

Example Based Processing For Image And Video Synthesis

A Dissertation
Presented to
The Academic Faculty

by

Antonio Haro

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
November 2003

Example Based Processing For Image And Video Synthesis

Approved:

Irfan Essa, Chairman

Aaron Bobick

Sing Bing Kang
(Microsoft Research)

Jim Rehg

Greg Turk

Date Approved: 11/13/2003

ACKNOWLEDGMENTS

Primero de todo, me gