of this approach stem from the dual discrete and continuous representations and from allowing the user to refine the result of the cut by fitting a new spline to it and modifying its points. Segmentation of an object in video is performed by a series of updates to the control points and computation of a minimum graph cut.

In conclusion, this work presents a framework for synthesis by example. Our data driven approach is centered around the decomposition of the signal into parts. It includes a search for similar parts, operations on components, and finally composition of the parts into a coherent whole.

Part I

Image and Shape Analysis and Synthesis

Chapter 2

Example-Based Style Synthesis

We introduce an example-based synthesis technique that extrapolates novel styles for a given input image. The technique is based on separating the style and content of image fragments. Given an image with a new style and content, it is first adaptively partitioned into fragments. Stitching together novel fragments produces a coherent image in a new style for a given content. The aggregate of synthesized fragments approximates a globally non-linear model with a set of locally linear models. We show the result of our method for various artistic, sketch, and texture filters and painterly styles applied to different image content classes. This part appeared in proceedings of IEEE Conference on Computer Vision and Pattern Recognition 2003 [35].

2.1 Introduction

Images and paintings can be regarded as a composition of content and style. An illustrative example is shown in Figure 2.1, where the four images on the top left are examples consisting of two pairs of images which have the same "content" and two pairs with the same "style". Let $\mathbf{p}_i$ represent the different underlying content of each row $i = 1, \ldots, m$, and f_j the different style operator of each column $j = 1, \ldots, n$. Then the $m \times n$ matrix of images is $\mathbf{A} = (a_{ij}) = f_j(\mathbf{p}_i)$.

Problem statement: Given the sub-matrix of images $\mathbf{A}(1 : m-1, 1 : n-1)$ as a small training set, and the input image $\mathbf{A}(m, n)$ shown on the lower right of Figure 2.1, which has a different content and a different style, the task of the algorithm is to complete the remaining images $\mathbf{A}(m, 1 : n-1)$ and

$\mathbf{A}(1:m-1,n)$. Loosely speaking we would like to synthesize a new image that contains the same content but with novel styles. In particular, the goal is to complete the image matrix and synthesize images with the same content as the input image, with the styles of the images of the training set.

Synthesizing images in various styles by example is related to the problem known as *image analogies* [63]: given training images $f_1(\mathbf{p}_1), f_2(\mathbf{p}_1)$ in different styles, and another example image in one of the known styles, $f_1(\mathbf{p}_2)$, the goal is to synthesize the example in the other known style $f_2(\mathbf{p}_2)$. In contrast, we deal with a two-factor problem, since the image $f_n(\mathbf{p}_n)$ has both unknown content $\mathbf{p}_n \neq \mathbf{p}_1, \mathbf{p}_2$ and unknown style $f_n \neq f_1, f_2$. Notice that once any of the images in the image matrix is synthesized, the remaining images can then be generated by analogy. In image analogies [63], the training set is used as a look-up table, which shows explicitly the mapping between styles of a given pixel in a given local content. Here we take a different approach: the training set is used to generate a model that factors style and content, and then novel images are synthesized based on the computed model. Factorization of style and content has been applied successfully to aligned images [52, 110]. However, in our setting there is no natural correspondence between the training images of different content, as shown in the rows of Figure 2.1, and no natural correspondence with the given input image, and thus the images cannot be aligned. Moreover, the input image has a different content, and its style is not necessarily present in any of the examples in the training set.

Separating style and content in the general case is a difficult problem. It has been successfully applied to factors such as illumination, that can be approximated by a linear subspace [8]. The method introduced in this work deals with models which are globally non-linear. The idea is to approximate a globally non-linear model by multiple locally linear models. The input image is decomposed into a set of overlapping fragments of various sizes, and fragments of the training images are considered at multiple positions, scales, and orientations. At the fragment level, similar fragments are approximately locally linear. Composing the local fragments back into a full image, approximates the globally non-linear model by locally linear models. This property is maintained by an adaptive choice of fragments, as will be described in detail below.

Building upon recent example-based image synthesis techniques [41, 49, 63, 80, 105, 110], our approach is to adaptively decompose the input image into fragments, and then synthesize a coherent image by stitching together synthesized fragments that follow multiple constraints. Taking an adaptive

approach captures features at multiple scales and avoids drastic blocking artifacts. It should be emphasized that this work takes into account image content that includes both stationary and local regions such as textures, as well as non-stationary and global information present in general images. However, our approach is limited to capturing styles that are local, and at the fragment level, the model is simple and fixed.

2.2 Previous work

Many operations ranging from low-level vision tasks to high-end graphics applications have been efficiently performed based on examples [7, 12, 17, 19, 21, 49, 50, 63, 64, 104, 110].

The idea of defining and factoring image style and content using a bilinear model was introduced to computer vision by Freeman and Tenenbaum [52]. Given a training set of aligned face images of different people under varying illumination, Tenenbaum and Freeman [110] refer to the identity of a face as content whereas the illumination is considered as style. Style and content are factored pixel-wise by fitting a symmetric bilinear model to the training images. Combinations of style and content are synthesized from a new face image under novel illumination by changing the estimated parameters of the new image.

Given a set of parameters describing facial expressions such as degree of happiness or surprise, along with a relative motion field of the face, Beymer *et al.* [11, 12] use a regression function that estimates a mapping from parameters to motion for synthesizing novel facial expressions from a set of labelled training images.

Rather than defining individual filters by hand, Hertzmann *et al.* [63] automatically studied a mapping of spatially local filters from image pairs in pixel-wise correspondence, one a filtered version of the other. A new target image is then filtered analogously to the training images. Pixels are rendered by comparing their neighborhood, and those of a coarser level in a multi-resolution pyramid, to neighborhoods of a training image pair. Considering the original image as output and taking its segmentation map as input allows the texture to be painted by numbers [58]. A new image that is composed of the various textures is synthesized by painting a new segmentation map. Swapping the output segmentation map with the original input image results in example-based segmentation [17]. In our case, we cannot only

approximate a common representation for images by blurring and median filtering and then proceed by analogy, as this results in the loss of image detail.

Freeman *et al.* [49, 50] derive a model used for performing super-resolution by example. The technique is based on examining many pairs of high-resolution and low-resolution versions of image patches from several training images. Baker and Kanade [7] apply this technique restrictively, to a class of face images. Given a new low-resolution face image its corresponding high-resolution image is inferred by re-using the existing mapping between individual low-resolution and high-resolution face patches.

The term *image quilting* [41] was coined for describing the process of synthesizing a new image by stitching together blocks of existing images. The technique focuses on handling boundaries between blocks and is applied to texture synthesis. It enforces a constraint that the error between overlapping blocks is minimal, and blocks are stitched along a minimum cost path [32] that is computed by dynamic programming. The results depend on the size of a block, which varies according to the texture properties. Hierarchical pattern mapping [105] takes into account that patterns in a texture are often in many different sizes. A 3D surface is progressively covered by texture patches of various shapes and sizes. Starting from a large patch, it is split into smaller ones based on the texture fitting error with already textured neighborhoods. By adding the constraint that the synthesized texture match an example image, Ashikhmin [5] presented texture transfer, in which an example image is rendered with a training texture. This technique was extended to color transfer [120], a special case of image analogies, by matching local image statistics. In our setting, we cannot simply transfer color and texture from one style to another, as applying properties of one style over the other could result in artifacts.

2.3 Image manifolds

A general mathematical perspective for describing the variability of images is to regard an image, which is represented by a vector of pixel intensities, as a point in an abstract image space, and then to characterize a set of images by a manifold in this high-dimensional image space [102]. The statistics of natural images are correlated [44] and far from random. Recently, several researchers [8, 96] have independently shown that irradiance environment maps, for rendering Lambertian objects under distant illumination, are well approximated by a nine-dimensional linear subspace. Thus, from any given view,

the appearance of an arbitrary Lambertian shape, under arbitrary distributed distant illumination, is approximately near a ten-dimensional manifold. By adding more parameters, the dimensionality of the manifold increases, and it becomes highly complex and non-linear, although its dimensionality remains far less than that of the image space.

In our work, we would like to be able to extrapolate between images in parameter space by moving along such a manifold and extrapolating novel points. This requires addressing three issues: (i) The manifold is complex and has a highly non-linear global structure, (ii) The "curse of dimensionality"; the dimensionality of the image space is too high, and (iii) There is no natural correspondence between the images. Classical approaches such as Principle Component Analysis or Multiple Discriminant Analysis, that find projections that best represent or separate data in a least squares sense, are unsuitable for our purposes since they require correspondence.

Our approach is based on the observation that although the manifold is globally non-linear, *locally* it is approximately linear. This alleviates the problem of complexity of the manifold. It permits applying simple local linear models as a good approximation of a global highly non-linear model. To reduce the high dimensionality of the space and to overcome the lack of correspondence, the images are decomposed into multi-scale fragments. A fragment is defined as a connected region cut from an image tile. The tiles are overlapping and their aggregation is an over-complete representation of the image. The key idea is to apply the synthesis at the tile level, and in a way that the composition of their fragments into a full image is coherent.

2.4 Style synthesis

As explained in the introduction, we would like to synthesize novel images based on the training set. Our solution is based on adaptively decomposing the input image into fragments and applying the synthesis locally. We construct a feature pyramid for each image to capture various features at multiple scales. The training set constitutes an over-complete representation, considering fragments at all scales, positions, and eight orientations. The input image is adaptively subdivided by a quad-tree whose leaves consist of overlapping tiles that are traversed in Morton's order [100]. Images are then synthesized from coarse to fine where each level is based on a coarser level. This captures features at multiple scales and accelerates

the computations.

At each level, tiles are synthesized from a number of example tiles. The choice of these examples is based on multiple constraints, which include the underlying image geometry. The extrapolated tiles are then cut into fragments which together form a coherent tessellation of the novel image. In the following we elaborate on each of these stages.

2.4.1 Image decomposition

We adaptively decompose the input image $\mathbf{A}(m,n)$ into overlapping tiles. Our algorithm proceeds in a top-down fashion maintaining a quad-tree as shown in Figure 2.2 for the input image, where each of its nodes represents an image tile (element), and its four children represent a partition of that element. The leaves of the tree correspond to elements that together cover the image. Each element is extended to include overlapping boundaries. These boundaries form the constraints among adjacent elements which are necessary to generate a coherent image. During image generation, elements are traversed in an order that preserves coherence with previously synthesized regions.

An adaptive subdivision requires a method for deciding whether an element should be split or not. Our initial subdivision of the input is based on the luminance and color values across an element. If any of the absolute differences between the maximum and minimum values is above a threshold δ, and the minimum element size $\varepsilon_{\min}$ has not yet been reached, then the element is recursively subdivided. In all the results shown we set δ between 0.25 and 0.4, and ε_{min} to 4 by 4 and 8 by 8.

This coarse subdivision is further refined at runtime in any of the following stages: (i) Multi-dimensional search: if a suitable set of candidate tiles is not found; (ii) Fragment synthesis: if the model fails to accurately reconstruct (validate) the input tile; and (iii) Image reconstruction: if the synthesized fragments do not agree with previously synthesized regions. There is a trade off here since finer elements meet the above conditions but do not capture large features.

2.4.2 Multi-dimensional search

An important part of the algorithm is the selection of a set of fragments which has a 2D (style and content) embedding that reflects the relation among the training set images in the higher dimension. At the same time, the synthesized fragments must agree with their neighbors. Moreover, in a single step,

the algorithm synthesizes a set of fragments, one for each style and content. Thus, a multi-dimensional search must simultaneously take into consideration constraints from multiple tiles and boundary values. Boundaries are crucial in capturing geometric configurations and maintaining consistency with previously synthesized regions.

A *search vector* is the concatenation of components from multiple origins, as illustrated in Figure 2.3. Formally, let $\mathbf{A}(i,j)^l$ denote an image in row i and column j of the matrix, at scale l, where $l=0$ is the coarsest level. Let $T_{p,\varepsilon,o}$ denote an extended tile around position p, of size ε, in orientation o, and similarly, let $B_{p,\varepsilon,o}$ denote the corresponding boundary. Each search vector is a concatenation of the boundaries B with previously synthesized fragments in styles $j=1,\ldots,n-1$, and tile T in style n, and the corresponding tiles at a coarser scale $l-1$, for $l>0$,

$$B_{p,\varepsilon,o}(\mathbf{A}(i,1)^l),\ldots,B_{p,\varepsilon,o}(\mathbf{A}(i,n-1)^l),T_{p,\varepsilon,o}(\mathbf{A}(i,n)^l),$$

$$T_{\frac{p}{2},\frac{\varepsilon}{2},o}(\mathbf{A}(i,1)^{l-1}),\ldots,T_{\frac{p}{2},\frac{\varepsilon}{2},o}(\mathbf{A}(i,n)^{l-1}). \tag{2.1}$$

Given the vector defined in Eq. (2.1), with input content $i=m$ and upright orientation, we find its nearest neighbors in each content row $i=1,\ldots,m-1$, over each position, and orientation. This enforces the same position p and orientation o, for the same content across different styles $j=1,\ldots,n$, but allows different positions and orientations for different contents $i=1,\ldots,m$.

The above multi-dimensional search is applied at multiple scales over several features. Our algorithm maintains a feature pyramid, created in a pre-processing stage, for each image in the matrix $\mathbf{A}$. The feature pyramid consists of color and luminance Gaussian pyramids, as well as four gradient pyramids in the horizontal, vertical and two diagonal directions, and a Laplacian pyramid, which are derived from the luminance values. Features are selected according to properties of the training set and the component type. Color is effective for a rich training set, which contains a large distribution of samples for a high dimensional search. Luminance is suitable for both the tiles (T) and boundaries (B). In addition, gradients and the Laplacian are effective features for tiles, but not used for thin boundaries. Gradients are computed by the Sobel operator and are correspondingly isotropic for horizontal, vertical and diagonal edges. The Laplacian is invariant under rotations for increments of 45 degrees. In this work

we experimented with L_1, L_2 norms, normalized correlation coefficient, and ordinal measures [13] for similarity of vectors. The normalized correlation coefficient is linearly invariant to contrast, and ordinal measure is too expensive for our purposes. Therefore, we use the L_1 and L_2 norms depending on the selected features.

The search vectors can be regarded as points in multi-dimensional space. Finding a best match is then equivalent to finding the nearest-neighbor. Three main strategies of searching for nearest neighbors are pre-structuring, partial distances, and pruning [39]. We use partial distances, which constitute a monotonic non-decreasing function, and test intermediate distances twice. In order to further improve matching performance, the search for a similar vector is performed hierarchically from coarse to fine. The position of the best matching vector is successively refined based on the best position found at a coarser level. The search is then confined to a window that decreases exponentially in size while traversing the pyramid from coarse to fine. For a full pyramid, each search is logarithmic in the number of pixels in the finest level times the number of features. We choose the coarsest level in the search for each vector such that the size is at least ε_{min}. Overall, the entire search is linear in the resolution of the input image times the logarithmic factor in the number of training images, their resolution, the number of orientations, and the number of features.

2.4.3 Fragment synthesis

In this section we describe the mathematical tool used to locally factor style and content of tiles and synthesize novel fragments. We begin by briefly reviewing multilinear and bilinear forms, and factoring equations for symmetric bilinear forms (the reader is referred to Tenenbaum and Freeman [110] for detailed derivations).

A multilinear form on V^n is a mapping $f : V^n \rightarrow F$ which is linear in each argument, that is, for all $\mathbf{x}_i, \mathbf{x}'_i \in V$, and all $\lambda \in F$:

$$\begin{aligned} f(\mathbf{x}_1, \ldots, \mathbf{x}_i + \mathbf{x}'_i, \ldots, \mathbf{x}_n) &= f(\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_n) + \\ & \quad f(\mathbf{x}_1, \ldots, \mathbf{x}'_i, \ldots, \mathbf{x}_n) \\ f(\mathbf{x}_1, \ldots, \lambda \mathbf{x}_i, \ldots, \mathbf{x}_n) &= \lambda f(\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_n). \end{aligned}$$

The special case of a 1-form is a linear mapping $f(\alpha\mathbf{x}+\beta\mathbf{y}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{y})$. A 2-form is a bilinear form. Defined on $V \times W$, it is a mapping $f : V \times W \to F$ which is linear in each argument, that is, for all $\mathbf{x}, \mathbf{x}' \in V$, all $\mathbf{y}, \mathbf{y}' \in W$ and all $\lambda \in F$:

$$\begin{aligned} f(\mathbf{x}+\mathbf{x}', \mathbf{y}) &= f(\mathbf{x}, \mathbf{y}) + f(\mathbf{x}', \mathbf{y}) \\ f(\mathbf{x}, \mathbf{y}+\mathbf{y}') &= f(\mathbf{x}, \mathbf{y}) + f(\mathbf{x}, \mathbf{y}') \\ f(\lambda\mathbf{x}, \mathbf{y}) &= \lambda f(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}, \lambda\mathbf{y}). \end{aligned}$$

If $f : V \times V \to F$ is a bilinear form then the matrix $\mathbf{M}$ of f relative to the basis $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ is given by $m_{ij} = f(\mathbf{v}_i, \mathbf{v}_j)$. If $\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{v}_i$ and $\mathbf{y} = \sum_{i=1}^{n} y_i \mathbf{v}_i$ then the polynomial and matrix representations of the form are:

$$f(\mathbf{x}, \mathbf{y}) = \sum_{i,j=1}^{n} x_i y_j m_{ij} = \mathbf{x}^t \mathbf{M} \mathbf{y}. \tag{2.2}$$

A bilinear form is symmetric if $f(\mathbf{x}, \mathbf{y}) = f(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in V$, and in matrix form $\mathbf{M} = \mathbf{M}^t$.

As described above, a bilinear form maps two vectors $\mathbf{x}$ and $\mathbf{y}$ by a matrix $\mathbf{M}$ to a scalar, and introducing subscripts for multiple vectors, matrices and scalars:

$$a_{ijk} = \mathbf{x}_i^t \mathbf{M}_k \mathbf{y}_j. \tag{2.3}$$

Now consider the inverse problem. Given only a set of scalars a_{ijk}, a bilinear model can be fitted by minimizing the total squared error:

$$\min_{\mathbf{x}_i, \mathbf{M}_k, \mathbf{y}_j} \sum_i \sum_j \sum_k (a_{ijk} - \mathbf{x}_i^t \mathbf{M}_k \mathbf{y}_j)^2. \tag{2.4}$$

A solution consists of a set of bilinear forms, defined by matrices $\{\mathbf{M}_k\}$, together with sets of vectors $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_j\}$.

We use an iterative SVD method as presented in [110] for Eq. (2.4). An exact solution exists when the number of vectors $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ equals their dimension. For example, for general matrices $\mathbf{M}_k$, an exact solution exists when choosing each set $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ to be a basis. For the special case of the standard basis, the matrices are simply defined by the scalar values themselves $\mathbf{M}_k(i, j) = a_{ijk}$.

Next, we would like to estimate the two vectors **u** and **v**, derived from the established mappings $\{\mathbf{M}_k\}$, for another vector **b**:

$$b_k = \mathbf{u}^t \mathbf{M}_k \mathbf{v}. \tag{2.5}$$

The above is solved iteratively, where the initial guess for the vector **v** is the mean of the vectors $\mathbf{y}_i$, and the superscript t denotes the vector transpose of a block matrix [110]:

$$\mathbf{u} = ((\mathbf{M}\mathbf{v})^t)^{-1}\mathbf{b} \qquad \mathbf{v} = ((\mathbf{M}^t\mathbf{u})^t)^{-1}\mathbf{b}. \tag{2.6}$$

Finally, the derivation above is applied to a style and content matrix of tiles. The tile positions, orientations and scales are of the boundaries and tile in scale l in Eq. (2.1), for the best match in each content row $i = 1,\ldots,m-1$, and the original search vector. The kth pixel value of a tile in content row i and style column j is denoted as a_{ijk}. The vectors $\mathbf{x}_i$ and $\mathbf{y}_j$ determine the style and content of each tile. The vector **b** denotes the input tile, and its estimated style and content parameters are **u** and **v**. In case the model fails to reconstruct (validate) the input tile, Eqs. (2.5,2.6), we first extend the number of nearest neighbors considered in the search, and then refine the input subdivision.

Having estimated the style and content parameters **u**, **v** of the remaining row and column, the tile matrix is completed by applying $\mathbf{x}_i\mathbf{M}_k\mathbf{v}$ and $\mathbf{u}\mathbf{M}_k\mathbf{y}_j$.

As explained, the multi-dimensional search is based on a number of selected features. However, after the search finds a set of tiles, we consider only luminance, and color for a sufficiently varied training set. Synthesis is performed in the perceptually-based (l,α,β) color space [97]. This space minimizes the correlation between channels, and so tile synthesis is performed separately for each channel. The results are then transformed back to (r,g,b) values for display.

2.4.4 Image reconstruction

Prior to synthesis, the search for a set of tiles enforces that they match previously synthesized regions. This is not sufficient to guarantee that the synthesized tiles match previously synthesized regions as well. The error after synthesis is computed for the overlapping boundaries, over all features, across the different styles. If it is proportional to the error before synthesis, then we repeat the synthesis process, extending the number of nearest neighbors. If a small number of nearest neighbors is exhausted, then

the search and synthesis process is refined by recursively subdividing tiles. Once a set of matching tiles is synthesized, we find the minimum cost path along the error surface of the overlapping boundaries by dynamic programming [41], breaking ties by taking the center values.

Traversal of the entire input image guarantees a cover of the images in the mth row, but not of the images in the nth column of $\mathbf{A}$. The synthesized tiles of the nth column are transformed back to their original orientation, and may appear at any image position according to the search. Thus, for each scale l, after synthesizing images $\mathbf{A}(m,j)^l$ for columns $j = 1,\ldots,n-1$, the images $\mathbf{A}(i,n)^l$ for rows $i = 1,\ldots,m-1$ are completed by extending image analogies. The roles of the training images and the inputs are then reversed. The images of each row $i < m$ serve separately as input, and the mth row serves as the training images. Additional modifications, compared with image analogies [63], consist of (i) adaptively decomposing the input, (ii) stitching together fragments instead of synthesizing pixels, and (iii) searching for a simultaneous match of previously synthesized boundaries and candidate regions, in multiple scales and across all styles as shown in Figure 2.3.

Since style synthesis is performed from coarse to fine we specify the initial conditions for the search and synthesis, and the inductive step applied from each level to the next. First, if the coarsest level $l = 0$ in the pyramid consists of a single tile, then the search is empty, and the remaining tiles are synthesized. Next, for $l > 0$, the image boundaries for each style in the remaining mth row are initialized, and the coarser level $l-1$ serves as an initial guess for the remaining nth column. In particular, for different luminance across styles, after initializing $A(m,j)$ with the target, linearly matching the histograms to $A(1:m-1,j)$ improves the result. In this case, $A(i,n)$ is initialized with random samples of $A(m:n)$. For the same colors across styles, initializing $A(i,n)$ with the mean of $A(i,1:n-1)$ yields comparable results.

Finally, we consider two special cases: (i) A single content training row; The search is performed under multiple constraints simultaneously, finding a single fragment for each image. Therefore instead of extrapolation, the nearest neighbor is selected. Notice that if there is no underlying model across the different content rows, then images in the same style and different contents are regarded as a single image; and (ii) A single style training column; Content rows are considered as one image, and the search finds a single fragment, selecting the nearest neighbor.

2.5 Results

We have experimented with our method with various sources and styles: (i) Artistic, sketch, and texture filters from an image editing tool [1]; (ii) Painterly renderings generated by applying layers of curved brush strokes [62]; and (iii) Real paintings. The filters and painterly styles were applied to a variety of image classes from a database of stock photographs. The subjects of these photographs are varied and include landscapes, buildings, people, products, and more. For all of these families of styles and image content classes, our method produces visually plausible results. Computation times range from one hour (Figure 2.5) to ten hours (Figure 2.4) for synthesizing a set of 256 by 256 images on a 1.8GHz PC, running a Java implementation.

The transposed matrix in Figure 2.4 shows natural images in four different artistic styles, in each row from top to bottom: poster edges, fresco, paint daubs and dry brush. The poster edges filter reduces the number of colors in an image and accentuates edges with black lines. The fresco filter applies short and rounded dabs creating a coarse style. The paint daubs filter paints an image with simple round brush strokes. The dry brush filter simplifies an image by painting edges with round brush strokes and reducing the range of colors. The two images in dry-brush on the left of the lower row, and the three images in poster-edges, fresco, and paint-daubs on the right column were synthesized by our algorithm, and are denoted by a thick frame.

The transposed matrix in Figure 2.5 shows landscape images in three different sketch styles: charcoal, reticulation, and graphic-pen. The charcoal filter draws major edges in bold, while mid-tones are sketched using diagonal strokes. The reticulation filter creates an image that appears clumped in the shadow areas and lightly grained in the highlights. The graphic pen filter uses fine linear strokes to capture image details. Only upright orientation was considered for synthesizing the images shown on the right column, to capture the directional strokes of the graphic-pen. The two images in graphic-pen on the left of the third row, and the two images in charcoal and reticulation on the right column were synthesized by our algorithm. We compare our results with applying the filter directly to the original images, as shown in the bottom row of Figure 2.5.

The matrix in Figure 2.6 shows landscape images. Each of the landscape images has a different appearance. We applied three different artistic and texture styles: craquelure, sponge, and water-color. The craquelure filter paints an image onto a relief surface, generating cracks that follow contours. The

sponge filter creates images with highly textured areas of contrasting color. The water-color filter simplifies image details by saturating color at edges. The images in sponge and craquelure on the lower row, and the two images in water color on the top right column were synthesized by our algorithm.

The matrix in Figure 2.7 shows images in two different painterly styles [62], impressionist and expressionist, and a real painting. The impressionist style applies moderately curved brush strokes without random jitter. The expressionist style paints an image with elongated brush strokes with random color jitter. The input image is from a painting in a post-impressionist style. The images in impressionist and expressionist style on the lower row, and the two images on the top right column were synthesized by our algorithm.

2.6 Comparison

We have compared our approach for style synthesis with the best space invariant linear filter from an image in the training set. To compute the optimal linear filter we solve for:

$$\theta^* = \arg\min_{\theta} \frac{1}{k} \|b - A\theta\|_2^2,$$

where A is a matrix containing k rows each with pixels within a source patch, b is a column vector of target pixels, and θ the filter parameter vector. This is equivalent to solving for:

$$\theta^* = \arg\min_{\theta} (\theta^T \tilde{R}_{aa} \theta - 2\theta^T \tilde{r}_{ab}),$$

where $\tilde{R}_{aa} = \frac{A^T A}{k}$ estimates the auto correlation, and $\tilde{r}_{ab} = \frac{A^T b}{k}$ estimates the cross correlation. The solution is the parameter vector $\theta^* = (\tilde{R}_{aa})^{-1} \tilde{r}_{ab}$.

Notice that in our case the novel image is in a new style that is not present in the training images. The comparison is shown in Figure 2.8, where the left column shows the result of the best space invariant linear filter and the right column shows the result of our method. The top left image shows the result using a filter computed from a style present in the training set to Craquelure and corresponds to Figure 2.6, whereas the bottom corresponds to Poster Edges in Figure 2.4. As observed, our method is better suited for handling adaptive patches, whereas applying one style onto another creates artifacts.

2.7 Conclusions

We have introduced a method for style synthesis that utilizes a training set of images. While the specific implementation here is image synthesis, the main motivation is to explore the separability of content and style, which can be viewed as a fundamental topic in image pattern analysis. Our method is based on an adaptive scheme to synthesize local fragments of an image by extrapolating multiple fragments. Future work will focus on problems with more than two factors, using a local multilinear model for approximating highly complex factors of appearance, such as changing weather conditions.

Our approach is example-based, which means that its performance is dependent on the richness of the available fragments in the training set. In addition, it is an image-based 2D method and does not incorporate high-level information about the underlying scene. Although we have applied several techniques for accelerating the search, our algorithm is still very slow. To speedup the search we would like to apply geometric hashing, and then use larger training sets and consider tile deformations.

Finally, an interesting extension of this work is to 3D surface patches of 3D models. This will require to partition a mesh into patches and incorporate mesh coordinates in the synthesis process.

Figure 2.1: A training set of painterly rendered images and an input from a Still Life by Paul Cezanne form an image matrix. Each column consists of various styles of the same image, and each row results from applying the same style to different images. The goal is to synthesize the remaining images.

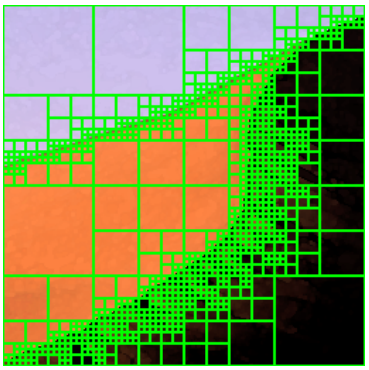


Figure 2.2: Quadtree image decomposition.

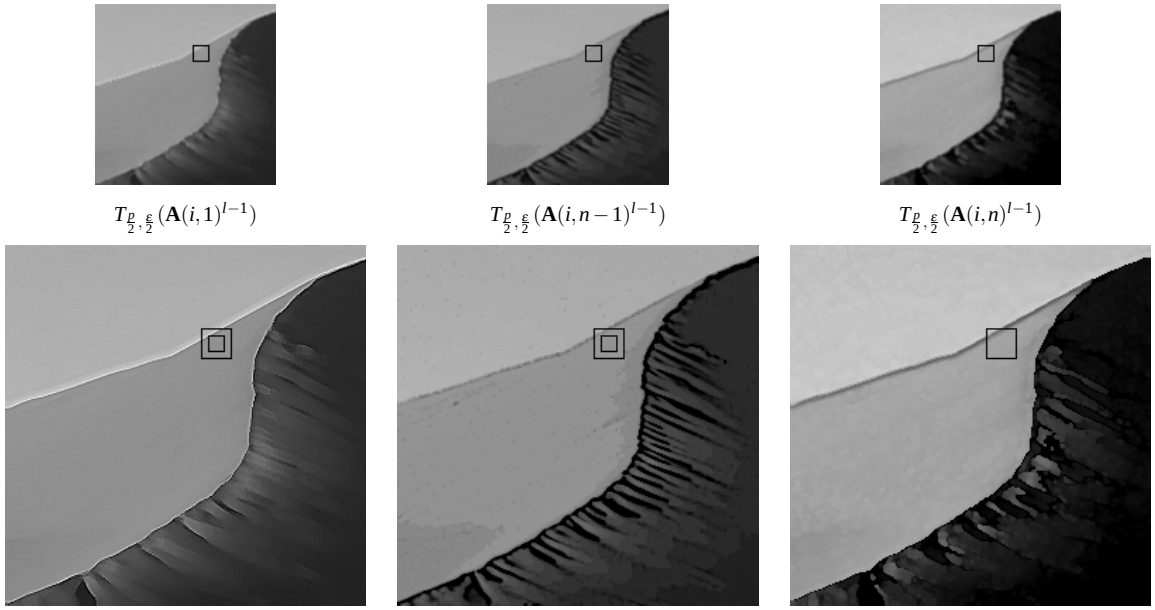


Figure 2.3: The components of a search vector for a single content row are highlighted. Notice that the extension of a uniform tile captures the edge.

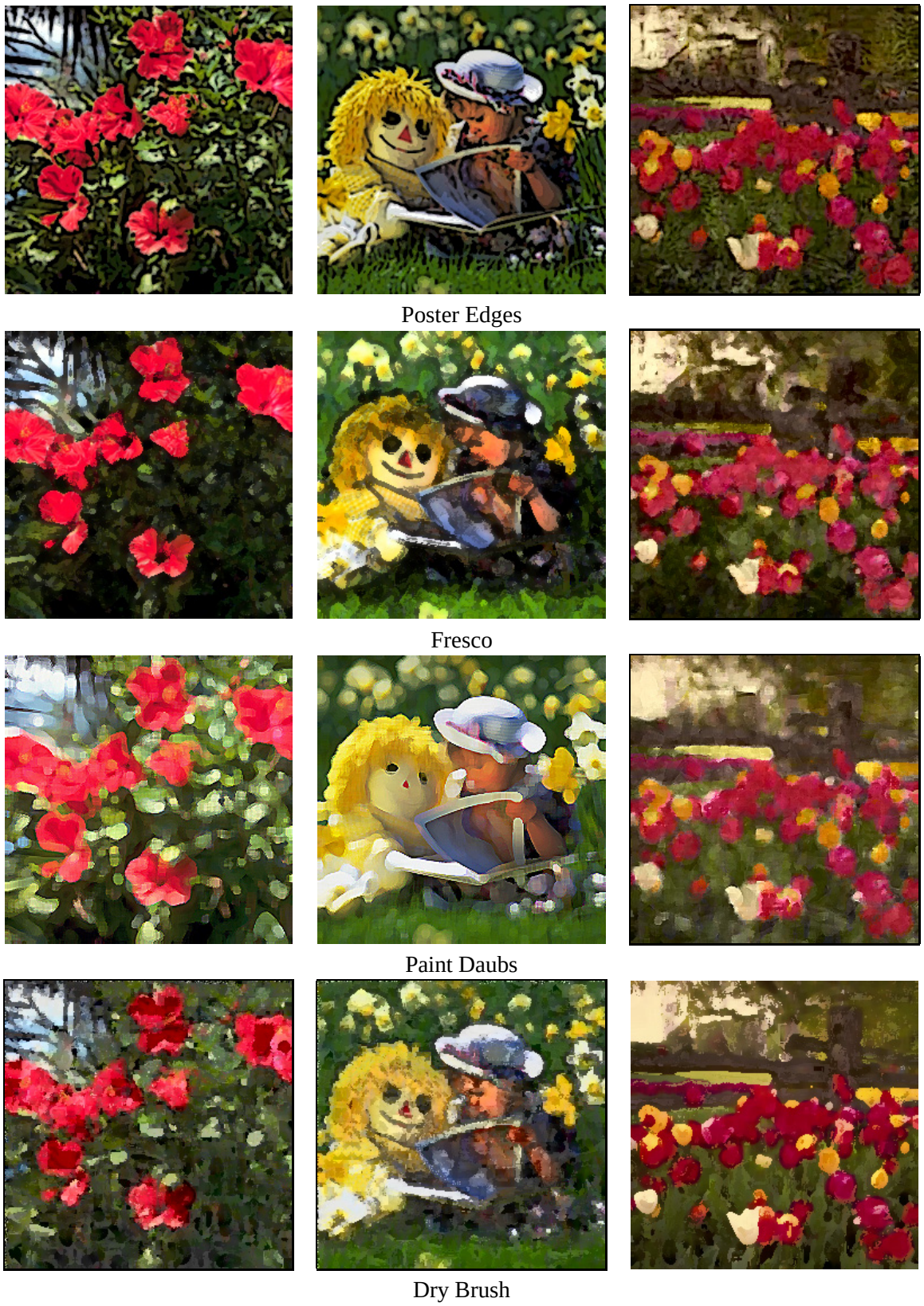


Figure 2.4: A set of natural images and artistic filters form an image matrix. The two images in dry-brush on the left of the lower row, and the three images in poster-edges, fresco, and paint-daubs on the right column were synthesized by our algorithm.

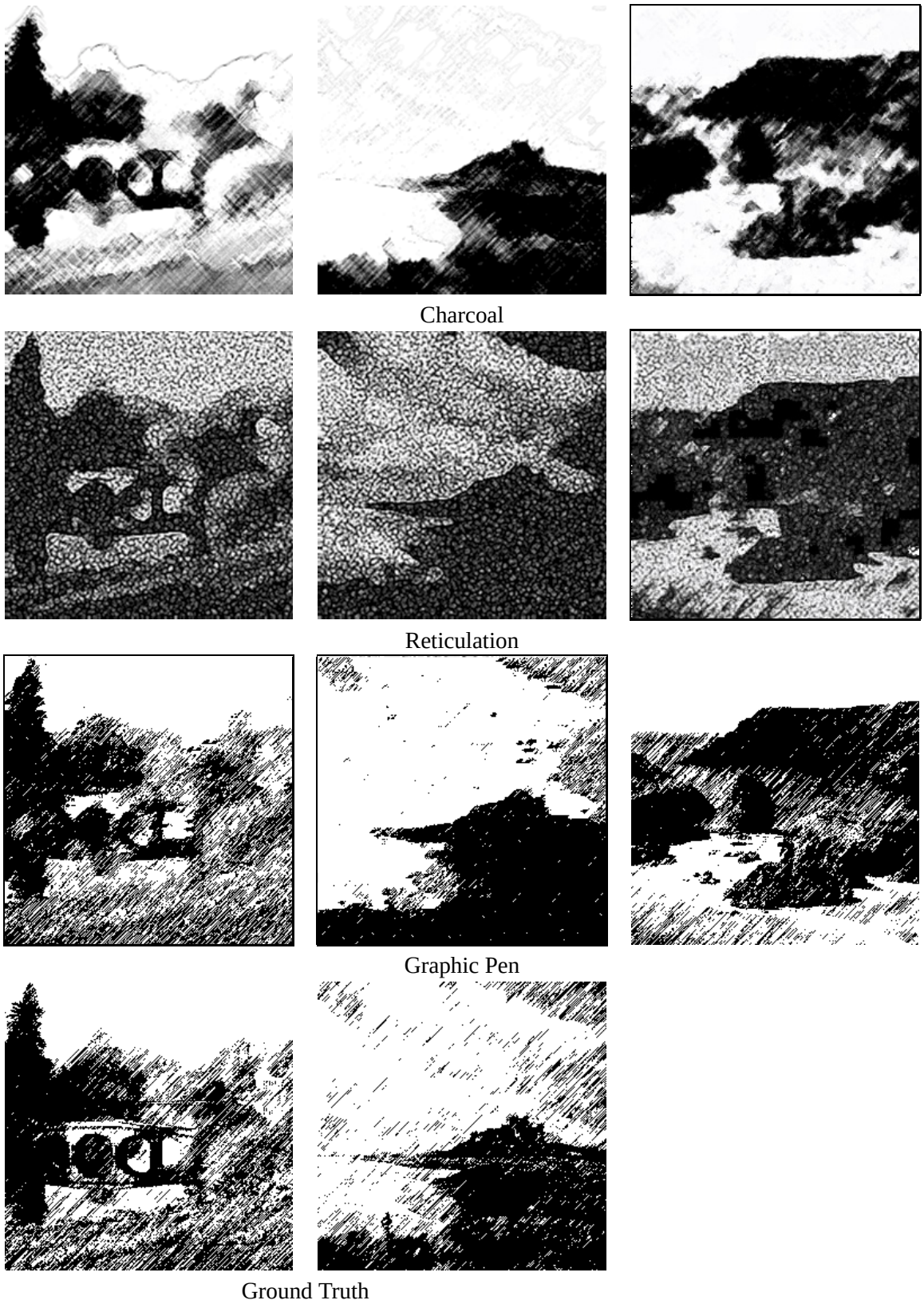
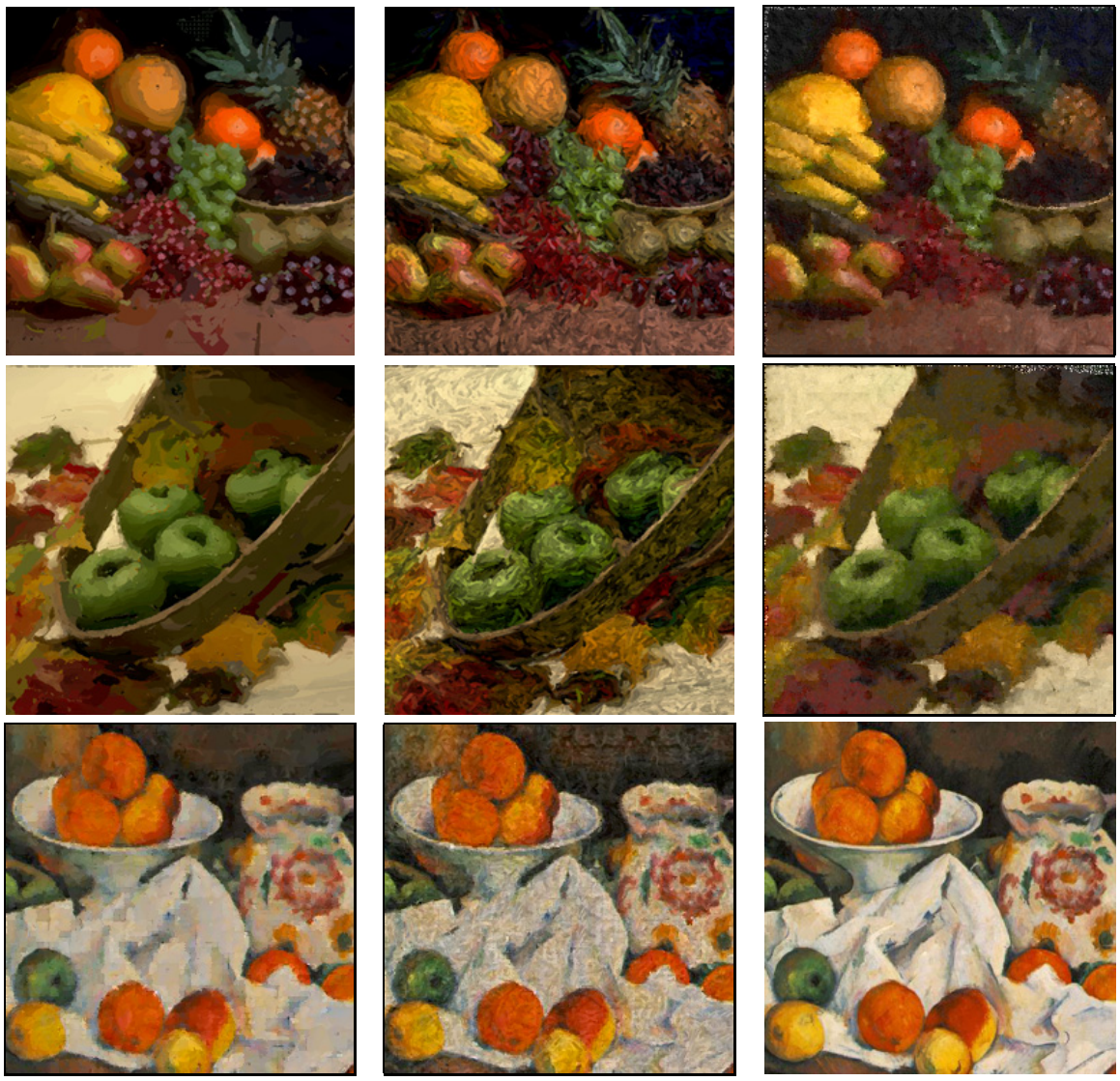


Figure 2.5: A set of landscape images and sketch filters form an image matrix. The two images in graphic-pen on the left of the third row, and the two images in charcoal and reticulation on the right column were synthesized by our algorithm.

Sponge Craquelure Water Color

Figure 2.6: A set of landscape images and artistic, texture filters form an image matrix. The images in sponge and craquelure on the lower row, and the two images in water color on the top right column were synthesized by our algorithm.



Impressionist Expressionist Cezanne

Figure 2.7: A set of painterly rendered images and an input from a Still Life by Paul Cezanne form an image matrix. The images in impressionist and expressionist style on the lower row, and the two images on the top right column were synthesized by our algorithm.

Best space invariant linear filter Our method

Figure 2.8: Comparison: (Left) Best space invariant linear filter. (Right) Result of our method.

Chapter 3

Multilinear Shape Analysis using Level Sets

This chapter provides a framework for learning, representing and querying large data sets of shapes by multilinear analysis of level sets. Shape regions are represented by an implicit shape model that does not require dense point correspondence and handles varying topology. We analyze a large training set of shape models by multi-linear factorization of the tensor produced from their level sets. This allows to synthesize canonical samples which are independent of any subset of factors, and to reduce dimensionality while maintaining the variability of known factors. We demonstrate the applicability of our approach by analysis of a multidimensional corpus of ceramics shapes and sketches of archaeological pottery.

3.1 Introduction

Analysis and organization of large collections of shape classes is a fundamental problem in pattern recognition. In this work we address the issues of shape representation, organization, and analysis of a large data set, to enable efficient classification and retrieval.

Real-world data sets are relatively large and characterized by varying shapes that consist of both smooth and highly curved regions. Point based shape representations require determining dense point correspondence or a set of corresponding features that describe each shape. For a collection consisting

of a few hundred shapes and which requires scalability, it is not feasible to manually match the shapes or find point-wise correspondence between them. In addition point-based shape representations are not suitable for handling shapes of different or changing topology, or shapes with regions of high curvature. We therefore use an implicit region-based shape model. The input shapes are encoded as binary images and represented using a function in which the shape boundaries are a level set.

We aim at finding a unified representation that captures the relationships among different shapes in the set while maintaining the information on the interaction between shape factors. In this regard, approaches such as principle component analysis or multiple discriminant analysis find projections that best represent or separate data in a least squares sense. In principle component analysis a set of random variables is transformed linearly and orthogonally such that the new variables are uncorrelated. The first few components account for a large proportion of the variance of the variables. Another view is as an approximation of the variance matrix by a matrix of lower rank. Therefore, principle component analysis is used as a linear dimensionality reduction technique which identifies orthogonal directions of maximum variance in the original data, and projects the data into a lower dimensionality space formed by a subset of the highest-variance components. However, representing each sample by a linear combination of the basis is suitable for collections consisting of a single linear factor.

A general mathematical perspective for describing the variability of shapes is to regard a shape, which is represented by a vector of implicit function values, as a point in an abstract shape space, and then to characterize a set of shapes by a manifold in this high-dimensional shape space. In this work we take a supervised approach and assume that multiple interesting or important factors of shape are known. We therefore organize the data set into a multi-dimensional array, in which each axis represents a specific factor or shape attribute, and each entry is an implicit function whose level set represents shape. This defines a grid of points in the high-dimensional shape space, and the representation takes into account the interactions between multiple factors as apposed to analysis of a single factor.

In a similar manner to which matrices correspond to bilinear forms, multi-dimensional arrays or tensors correspond to general multi-linear forms, and are suitable for analysis and processing of multi-dimensional data sets. Tensor decompositions are useful for multi-mode component analysis, compressing multi-dimensional arrays, and clustering. For a large data set consisting of a multi-dimensional array or tensor, multi-mode component analysis attempts to discover relations in the multi-way data set and

reduce each of the dimensions. This is realized by a decomposition of the tensor into tensor-matrix products. This decomposition provides a multi-linear model of the collection. Each matrix in the decomposition represents a linear model and their tensor-matrix composition is a multi-linear model of interactions between them. In particular, one of the matrices involved in the decomposition represents the eigenvectors as in PCA, and therefore this analysis supersedes a single factor linear model. Perhaps more importantly, it allows representing canonical samples from the data set independent of any subset of factors, paying attention to certain factors while being independent of others, which is key for classification.

We demonstrate the applicability of our approach for the specific domain of archeological data of pottery shapes which is characterized by multiple factors. In a general setting, this work is the first on multi-linear analysis of shape data represented by level sets of functions.

3.1.1 Related work

Principle component analysis transforms a basis to an orthogonal coordinate system formed by eigenvectors of the covariance matrix. Application of PCA to a set of aligned face images, eigen-faces [114], represents each face by a linear combination of basis images.

Based on the ideas developed by Osher and Sethian [87], Malladi *et al.*[82] use a level set approach for modeling shape. Tasi *et al.*[111] apply PCA to a collection of signed distance representations of shape for segmentation of medical images. Morphable models of 3D faces [16] and whole body scans [2] morph between all meshes and textures in full correspondence by linear combinations. The weights do not correspond to natural shape attributes as some attributes are related to shape measurements such as the width or height, and others such as expression are not. Synthesis is performed by adding the difference between two instances of the same shape with different values for a specific attribute. This is not suitable for attributes that are invariant for each shape. Therefore a label is given for the markedness of each attribute of each sample and a weighted sum is computed using these labels as weights assuming that each attribute is a linear function.

The goal of local linear embedding (LLE) [99], Isomap [109], and Hessian LLE [34], is to recover a low-dimensional parameterization of high-dimensional data. Recent application to continuous data such as image sequences [92] allows analysis of the trajectory through these embedded spaces.

There are various generalizations of data decomposition to multiple dimensions. A first type, presented by Tucker [112] for three-mode analysis is known as component analysis [81]. Kroonenberg and de Leeuw [74] provide an alternating least squares algorithms for component analysis. Kapteyn *et al.*[69] extend these to n-mode component analysis. Lathauwer *et al.*[78] present tensor notation and foundations for approximations for tensor decompositions. A second generalization to multimode arrays developed by Harshman [59], and by Caroll and Chang [24] is known as PARAFAC-CANDECOMP. Kolda [73] explores different notions of orthogonality for tensor decomposition, such as the rank-one approximation to higher order tensors presented by Zhang and Golub [124].

Applications of these tools are emerging in diverse fields ranging from pattern recognition, communications, medical imaging, computer vision, to data mining. In this regard, a challenge is to apply these methods to large data sets and in this work we deal with data sets consisting of thousands to millions of entries. Harshman [60] performs factor analysis of tongue shapes in which vocal-tract shapes are represented by a set of features. These features are displacement measurements of positions along reference lines defined on tongue x-rays taken during vowel pronunciation. Tenenbaum and Freeman [110] use bilinear models to factor style and content of warped letters and aligned face images, synthesizing novel images based on the computed model. Davis and Gao [33] use three-mode component analysis for representing and recognizing styles of human actions. Vasilescu and Terzopoulos [116] apply a multilinear model as presented by Lathauwer *et al.*[78] to aligned face images. In this work we use a multilinear model of entire functions whose level set is the shape as described in the following section.

3.2 Shape representation

Shapes are commonly represented by points describing their boundaries. An alternative is to represent shape by a signed distance function, in which the shape boundaries are the zero level set as shown in Figure 3.1. Formally, the level set of c is the set of points defined by $\{(x_1,\ldots,x_n) : f(x_1,\ldots,x_n) = c\} \in R^n$, and for $n = 2$ the level set is a level curve. The advantage of such a representation over a point-based approach is the ability to handle varying topology, represent areas of high curvature, and analyze a collection of roughly aligned shapes without dense pointwise or feature correspondence.

The Euclidean distance transform of a binary image is the distance between each pixel and the

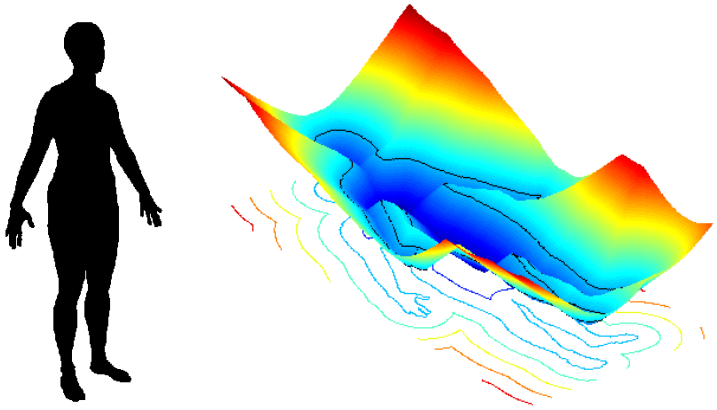


Figure 3.1: Shape representation.

nearest pixel of inverted value. The signed distance transform assigns negative values inside the shape and positive values outside. For gray scale images, in which black pixels represent the inside of a shape and white pixel the outside, there are also possible gray pixels on the boundary which is then detected by comparing the value of each pixel and two of its neighbor to a threshold. Linear interpolation using the distance transform is shown in Figure 3.2. A continuous implicit function enforcing a smoothness constraint [113] is also suitable for our purposes.

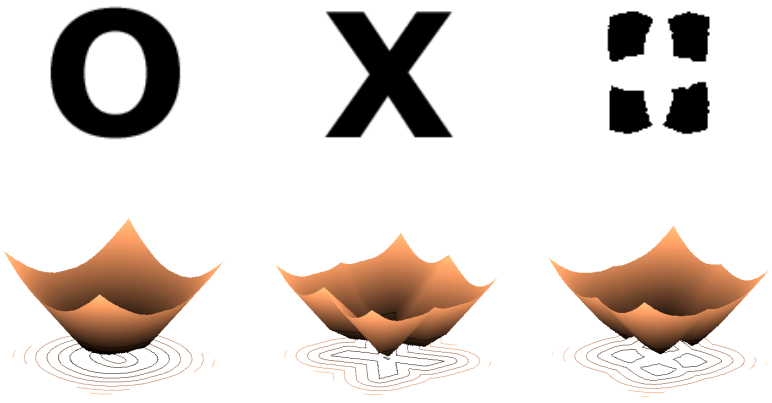


Figure 3.2: Linear shape interpolation.

3.3 Multilinear analysis

We begin by briefly describing the necessary definitions and operations on multi-dimensional arrays or tensors. The definitions are mode-k vectors, mode-k flattening, mode-k product, and k-rank (the reader is referred to Lathauwer [77] for detailed derivations). We apply a multilinear model to a large data set consisting of functions, each representing a shape by the zero level set [87] of the function.

Let A denote an n-dimensional $d_1 \times d_2 \times \cdots \times d_n$ array, or tensor of order n. The kth dimension of A is d_k, and an element of A is specified by $a_{i_1 \cdots i_n}$, where $i_j \in [1 \ldots d_k]$ for each $j \in [1 \ldots n]$. In our case, the tensor represents a collection of functions in which $\prod_{k=1}^{n-1} d_k < d_n$, where d_n is the dimension of the domain of the functions.

3.3.1 Flattening

The mode-k vectors of an n-dimensional $d_1 \times d_2 \times \cdots \times d_n$ tensor A are the vectors of dimension d_k obtained by changing i_k while fixing $i_1 \ldots i_{k-1} i_{k+1} \ldots i_n$, for a total of $d_1 \cdots d_{k-1} d_{k+1} \cdots d_n$ vectors. For example, the columns of a matrix are mode-1 vectors and the rows are mode-2 vectors (and the time spans of a video cube are mode-3 vectors).

The mode-k flattening of an n-dimensional $d_1 \times d_2 \times \cdots \times d_n$ tensor A into a $d_k \times (d_{k+1} \cdots d_n d_1 \cdots d_{k-1})$ matrix $\mathbf{A}_k$ is defined by unfolding the tensor such that the d_k dimensional mode-k vectors form the columns of the mode-k flattening. For example, for a three dimensional array (yxz cube), which is an order three $d_1 \times d_2 \times d_3$ tensor A, the mode-1 flattening $\mathbf{A}_1$ is the matrix $[A(:,1,:) \cdots A(:,d_2,:)]_{d_1 \times (d_2 d_3)}$ (concatenating d_2 slices of yz planes), the mode-2 flattening $\mathbf{A}_2$ is the matrix $[A(:,:,1)^{\mathrm{T}} \cdots A(:,:,d_3)^{\mathrm{T}}]_{d_2 \times (d_3 d_1)}$ (concatenating d_3 slices of xy planes), and the mode-3 flattening $\mathbf{A}_3$ is the matrix $[A(1,:,:)^{\mathrm{T}} \cdots A(d_1,:,:)^{\mathrm{T}}]_{d_3 \times (d_1 d_2)}$ (concatenating d_1 slices of xz planes).

3.3.2 Decomposition

The mode-k product $\times_k$ of an n-dimensional $d_1 \times d_2 \times \cdots \times d_n$ tensor A by a $r \times d_k$ matrix $\mathbf{U}$ is denoted by $A \times_k \mathbf{U}$ and defined as the n-dimensional $d_1 \times d_{k-1} \times r \times d_{k+1} \ldots \times d_n$ tensor B such that:

$$b_{i_1 \ldots i_{k-1} j i_{k+1} \ldots i_n} = \sum_{i_k=1}^{d_k} a_{i_1 \ldots i_k \ldots i_n} u_{j i_k} \tag{3.1}$$

for $j \in [1 \dots r]$. For a tensor A and matrices $\mathbf{U}_{p\times p}$, $\mathbf{V}_{q\times q}$, the mode-p and mode-q products maintain $(A \times_p \mathbf{U}) \times_q \mathbf{V} = A \times_q (\mathbf{V} \times_p \mathbf{U}) = A \times_p \mathbf{U} \times_q \mathbf{V}$, and for a matrix $\mathbf{V}_{p\times p}$ the mode-p products satisfy $A \times_p \mathbf{U} \times_p \mathbf{V} = A \times_p \mathbf{UV}$.

The mode-k product $B = A \times_k \mathbf{U}$ is computed according to $\mathbf{B}_k = \mathbf{U}\mathbf{A}_k$, by multiplying $\mathbf{U}$ by the flattened A and reorganizing the tensor B from its mode-k flattening $\mathbf{B}_k$.

A tensor is decomposed into a multilinear model of factors defined by the tensor-matrix products:

$$A = C \times_1 \mathbf{U}_1 \times_2 \cdots \times_n \mathbf{U}_n, \tag{3.2}$$

where C is the core tensor, and $\mathbf{U}_1, \dots, \mathbf{U}_n$ are the mode matrices. Each mode matrix $\mathbf{U}_k$ is computed by taking the left matrix of the SVD of the mode-k flattening $\mathbf{A}_k$. For large rectangular matrices this is performed efficiently by SVD of a multiplication by transpose, and with QR factorization by modified Gram-Schmidt, according to dimensions.

The core tensor is then computed by the tensor-matrix products $C = A \times_1 \mathbf{U}_1^{\mathrm{T}} \times_2 \cdots \times_n \mathbf{U}_n^{\mathrm{T}}$ by reorganizing $\mathbf{U}_1^{\mathrm{T}}\mathbf{A}_1(\mathbf{U}_2 \otimes \cdots \otimes \mathbf{U}_n)$ which is the Kronecker tensor product.

3.3.3 Truncation

The k-rank of a tensor A is the dimension of the vector space generated by the mode-k vectors of A, which is the rank of the mode-k flattening of A, and is denoted by $R_k = rank(\mathbf{A}_k)$.

3.4 Multilinear shape analysis

We provide an example illustrating the strengths of multilinear decomposition combined with an implicit shape representation. Figure 3.3 shows a $2 \times 2 \times 2$ array of functions where the indices appear in parenthesis. The domain of each function is a 32^2 grid (a total of 8192 real values), defining a four dimensional tensor $2 \times 2 \times 2 \times 32^2$. The factors are the prototype (column or jar), width (thin or fat) and convexity (or concavity). This can be considered as a grid of points in a high-dimensional shape space in which the level set of each implicit function is a point, and varying the value of the level set adds a degree of freedom in the space.

The tensor decomposition allows to form canonical samples independent of any subset of factors.

This is demonstrated for a one and two factors in Figure 3.4 where the independent factors are denoted by an asterisk. The first row show independence of the second factor by computing the tensor $I_2 = C \times_1 \mathbf{U}_1 \times_3 \mathbf{U}_3 \times_4 \mathbf{U}_4$ and varying $I_2(i,1,k,:)$ for $i,k \in 1,2$. The second row shows independence of the third factor resulting from computation of the tensor $I_3 = C \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_4 \mathbf{U}_4$ and varying $I_3(i,j,1,:)$ for $i,j \in 1,2$. The results take into account the interaction between factors and the remaining components of the tensors contain the variability for the factors, as in principle component analysis. The last row demonstrates dependence on a single factor, and is computed from the tensor $I_{23} = C \times_1 \mathbf{U}_1 \times_4 \mathbf{U}_4$ by changing $I_{23}(i,1,1,:)$ for $i \in 1,2$. This allows paying attention to certain factors while being invariant of others which is important for classification. Total computation time is 19 seconds on a 1.8 GHz PC processor with 1GB of RAM, including the flattening, tensor decomposition and tensor-shape computations.

3.4.1 Pottery shape analysis

Pottery is used for storage, processing, and transfer. Many vessels have multiple uses and each function may require different attributes of shape, and therefore form and function are often linked. Archaeologist use various methods to learn the function of vessels by analogy and inference. The methods are based on extrinsic factors such as written records, location of archeological recovery, physical residues, and usage, and on intrinsic factors such as decoration style and shape [98].

Ceramics can be characterized according to parts and their proportions. Three main components are orifice, body and base. Secondary features include spout, handle, supports, flange, neck, collar, lip and rim. Archaeological ceramic reports typically illustrate these using profiles with thickness as shown in Figure 3.5 (left), and older excavation references use sketches as shown in Figure 3.5 (right).

Abydos data set analysis

A common classification scheme related to usage distinguishes between classes such as plate, dish, bowl, jar and vase based on ratios of height and maximum diameter, with additional refinements. A coarser classification divides vessels into flat and hollow, and differentiates based on volume. Additional vessel shape classification schemes are based on contours, characteristic points on the silhouette, and solid geometry [42].

The Abydos data set includes 491 sketches of ancient pottery from the Abydos excavation by Petrie in 1902 [90]. The excavation manuscripts were scanned and manually preprocessed by cropping each individual sketch, and applying a threshold. Of the 491 sketches we use 96 samples organized in a $8 \times 3 \times 4 \times (128)^2$ multidimensional array (a total of 1572864 real values). The first factor includes the prototypes shown in Figure 3.6. Several functions of the two remaining factors, width and height, for a single prototype are shown in Figure 3.8. Organization of the set is performed manually, in a hierarchical fashion, each factor at a time, and requires a few manual comparisons within each factor, as the notion of prototypes and largest and smallest value of an attribute are quite intuitive. The level set functions of all the samples in the data set are pre-computed once and the tensor is then flattened for each mode $k = 1 \ldots 4$. The tensor is then decomposed as described in Section 3.3.2 and additional tensors are computed as shown in Figure 3.7. Computation time is 126 seconds for flattening all the modes, 13 seconds for computing the mode matrices, and 233 seconds for incrementally computing the core tensor C. Computation of tensors independent of factors is performed by a tensor-matrix product of the core with the dependent mode and is between 39 and 43 seconds for each tensor. This analysis supersedes PCA as the columns of matrix $\mathbf{U}_4$ contain the eigen-shapes of level set functions which describe the variation for the entire collection.

We have compared our method for shape analysis with the mean level sets over the independent factors using the Abydos dataset. The top row of Figure 3.9 shows the mean denoted by μ over the second and third factors given that the other two are fixed, as well as the mean over both of these factors given the that first is constant. The bottom row shows our results which better preserve the features. The independent factor is denoted by an asterisk and correspond to Figure 3.4.

Ceramics data set analysis

The types and appearance of vessels used in communities extend beyond function and shape. In contemporary communities, pottery may be classified into categories by its makers and users. These labels are called folk classifications [71] and are based on shape characteristics such as width to height ratio, neck position, emphasizing features such as handle and spout.

Our second data set includes 720 ceramics shapes [71] and the factors are the set of prototypes with two independent and continuous shape variations. The level set functions of all the samples in the

data set form a four dimensional tensor $10 \times 8 \times 9 \times (140^2)$ for a total of over 14 million real values. This raw tensor is first stored in a file. Several samples of functions for three factors of the tensor are shown in Figure 3.10. The tensor is then flattened and each mode-k flattened matrix for $k = 1 \ldots 4$ is stored. The tensor is then decomposed and the core tensor computed incrementally, discarding the files. Computation time is for flattening is between 270 and 313 seconds for each mode. Total computation time is thirty minutes including the tensor decomposition, computation of mode matrices, core tensor, and factor independent tensors.

3.5 Conclusions and future work

This work handles a large data set of shapes represented by a tensor of implicit functions for the first time. This is a supervised approach as the data set is organized into a tensor according to known factors. Therefore, future work will focus on unsupervised approaches such as clustering for automatically organizing the data set. This work opens extensions to surfaces and analysis of large 3D databases which will require part-based representations and decompositions.

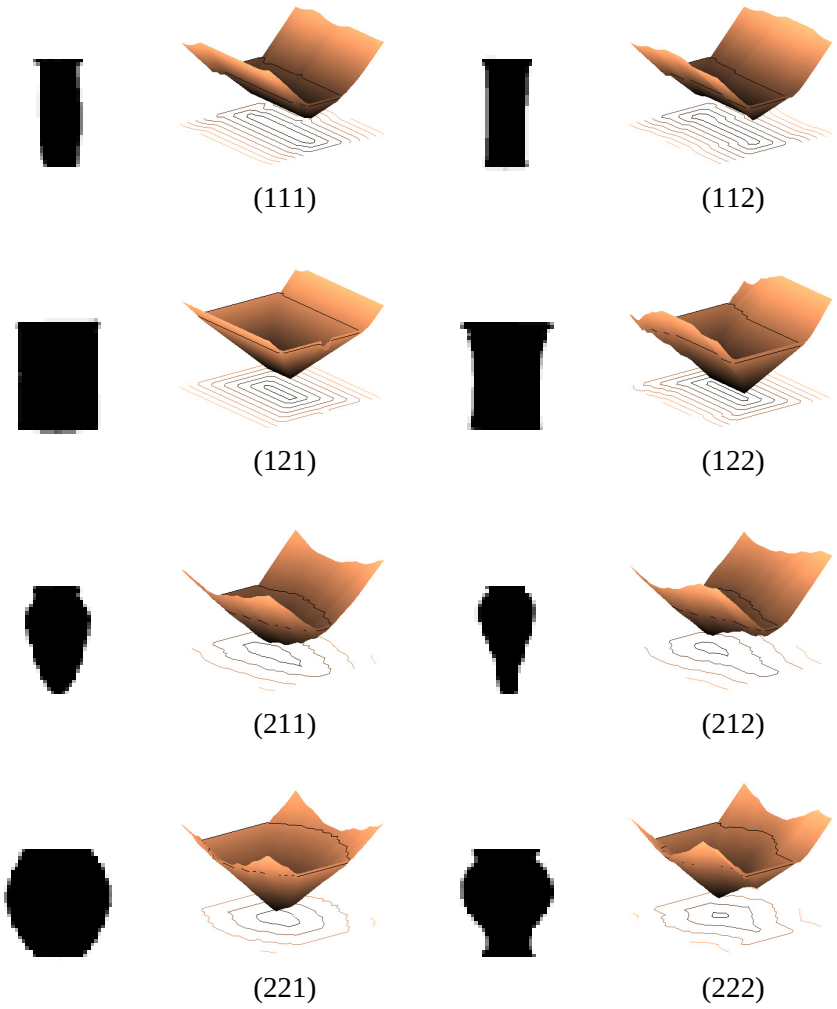


Figure 3.3: Input tensor: 2 (prototype) $\times$2 (width) $\times$2 (convexity) $\times 32^2$ (function value).

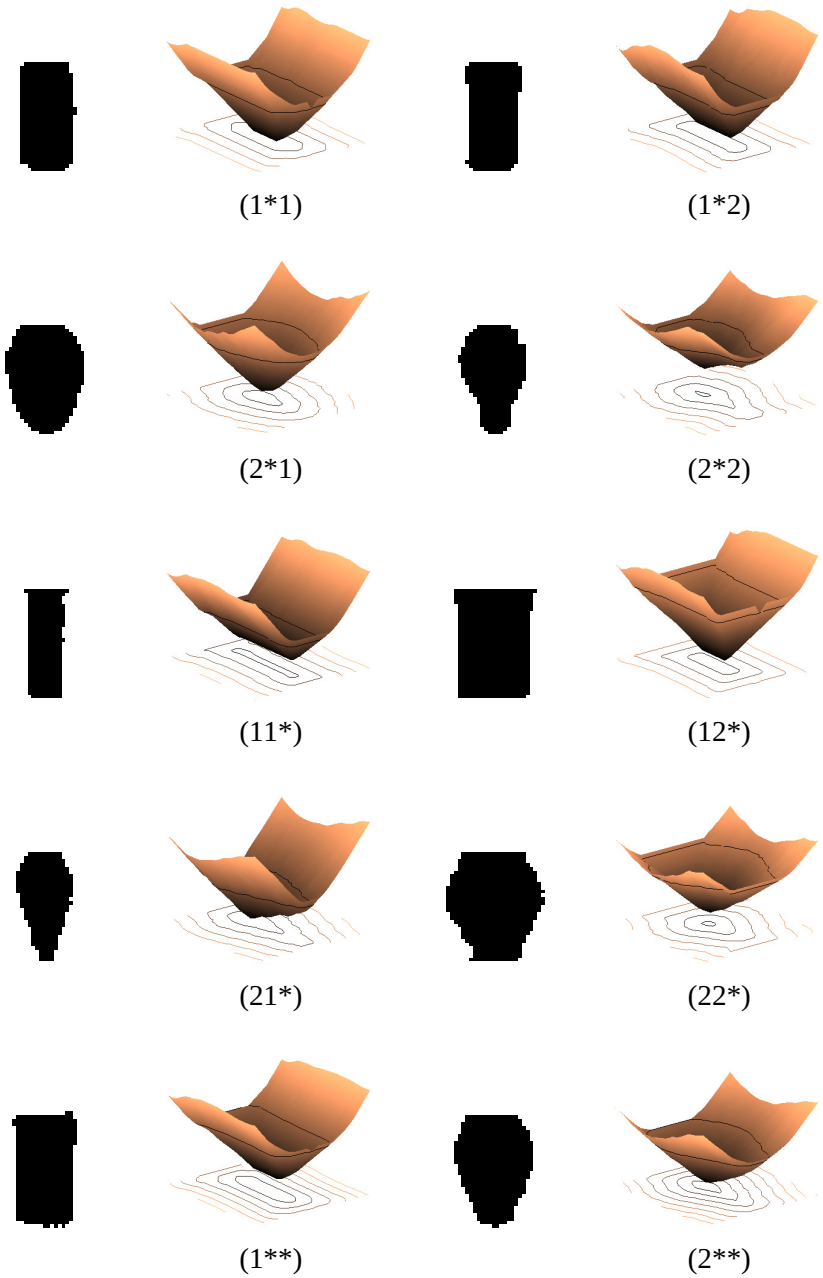


Figure 3.4: First to fourth rows: shapes independent of a single factor (indicated by *).Last row: shapes independent of two factors.

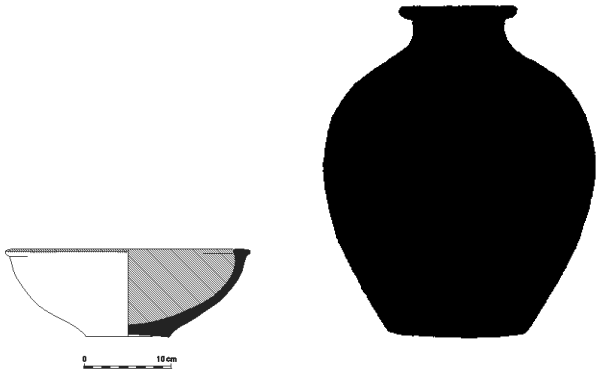


Figure 3.5: Archaeological ceramic profile and excavation sketch.

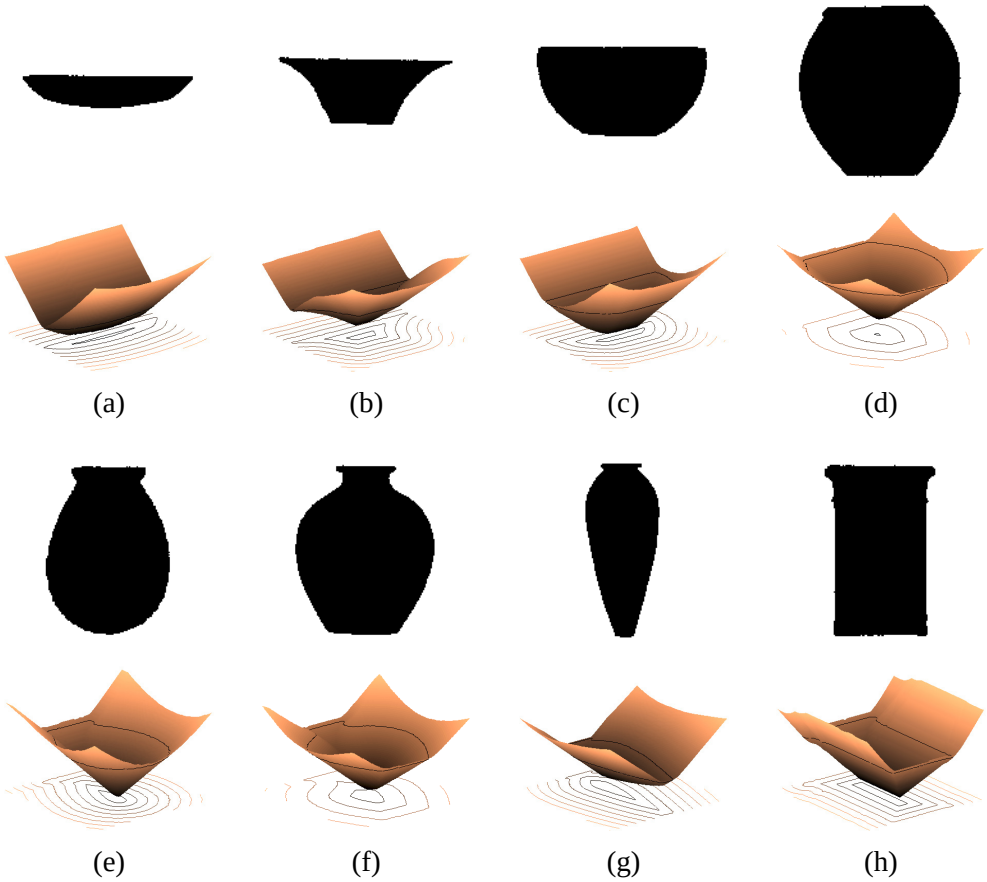
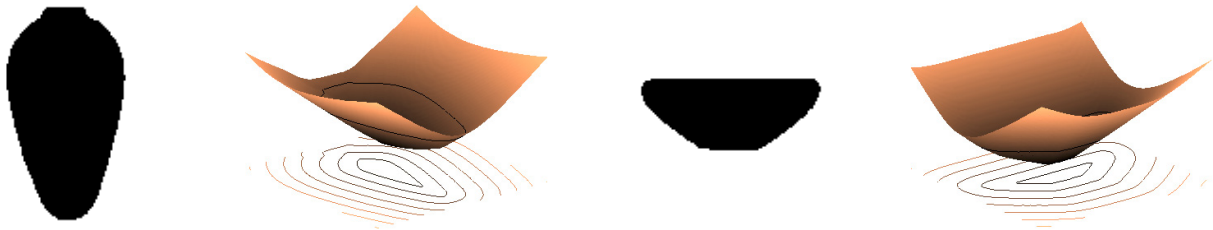
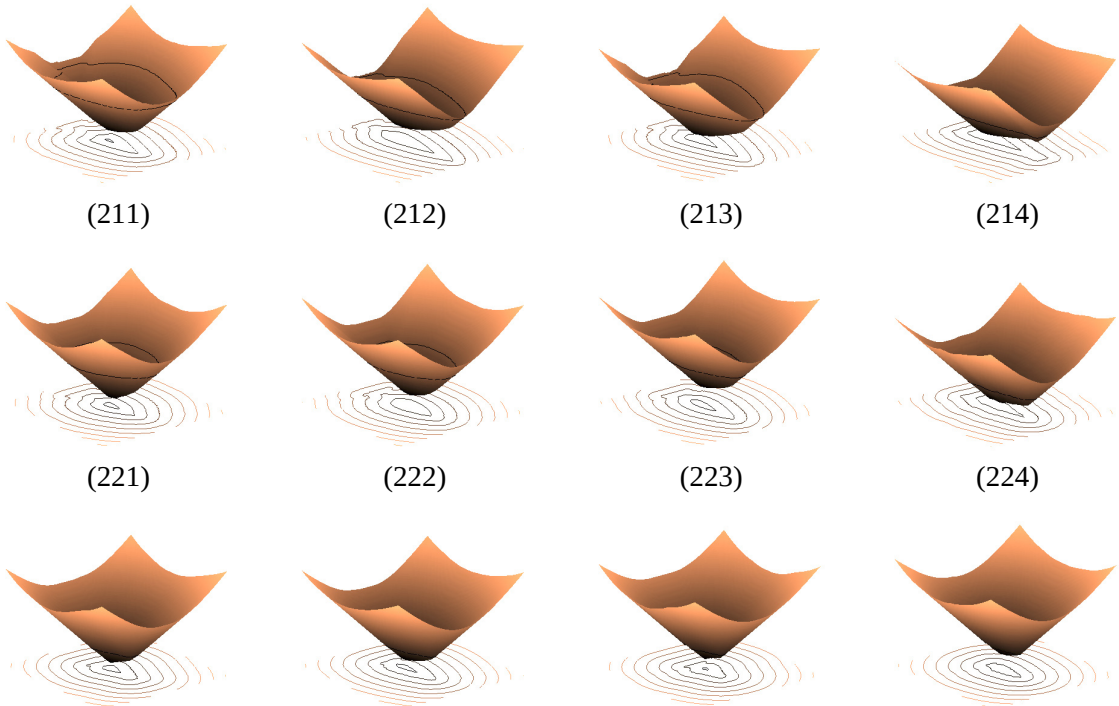


Figure 3.6: Prototype shapes from the Abydos data set: (a) plate. (b) dish. (c) bowl. (d) pot. (e) flask. (f) jar. (g) vase. (h) column.



(21*) (5*1)

Figure 3.7: Shapes independent of one factor synthesized from Abydos data set core tensor and mode matrices.



(211) (212) (213) (214)

(221) (222) (223) (224)

(231) (232) (233) (234)

Figure 3.8: Sampling of Abydos data set for a single prototype: two shape factors from four dimensional ($8 \times 3 \times 4 \times 128^2$) level set tensor.

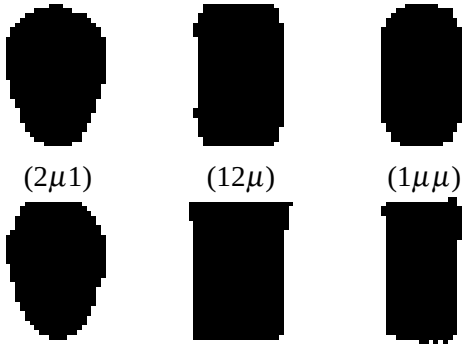


Figure 3.9: Comparison: (Top) mean. (Bottom) Result of our method.

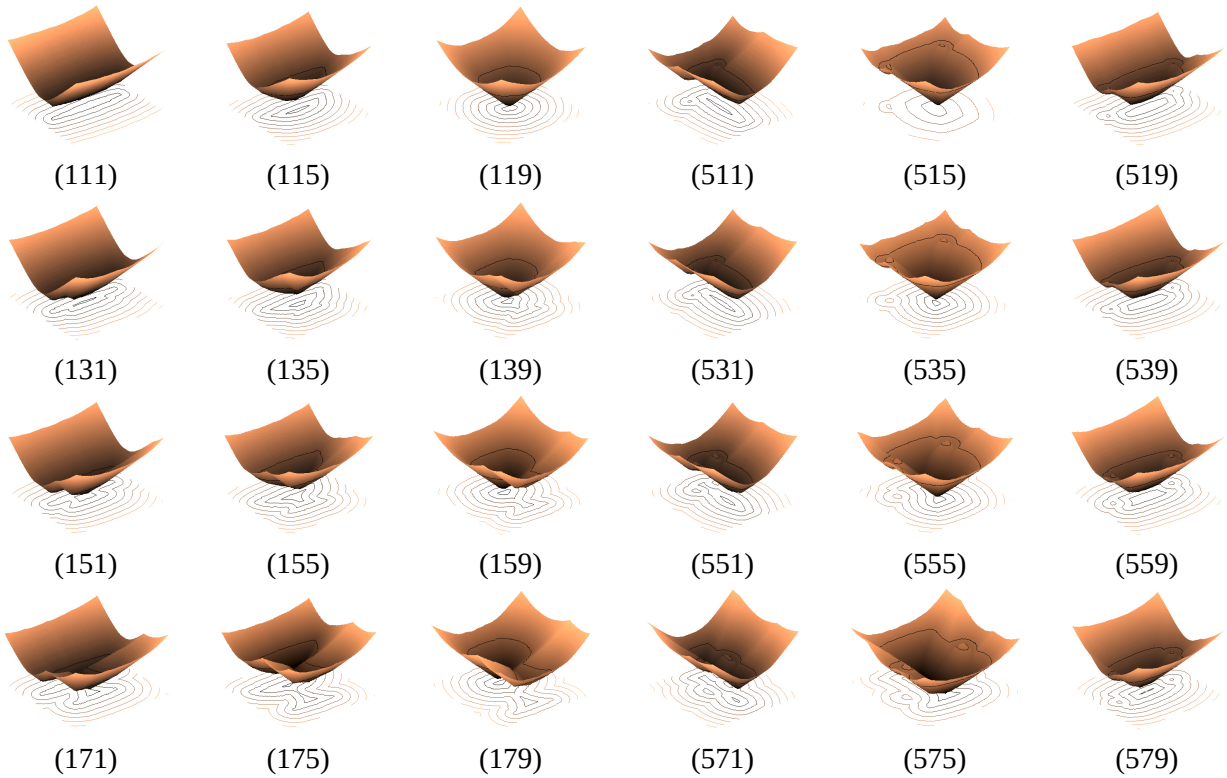


Figure 3.10: Sparse sampling of ceramics data set: three shape factors from four dimensional ($10 \times 8 \times 9 \times 140^2$) level set tensor. Notice the varying shape topology.

Part II

Image and Sound Completion

Chapter 4

Fragment-Based Image Completion

We present a new method for completing missing parts caused by the removal of foreground or background elements from an image. Our goal is to synthesize a complete, visually plausible and coherent image. The visible parts of the image serve as a training set to infer the unknown parts. Our method iteratively approximates the unknown regions and composites adaptive image fragments into the image. Values of an inverse matte are used to compute a confidence map and a level set that direct an incremental traversal within the unknown area from high to low confidence. In each step, guided by a fast smooth approximation, an image fragment is selected from the most similar and frequent examples. As the selected fragments are composited, their likelihood increases along with the mean confidence of the image, until reaching a complete image. We demonstrate our method by completion of photographs and paintings. This part was presented in SIGGRAPH 2003, and published in special issue of ACM Transactions on Graphics [36].

4.1 Introduction

The removal of portions of an image is an important tool in photo-editing and film post-production. The unknown regions can be filled in by various interactive procedures such as clone brush strokes, and compositing processes. Such interactive tools require meticulous work driven by a professional skilled artist to complete the image seamlessly. Inpainting techniques restore and fix small-scale flaws in an image, like scratches or stains [65, 9]. Texture synthesis techniques can be used to fill in regions with

stationary or structured textures [40, 118, 41]. Reconstruction methods can be used to fill in large-scale missing regions by interpolation. Traditionally, in the absence of prior knowledge, reconstruction techniques rely on certain smoothness assumptions to estimate a function from samples. Completing large-scale regions with intermediate scale image fragments remains a challenge.

Visual perceptual completion is the ability of the visual system to "fill in" missing areas [85] (partially occluded, coinciding with the blind spot, or disrupted by retinal damage). While the exact mechanisms behind this phenomenon are still unknown, it is commonly accepted that they follow some Visual Gestalt [72] principles, namely, completion by frequently encountered shapes that result in the simplest perceived figure [88]. Motivated by these general guidelines, we iteratively approximate the missing regions using a simple smooth interpolation, and then add details according to the most frequent and similar examples.

Figure 4.1: From left to right: the input image, and inverse matte that defines the removal of an element, the result of our completion, and the region completed by our algorithm.

Problem statement: Given an image and an inverse matte, our goal is to complete the unknown regions based on the known regions, as shown in Figure 4.1.

In this work, we present an iterative process that interleaves smooth reconstruction with the synthesis of image fragments by example. The process iteratively generates smooth reconstructions to guide the completion process which is based on a training set derived from the given image context.

The completion process consists of compositing image fragments and can be regarded as a "push-background" process, in contrast to the "pull-foreground" processes associated with image matting. Our completion approach requires a relevant training set and a degree of self-similarity within the input image. It is an image-based 2D method that does not incorporate high-level information and can therefore produce unnatural looking completions.

This chapter is organized as follows. After a survey of related work, we present an overview of the entire completion process (Section 4.2), and then describe each component of the method in detail. We introduce a fast approximation method (Section 4.3), and present a confidence map and traversal order (Section 4.4). Next, we describe the search for similar fragments and their adaptive neighborhood size (Section 4.5). In Section 4.6 we present the compositing of image fragments. Completion is performed from coarse-to-fine, as described in Section 4.7. Finally we show the results of our method on various photographs and paintings (Section 6.7), and discuss its limitations (Section 4.9).

4.1.1 Related work

Many operations ranging from low-level vision tasks to high-end graphics applications have been efficiently performed based on examples:

Hertzmann *et al.* [63] study a mapping of spatially local filters from image pairs in correspondence, one a filtered version of the other. A new target image is filtered by example. Pixels are assigned values by comparing their neighborhood, and those of a coarser level in a multi-resolution pyramid, to neighborhoods of a training image pair. Considering the original image as output and taking its segmentation map as input allows texture to be painted by numbers [58]. A new image that is composed of the various textures is then synthesized by painting a new segmentation map. Swapping the output segmentation map with the original input image results in example-based segmentation [17].

Freeman *et al.* [51, 49] derive a model used for performing super-resolution by example. The technique is based on examining many pairs of high-resolution and low-resolution versions of image patches from several training images. Baker and Kanade [6] apply this technique restrictively to the class of face images. Given a new low-resolution face image, its corresponding high-resolution image is inferred by re-using the existing mapping between individual low-resolution and high-resolution face patches.

The example-based image synthesis methods described above use a supervised training set of corresponding pairs. In our work, the examples provide the likelihood of a given context appearing in the given image.

There is considerable work on texture synthesis, of which the most notable are based on Markov Random Fields [40]. A new texture is incrementally synthesized by considering similar neighborhoods in the example texture. These techniques synthesize texture which is both stationary and local [118].

Igehy and Periera [66] replace image regions with synthesized texture [61] according to a given mask. Texture transfer [5] adds the constraint that the synthesized texture match an example image. This yields the effect of rendering a given image with the texture appearance of a training texture. This technique is extended to color transfer [120], a special case of image analogies, by matching local image statistics. Efros and Freeman [41] introduce a simple and effective texture synthesis technique that synthesizes a new texture by stitching together blocks of existing example texture. The results depend on the size of a block which is a parameter tuned by the user that varies according to the texture properties. In this work, we synthesize both local and global structures. To capture structures of various sizes, similarly to hierarchical pattern matching [105], we take an adaptive approach, where fragments have different sizes based on the underlying structure.

Image inpainting [9, 26] fills in missing regions of an image by smoothly propagating information from the boundaries inwards, simulating techniques used by professional restorators. However, the goals of image inpainting techniques and image completion are different. Image inpainting is suitable for relatively small, smooth, and non-textured regions. In our case the missing regions are large, and consist of textures, large-scale structures, and smooth areas. Recently, Bertalmio *et al.* [10] combine image inpainting with texture synthesis by decomposing an image into the sum of two components. Inpainting [9] is applied to the component representing the underlying image structure, whereas texture synthesis [40] is separately applied to the component representing image detail, and the two components are then added back together.

There are a number of approaches related to image completion in the Computer Vision literature, but most of them focus on the edge and contour completion aspect of the problem. Edge completion methods find the most likely smooth curves that connect edge elements, usually by minimizing a function based on curvature [57]. Given a grid of points and orientations as elements, completion methods consider the space of all curves between pairs of elements, and the pairwise interaction between elements. The likelihood that any two elements are connected defines a field for each element, and completion is performed by a summation over all fields [121]. Sharon *et al.* [103] take into account edge elements at various scales, and use a multi-grid method to accelerate computations.

Our work is also related to photo-editing techniques. Oh *et al.*[86] incorporate some depth information into photo-editing, whereas our method is a 2D image-based technique, which has no notion of the

underlying scene. Brooks and Dodgson [22] present an image editing technique that is based on texture self-similarity, editing similar neighborhoods in different positions. Our method finds self-similarities in the image under a combination of transformations: translation, scale, rotation and reflection.

4.2 Image completion

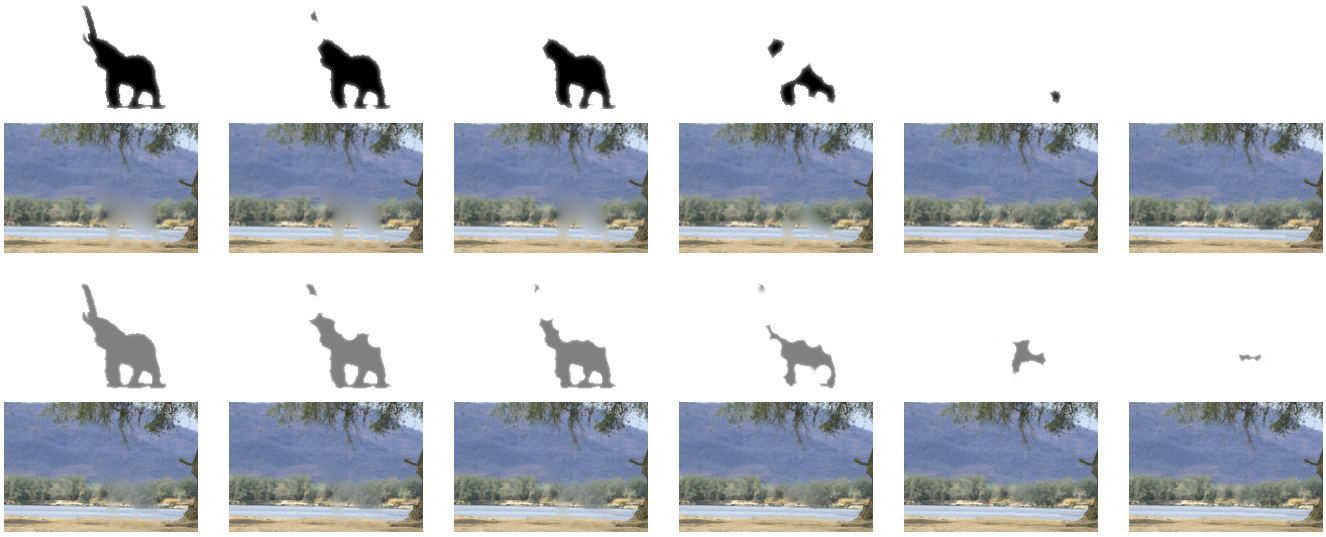


Figure 4.2: Completion process: confidence and color of coarse level (top row) and fine level (bottom row) at different time steps. The output of the coarse level serves as an estimate in the approximation of the fine level.

We assume that foreground elements or background regions are roughly marked with an image editing tool, or a more accurate α channel is extracted using a matting tool. This defines an *inverse matte* $\bar{\alpha}$ that partitions the image into three regions: the known region, where $\bar{\alpha}_i = 1$; unknown region, where $\bar{\alpha}_i = 0$; and, optionally, a gray region, where $0 < \bar{\alpha}_i < 1$ for each pixel i, and "inverts" the common definition of trimaps that are generated in the process of pulling a matte and foreground elements from an image [27]. We require a *conservative* inverse matte that, at least, contains the entire extracted region. As in digital image matting, the regions of the inverse matte are not necessarily connected. The inverse matte defines a confidence value for each pixel. Initially, the confidence in the unknown area is low. However, the confidence of the pixels in the vicinity of the known region is higher. The confidence

values increase as the completion process progresses.

Our approach to image completion follows the principles of *figural simplicity* and *figural familiarity*. Thus, an *approximation* is generated by applying a *simple* smoothing process in the low confidence areas. The approximation is a rough classification of the pixels to some underlying structure that agrees with the parts of the image for which we have high confidence. Then the approximated region is augmented with *familiar* details taken by example from a region with higher confidence.

All of these processes are realized at the image fragment level. A fragment is defined in a circular neighborhood around a pixel. The size of the neighborhood is defined adaptively, reflecting the scale of the underlying structure. Image completion proceeds in a multi-scale fashion from coarse to fine, where first, a low resolution image is generated and then the results serve as a coarse approximation to the finer level. For every scale we consider neighborhoods in level sets from high to low confidence. Figure 4.2 shows the confidence and color values at different time steps in each scale.

At each step, a *target* fragment is completed by adding more detail to it from a *source* fragment with higher confidence. Typically, the target fragment consists of pixels with both low and high confidence. The pixel values which are based on the approximation generally have low confidence, while the rest of the fragment has higher confidence. For each target fragment we search for a suitable matching source fragment, as described in Section 4.5, to form a coherent region with parts of the image which already have high confidence. The search is performed under combinations of spatial transformations to extend the training set and make use of the symmetries inherent in images. The source and target fragments are composited into the image as described in Section 4.6. The algorithm updates the approximation after each fragment composition. As fragments are added, the mean confidence of the image converges to one, completing the image.

A high-level description of our approach appears in Figure 4.3. In the pseudocode, the following terms are emphasized: (i) approximation, (ii) confidence map, (iii) level set, (iv) adaptive neighborhood, (v) search, and (vi) composite. These are the building blocks of our technique. In the following sections we elaborate on each in detail.

Input: image C, inverse matte $\bar{\alpha}$ ($\exists$ pixel with $\bar{\alpha} < 1$)

Output: completed image, $\bar{\alpha} = \mathbf{1}$

Algorithm:

for each scale from coarse to fine
    approximate image from color and coarser scale
    compute **confidence map** from $\bar{\alpha}$ and coarser scale
    compute **level set** from confidence map
    while mean confidence $< 1 - \varepsilon$
        for next target position p
            compute **adaptive neighborhood** $N(p)$
            search for most similar and frequent source match $N(q)$
            composite $N(p)$ and $N(q)$ at p, updating color and $\bar{\alpha}$
            compute **approximation**, **confidence map** and update **level set**

Figure 4.3: Image completion pseudocode.

4.3 Fast approximation

A fast estimate of the colors of the hidden region is generated by a simple iterative filtering process based on the known values. The estimated colors guide the search for similar neighborhoods, as described in Section 4.5. Our completion approach adds detail by example to the smooth result of the fast approximation. The process of approximating a given domain with values that "agree" with some known values is known as scattered data interpolation. Typically, the assumption is that the unknown data is smooth, and various methods aim at generating a smooth function that passes through or close to the sample points. In image space methods, the approximation can be based on simple discrete kernel methods that estimate a function f over a domain by fitting a simple model at each point such that the resulting estimated function $\hat{f}$ is smooth in the domain. Localization is achieved either by applying a kernel K that affects a neighborhood ε, or by a more elaborate weighting function. A simple iterative filtering method known as push-pull [54] is to down-sample and up-sample the image hierarchically using a local kernel at multiple resolutions. In the coarser levels, the kernel filter affects larger regions and the values of the samples percolate, yielding smoother data. When applied to finer resolutions, the effect is localized and higher frequencies are approximated.

The above simple process is accelerated and refined by employing a multi-grid method. The image C

is pre-multiplied by the inverse matte $\bar{C} = C\bar{\alpha}$, and the approximation begins with $\bar{C} + \alpha$. Let $l = L, \ldots, 1$ denote the number of levels in a pyramid constructed from the image. Starting from an image $Y_{t=0}^{l=L}$ of ones, the following operations are performed iteratively for $t = 1, \ldots, T(l)$:

$$Y_{t+1}^{l} = (Y_t^l \alpha + \bar{C})(*K_\varepsilon \downarrow)^l (\uparrow *K_\varepsilon)^l. \tag{4.1}$$

Equation (4.1) consists of re-introducing the known values $\bar{C}$, then l times down-sampling $\downarrow$ with a kernel K_ε, and l times up-sampling $\uparrow$ with a kernel K_ε. It is applied repeatedly (subscript t is incremented) until convergence $Y_{t+1}^l = Y_t^l$.

Applying this process for $l = L$ scales results in a first approximation $Y_{t=T(L)}^{L}$. Then (superscript l is decremented) the known values are re-introduced, and the process is repeated for $l - 1$ scales,

$$Y_{t=0}^{l-1} = Y_{t=T(l)}^{l} \alpha + \bar{C}. \tag{4.2}$$

For $l = 1$, the approximation is $Y_{t=0}^{l=1} = Y_{t=T(2)}^{l=2}$, and the following iterations are performed:

$$Y_{t+1}^{1} = (Y_t^1 \alpha + \bar{C}) * K_\varepsilon. \tag{4.3}$$

The final output is $C = Y_{T(1)}^{1}$.

As shown in Figure 4.4 the iterations applied to many levels (b-c), for large l, approximate the lower frequencies, while the iterations applied to few levels (d) or a single scale (e) handle higher frequencies. Table 4.1 summarizes the number of iterations, error, and computation time for each set of levels. The running times are for a 512 by 512 image, shown in Figure 4.4. Note that in the completion process, the bounding box of the unknown regions is typically much smaller, and decreases with each completion step.

Figure 4.5(a) shows an input image where part of the image is hidden by text, while the rest of the image is only visible through the text. The result of the approximation is shown in (b). The global root mean square error of luminance values in [0,1] is 0.049. Our completion method is based on image fragments, and therefore we are interested in the local error in a neighborhood around each pixel. This is illustrated in (c), which shows the inverse of the RMSE of luminance neighborhoods of radius 4 around

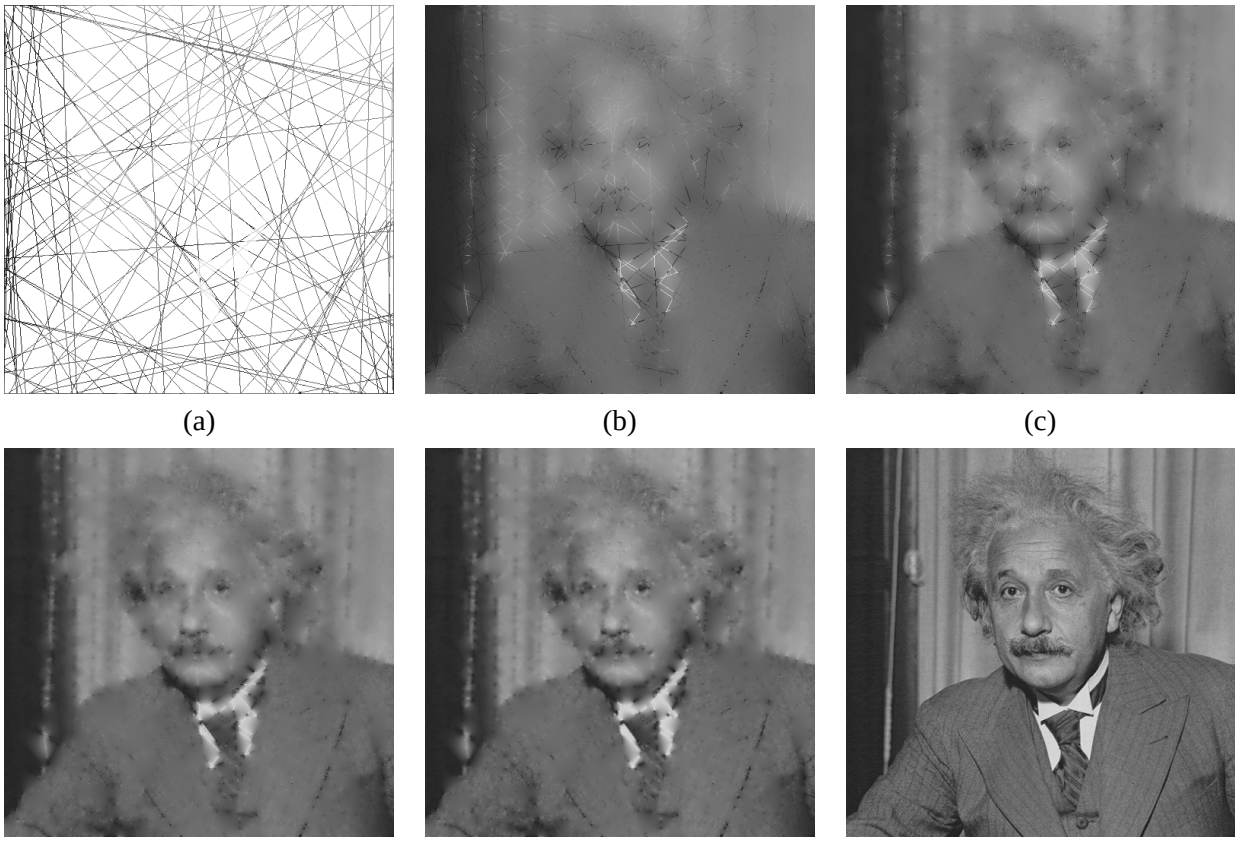


Figure 4.4: (a) Input, $\bar{C}+\alpha$: 100 lines are randomly sampled from a 512 by 512 image and superimposed on a white background. (b-e) $Y^l_{T(l)}$ for $l = 4,\ldots,1$. (e) the result of our approximation. (f) source image.

each pixel (37 samples). The source image is shown in (d).

4.4 Confidence map and traversal order

In our setting we assume an inverse alpha matte that determines the areas to be completed. The matte assigns each pixel a value in $[0,1]$. We define a confidence map β by assigning a value to each pixel i:

$$\beta_i = \begin{cases} 1 & \text{if } \bar{\alpha}_i = 1 \\ \sum_{j\in N(i)} g_j \bar{\alpha}_j^2 & \text{otherwise,} \end{cases} \tag{4.4}$$

# of levels	# of iterations	RMSE	Time (sec.)
4	197	0.0690	4.2
3	260	0.0582	5.3
2	739	0.0497	13.7
1	1269	0.0470	22.9

Table 4.1: Statistics and running times for the approximation in Figure 4.4. The levels correspond to Figure 4.4 (b-e), and the RMSE is of luminance in values [0,1]. Total computation time is 46.1 seconds for a 512 by 512 image.

where N is a neighborhood around a pixel, g is a Gaussian falloff, and $\bar{\alpha}$ is the inverse matte. The map determines how much confidence to place in image information in each pixel as new information is generated during the course of the algorithm's progress, and is used for comparing and selecting fragments (Section 4.5).

In each scale, the image is traversed in an order that proceeds from high to low confidence, while maintaining coherence with previously synthesized regions. To compute the next target position from the confidence map, we set pixels that are greater than the mean confidence $\mu(\beta)$ to zero and add random uniform noise between 0 and the standard deviation $\sigma(\beta)$:

$$v_i = \begin{cases} 0 & \text{if } \beta_i > \mu(\beta) \\ \beta_i + \rho[0, \sigma(\beta)] & \text{otherwise.} \end{cases} \tag{4.5}$$

This defines a level set of candidate positions. At each iteration we retrieve the image coordinates of the pixel with maximal value in Eq. 4.5. This determines the position of the next search and composite. After compositing (Section 4.6), the set of candidate positions is updated by discarding positions in the set that are within the fragment. Once the set of candidate positions is empty it is recomputed from the confidence map. As the algorithm proceeds, the image is completed, $\mu(\beta) \to 1$ and $\sigma(\beta) \to 0$. The width of the zone from which the candidates for the next target position are selected, and the tolerance, decrease as the image is completed.

Figure 4.6 shows, from left to right, the inverse matte, confidence map, and level set, at two different time steps. The confidence map β is visualized on a logarithmic scale $h(\beta) = \lg(\beta + 1)$, with $h = 1$ shown in green, $h \leq 0.01$ in purple, and $0.01 < h < 1$ in shades of blue.

4.5 Search

This section describes a 2D pattern matching method that considers the confidence of each pixel, in addition to the similarity of features used in traditional template matching. For each target fragment T, we search for the best source match S over all positions x,y, five scales l, and eight orientations θ: denoted as the parameter $r = (l,x,y,\theta)$. The algorithm adds detail to the approximated pixels ($\beta_i < 1$) by example without modifying the known pixels ($\beta_i = 1$). The confidence map is used for comparing pairs of fragments by considering corresponding pixels $s = S(i)$ and $t = T(i)$ in both target and source fragment neighborhoods N. For each pair of corresponding pixels, let β_s, β_t denote their confidence, and $d(s,t)$ denote the similarity between their features. The measure is L_1 and the features are color and luminance for all pixels, and gradients (1st order derivative in the horizontal, vertical, and diagonal directions) in the known pixels. We find the position, scale and orientation of the source fragment that minimizes the function:

$$r^* = \arg\min_r \sum_{s=S_r(i),t=T(i),i\in N} (d(s,t)\beta_s\beta_t + (\beta_t - \beta_s)\beta_t). \tag{4.6}$$

The first term of this function, $d(s,t)\beta_s\beta_t$, penalizes different values in corresponding pixels with high confidence in both the target and source fragments. The second term, $(\beta_t - \beta_s)\beta_t$, rewards pixels with a higher confidence in the source than in the target, while penalizing pixels with lower confidence in the source than in the target, normalized by the target confidence. The goal of these two terms is to select an image fragment that is both coherent with the regions of high confidence and contributes to the low confidence regions.

Source fragments are first trivially rejected based on luminance mean and variance. We consider a small set of $k = 5$ nearest neighbors according to Eq. 4.6 and take the most frequent, based on the luminance statistics.

Figures 4.7, 4.8 and 4.9 show pairs of matching neighborhoods outlined by the same color. The center of each target neighborhood is marked with a cross. A smaller or larger source neighborhood than target means a match across different scales. Note that in the completed region in Figure 4.7 (d) there are artifacts in the lower left portion, along the coast-line, where the dark and bright colors of opposite sides of the large missing region meet.

4.5.1 Adaptive neighborhood

An adaptive scheme to determine the size of the neighborhood is important for capturing features of various scales. Our algorithm uses existing image regions whose size is inversely proportional to the spatial frequency. We therefore tested several criteria to determine the adaptive size of a neighborhood [53]. The first is the set of statistical moments of the luminance histogram of an element. If x is a random variable denoting luminance values and $p(x_i)$ the corresponding histogram for k distinct values, then the n-th moment of x about the mean m is $\mu_n(x) = \sum_{i=0}^{k}(x_i - m)^n p(x_i)$. The second is the measure of uniformity of an element $u = \sum_{i=0}^{k} p^2(x_i)$. The third is the average entropy $e = -\sum_{i=0}^{k} p(x_i) \lg p(x_i)$. Considering only a small number of radii 2^j for $j = 1, \ldots, l = 5$ is a quantization of the neighborhood size map. We have found that a simple contrast criterion, the absolute of difference between extreme values across channels, yields results which are nearly as good as the other, more elaborate measures that we have tried.

Allowing nearby matches of large smooth neighborhoods, while discarding nearby matches of smaller textured neighborhoods avoids smearing artifacts. Searching in scale factors above 1 only for large neighborhoods allows interpolation of smooth areas while avoiding blurring of detailed regions.

Figure 4.10 shows an image and its corresponding neighborhood size map. These values are updated every completion iteration. Each pixel in (b) reflects an estimate of the frequencies in a neighborhood around the corresponding pixel in (a). Regions with high frequencies are completed with small fragments whereas those with low frequencies are completed with larger fragments.

4.6 Compositing fragments

We would like to superimpose the selected source fragment S over the target fragment T such that the regions with high confidence seamlessly merge with the target regions with low confidence. Since we give priority to the target, we need to compose the source fragment "under" the target fragment. Taking the alpha values into consideration, we apply the compositing operator: T OVER S. To create a seamless transition between T and S we apply this operator in frequency bands.

The Laplacian pyramid [23] can be used to smoothly merge images according to binary masks. The color components C of each image A and B are decomposed into Laplacian pyramids L, and the binary

masks M into Gaussian pyramids G. The Gaussian and Laplacian pyramids are separately multiplied for each image at corresponding scales k, and then added, to form a Laplacian pyramid of the merged image $L(C_{merge})$, which is then reconstructed,

$$L_k(C_{merge}) = L_k(C_A)G_k(M_A) + L_k(C_B)G_k(M_B). \tag{4.7}$$

An alpha value in $[0,1]$ is traditionally considered as either having partial coverage or as semi-transparent. The alpha channel, commonly represented by 8 bits, is used for capturing fractional occlusion to combine anti-aliased images. The classical associative operator for compositing a foreground element over a background element, F OVER B, is:

$$\begin{aligned} C_{out} &= C_F\alpha_F + C_B\alpha_B(1-\alpha_F) \\ \alpha_{out} &= \alpha_F + \alpha_B(1-\alpha_F). \end{aligned} \tag{4.8}$$

Porter and Duff [93] pre-multiplied color by alpha, and the volume rendering ray casting integral is often discretely approximated by a sequence of OVER operations.

We represent color and alpha values as matrices and plug Eq. 4.8 into Eq. 4.7 to get the Laplacian OVER operator:

$$L_k(C_{out}) = L_k(C_F)G_k(\alpha_F) + L_k(C_B)G_k(\alpha_B)G_k(\mathbf{1}-\alpha_F). \tag{4.9}$$

The alpha values α_{out} are obtained by setting $C_F = C_B = \mathbf{1}$ in Eq. 4.9.

The Laplacian OVER operator uses a wide overlapping region for the low frequencies and a narrow overlap for the high frequencies. The OVER operator is traditionally used for compositing a figure over a background, while maintaining sharp transitions according to alpha values. We use the Laplacian OVER operator for compositing two similar fragments, one over the other, to create a smooth composite.

The output is a fragment R that is premultiplied, and is masked by a circular neighborhood. To insert this result into the image, we multiply the previous approximated image by its inverse matte $\bar{C} = C\bar{\alpha}$, and then put the fragment values $R(\bar{C})$ into $\bar{C}$ and $R(\bar{\alpha})$ into $\bar{\alpha}$. In the following iteration the approximation begins with these premultiplied values.

Fragment compositing is illustrated in Figure 4.12. The top row shows the matching neighborhoods

(left) and the inverse matte (right). The alpha values, $G_k(\alpha_F)$ and $G_k(\alpha_B)$, are shown in the center row. The reconstructed color values of each intermediate term in Eq. 4.9 are shown on the bottom left. The premultiplied color and alpha output are shown on the bottom right. The output is masked by a circular neighborhood and quilted on the borders.

4.7 Implementation

Our implementation completes the image from coarse to fine. Aside from computational efficiency, coarse-to-fine completion is important for capturing features at several scales. Starting with a coarse scale corresponds to using a range of relatively large target fragments, and then using target fragments that become gradually smaller at every scale, finer and finer details are added to the completed image. The result of the coarse level C^l is up-sampled by bicubic interpolation and serves as an estimate for the approximation of the next level:

$$\hat{f}^{l+1} = \lambda \hat{f}^{l+1} + (1-\lambda)(C^l \uparrow *),$$

where $\lambda = 0.5$. The confidence values of the next level are also updated:

$$\beta^{l+1} = \lambda \beta^{l+1} + (1-\lambda)(\beta^l \uparrow),$$

where the confidence values of the coarse level are $\beta^l = \mathbf{1}$ upon completion.

The following are implementation details. We perform completion using two scales, with 192 by 128 and 384 by 256 resolution. To create a conservative inverse matte for the fine level, the inverse matte of the coarse level is up-sampled by nearest neighbors and dilated. The approximation described in Section 4.3 is a local process, and therefore performed in the bounding box of the unknown regions. We use three scales $L = 3$ in Eq. 4.2, a Gaussian kernel with standard deviation 1 in Eq. 4.1, and 0.85 for the final pass in Eq. 4.3. It is wasteful to run the approximation until convergence since the error diminishes non-linearly. First, we choose a representative subset of $\sqrt{N}$ random positions in the unknown regions, where N is the number of pixels. At each set of levels in Eq. 4.1 the approximation stops when the $\frac{\sqrt{N}}{L-l+1}$ representative values converge. The search based on Eq. 4.6 is performed at five

scales with factors spaced equally between 0.75 and 1.25, at each x,y position, and eight orientations (four rotations in increments of 90 degrees, and their four reflections). Compositing in Eq. 4.10 is performed using at most three levels *k* in the Gaussian and Laplacian pyramids for the largest fragments. The completion process terminates when $\mu(\beta) \geq 1 - \varepsilon$, as described in the pseudocode in Figure 4.3, and we set $\varepsilon = 0.05$.

4.8 Results

We have experimented with the completion of various photographs and paintings with an initial mean confidence $\mu(\beta) > 0.7$. The computation times range between 120 and 419 seconds for 192 by 128 images, and between 83 and 158 minutes for 384 by 256 images, on a 2.4 GHz PC processor. Slightly over 90 percent of the total computation time is spent on the search for matching fragments. Note that computation time is quadratic in the number of pixels.

Figure 4.1 shows a 384 by 256 image with $\mu(\beta) = 0.876$. The image consists of various layers of smooth and textured regions. Our method synthesizes textures of different scale and completes the shorelines. Completion consists of 137 synthesized fragments (of them 26 for the coarse level), and total computation time is 83 minutes (220 seconds for the coarse level).

Figure 4.11 shows a 384 by 256 image of the *Universal Studios* globe with $\mu(\beta) = 0.727$. In the center of the globe there are pixels which are used for completion. However, the color is contaminated by the neighboring pixels of the globe. Marking these pixels as completely known in the matte creates artifacts in the completed regions. An alternative is to assign values that are less than 1 to the corresponding matte positions, as shown in (b). This way, these pixels are not totally discarded, and are taken as strong hints to the location of the smooth area and the textured regions. Completion consists of 240 synthesized fragments (of them 57 for the coarse level), and total computation time is 158 minutes (408 seconds for the coarse level).

Figure 4.13 shows more results for 192 by 128 images with $\mu(\beta) > 0.87$. Our approach completes shapes, smooth and textured regions, and various layers of both, as well as transparency. Completion requires between 20 and 53 synthesized fragments, with average neighborhood radii between 6 and 14.

Figure 4.14 compares our result of image completion (c) with image inpainting [9] (d) and texture

synthesis.

4.9 Limitations

Our approach to image completion is example-based. As such, its performance is directly dependent on the richness of the available fragments. In all the examples presented, the training set is the known regions in a single image, which is rather limited. The building blocks for synthesizing the unknown region are image fragments of the known region. To utilize the training set, new fragments are synthesized by combining irregular parts of fragments, applying combinations of transformations to fragments (scale, translation, rotation, and reflection), and compositing fragments together.

To evaluate the performance of our method, we can use examples where the unknown region is available, and quantify and measure the completion with respect to the ground truth. Specifically, we can use the ground truth data for the search and reconstruct it in the composite, calculating the best alpha for the blend that reconstructs the ground truth.

Our technique is an image-based 2D method. It has no knowledge of the underlying 3D structure in the image. For example, in Figure 4.15, the front end of the train is removed. Our method completes the image as shown in (c). The matching fragments of completion are shown in (d).

Note that even two fragments of the same part of an object may differ greatly under slight illumination changes and transformations. While this is obviously a notable limitation, it is also an advantage as it is based on a simple mechanism, that does not require building models from a single image. An alternative to an automatic image completion is to let the user manually build an image-based model and apply photo-editing operations with some 3D knowledge as in [86].

Our image completion approach does not distinguish between figure and ground. This presents a limitation for completion when the inverse matte is on the boundary of a figure, since both the figure and background can be synthesized by example. For example, in Figure 4.16(a), the unknown region meets the boundary of the figure of the apple. Our approach does not handle these cases. However, note that the known regions of this painting contain similar patterns, and the search finds matches under combinations of scale and orientation in (b), and the completion results in (c). Note the ghost artifact on the right portion of the completed figure.

Our approach does not handle ambiguities in which the missing area covers the intersection of two perpendicular regions as shown in Figure 4.17.

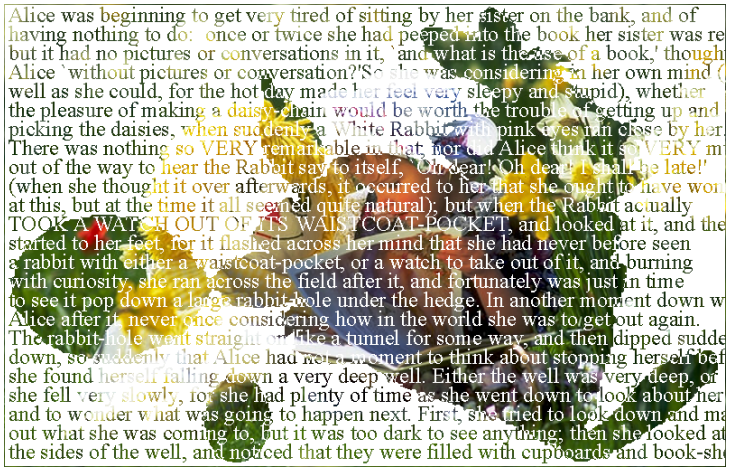
Visual grouping and background-foreground separation are an open problem, usually addressed by a subset of the Gestalt principles (proximity, similarity, good continuation, closure, smallness, common fate, symmetry and pragnanz) [72]. These principles can be incorporated into a photo-editing tool, in which it is practical to give the user some degree of control over the completion process, provided it be done in a simple and intuitive fashion. By specifying a *point of interest* in the image or a direction, the user can optionally favor symmetry, proximity, or the horizontal or vertical axis. This requires adding bias to the search in Eq. 4.6 by a function of the horizontal or vertical distances between target and source fragments, or by their distance to the point of interest. Guiding the traversal order requires adding a wide Gaussian centered at a point or a soft directional gradient to Eq. 4.5. Figure 4.18 shows completion using a point of interest located near the center of the image.

4.10 Summary

We have introduced a new method for image completion that interleaves a smooth approximation with detail completion by example fragments. Our method iteratively approximates the unknown region and fills in the image by adaptive fragments. Our approach completes the image by a composition of fragments under combinations of spatial transformations.

To improve completion, future work will focus on the following: (i) Performing an anisotropic filtering pass in the smooth approximation, by computing an elliptical kernel, oriented and non-uniformly scaled at each point, based on a local neighborhood; (ii) Locally completing edges in the image based on elasticity, and then using the *completed* edge map in the search; and (iii) Direct completion of image gradients, reconstructing the divergence by the full multi-grid method.

In addition, we would like to extend our input from a single image to classes of images. Finally, increasing dimensionality opens exciting extensions to video and surface completion.



(a)

(b)

(c)

(d)

Figure 4.5: The input image in (a) is partly covered by a white text, while the rest of the image is only visible through the text. The result of our approximation is in (b). The RMSE of local neighborhoods in radius 4 is in (c) and the ground truth is in (d).

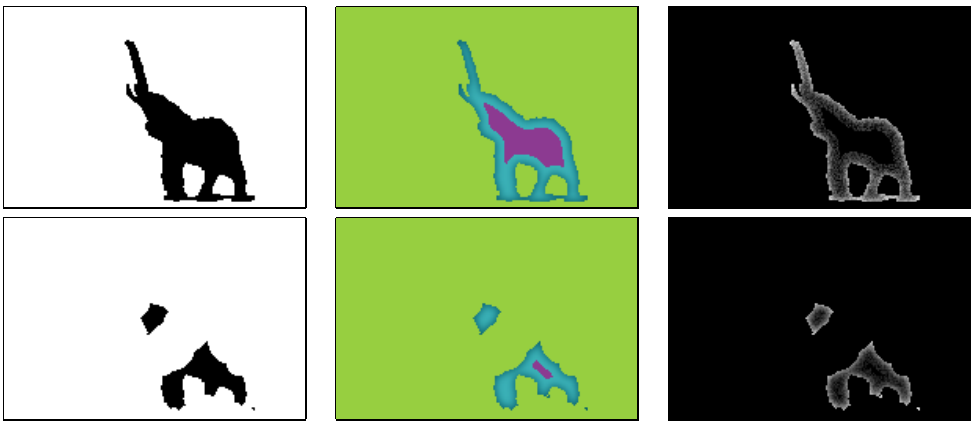


Figure 4.6: From left to right: inverse matte, visualization of confidence values on a logarithmic scale, and level set, at two different time steps.

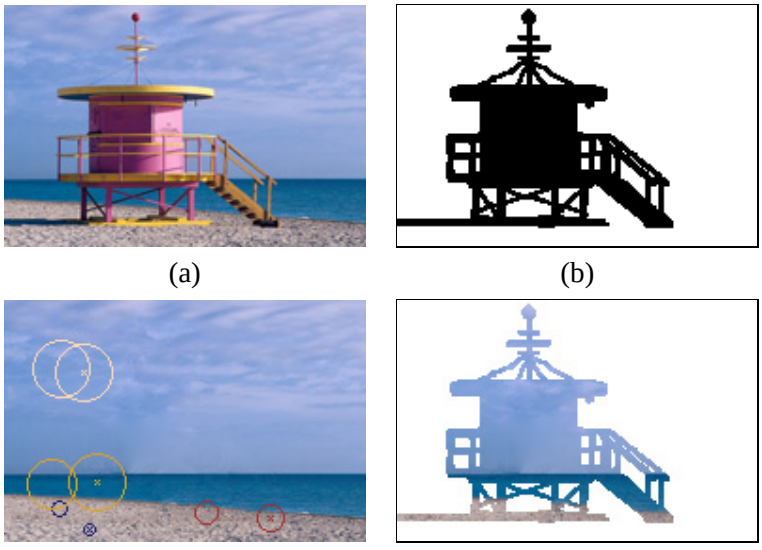


Figure 4.7: (a-b) Input color and inverse matte. (c-d) The result of our completion, several matching neighborhoods outlined with circles of the same color, the target center marked by a cross. Our approach completes the smooth areas with large fragments, the textured regions with smaller fragments, and the shoreline by searching in different *scales*.

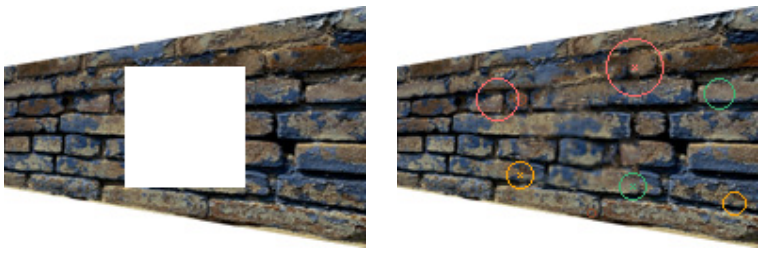


Figure 4.8: Our approach completes structured texture in perspective by searching in different *scales*.

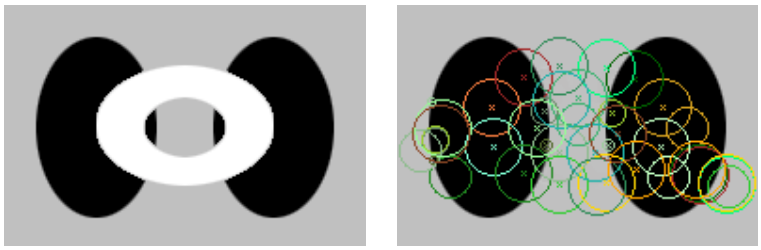
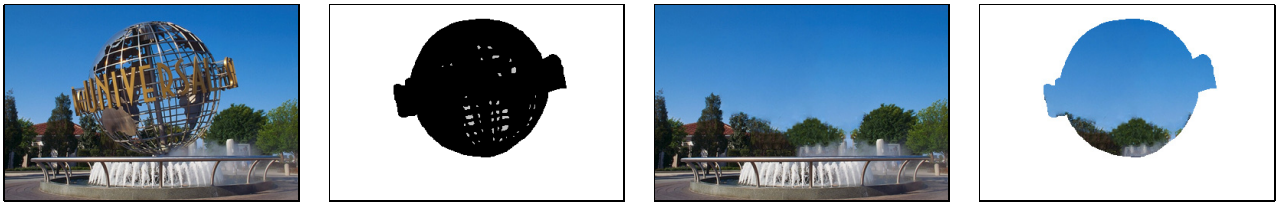


Figure 4.9: Our approach completes symmetric shapes by searching under *rotations and reflections*. Completion consists of a total of 21 fragments, marked on the output image, with mean radius of 14.

(a) (b)

Figure 4.10: An image (a) and its corresponding neighborhood size map (b), where brighter regions mark larger neighborhoods.



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

Figure 4.11: (a) The *Universal Studios* globe. (b) Inverse matte. (c) Completed image. (d) Content of the completed region.

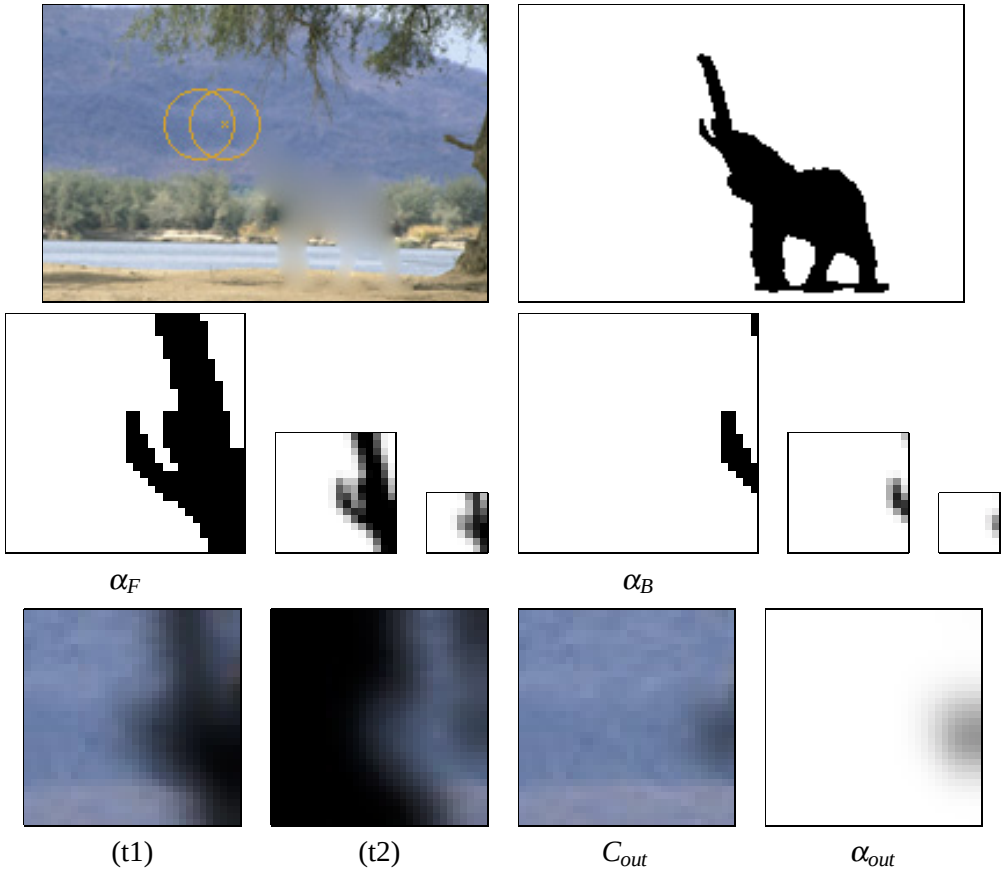


Figure 4.12: Fragment compositing: matching fragments (top left), inverse matte (top right), and Gaussian pyramids of the fragments alpha (center row). (t1) First term of Eq. 4.9. (t2) Second term of Eq. 4.9. The output color and alpha of compositing (bottom right).

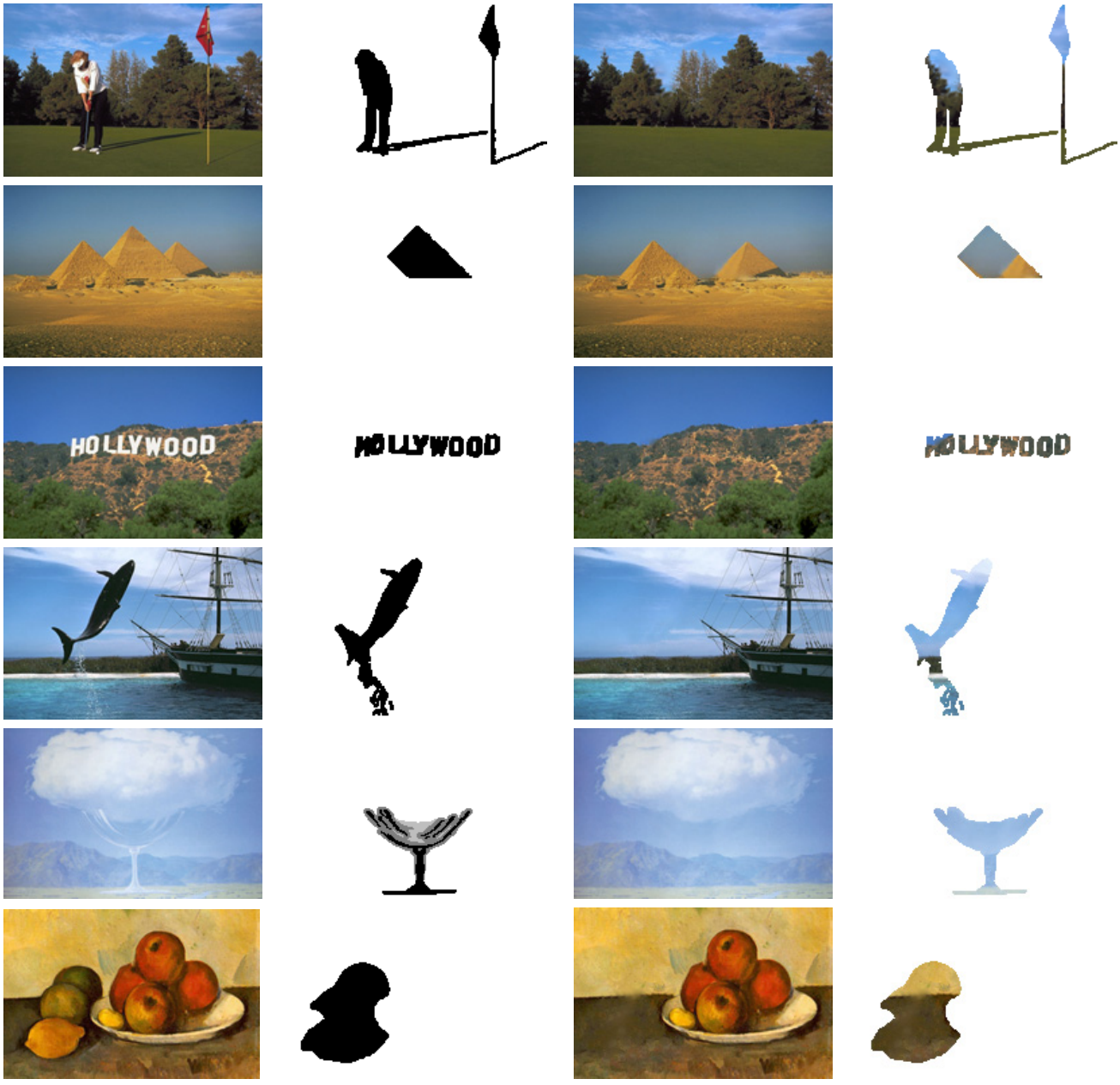
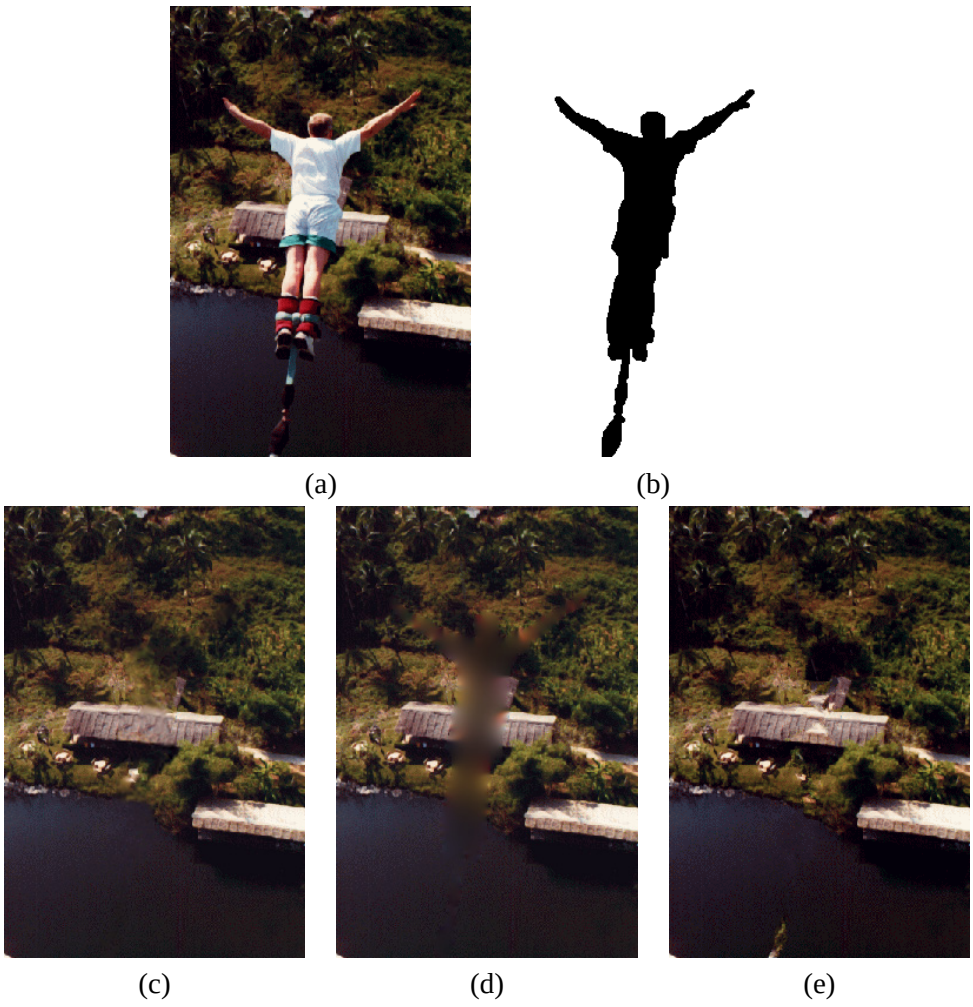


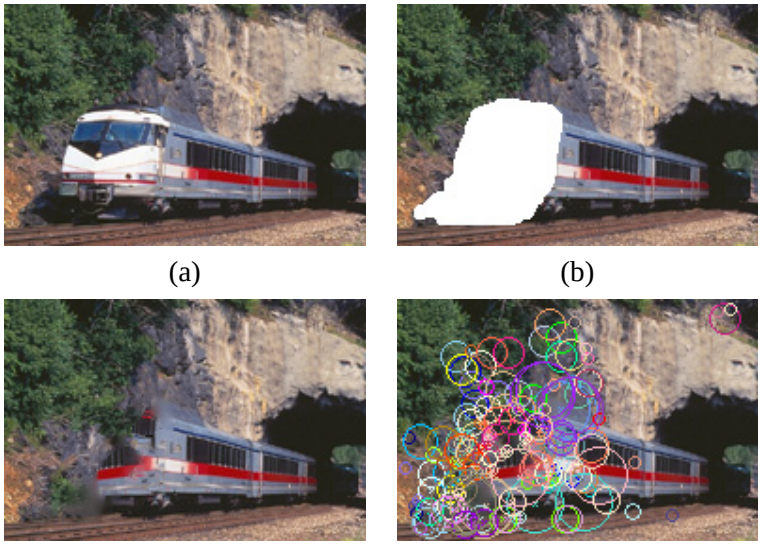
Figure 4.13: Completion results for some photographs and well-known paintings.



(a) (b)

(c) (d) (e)

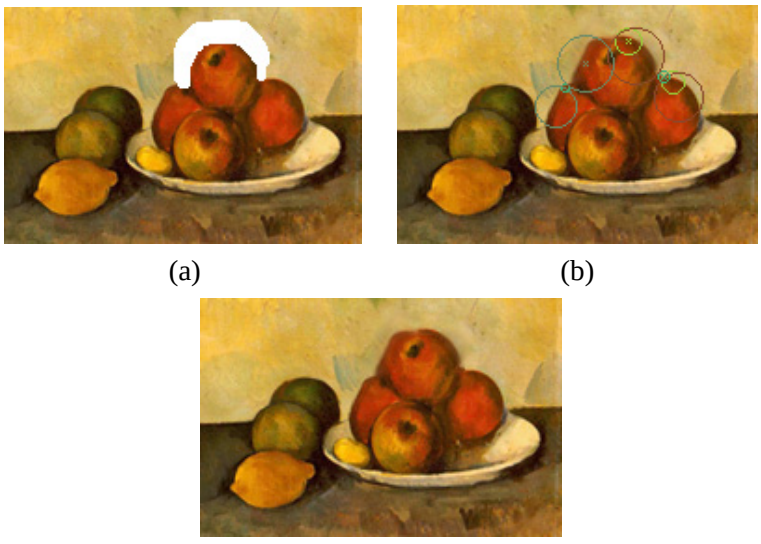
Figure 4.14: Comparison of completion result (c) with image inpainting (d) and texture synthesis (e).



(a) (b)

(c) (d)

Figure 4.15: Our approach is an image-based 2D technique and has no knowledge of the underlying 3D structures. (a) Input image. (b) The front of the train is removed. (c) The image as completed by our approach. (d) Matching fragments marked on output.



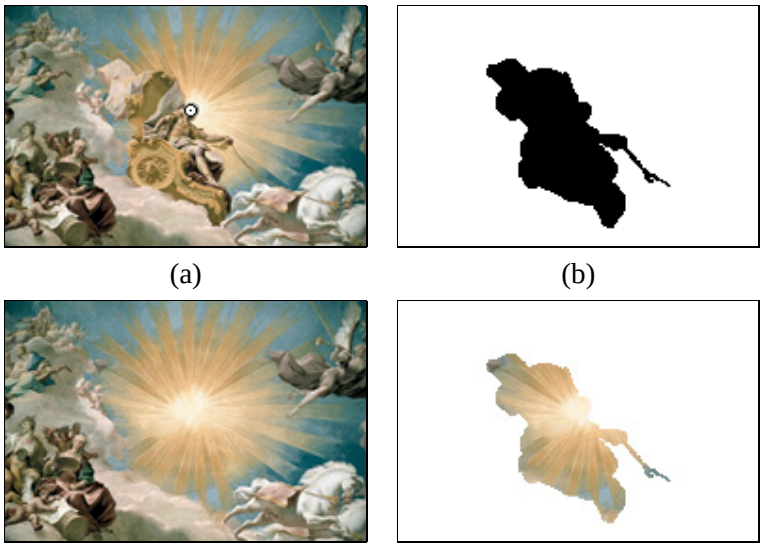
(a) (b)

(c)

Figure 4.16: Our approach does not handle cases in which the unknown region is on the boundary of a figure as in (a). However, the known regions of this painting contain similar patterns, and the search finds matches under combinations of scale and orientation in (b), and the completion results in (c).

(a) (b) (c)

Figure 4.17: Our approach does not handle ambiguities such as shown in (a), in which the missing area covers the intersection of two perpendicular regions as shown above. (b) The same ambiguity in a natural image. (c) The result of our completion.



(a) (b)

(c) (d)

Figure 4.18: A *point of interest* is specified with a white circle near the center of the image in (a). Our result using the point of interest to complete the circularly symmetric shape is shown in (c).

Chapter 5

Spectral Sound Gap Filling

We present a new method for automatically filling in gaps of textural sounds. Our approach is to transform the signal to the time-frequency space, fill in the gap, and apply the inverse transform to reconstruct the result. The complex spectrogram of the signal is partitioned into separate overlapping frequency bands. Each band is fragmented by segmentation of the time-frequency space and a partition of the spectrogram in time, and filled in with complex fragments by example. We demonstrate our method by filling in gaps of various types of textural sounds. This part appeared in proceedings of IEEE International Conference on Pattern Recognition 2004 [37].

5.1 Introduction

Automatically filling in gaps of sound and synthesizing textural sounds with similar characteristics to a given input are important in many applications. Since accurate reconstruction of the gap is impossible, the goal of our algorithm is to fill in the signal to produce a perceptually coherent output. In this work we apply techniques used in texture and image synthesis for context-based sound synthesis. We adopt direct image space methods for synthesizing a time varying audio signal by using the complex spectrogram.

The complex spectrogram is an invertible two-dimensional time-frequency representation resulting from the short-time Fourier transform. In each time-frequency coordinate we consider both magnitude and phase, as well as their gradients in time and frequency. Most of the work done in auditory signal processing and scene analysis is based on time-frequency representations that use spectral properties

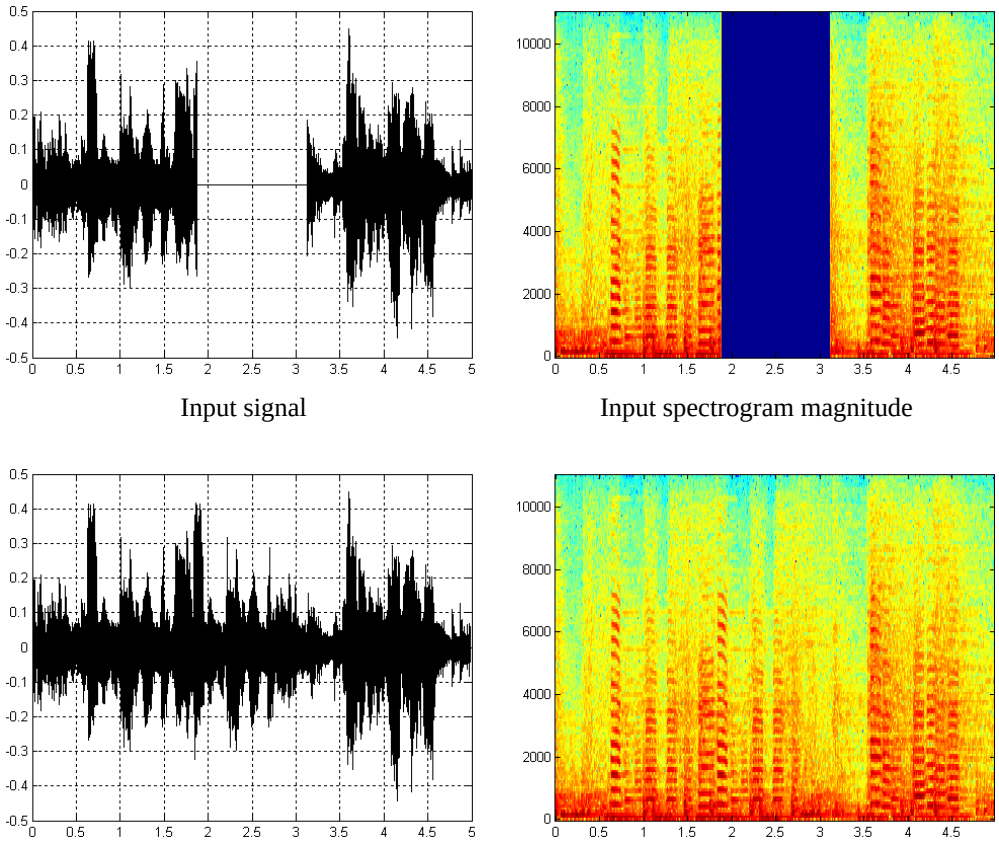


Figure 5.1: Spectral gap filling of Jazz segment.

of the signal within time windows. Additional motivation for representing sound in the time-frequency space is that the ear transforms time oscillations into frequency-dependent nerve firings, and that roughly speaking, sound is perceived in the frequency domain. More perceptually motivated, two-dimensional representations that are based on the short-time Fourier transform, such as the mel-frequency cepstral coefficients, are also suitable for our purposes.

Given an input sound signal with a gap as shown on the top left of Figure 5.1, the result of our gap filling algorithm is shown on the lower left, and the corresponding spectrogram magnitudes are shown on the right column.

5.1.1 Related work

The short-time Fourier transform is a well-established tool used in sound analysis and synthesis [3, 94]. It allows the reconstruction of a signal from its modified short-time Fourier transform [31, 56]. Recently, various algorithms for texture and image synthesis were proposed and applied to the task of filling in missing regions in images. In this work we aim at using similar approaches by transforming the real 1D time signal into a complex 2D time-frequency space. This is performed by matching similar spectral sound fragments and segmentation of the time-frequency space. Ullman *et al.*[115] emphasize the importance of intermediate-level fragments for the tasks of visual classification and segmentation. Kwatra *et al.*[75] perform image and video texture synthesis by finding the min-cut of a graph using a cost function defined on edges between adjacent pixels. Bertalmio *et al.*[10] combine image inpainting with texture synthesis by decomposing an image into the sum of two components. Inpainting is applied to the component representing the underlying image structure, whereas texture synthesis is separately applied to the component representing image detail, and the two components are then added back together. Fragment-based image completion [36] iteratively approximates the unknown regions and searches for adaptive image fragments under combinations of spatial transformations. Completion is performed from coarse to fine scales, proceeding from regions of high to low confidence. Criminisi *et al.*[30] fill in an order that gives priority to high gradients. This is achieved in the former by multiplying the traversal map by an adaptive neighborhood size map. Jia and Tang [68] first perform complete segmentation of the input and then continue the segmentation boundaries of the missing regions by tensor voting.

5.2 Time-frequency representation

In this work we use the short-time Fourier transform despite its inherent limitations, namely; a uniform partition of the time frequency plane, with single time resolution for different frequencies (an alternative is a multi-resolution wavelet representation). The advantages of this representation for synthesis are its simplicity - directly synthesizing the complex spectrogram using recent image space techniques, and working with an invertible time-frequency representation that is robust to large modifications [94].

Given a sampled sound signal $f(t)$, a symmetric window $g(t)$ is translated by time x and modulated

by frequency y, where $g_{x,y}(t) = e^{iyt}g(t-x)$, defining the continuous short-time Fourier transform by:

$$S_f(x,y) = \langle f, g_{x,y} \rangle = \int f(t)g(t-x)e^{-iyt}dt. \tag{5.1}$$

The window is normalized so that $\|g_{x,y}\| = 1$, and multiplying the signal $f(t)$ by $g(t-x)$ localizes the Fourier integral in the neighborhood of $t = x$. More specifically, we use a Hamming window $g(t) = 0.54 + 0.46cos(2\pi t)$ with an overlap of $\frac{1}{4}$ window size. The notation (x,y) represents (time,frequency) emphasizing the image nature of the complex spectrogram. Let $M(S_f(x,y))$ denote the spectrogram magnitude, and let $\Phi(S_f(x,y))$ denote the phase. To reconstruct the signal from its complex spectrogram, the inverse short-time Fourier transform is applied to each column, and the overlap-addition method [94, 31] is used to recover the signal. This allows reconstructing spectrograms that have undergone large modifications.

5.3 Sound gap filling

Our approach is to map the signal to the time-frequency space by the short-time Fourier transform, fill in the gap, and apply the inverse mapping to reconstruct the result, as illustrated by:

$$f(t) \mapsto S_f(x,y) \underset{gap\ filling}{\longrightarrow} S_{f'}(x,y) \mapsto f'(t)$$

The complex spectrogram of the signal is separated into overlapping frequency bands and partitioned in time. Each frequency band is filled in by matching fragments and the synthesized result is reconstructed. Gap filling proceeds by matching complex time-frequency fragments to the overlapping regions of existing data from the input and synthesized signal. The criteria for matching fragments is based on magnitude and phase and is performed separately in each frequency band while maintaining coherence between overlapping bands. The complex time-frequency plane is filled in by fragments with adaptive extent in time and frequency. In addition, each fragment forms irregular boundaries in the time-frequency plane within causal neighborhoods, which are determined by local segmentation based on magnitude and phase, and their gradients in both time and frequency. Once the complex time-frequency plane is covered, we apply the inverse transform, and use the overlap-addition method to further blend

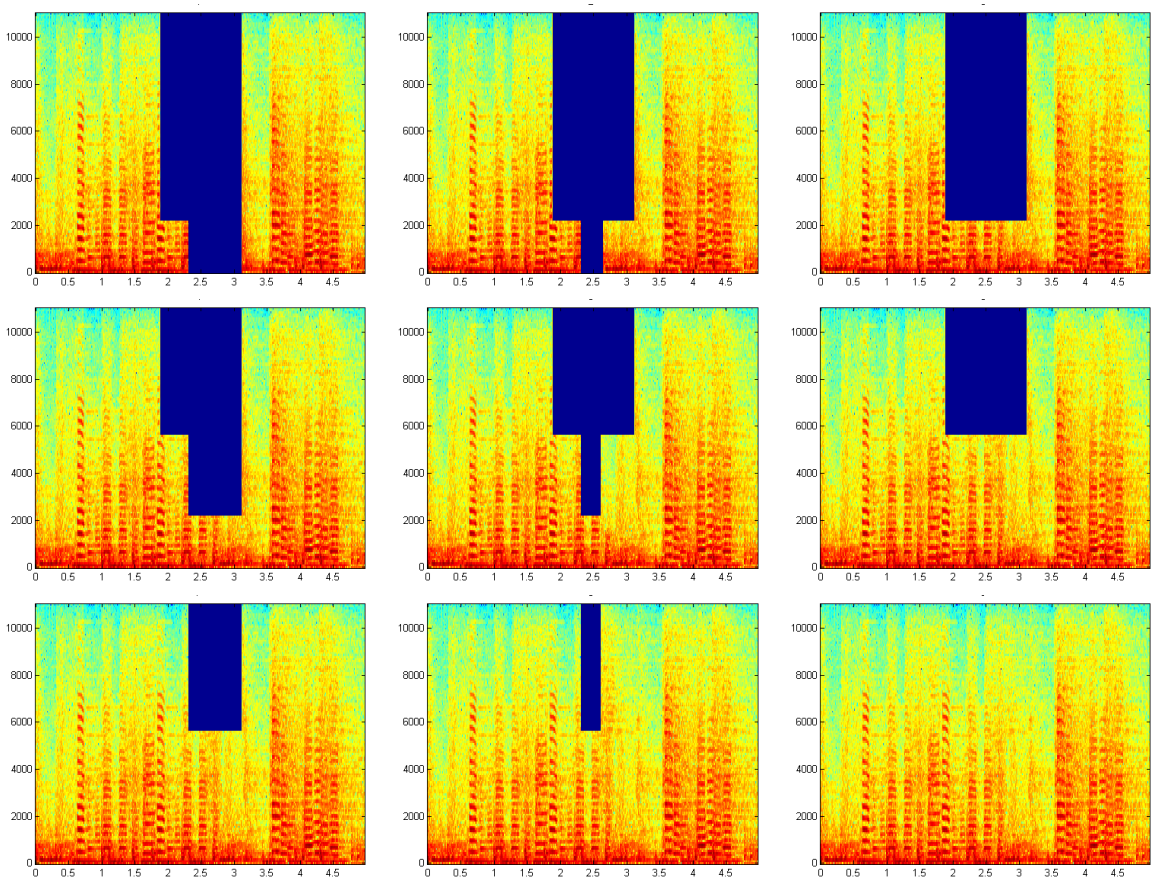


Figure 5.2: Spectrogram magnitudes in consecutive gap filling steps.

together the fragments into a coherent output sound stream. Following is a detailed description of each part of our algorithm.

5.3.1 Frequency partition and synthesis order

The complex spectrogram of the signal S_f is partitioned into separate overlapping frequency bands $F_k = S_f(:, b_k)$. Low frequencies of most natural stimuli usually contain more energy than high frequencies and therefore are less affected by noise. Therefore, gap filling of the complex spectrogram proceeds from low to high frequency bands. Perceptual time-frequency representations use a logarithmic frequency scale. Therefore, in our linear frequency scale, the frequency extents are spaced exponentially by multiplying each one from low to high frequencies, such that $|b_k| = 2|b_{k-1}|$. Figure 5.2 shows the spectrogram

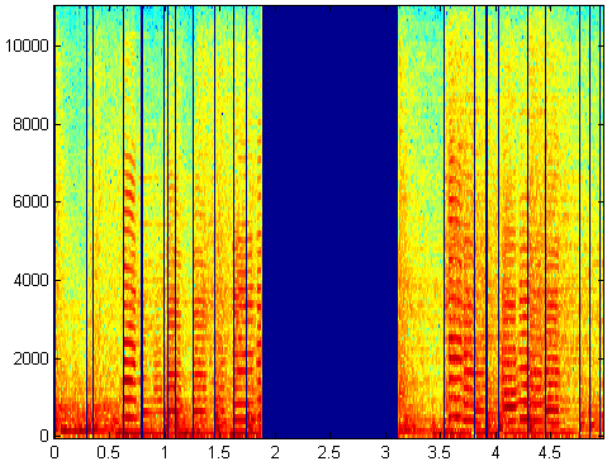


Figure 5.3: Spectrogram time partition.

magnitudes $M(S_f)$ in consecutive steps of the algorithm for the signal shown in Figure 5.1. Within each frequency band F_k we consider fragments $T_k = S_f(a, b_k)$ that overlap the known regions, and fill in each frequency band from the known to unknown regions of the spectrogram.

5.3.2 Time partition and spectral search

The spectrogram is partitioned in time $P(S_f)$ as shown in Figure 5.3 by summing the time gradient magnitudes over all frequencies $\sum_y |\frac{\partial M(S_f(x,y))}{\partial x}|$ for each time x. At each step of filling in a frequency band F_k, a target fragment T_k is defined with a causal region of overlap $O(T_k)$ with the input and previously synthesized regions. To maintain a coherent sound stream, the time extents $a = [a_1, a_2]$ of each fragment are determined by the nearest times in the partition of all previously (lower) synthesized frequencies with the greatest response inside the gap. Motivated by our auditory working memory, the maximum overlap in time is 250ms. We search the known complex spectrogram for source matches $O(T'_k)$ within the same frequency band, across all time intervals outside the gap. This is a linear one-dimensional search in time which is very efficient. A fundamental task in sound analysis is the comparison of pairs of local spectral representations, and several spectral distance and distortion measures were proposed and analyzed [55]. The amplitude of natural sound signals can rapidly change over several orders of magnitude. Therefore, the similarity of two spectral regions is based on the RMS *logarithmic* spectral

distortion [55] used in many speech recognition systems. We have also experimented with the the distortion measure $\frac{1}{2} - \frac{1}{2}\frac{O(T)\cdot O(T')}{\|O(T)\|\|O(T')\|} \in [0,1]$ used in indexing [48, 25]. The features are the magnitude and phase (M,Φ), separately normalized taking a weighted average. We find the best overlap match $O(T'_k)$, which in turn defines the best fragment match T'_k. The time extents a' of T'_k are similarly updated according to the partition $P(S_f)$. The spacing between time extents a of T_k inside the gap are set to match corresponding time extents a' of T'_k outside the gap, such that $|a| = |a'|$, which gives priority to filling the gap with structured sound fragments partitioned in time.

5.3.3 Spectral boundaries

Incrementally, each matching fragment T'_k fills in a portion of the spectral gap. Its spectral boundaries in the time-frequency space are irregular and based on a local segmentation that determines which disjoint parts to take from $O(T_k)$ and $O(T'_k)$. Locally, the boundaries between fragments define a spectral segmentation with the input and previously synthesized regions. The distortion between spectral features, both magnitude and phase, with priority to high gradient regions of magnitude and phase in time and frequency [46], defines the spectral boundaries which are computed by dynamic programming. The features are magnitude, phase, and their gradients in both time and frequency $(M,\Phi,\frac{\partial M}{\partial x},\frac{\partial M}{\partial y},\frac{\partial \Phi}{\partial x},\frac{\partial \Phi}{\partial y})$. Magnitude and phase information are separately divided by their respective normalized gradient magnitudes in time and frequency, and are approximated by central and forward differences.

Finally, the output is reconstructed from the filled in complex spectrogram, and the overlap-addition method blends together the fragment boundaries to form a coherent output $S_{f'}(x,y) \mapsto f'(t)$.

5.4 Results

We have experimented with our algorithm for gap filling of various types of textural sounds. Computation time is $O(nlogn)$ in the number of samples n, and is between 10 and 270 seconds for $44k$ and $357k$ samples, on a 1.8Ghz PC processor running Matlab. We use three separate frequency bands F_k, a Hamming window of length 256 samples, and a gap size of $\frac{n}{8}$ samples positioned around the $\frac{n}{3},\frac{n}{2},\frac{2n}{3}$ marks. Our algorithm fills in gaps in both abrupt and more continuous textural sounds.

Figure 5.1 shows the result of filling in a gap of a Jazz segment [101]. Figure 5.4 demonstrates

the result of our gap filling algorithm for various types of natural sounds. In each row the input signal is shown on the leftmost column, its spectrogram magnitude in the second column, the magnitude of the spectrogram filled in by our algorithm in the third column, and the resulting signal in the rightmost column. The first five rows demonstrate the results of gap filling of natural sounds. The top row shows the result of filling in a gap of a rapidly changing bird song [29], and the second row the result of filling in a gap of the sound of an elk. The third to fifth rows show the results of filling in a gap of a frogs' vocalization [29] for various positions of the gap.

Figure 5.5 demonstrates the results of filling in synthetic sounds of a siren, engine, and music. Our approach to sound gap filling is example-based and therefore its performance is limited to the richness of the available fragments, as shown in the example of filling in a gap of a musical segment with vocals [101] in Figure 5.6. Statistics for each sound in Figures 5.4 and 5.5 appear in Table 5.1.

Sound	Samples (n)	Rate (Hz)	Gap (size,pos)	Time (sec.)
Jazz segment	110250	22050	$\frac{n}{8}, \frac{n}{2}$	57.1
Bird song	356929	22050	$\frac{n}{8}, \frac{n}{2}$	270
Elk	109667	11025	$\frac{n}{8}, \frac{2n}{3}$	35.2
Frogs' vocals	145217	16000	$\frac{n}{8}, (\frac{n}{3}, \frac{n}{2}, \frac{2n}{3})$	55.5,57.9,56.9
Siren	43951	16000	$\frac{n}{8}, \frac{n}{2}$	9.9
Engine	134151	11025	$\frac{n}{8}, \frac{n}{2}$	53.7
Blade Runner segment	250000	22050	$\frac{n}{8}, \frac{n}{2}$	214.2
Chariots of Fire segment	125000	11025	$\frac{n}{8}, \frac{n}{2}$	58.6
Music vocals	110250	22050	$\frac{n}{8}, \frac{n}{2}$	65.4

Table 5.1: Statistics and running times for gap filling of sounds in Figures 5.1 and 4.13.

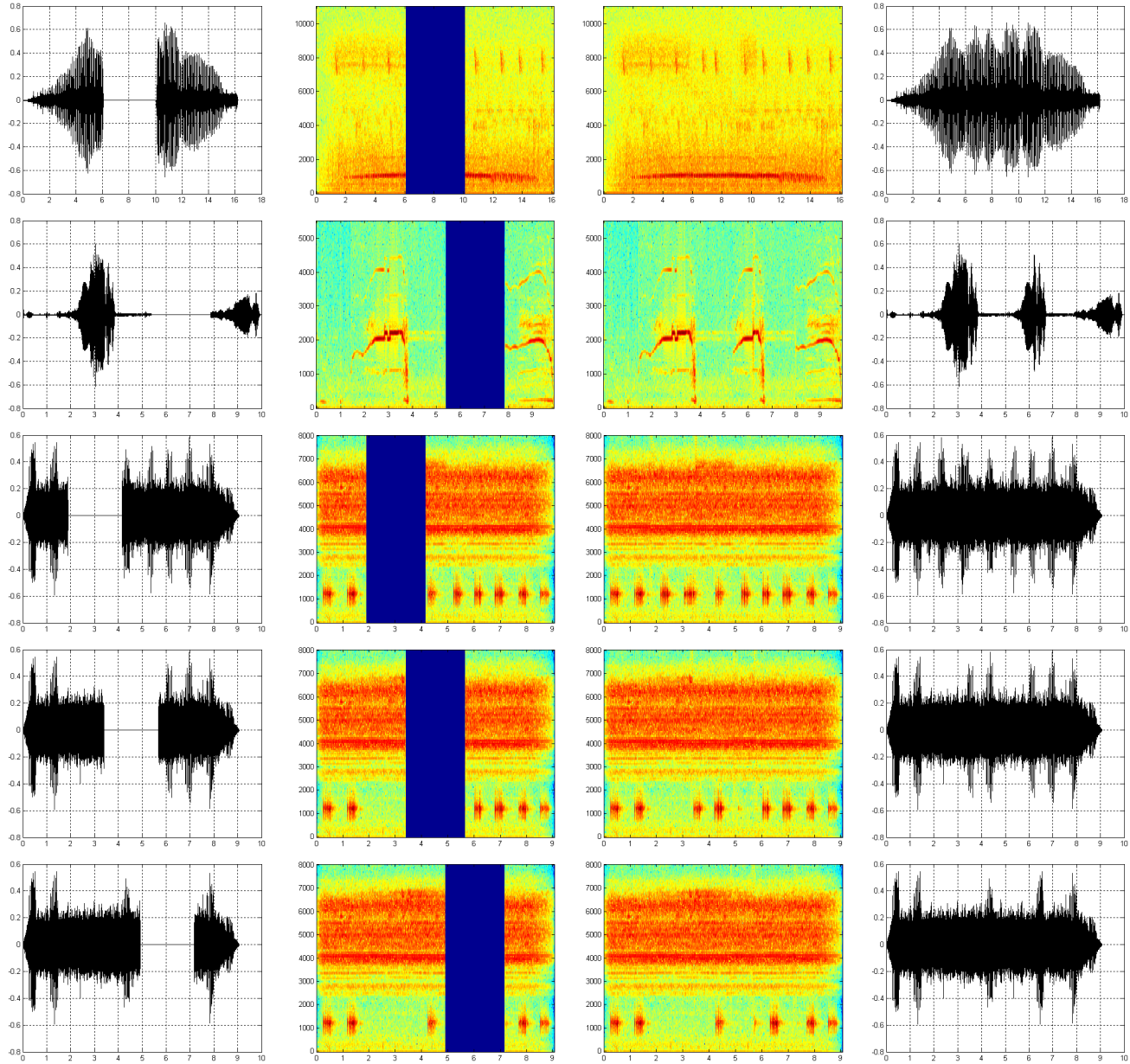
5.5 Url of auditory results

The sound files and auditory results are available at: http://www.cs.tau.ac.il/∼idrori/sounds.zip.
In addition they are available on a public free site: http://www.geocities.com/supplemental04audio/icpr/.

5.6 Future work

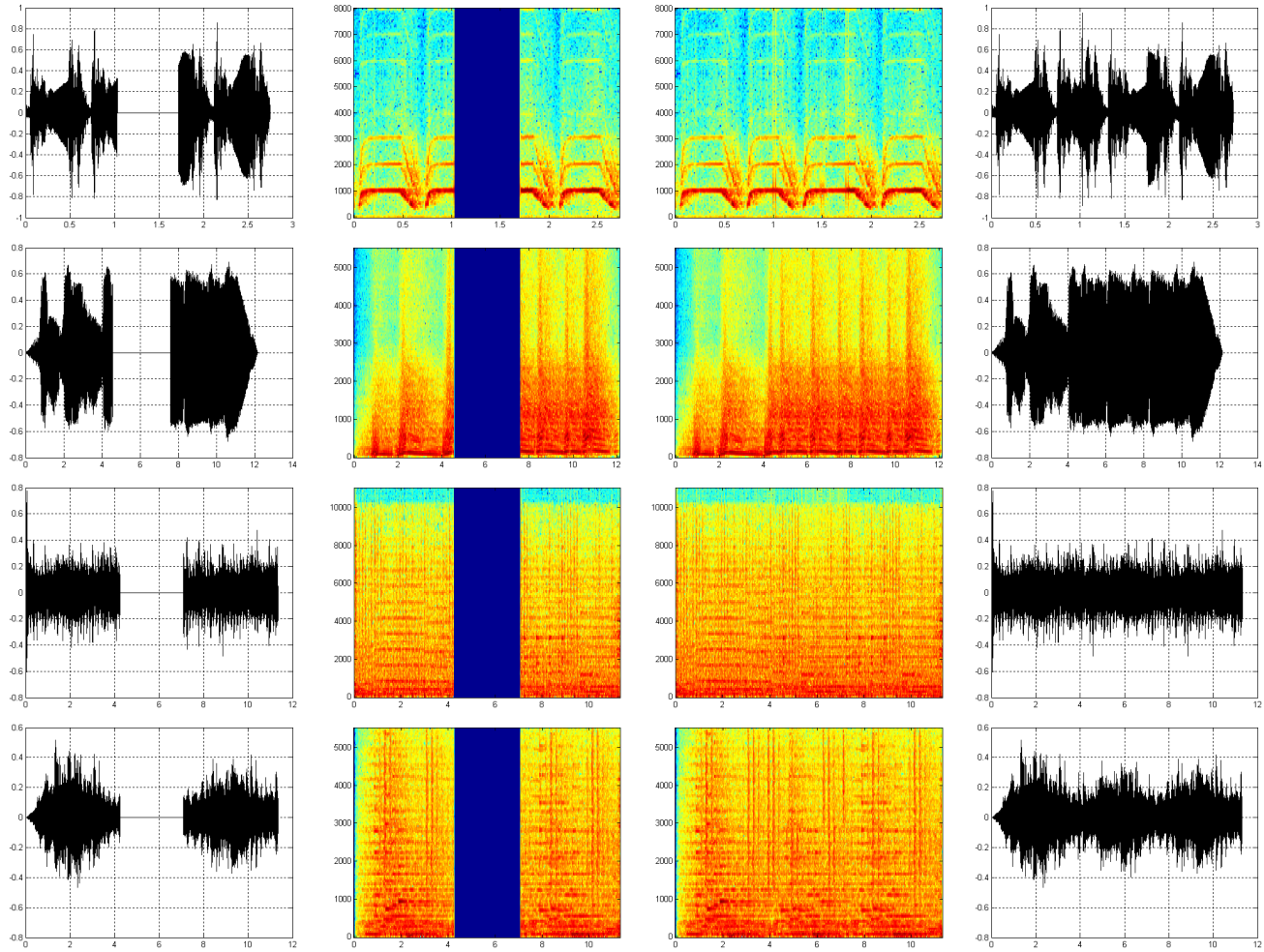
To improve the local segmentation we would like to fuse information from multiple candidate target fragments by having each fragment vote on the segmentation. Additional applications include transfer-

ring spectral attributes between pairs of signals by constrained synthesis, and extensions of this work to spectral synthesis of image and volumetric data.



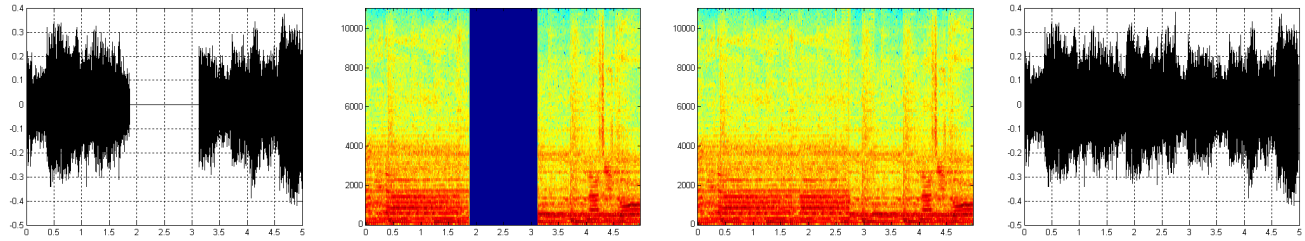
Input $f(t)$ $M(S_f)(x,y)$ $M(S_{f'})(x,y)$ Output $f'(t)$

Figure 5.4: Spectral sound gap filling of natural sounds.



Input $f(t)$ $M(S_f)(x,y)$ $M(S_{f'})(x,y)$ Output $f'(t)$

Figure 5.5: Spectral sound gap filling of synthetic sounds.



Input $f(t)$ $M(S_f)(x,y)$ $M(S_{f'})(x,y)$ Output $f'(t)$

Figure 5.6: Limitation: spectral sound gap filling of music with vocals.

Part III

Video Operations in the Gradient Domain

Chapter 6

Video Operations in the Gradient Domain

Fusion of image sequences is a fundamental operation in numerous video applications and usually consists of segmentation, matting and compositing. We present a unified framework for performing these operations on video in the gradient domain. Our approach consists of 3D graph cut computation followed by reconstruction of a new 3D vector field by solving the Poisson equation. We introduce new methods for fusing video smoothly and separating foreground elements from a video background for compositing by defining smooth and sharp transition constraints on a gradient video compositing equation. We demonstrate the applicability of smooth video transitions by fusing pairs for video mosaics, video folding, and video texture synthesis, and demonstrate the applicability of sharp video transitions by video segmentation, video trimap extraction and 3D compositing into a new sequence. Our results demonstrate that our method maintains coherence of the video matte and composite, and avoids temporal artifacts.

6.1 Introduction

Compositing video and separating foreground video elements from a video background for compositing, as well as background video completion, are operations required in the production of most modern motion pictures. Smoothly fusing parts of video is important for a number of applications: video mosaics, video texture synthesis, and video folding by reusing parts of video footage. Separating foreground and background is required for creating a video matte which is used for compositing elements into a new

background.

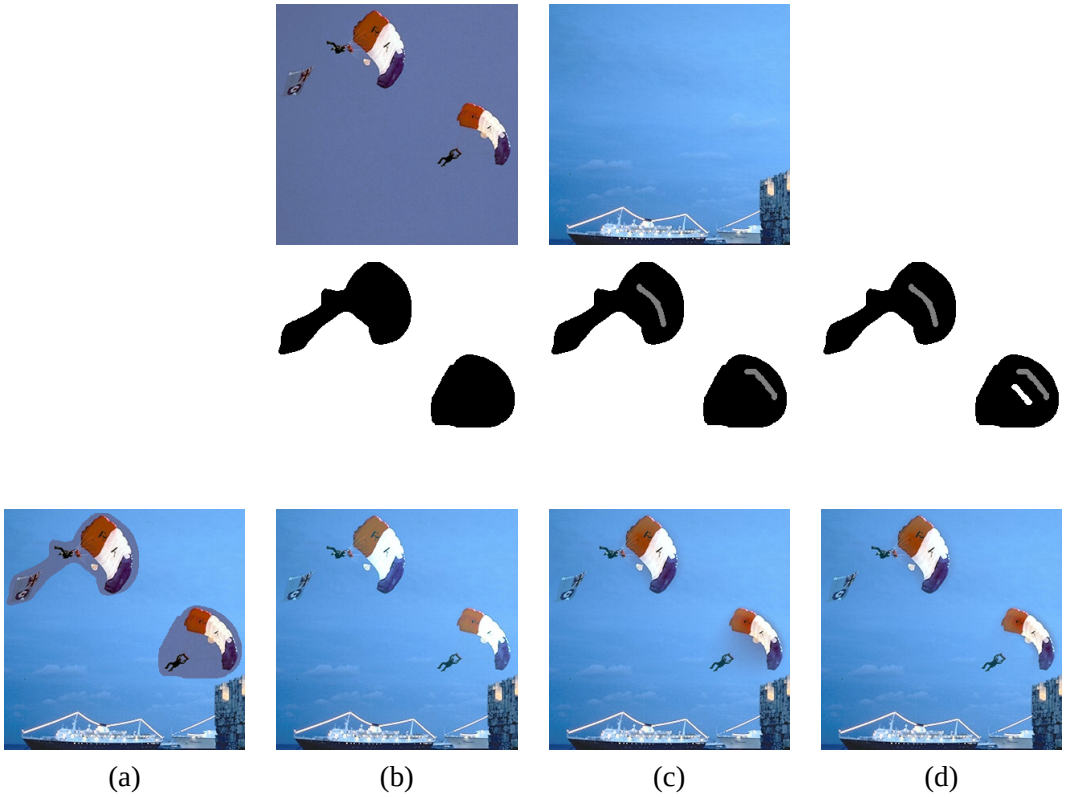
Compositing is the process of seamlessly combining multiple image or video regions. It is commonly used in creating mosaics, editing, and texture synthesis. Image and video mosaics are used in applications in which there are multiple sources, each with a limited field of view or from a different origin. The goal is to merge these sources to form a single mosaic. This is performed by first aligning the sources by warping and then smoothly blending the overlapping regions. Image and video editing operations include temporal transitions, placing a figure over a background, and seamlessly combining different image or video regions. Texture synthesis commonly consists of searching for similar patches followed by merging them together. These applications involve compositing as an essential step of the process.

Natural matting is the problem of deriving alpha values around object boundaries by which background and foreground pixels are blended. The operations of compositing and matting are therefore linked by the compositing equation which defines the color of a pixel as a convex combination of foreground and background pixels. The input to video matting is the composite video and a trimap which consists of three grayscale values denoting background, foreground and unknown video regions. Matting is derived from the compositing equation and requires background video completion and foreground estimation as well as a trimap. Background video completion is the process of filling in missing regions in video based on temporal and spatial information. Segmentation of a video into layers, compositing, matting and completion are all inherently linked by their required input-output and usage. Computing a trimap and background video completion, in turn, require rough segmentation of the foreground and compositing of the completed video background.

6.1.1 Overview

Our approach is to work in 3D with video volumes in the gradient domain. Therefore, we represent the compositing equation for video volumes and differentiate. Working in the gradient domain allows performing local operations to get a global effect, as shown for the operations of compositing and matting of images in Figures 6.1, 6.2.

The differentiation transforms the equation into differential coordinates and results in the *gradient video compositing equation*. Next, we zero out terms in the equation according to the application,

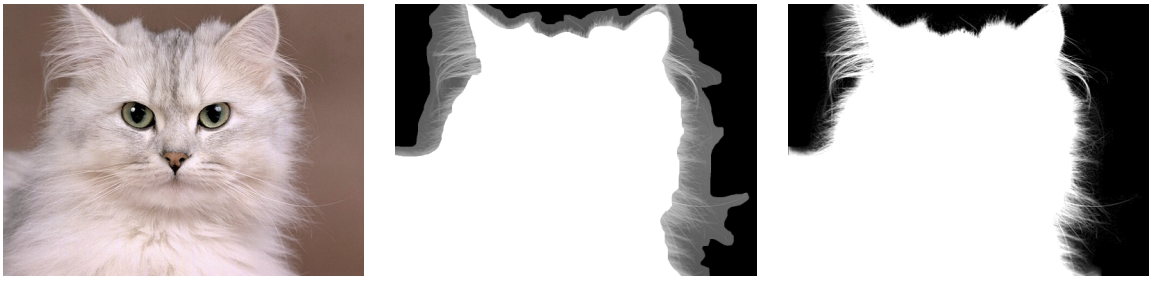


(a) (b) (c) (d)

Figure 6.1: (a) Cut and paste. (b) Gradient compositing changes the color of the parachutes. (c) Constraints from a single image results in a halo around the right parachute. (d) Constraints from both images.

whether it requires a smooth transition or sharp transitions between uniform regions. A new three-dimensional vector field results from removing terms from the gradient video compositing equation. This vector field is reconstructed to get the final result. Frame-by-frame gradient operations give rise to strong temporal artifacts which are noticeable in video, and therefore, we extend the reconstruction equations to three dimensions.

We develop and use several tools in this work. The first is 3D minimum graph cuts and the second is reconstruction from a modified 3D vector field by solving the Poisson equation. The types of cuts and iterations differ for the different applications of video fusion and separation. The new vector fields are defined with respect to the cut, and the result of solving the 3D Poisson equation is a new video or matte according to the application. In addition we use 3D splines to fit positions of tracker data and 2D splines to mark regions around foreground elements in a few key-frames and interpolate between them. Finally,



(a) (b) (c)

Figure 6.2: (a) Input color. (b) Input mask in which foreground is in white, background in black, and unknown as luminance. (c) Result of gradient matting.

motion estimation is used for the cut computation and in temporal background video completion.

Graph partitioning techniques are commonly used for image and object segmentation, for stitching together aligned images to form mosaics, and for texture synthesis. The graph is partitioned into two disjoint regions by a cut that runs through an error surface defined by pixels in the overlapping region or through graph edges defined between adjacent pixels. Traditionally, the cost function that determines the degree of dissimilarity between the two pieces is based on intensities and gradients.

In conjunction, a variety of operators such as blending, maximum, and bilinear functions are used for merging image regions. Performing these operations in the spatial, frequency, or gradient domains provides different effects. Inspired by the retinex theory [76] and its extensions [95], reconstructing an image from its modified gradient field has been used for removing shadows from images [45], compressing the dynamic range of images [43], and image editing operations [89]. Operating in the gradient domain is especially suitable for compositing as it focuses on the different types of transitions between regions.

6.1.2 Poisson equation

The Poisson equation allows reconstruction of a scalar function given a vector field and boundary conditions. In our context, modification of the function is performed by modifying the gradient field and the boundary conditions and then reconstruction of the result. The formulation can be viewed as a least squares minimization. Local artifacts are avoided as the least squares solution distributes the error throughout the reconstructed video.

6.1.3 Video compositing

Our approach to video compositing begins by applying a smoothness condition on the gradient video compositing equation. We proceed by first cutting along overlapping sequences in regions where both the motions do not conflict and the colors are similar. Instead of considering only similarity in appearance, our criterion for computing the cut also takes into account motion by computing the error between the motion in overlapping regions and the difference between their colors. Next, the video is reconstructed from a new 3D vector field. This is important for alleviating the problem of fusing together videos with varying appearance as shown in Figure 6.3 (a),(b). The seam in a 2D slice of a 3D spatial cut between the two sequences shown in (c) is noticeable due to the different scenes and their illumination. Reconstruction of the merged 3D gradients fuses together the regions avoiding the discontinuities as shown in (d). In images, feathering or multi-resolution blending [23] locally smooths the sharp transition. In image sequences temporal effects are enhanced, and therefore 3D gradient compositing is important for maintaining the detail and the temporal coherence of the result.

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

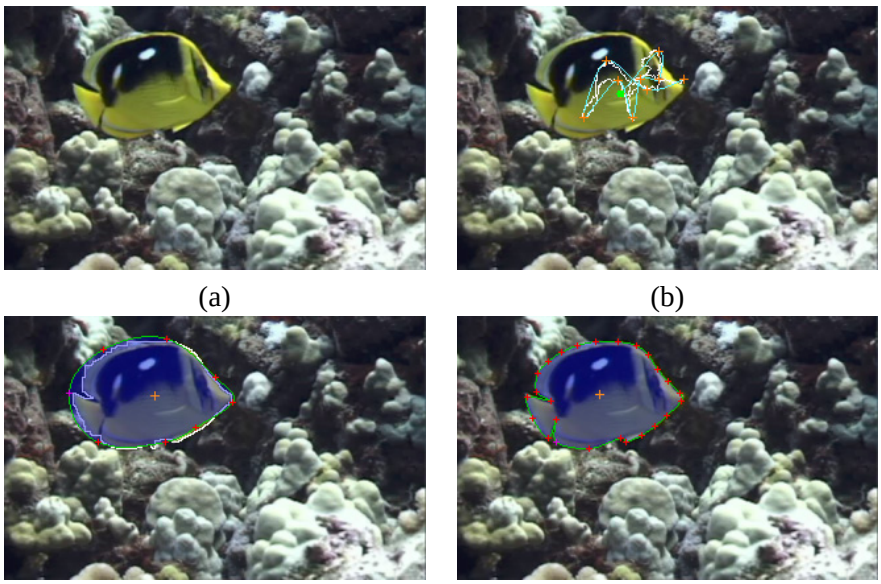
Figure 6.3: (a-b) Input frames from a pair of time-lapse sequences. (c) Cut. (d) Result of gradient video compositing.

6.1.4 Video segmentation, matting and completion

Video segmentation begins by computing a rough binary mask, which is refined to a trimap that consists of three grayscale values, and finally ends with an accurate alpha video matte in [0,1]. For the initial binary mask, the user marks a point on the object and a set of points around the object in a single frame. We then track the object forwards and backwards in time and fit a 3D spline to the tracker data as shown in Figure 7.1 (b) projected onto a single frame. This is followed by refinement of key-frames using

closed 2D shape splines as shown in Figure 7.1 (c) in green. This serves as input to the next part of our process which requires extracting a more accurate result. This is computed by iterative 3D graph cuts that snap to the object boundaries. We iterate between steps of dilation and cut computation to find the contour of the video element as shown in (c) overlaid in white. Usually the graph cut results in a discrete representation over which the user has no control, and which is not always our desired final solution. Therefore our approach is to fit a spline to the resulting cut in key-frames. This allows the user to change the control points of the spline as shown in (d) and then perform additional iterations of cut computation.

The resulting contour is dilated once more to form a video trimap which is refined by an accurate computation of a video matte. The video matte is computed by using the 3D Poisson equation. This requires completion of the video background and estimation of the foreground. Our approach to background video completion is using the temporal information present in video by searching forwards and backwards in time spans for background pixels whose motion field is small, and filling in any remaining foreground pixels by diffusion.



(a) (b)

(c) (d)

Figure 6.4: (a) Input frame. (b) Projected tracker data and 3D spline. (c) Key-frame interpolation and minimum graph cut. (d) Spline fitting to graph cut and its modification.

6.2 Related work

Porter and Duff [93] introduced compositing arithmetic for anti-aliased accumulation of separate elements with alpha into a single image. The associative over operator is used to composite a foreground element over a background. A common approach used for integrating images to form a mosaic is feathering [107], which weighs the contribution of pixels in overlapping regions as a bilinear function of distance from the region boundary or cut. Burt and Adelson [23] introduced the Laplacian pyramid to smoothly merge images according to a binary mask, resulting in a wide overlapping region for low frequencies and a narrow overlap for fine details. The Poisson equation has been widely used in computer vision. The idea of reconstructing an image from its modified gradient field was recently used for recovering reflectance images [119], removing shadows from images [45], and introduced to computer graphics by Fattal *et al.*[43] for compressing the dynamic range of images. Subsequently it was used for image editing operations such as seamless blending [89, 79] by mixing image gradients. Perez *et al.*[89] generalized the framework from rectangular domains to arbitrary patches by requiring Dirichlet boundary conditions, and applied the maximum operator to gradients and suppressed small gradients.

Davis [32] stitched together images to form a mosaic by enforcing a constraint that the error between overlapping regions is minimal. The images are stitched along a minimum cost path computed by Dijkstra's algorithm. The cost function used to determine the cut is the difference between pixel intensities in the overlapping regions normalized by the difference between extreme values. Efros and Freeman [41] applied this method for stitching together texture blocks. The cost function is defined as the difference between pixel intensities in the overlapping regions. A minimum cost path through the error surface is computed by dynamic programming and is linear in the number of pixels. Kwatra *et al.*[75] performed texture synthesis by finding the min-cut of a graph using a cost function defined on the edges between adjacent pixels from the overlapping regions. Taking into account the spatial frequencies in the cost function is especially important for perceptual masking. Dividing color by gradient magnitudes gives priority to high gradient regions which provide a snapping effect.

Object segmentation in image sequences is one of the fundamental problems in computer vision. This problem is usually addressed either by discrete representations which are currently manifested by graph partitioning techniques, or by continuous methods typically referred to as active contours. Active contours [70, 15] are used for segmentation by minimizing an energy function which consists

of terms that guide the contour towards image features and prefer smooth boundaries. The snake is sensitive to the initial contour and approaches the nearest local minimum. Mortensen and Barrett [84] use dynamic programming for segmenting an object in an image and coined the term intelligent scissors. Xu *et al.*[122] use iterative graph cuts for segmentation of an object from a background. The image is represented by an adjacency graph, and the user defines a polyline which is dilated to form an initial region around the object. The outer and inner boundaries of this region define the source and sink of the graph and the computed minimum cut is iteratively dilated to form a new boundary region, until converging to a global minimum. Irani *et al.*[67] and Wang and Adelson [117] use optical flow to separate a video into layers. We apply 3D iterative graph cuts of a video volume to find the initial segmentation of a foreground video element.

Mitsunaga *et al.*[83] take the derivative of the image compositing equation and assume relatively small changes in the foreground and background regions to derive the relationship between image gradients and matte gradients. Apostoloff and Fitzgibbon [4] learn the joint distribution between image gradients and alpha gradients from a collection of image sequences. Yu *et al.*[123] perform mesh denoising by bilateral filtering [47] shifting vertex positions using the Poisson equation. Recently, Jian et al. [106] use the Poisson equation for 2D matting and perform a set of local operations to refine the resulting matte. We present a unified framework for performing different operations on video in the gradient domain, and solve a 3D Poisson equation to extract a video matte avoiding temporal artifacts. Chuang *et al.*[27] extend Bayesian image matting [28] to video and use optical flow to interpolate a trimap. In contrast, we use an iterative 3D graph-cut approach to segment the foreground and compute a trimap. We show the relationship between completion and compositing, and complete the video background by motion estimation and diffusion.

6.3 Gradient video operations

We present a unified framework for performing operations on video in the gradient domain. We begin by extending a single dot product $c = a \cdot b$ for a single pixel c and vectors a and b to multiple dot-products, $C_p = A_p \cdot B_p = \sum_i A_p^i B_p^i$, one for each coordinate $p = (x,y,z)$ of the video volume C. We then

differentiate and work with differential coordinates:

$$\nabla C_p = \sum_i (\nabla A_p^i B_p^i + A_p^i \nabla B_p^i). \tag{6.1}$$

Finally we create a new vector field by modifying and setting terms to zero according to the desired operation, and reconstructing the result.

6.3.1 Gradient video compositing equation

In the spatial domain the *compositing equation* defines the value of each pixel as a linear combination of foreground and background:

$$C = \alpha F + (1-\alpha)B. \tag{6.2}$$

In video compositing, C, F, B denote video volumes, α a 3D matte, 1 a volume of ones, and the operations are point-wise multiplication, addition and subtraction between three-dimensional arrays. Differentiating we get the *3D gradient video compositing equation*:

$$\nabla C = \alpha \nabla F + (1-\alpha)\nabla B + \nabla \alpha (F-B). \tag{6.3}$$

Our observation is that a smooth transition between foreground and background videos along the 3D matte boundaries yields:

$$\nabla \alpha (F-B) = 0,$$

$$G = \alpha \nabla F + (1-\alpha)\nabla B, \tag{6.4}$$

where G is a new vector field. The volume is then reconstructed from the new vector field by solving a 3D Poisson equation. Whereas in the inverse problem, a sharp transition between uniform foreground and background yields:

$$\alpha \nabla F + (1-\alpha)\nabla B = 0,$$

$$\nabla C = \nabla \alpha (F-B), \tag{6.5}$$

which defines the relationship between video matte gradients and video gradients for uniform regions.

6.4 3D Poisson

In this work we perform operations on video in the gradient domain. Frame-by-frame gradient compositing and matting gives rise to strong temporal artifacts. We therefore extend the reconstruction equations to three dimensions. Following the notation of [43], we search the space of all 3D potential functions for a function V whose gradients are closest to G in the least squares sense; V should minimize the integral:

$$\int\int\int F(\nabla V, G)\,dx\,dy\,dz, \tag{6.6}$$

where

$$\begin{aligned} & F(\nabla V, G) = \|\nabla V - G\|^2 \\ = \quad & (\frac{\partial V}{\partial x} - G_x)^2 + (\frac{\partial V}{\partial y} - G_y)^2 + (\frac{\partial V}{\partial z} - G_z)^2. \end{aligned} \tag{6.7}$$

According to the variational principle, a function V that minimizes the integral in Eq. 6.6 must satisfy the Euler-Lagrange equation, that for three independent variables generalizes to:

$$\frac{\partial F}{\partial V} - \frac{d}{dx}\frac{\partial F}{\partial V_x} - \frac{d}{dy}\frac{\partial F}{\partial V_y} - \frac{d}{dz}\frac{\partial F}{\partial V_z} = 0, \tag{6.8}$$

which is a partial differential equation in V. Substituting F results in satisfying the Poisson equation:

$$\nabla^2 V = \nabla \cdot G, \tag{6.9}$$

where

$$\nabla^2 = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} \tag{6.10}$$

is the three-dimensional Laplacian operator, and

$$\nabla \cdot G = \frac{\partial G_x}{\partial x} + \frac{\partial G_y}{\partial y} + \frac{\partial G_z}{\partial z} \tag{6.11}$$

is the divergence of the vector field measuring the extent to which a point in the field is a source or sink.

6.4.1 Discrete solution

To solve for V we consider Eq. 6.9 as a boundary value problem of the form $L\phi = f$, where L is an operator, ϕ an unknown scalar field, and f a non-homogeneous term. When there is no analytical solution, the domain is a discrete volume grid. Methods for solving the Poisson equation in 3D are well known in scientific computation [20].

The divergence is approximated by the finite backward difference:

$$\begin{aligned} divG &\approx G_x(x,y,z) - G_x(x-1,y,z) \\ &+ G_y(x,y,z) - G_y(x,y-1,z) \\ &+ G_z(x,y,z) - G_z(x,y,z-1). \end{aligned} \tag{6.12}$$

The intensity gradient ∇V is approximated by the finite forward difference:

$$\begin{aligned} \nabla V(x,y,z) &\approx (V(x+1,y,z) - V(x,y,z), \\ &\quad V(x,y+1,z) - V(x,y,z), \\ &\quad V(x,y,z+1) - V(x,y,z)). \end{aligned} \tag{6.13}$$

The Laplacian is approximated using finite differences by a Taylor series expansion of each voxel. We considering the six connected neighbors of each voxel and their three directions. Then expand in the positive and negative directions to fourth order, add each pair of equations, solve for the second derivative, and sum the solutions in each of the coordinates to get:

$$\begin{aligned} \nabla^2 V(x,y,z) &\approx V(x+1,y,z) + V(x-1,y,z) \\ &+ V(x,y+1,z) + V(x,y-1,z) \\ &+ V(x,y,z+1) + V(x,y,z-1) \\ &- 6V(x,y,z). \end{aligned} \tag{6.14}$$

This results in a set of linear equations that in matrix form contains only seven non-zero elements in each row. For compositing of video pairs the computation is performed on the entire volume and

therefore we use a 3D Poisson solver with multi-grid [20] on an entire regular domain. We solve Eq. 6.9 separately for each channel. For video matting the computation is performed only on video regions corresponding to the gray area of the video matte and therefore we use a direct solver [108] on an iregular grid. The boundary conditions are defined by the known (0 or 1) regions of the trimap. We solve a linear system whose unknowns are voxels only in the gray region.

6.4.2 Boundary conditions

We apply two types of boundary conditions in 3D for the respective applications of smooth and sharp transitions. Neumann boundary conditions specify the normal derivative of a function on a surface, which in our case means that for compositing of video pairs the derivative on the volume boundary is zero, and pad the volume boundaries. Dirichlet boundary conditions specify the value of a function on a surface:

$$V|_{\partial\Omega} = V^*|_{\partial\Omega} \tag{6.15}$$

where V^* provides the values on the boundary $\partial\Omega$. In our case this means that known matte voxels (0 or 1) constitute boundary conditions for neighbouring unknown voxels in the gray region of the trimap. Dirichlet boundary conditions allow working with arbitrary 3D outlines.

6.5 Smooth transition

6.5.1 Appearance and motion cuts

To compute the vector field in Eq. 6.4 requires computing a mask which determines the contributions of the gradients from the videos. Therefore, prior to reconstruction, our goal is to find a good criterion for computing a cut between overlapping videos. We would like the cut to pass through areas where both the motions do not conflict and the colors are similar. Therefore, in the overlapping region we look at the error volume between both the colors and the motions. The matching cost between two overlapping regions A and B in each 3D coordinate p is defined as:

$$cost(p) = \lambda \|\mathbf{c}_A(p) - \mathbf{c}_B(p)\| + (1-\lambda)\|\mathbf{f}_A(p) - \mathbf{f}_B(p)\|, \tag{6.16}$$

where **c** denotes color channels, **f** denotes motion field values, and λ a scalar that determines the balance between color and motion.

The cut is followed by reconstructing a function from a 3D vector field. In this respect the motion constraint is more elaborate than directly requiring that the gradients also agree along the cut by the constraint $\|\mathbf{G}_A(p) - \mathbf{G}_B(p)\|$. We use an optical flow technique of Black and Anandan [14] which estimates the flow in a multi-scale fashion, from coarse to fine, to handle large motions. It uses robust statistics to avoid large errors from outliers and allows for flow discontinuities.

6.5.2 Efficiency

Working with video, our goal is to limit the computation time of the entire compositing process to be linear in the number of voxels. We have experimented with finding a minimum graph cut in two and three dimensions [18] as shown in Figure 6.9 (c). However, when compositing pairs of video the cut computation is performed on the entire video volume and therefore computation time is dependent on the number of vertices times the number of edges in the graph, which is quadratic in the number of voxels. A direct extension of image quilting [41] to 3D using dynamic programming would result in a minimum cost 1D path in the volume. Using dynamic programming for each spatial or temporal 2D slice separately would not preserve coherence, resulting in noticeable artifacts.

We therefore combine dynamic programming with a greedy heuristic, modifying the cost according to adjacent slices. More specifically, in the initialization phase, the cost of each position is the sum of three functions: (i) a function of appearance and motion as defined in Eq. 6.16; (ii) a function of the matte which is the distance from the corresponding previous cut maintaining temporal coherence; and (iii) a Gaussian function of position which avoids trivial cuts in the overlapping region by assigning a penalty to voxels near the boundaries. In the update phase, this sum is added to the minimum over the previous positions. We perform this computation for finding both spatial and temporal cuts. Temporal cuts are found by rotating the inputs, performing the computation, and then rotating back the output. We found the results of this approach to be sufficient for our purposes as it avoids extreme changes of the cut, while maintaining linear computation time.

6.6 Sharp transition

Our approach for extracting a foreground video element consists of three stages: (i) key-frame tracking; (ii) iterative graph-cuts; and (iii) matte extraction. First, the user marks a point on the object and a region around the object in a key-frame. The position is tracked throughout the entire sequence and the scale is estimated. Next we fit a 3D spline to the tracker data and define closed 2D splines around the object in each key-frame. The splines are interpolated across the remaining frames resulting in a binary mask. Next, we apply an iterative 3D graph cut algorithm for extracting a trimap from the binary mask, and finally we extract an accurate 3D alpha matte from the trimap by completing the background and estimating the foreground in the gray region, and then solving a 3D Poisson equation. The different stages are characterized by the accuracy of data maintained in each stage: starting from a rough binary mask, onto a more accurate trimap, and finally a fine 3D alpha channel used for 3D compositing.

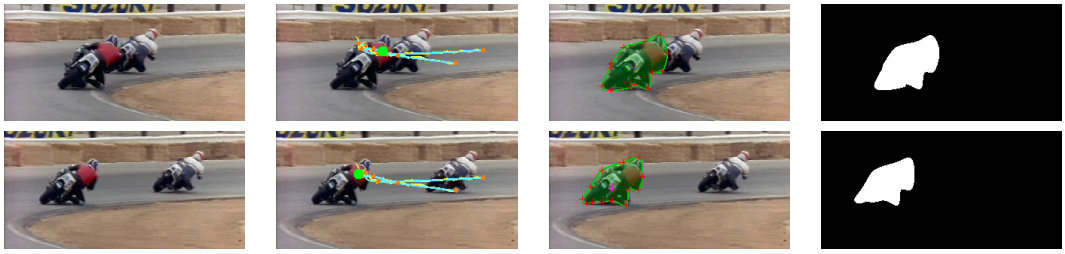
6.6.1 Key-frame tracking

We would like to extract an object from an input sequence. First, the user marks a point on the object to track and an area around it in a single frame. According to this point we track the object and estimate its location in each frame to get an ellipse that denotes the change in scale in two orthogonal axes. Let T denote the set of tracker positions in each frame. We fit a 3D spline to the tracker position as shown projected onto a single frame in Figure 6.5 (b). Fitting the 3D spline is performed incrementally using a greedy algorithm. We begin by fitting a 3D spline through all positions $P = T$. Next, in each iteration of the algorithm we find a point q whose removal from the set P results in a new spline minimizing an overall error between the tracker data T and the spline $S_{P\setminus q}$:

$$q^* = \arg\min_{q\in P} d(S_{P\setminus q}, T) \tag{6.17}$$

$$d(S_P, T) = \sum_{t\in T} d(S_P, t), \tag{6.18}$$

where $d(S_P, t)$ is the distance between the 3D spline curve S_P and a tracker point t according to the L_2 norm. Fitting is performed until reaching a minimum number of key-frames. This result is refined by the user by adding key-frames and refining the 2D shape splines around the object as shown in Figure



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

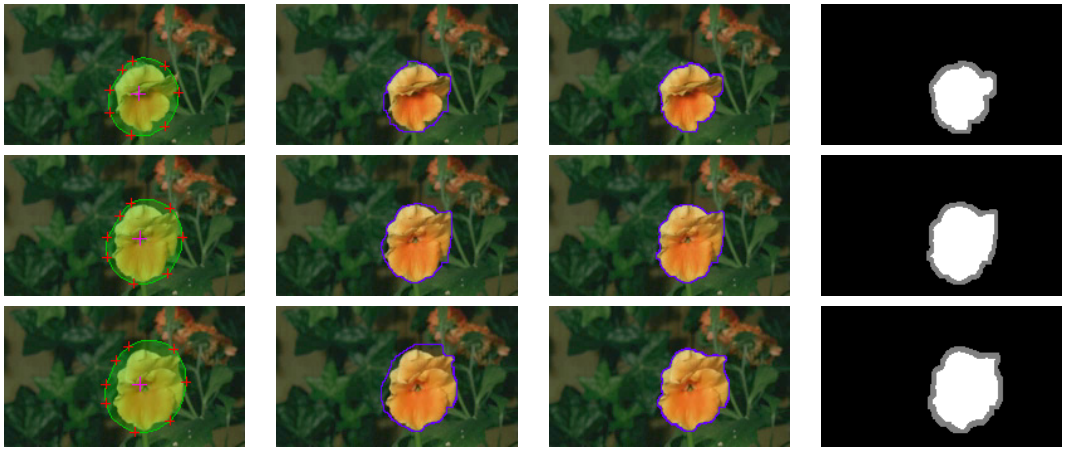
Figure 6.5: Tracking: (a) Input frames. (b) Tracker data and 3D spline projected on frame. (c) Closed 2D shape spline. (d) Binary mask.

6.5 (c). Each key-frame the user updates is propagated to the following frames. The output defines an initial binary mask as shown in (d) for the iterative graph-cut algorithm which extracts a trimap.

6.6.2 Iterative graph-cuts

In order to extract the foreground object from a video and composite it onto a new background we need to find a trimap that defines the foreground and background regions and an unknown region (where alpha values are in $[0,1]$). Given a rough binary mask line around the object, we perform iterations of dilation and computation of the minimum graph cut in the dilated region. Each iteration defines a more accurate region and the cut typically converges between three to ten iterations depending on the accuracy of the spline and the initial kernel size which is decreased in each iteration. The cut is computed by taking the inner and outer vertices of the dilated region to be source and sink. This provides the cut with an initial estimate of the object boundary which is refined in each iteration. This iterative process defines a contour around the object. Nodes of the graph are created only in the boundary region, resulting in a relatively small graph and efficient computation (rather than a regular graph of the entire volume with n^3 nodes). The weights of vertical, horizontal and temporal edges are set to the absolute of color difference divided by the sum of directional luminance gradient magnitudes.

Figures 6.6 demonstrates the various stages using our graph cut trimap extraction: The user marks point on the object and a few points around the object in a key-frames. The object is tracked and a 3D spline is computed through the tracker data. In addition 2D shape splines are computed through the point around the object and interpolated between all key-frames as shown in (a). This defines an initial binary



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

Figure 6.6: Video trimap extraction: (a) Keyframe spline interpolation. (b) Initial slice of 3D graph cut. (c) Result of iterative 3D graph cut. (d) Graph cut trimap.

video mask which serves as an initial estimate to the 3D iterative graph cut algorithm. The boundary is dilated to form a region in which a 3D minimum cut is computed as shown in (b). Dilation and minimum cut computation are performed iteratively, as described in Figure 6.7, resulting in a contour around the object shown in (c) which is once more dilated to form a video trimap shown in (d).

6.6.3 Video matting and completion

The trimap is used to compute an accurate video matte. We extract the matte by assuming that the background and foreground are relatively smooth and setting the corresponding terms in the gradient compositing equation to zero. The 3D gradients of the video are then proportional to the 3D gradients of the matte. The scaling factor for each voxel in the unknown region is the difference between the foreground and the background. The resulting matte is used to composite the video element onto a new background or video.

To extract the matte we need to complete the video background B and estimate the foreground F only in the unknown (gray) trimap regions. In this respect using a 3D video volume presents a clear advantage over 2D matting. Temporal information is commonly used for video completion [91]. Approximating camera motion with a global transformation by tracking a sparse set of features allows warping the

image sequence to a single reference frame, applying the inverse transformation after completion. We utilize the temporal information present in image sequences by searching in temporal neighborhoods along the missing volume. We search along time spans, forward and backward in time, to find the nearest existing background in the same spatial coordinate whose flow magnitude is near zero. When a foreground object cannot be filled in by pixels from a temporal neighborhood, for example, when background information is unavailable or its temporal distance is too far, we perfom diffusion. We have also found useful the temporal median computed over span points with no or little motion. Background completion is described in pseudocode in Figure 6.8. The foreground video in the gray trimap region is estimated by diffusion.

6.7 Results

6.7.1 Video compositing

We first demonstrate some 2D results of reconstructing an image from merged gradients of a pair of input images with respect to a 2D minimum graph cut. The top row of Figure 6.9 (d) shows the reconstructed result is coherent in both texture and color. The lower row of Figure 6.9 shows a pair of input images with different skies and actors in the scenes (a-b). The third figure (c) shows the result of a 2D minimum graph cut, and the rightmost figure (d) shows the result of gradient image compositing with respect to (c). Gradient image compositing merges the sky seamlessly while maintaining the silhouettes around the shield. Computation time is under a minute for a pair of 720 by 480 input images on a 1.8 GHz P3 processor, for both the quadratic 2D minimum graph-cut computation and for solving the 2D Poisson equation using the full multigrid method.

We have experimented with compositing various pairs of sequences with varying geometric and photometric differences from a database of video footage. The subjects of these sequences are varied and include time-lapse, scenery, textures, and more. We set an equal balance between color and motion, $\lambda = 0.5$, in computation of the cut. Total computation time of the *entire video* is between 30 and 60 seconds for input pairs of size between 112^3 and 148^3 on a 1.8 GHz P3 processor with 1GB of RAM. In the following we show the spatial and temporal effects resulting from gradient video compositing. We

use video pairs to demonstrate the applicability and advantages of our compositing method for mixing, extending and folding videos in space and time. These operations are commonly used in creating video mosaics, in video editing, and video texture synthesis.

Video mosaics

Figure 6.10 shows frames from a pair of time-lapse image sequences from different sources (a-b), and a frame from the result of gradient video compositing is shown in (c). Notice that while the cut is found only in the overlapping region, the sequence is reconstructed from gradients on the entire domain to get a single coherent time-lapse output. We applied our technique for temporally merging pairs of different sequences which is used for extending videos in time and space for creating video mosaics.

Video folding

Figure 6.11 demonstrates the application of our method to folding an image by placing the left and right portions of the image one over the other, computing a 2D cut in the overlapping region, and then reconstructing the image from the merged gradients. Figure 6.12 shows the application of our method to spatially folding a video by placing the left and right volumes one over the other, computing a 3D cut in the overlapping volume, and reconstructing the video from a 3D vector field. We demonstrate compositing results for both static and moving cameras, as well as a scene with noisy snowfall background, while maintaining temporal coherence of the output. Figure 6.16 shows the application of our method to folding image sequences by cutting a temporal band to change the sequence duration. A 2D slice of a 3D temporal cut of the first and last sequence segments is shown in (c), and the result of gradient video compositing in (d). This is used for folding videos in space and time.

Video textures

Figure 6.13 shows frames from different pairs of image sequences (a-b). A 2D slice of 3D spatial and temporal cuts is shown in (c). Our method spatially and temporally composites the clouds, fire and smoke textures, and mixes the water textures seamlessly as shown in (d). This is used to extend videos in space and time for video textures.

In these results we demonstrate the compositing of pairs of sequences. For applications such as video mosaics, editing, and texture synthesis this extends to multiple matching sequences. The sequences are sequentially cut, storing each mask, and the resulting 3D gradient field is reconstructed once. This ensures that the spatial and temporal appearance of the entire sequence is coherent.

6.7.2 Video trimap extraction

We have experimented with our method for video trimap extraction on various videos with uniform, textured and complex backgrounds. Figure 6.14 shows a frame from the result of video trimap extraction for a time-lapse sequence of a flower opening. The figure demonstrates the various steps of our method: (a) keyframe spline interpolation, (b-c) iterative 3D graph cuts, and a frame of the resulting trimap (d).

6.7.3 Video matting and completion

We first demonstrate 3D gradient video matting on a simple sequence with noticeable transparencies. A frame of a short sequence of a lioness is shown on the left of Figure 6.15 and a frame of the resulting gradient video matte on the right. Notice the fine detail on the chin and whiskers. Two-dimensional, frame-by-frame, gradient compositing and matting suffers from temporal artifacts and the advantage of our video matting approach is that it is performed in 3D and therefore results in a consistent matte which maintains temporal coherence. Figure 6.17 demonstrates the results of gradient video matting. The sequence contains 192 frames of 192 by 112 resolution, and contains four user specified key-frames. An input frame is shown in (a) and the corresponding graph cut trimap in (b). Foreground estimation in the unknown trimap region is shown in (c) and video background completion of the unknown trimap region in (d). The accurate video matte extracted by our 3D method is shown in (d) and a composite on a new background in (e). Total computation time is 170 seconds, including seven graph cut iterations and solving for 138565 unknowns in the matte computation.

6.8 Conclusions

The contributions of this work include the introduction and use of the gradient video compositing equation for smooth and sharp transitions between videos. This is performed by cutting followed by reconstruction of a 3D vector field, utilizing and maintaining the temporal coherence present in the video and performing iterative 3D graph cuts for capturing the contours of foreground video elements. It is based on extending the reconstruction of a vector field to 3D using the Poisson equation and considers motion in the cut computation. We provide a 3D formulation which reconstructs a temporally coherent output, avoiding temporal artifacts by spreading the reconstruction error of our least squares solution over the entire video volume. Finally, our results show useful applications for video mosaics, folding, video textures, compositing, video trimap extraction, video matting and background completion, within a unified framework.

Input: color video, binary foreground video mask

Output: foreground video trimap

Algorithm:

```
do
    set mask boundary to gray
    dilate gray boundary
    compute 3D minimum graph cut in boundary
    update new binary mask
until convergence
```

Figure 6.7: Iterative graph-cuts: trimap extraction pseudocode.

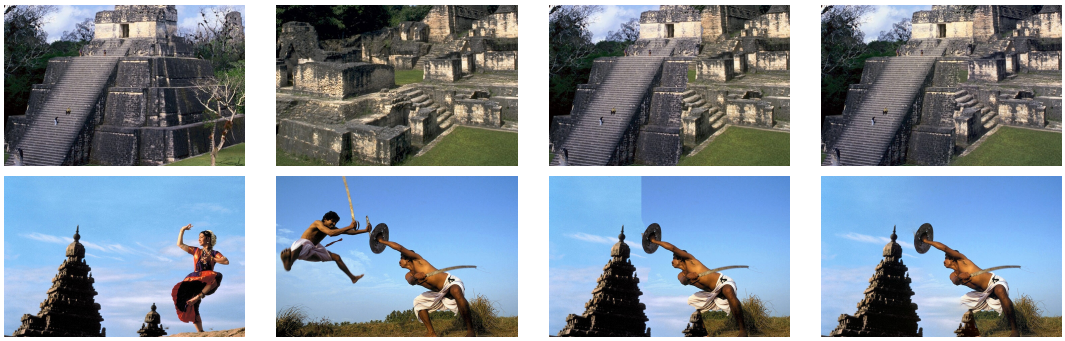
Input: color video, video trimap

Output: completed video background in gray trimap region

Algorithm:

```
compute optical flow
for each frame
    for each unknown pixel
        search time span from near to far
            if background pixel and
            flow magnitude near zero
                assign color, update mask, and end search
if unknown pixels remain
    fill in by diffusion
```

Figure 6.8: Background video completion pseudocode.



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

Figure 6.9: (a-b) Pair of input images. (c) 2D minimum graph cut. (d) Result of gradient image compositing.

(a) (b) (c)

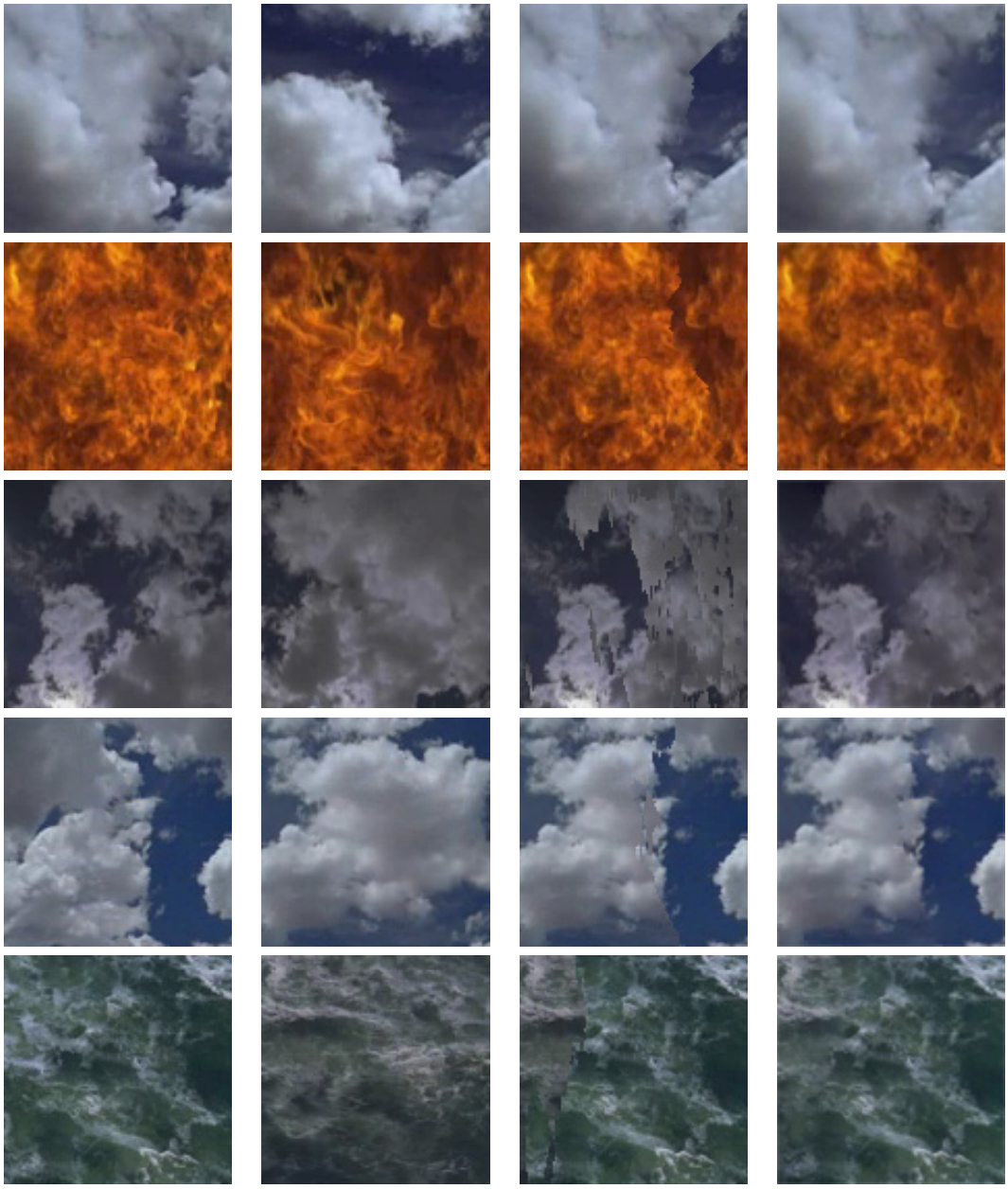
Figure 6.10: Video mosaic: (a-b) Input frames from a pair of time-lapse sequences. (c) Result of gradient video compositing.

(a) (b) (c)

Figure 6.11: Image folding: (a) Input image. (b) Cut. (d) Result of gradient image compositing.

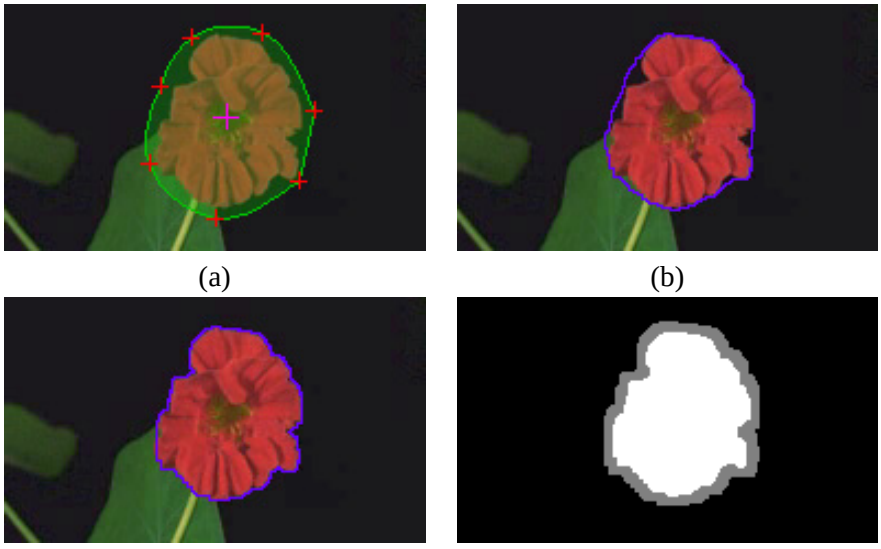
(a) (b) (c)

Figure 6.12: Spatial video folding: (a) Input frame. (b) Cut. (c) Result of gradient video compositing.



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

Figure 6.13: Spatial and temporal video textures: (a-b) Input frames from sequence pairs. (c) Cut. (d) Result of gradient video compositing.



(a) (b)

(c) (d)

Figure 6.14: Video trimap extraction: (a) Keyframe spline interpolation. (b) Initial slice of 3D graph cut. (c) Result of iterative 3D graph cut. (d) Graph cut trimap.

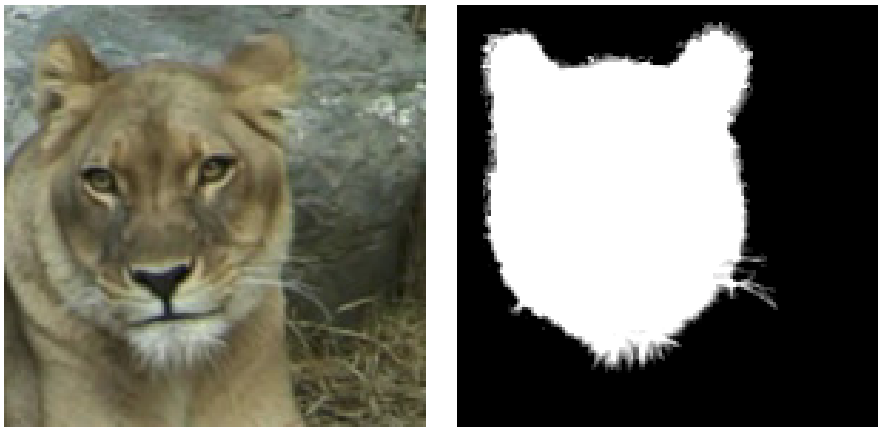
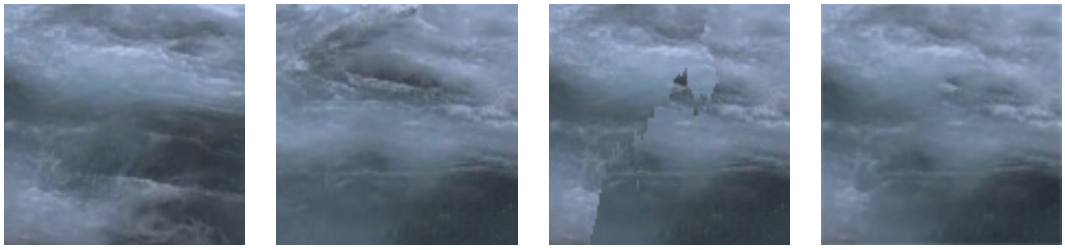
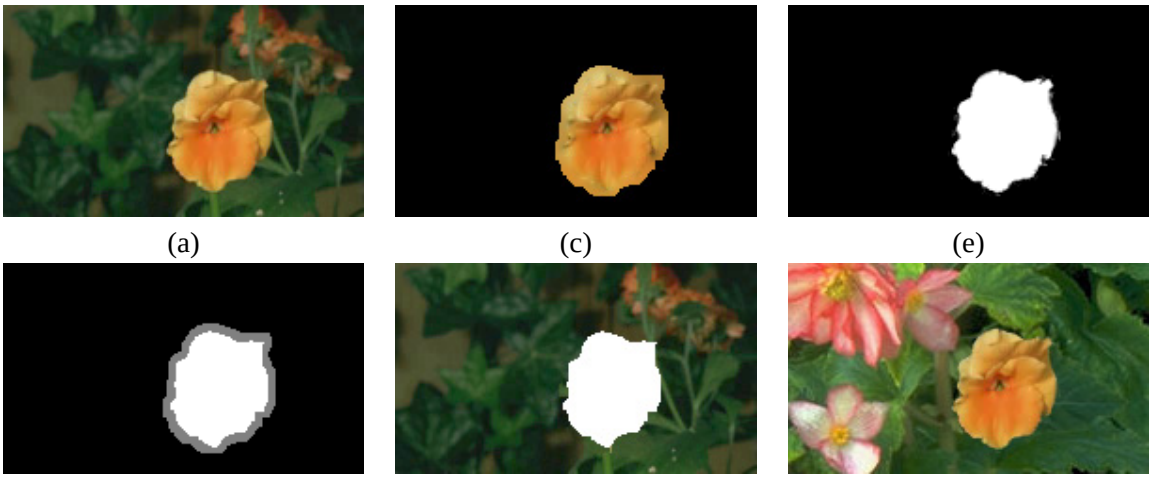


Figure 6.15: Video matting: frame from input video (left), and frame from resulting video matte (right).



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

Figure 6.16: Temporal video folding: (a) Input frame. (b) Cut. (c) Result of gradient video compositing.



(a) (c) (e)

(b) (d) (f)

Figure 6.17: Video matting and completion: (a) Frame from input video. (b) Graph cut trimap. (c) Foreground estimation in unknown region. (d) Background completion of unknown region. (e) Alpha matte. (f) Composite.

Chapter 7

Interactive Object Segmentation by Fitting Splines to Graph Cuts

Object segmentation in image sequences is one of the fundamental problems in computer vision and graphics. This problem is usually addressed either by discrete representations which are currently manifested by graph partitioning techniques, or by continuous methods typically referred to as active contours. In this work [38] we take a unified approach by fitting splines to graph cuts. The strengths of this approach stem from the dual discrete and continuous representations and from allowing the user to refine the result of the cut by fitting a new spline to it and modifying its points. Segmentation of an object in video is performed by a series of updates to the control points and computation of a minimum graph cut. Usually the graph cut results in a discrete representation over which the user has no control, and which is not always our desired result. Therefore our approach is to fit a spline to the resulting cut in key-frames. This allows the user to change the control points of the spline and then perform additional iterations of cut computation.

In order to extract an object from an input sequence the user first marks a point on the object and an area around it in a single frame. The object position is tracked forwards and backwards in time and the result is a set of discrete points (x,y,t). We fit a 3D spline to this set of points by a greedy algorithm.

First, all of the discrete points are part of the spline. Next, in each iteration, we find the point whose removal from the set results in a new spline minimizing the overall error between the tracker data and spline. The error is global and measures the distance between the spline and the collection of points. Fitting is performed until reaching a minimum number of key-frames. The result is refined by the user by adding key-frames and modifying positions.

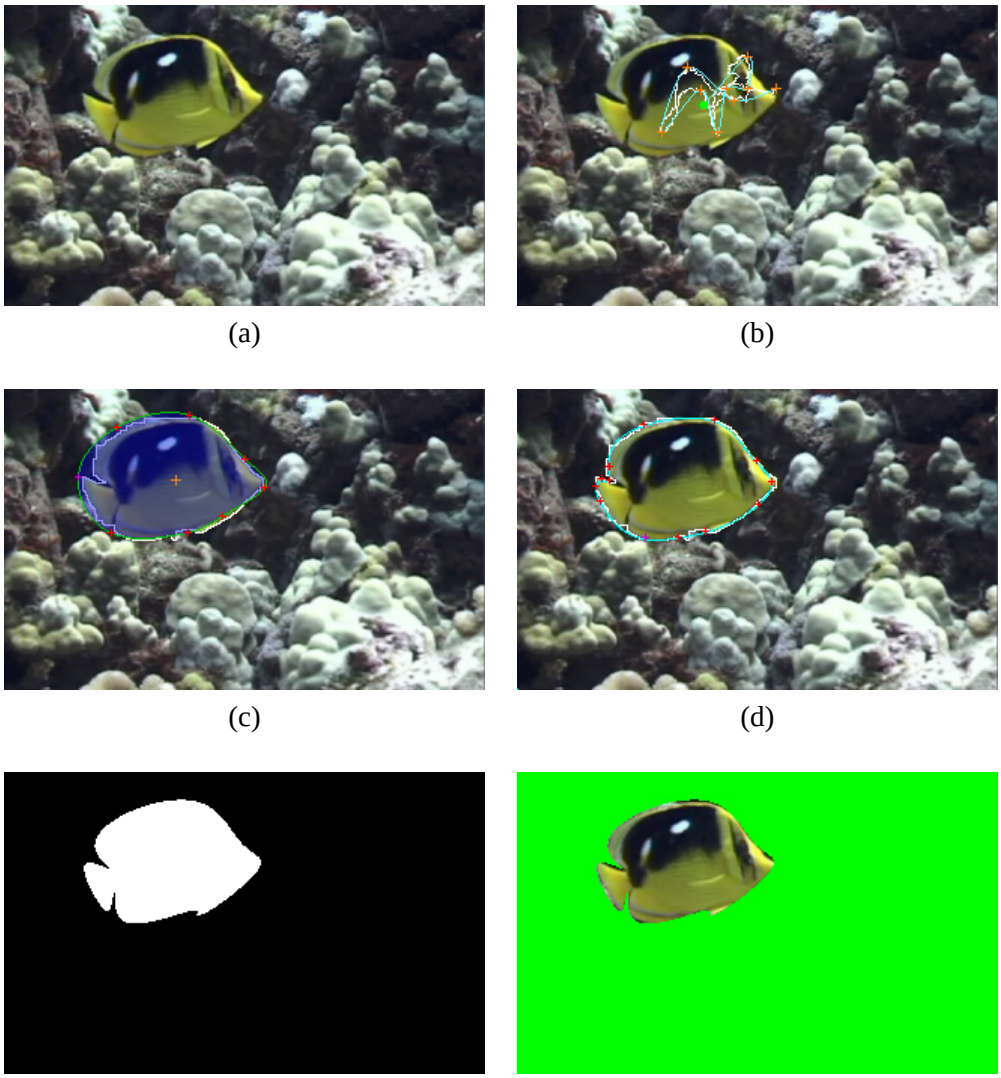
We use discrete and continuous representations for both object position and shape. After tracking the object position in each frame and fitting a 3D spline, the user roughly marks the object shape in a number of key-frames by points on a 2D closed spline. The last updated key-frame shape is propagated forward in time as a starting point for the following frames which the user selects to modify. The user can refine key-frame shapes by modifying spline points and applying rigid transformations. Shapes in the remaining frames are initially interpolated from key-frames. Next, the 2D shape is refined by iterative graph cuts.

Graph partitioning techniques are commonly used for object segmentation. We perform dilation on the spline boundaries and compute the minimum graph cut in the boundary region. Voxels on the inner boundary of the dilated region are connected to a source node and voxels on the outer boundary are connected to a sink. The rest of the voxels in the dilated region are connected regularly. The weight of each internal edge is inversely proportional to the gradient magnitude so that the resulting cut sticks to edges and transition regions in the video. We compute a spline approximation to the discrete set of data points resulting from the cut computation. We use a cubic B-spline representation and solve a constrained minimization problem iteratively. The knots of the spline are located automatically, given a single parameter which controls the tradeoff between the smoothness and closeness of fit. A small value results in an accurate spline with many knots, whereas a large value results in a smooth weighted least-squares polynomial with a few knots. Fitting is performed by computing successive least-squares splines. At each iteration computation of spline knots is adaptive, adding more in regions in which the function is difficult to approximate than where it is smooth. This gives a continuous representation which the user can further refine by modifying the spline and then re-computing a new graph cut.

The process of interleaving a discrete cut computation and spline fitting is performed until the user is satisfied with the mask. We have found this method of interaction to be simple and efficient, producing a detailed mask using relatively few steps, benefiting from the advantages of both representations. We

demonstrate the result of the process for an underwater image sequence by segmenting a fish. A frame of the input is shown in (a). The projection of the tracker data and the 3D spline fitted to the data are shown in (b). A user specified key-frame and the result of the iterative graph cut are shown in (c). A new spline is fitted to the graph cut as shown in (d) and modified. The resulting mask is shown in (e) and a composite on a green screen is shown in (f). Our approach to object segmentation has useful applications in video compositing, matting and video completion.

We have compared our approach for object segmentation in video with graph cut segmentation on an image sequence. Figure 7.2 show the graph cut result [122] in white compared with our result of spline fitting in cyan. Notice the various degrees of smoothness shown in the close up images.



(a) (b)

(c) (d)

(e) (f)

Figure 7.1: Object segmentation in video: (a) Input frame. (b) Projected tracker data and 3D spline. (c) Key-frame interpolation and minimum graph cut. (d) Spline fitting to graph cut. (e) Binary mask. (f) Composite on green screen.

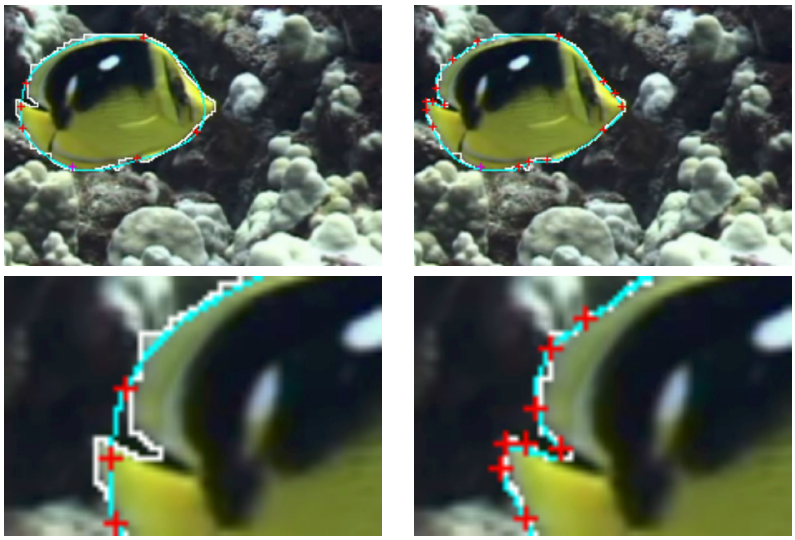


Figure 7.2: Comparison: Graph cut is shown in white. Our result of spline fitting with various degrees of smoothness is shown in cyan. (Bottom) close up view.

Bibliography

[1] Adobe Systems Incorporated. http://www.adobe.com.

[2] Brett Allen, Brian Curless, and Zoran Popovic. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Transactions on Graphics*, 22(3):587–594, 2003.

[3] Jont B. Allen. Short term spectral analysis, synthesis, and modification by discrete Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238, 1977.

[4] Nicholas Apostoloff and Andrew Fitzgibbon. Bayesian video matting using learnt image priors. In *BMVA symposium on Spatiotemporal Image Processing*, 2004.

[5] Michael Ashikhmin. Synthesizing natural textures. In *ACM Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[6] Simon Baker and Takeo Kanade. Limits on super-resolution and how to break them. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 372–379, 2000.

[7] Simon Baker and Takeo Kanade. Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1167–1183, 2002.

[8] Ronen Basri and David Jacobs. Lambertian reflectance and linear subspaces. In *IEEE International Conference on Computer Vision*, volume 2, pages 383–390, 2001.

[9] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of ACM SIGGRAPH 2000*, pages 417–424. ACM Press, 2000.

[10] Marcelo Bertalmio, Luminita Vese, Guillermo Sapiro, and Stanley Osher. Simultaneous structure and texture image inpainting. *IEEE Transactions on Image Processing*, 12(8):882–889, 2003.

[11] David Beymer and Tomaso Poggio. Image representation for visual learning. *Science*, 272:1905–1909, 1996.

[12] David Beymer, Amnon Shashua, and Tomaso Poggio. Example based image analysis and synthesis. MIT AI Lab Technical Memo 1431, 1993.

[13] Dinkar N. Bhat and Shree K. Nayar. Ordinal measures for image correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:415–423, 1998.

[14] Michael J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[15] Andrew Blake and Michael Isard. *Active Contours*. Springer-Verlag, 1998.

[16] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of ACM SIGGRAPH 99*, pages 187–194, 1999.

[17] Eran Borenstein and Shimon Ullman. Class-specific, top-down segmentation. In *European Conference on Computer Vision*, pages 109–124, 2002.

[18] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[19] Matthew Brand and Aaron Hertzmann. Style machines. In *ACM Siggraph*, pages 183–192, 2000.

[20] Achi Brandt. Multiscale scientific computation: Review 2001. In T.J. Barth, T.F. Chan, and R. Haimes, editors, *Multiscale and Multiresolution Methods: Theory and Applications*. Springer Verlag, 2001.

[21] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video rewrite: driving visual speech with audio. In *ACM Siggraph*, pages 353–360, 1997.

[22] Stephen Brooks and Neil Dodgson. Self-similarity based texture editing. *ACM Transactions on Graphics,*, 21(3):653–656, 2002.

[23] Peter J. Burt and Edward H. Adelson. Merging images through pattern decomposition. *Applications of Digital Image Processing VIII*, 575:173–181, 1985.

[24] J. D. Carroll and J. J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35:283–319, 1970.

[25] Wei Chai and Barry Vercoe. Structural analysis of musical signals for indexing and thumbnailing. In *Joint Conference on Digital Libraries*, pages 27–34, 2003.

[26] Tony Chan and Jianhong Shen. Mathematical models for local nontexture inpainting. *SIAM Journal on Applied Mathematics*, 62(3):1019–1043, 2001.

[27] Yung-Yu Chuang, Aseem Agarwala, Brian Curless, David H. Salesin, and Richard Szeliski. Video matting of complex scenes. *ACM Transactions on Graphics (TOG)*, 21(3):243–248, 2002.

[28] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, 2001.

[29] Cornell Lab of Ornithology. http://birds.cornell.edu.

[30] Antonio Criminisi, Patrick Perez, and Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 2004, to appear.

[31] Ronald E. Crochiere. A weighted overlap-add method of short-time Fourier analysis/synthesis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):99–102, 1980.

[32] James Davis. Mosaics of scenes with moving objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 354–360, 1998.

[33] James W. Davis and Hui Gao. An expressive three-mode principal components model of human action style. *Image and Vision Computing*, 21(11):1001–1016, 2003.

[34] David L. Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[35] Iddo Drori, Daniel Cohen-Or, and Yehezkel Yeshurun. Example-based style synthesis. In *IEEE International Conference on Computer Vision and Pattern Recognition*, volume 2, pages 143–150, 2003.

[36] Iddo Drori, Daniel Cohen-Or, and Yehezkel Yeshurun. Fragment-based image completion. *ACM Transactions on Graphics (TOG)*, 22(3):303–312, 2003.

[37] Iddo Drori, Alon Fishbach, and Yehezkel Yeshurun. Spectral sound gap filling. In *IEEE International Conference on Pattern Recognition*, volume 2, pages 871–874, 2004.

[38] Iddo Drori, Tommer Leyvand, Daniel Cohen-Or, and Yehezkel Yeshurun. Interactive object segmentation by fitting splines to graph cuts. In *Siggraph Posters session*, 2004.

[39] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd ed., 2001.

[40] Alexei Efros and Thomas Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, 1999.

[41] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of ACM SIGGRAPH 2001*, pages 341–346. ACM Press, 2001.

[42] L. E. Ericson and E. G. Stickel. A proposed classification system for ceramics. *World Archaeology*, 4(3):357–367, 1973.

[43] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. *ACM Transactions on Graphics,*, 21(3):249–256, 2002.

[44] David J. Field. Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America*, 4(12):2379–2394, 1987.

[45] Graham D. Finlayson, Steven D. Hordley, and Mark S. Drew. Removing shadows from images. In *European Conference on Computer Vision*, pages 823–836, 2002.

[46] Alon Fishbach. Primary segmentation of auditory scenes. In *Proceedings of IEEE International Conference on Pattern Recognition*, pages 113–117, 1994.

[47] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. *ACM Transactions on Graphics (TOG)*, 22(3):950–953, 2003.

[48] Jonathan Foote. Automatic audio segmentation using a measure of audio novelty. In *Proceedings of IEEE International Conference on Multimedia and Expo*, pages 452–455, 2000.

[49] William T. Freeman, Thouis R. Jones, and Egon Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, pages 56–65, 2002.

[50] William T. Freeman and Egon Pasztor. Learning low-level vision. In *IEEE International Conference on Computer Vision*, pages 182–189, 1998.

[51] William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[52] William T. Freeman and Joshua B. Tenenbaum. Learning bilinear models for two-factor problems in vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 554–560, 1997.

[53] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 2002.

[54] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of ACM SIGGRAPH 96*, pages 43–54. ACM Press, 1996.

[55] Robert Gray, Andres Buzo, Augustine Gray, and Yasuo Matsuyama. Distortion measures for speech processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):367–376, 1980.

[56] D. W. Griffin and J. S. Lim. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

[57] Gideon Guy and Gerard Medioni. Inferring global perceptual contours from local features. *IEEE International Journal of Computer Vision*, (1–2):113–133, 1996.

[58] Paul Haeberli. Paint by numbers: Abstract image representations. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, pages 207–214. ACM Press, 1990.

[59] R. Harshman. Foundations of the parafac procedure: Model and conditions for an explanatory multi-mode factor analysis. *UCLA Working Papers in phonetics*, 16:1–84, 1970.

[60] R. Harshman, P. Ladefoged, and L. Goldstein. Factor analysis of tongue shapes. *Journal of the Acoustical Society of America*, 62(3):693–707, 1977.

[61] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of ACM SIGGRAPH 95*, pages 229–238. ACM Press, 1995.

[62] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *ACM Siggraph*, pages 453–460, 1998.

[63] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of ACM SIGGRAPH 2001*, pages 327–340. ACM Press, 2001.

[64] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M. Seitz. Curve analogies. In *Proceedings of 13th Eurographics Workshop on Rendering*, pages 233–245, 2002.

[65] Anil N. Hirani and Takashi Totsuka. Combining frequency and spatial domain information for fast interactive image noise removal. In *Proceedings of ACM SIGGRAPH 96*, pages 269–276. ACM Press, 1996.

[66] Homan Igehy and Lucas Pereira. Image replacement through texture synthesis. In *IEEE International conference on Image Processing*, volume 3, pages 186–189, 1997.

[67] Michal Irani, Benny Rousso, and Shmuel Peleg. Computing occluding and transparent motions. *International Journal of Computer Vision*, 12:5–16, 1994.

[68] Jiaya Jia and Chi Keung Tang. Inference of segmented color and texture description by tensor voting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004, to appear.

[69] Arie Kapteyn, Heinz Neudecker, and T. Wansbeek. An approach to n-mode components analysis. *Psychometrika*, 51:269–275, 1986.

[70] M. Kass, A. Witkin, and D. Trezopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[71] Willett Kempton. *The Folk Classification of Ceramics: A Study of Cognitive Prototypes*. Academic Press, 1981.

[72] Kurt Koffka. *Principles of Gestalt Psychology*. New York, Hartcourt, Brace and World, 1935, 1967.

[73] Tamara G. Kolda. Orthogonal tensor decompositions. *SIAM Journal on Matrix Analysis and Applications*, 23(1):243–255, 2001.

[74] Pieter M. Kroonenberg and J. de Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45:69–97, 1980.

[75] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (TOG)*, 22(3):277–286, 2003.

[76] Edwin H. Land and John McCann. Lightness and retinex theory. *Journal of the Optical Society of America*, 1(61):1–11, 1971.

[77] Lieven De Lathauwer. *Signal Processing Based on Multilinear Algebra*. PhD thesis, Katholieke Universiteit Leuven, 1997.

[78] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. On the best rank-1 and rank-(r1,r2,...,rn) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2001.

[79] Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. Seamless image stitching in the gradient domain. In *Proc. of the European Conference on Computer Vision, to appear*, 2004.

[80] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150, 2001.

[81] Jan R. Magnus and Heinz Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. Wiley series in probability and statistics, 1999.

[82] Ravikanth Malladi, James A. Sethian, and Baba C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175, 1995.

[83] Tomoo Mitsunaga, Taku Yokoyama, and Takashi Totsuka. Autokey: Human assisted key extraction. In *Proceedings of ACM SIGGRAPH 95*, pages 265–272, 1995.

[84] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *Proceedings of ACM SIGGRAPH 95*, pages 191–198, 1995.

[85] Alva Noe, Luiz Pessoa, and Evan Thompson. Finding out about filling-in: A guide to perceptual completion for visual science and the philosophy of perception. *Behavioral and Brain Sciences*, (6):723–748, 796–802, 1998.

[86] Byong Mok Oh, Max Chen, Julie Dorsey, and Fredo Durand. Image-based modeling and photo editing. In *Proceedings of ACM SIGGRAPH 2001*, pages 433–442. ACM Press, 2001.

[87] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[88] Stephen Palmer. *Vision Science*. MIT Press, 1999.

[89] Patrick Perez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics (TOG)*, 22(3):313–318, 2003.

[90] William Matthew Flinders Petrie. *Abydos Part I: Twenty-Second Memoir of the Egypt Exploration Fund/Society*. University College London Department of Egyptology, 1902.

[91] Pixeldust@. 2d3. http://www.2d3.com, 2003.

[92] Robert Pless. Image spaces and video trajectories: Using isomap to explore video sequences. In *IEEE International Conference on Computer Vision*, pages 1433–1440, 2003.

[93] Thomas Porter and Tom Duff. Compositing digital images. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, pages 253–259, 1984.

[94] Lawrence R. Rabiner and Ronald W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.

[95] Zia Rahman, Daniel J. Jobson, Glenn A. Woodell, and G. D. Hines. Multi-sensor fusion and enhancement using the retinex image enhancement algorithm. In *Visual Information Processing XI Proceedings of SPIE*, volume 4736, 2002.

[96] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *ACM Siggraph*, pages 117–128, 2001.

[97] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer Graphics and Applications*, pages 34–40, 2001.

[98] Prudence M. Rice. *Pottery Analysis*. The University of Chicago Press, 1987.

[99] Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[100] Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley Publishing, 1989.

[101] Eric Scheirer and Malcolm Slaney. Construction and evaluation of a robust multifeature speech/-music discriminator. In *Proceedings of International Conference on Acoustic, Speech, and Signal Processing*, pages 1331–1334, 1997.

[102] Sebastian H. Seung and Daniel D. Lee. The manifold ways of perception. *Science*, 290:2268–2269, 2000.

[103] Eitan Sharon, Achi Brandt, and Ronen Basri. Completion energies and scale. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1117–1131, 2000.

[104] Peter-Pike Sloan, Charles Rose, and Michael Cohen. Shape by example. In *ACM Symposium on Interactive 3D Graphics*, pages 135–143, 2001.

[105] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. *ACM Transactions on Graphics,*, 21(3):673–680, 2002.

[106] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. *ACM Transactions on Graphics, to appear*, 2004.

[107] Richard Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, pages 22–30, 1996.

[108] TAUCS. A Library of Sparse Linear Solvers, Tel-Aviv University. http://www.tau.ac.il/ stoledo/-taucs/, 2003.

[109] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[110] Joshua B. Tenenbaum and William T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12(6):1247–1283, 2000.

[111] Andy Tsai, Anthony Yezzi Jr., William Wells, Clare Tempany, Dewey Tucker, Ayres Fan, W. Eric Grimson, and Alan Willsky. A shape-based approach to the segmentation of medical imagery using level sets. *IEEE Transactions on Medical Imaging*, 22(2):137–154, February 2003.

[112] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

[113] Greg Turk and James F. OBrien. Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH 99*, pages 335–342, 1999.

[114] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuro Science*, 3(1):71–86, 1991.

[115] Shimon Ullman, Michel Vidal-Naquet, and Erez Sali. Visual features of intermediate complexity and their use in classification. *Nature Neuroscience*, 5(7):1–6, 2002.

[116] M. Alex O. Vasilescu and Demetri Terzopoulos. Multilinear subspace analysis of image ensembles. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 93–99, June 2003.

[117] John Y. A. Wang and Edward H. Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5):625–638, 1994.

[118] Li Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM SIGGRAPH 2000*, pages 479–488. ACM Press, 2000.

[119] Yair Weiss. Deriving intrinsic images from image sequences. In *Proceedings of International Conference on Computer Vision*, pages 68–75, 2001.

[120] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. *ACM Transactions on Graphics,*, 21(3):277–280, 2002.

[121] Lance Williams and David W. Jacobs. Stochastic completion fields: A neural model of illusory contour shape and salience. *Neural Computation*, 9(4):837–858, 1997.

[122] Ning Xu, Ravi Bansal, and Narendra Ahuja. Object segmentation using graph cuts based active contours. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 46–53, 2003.

[123] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics (TOG)*, 2004.

[124] Tong Zhang and Gene H. Golub. Rank-one approximation to high order tensors. *SIAM Journal on Matrix Analysis and Applications*, 23(2):534–550, 2001.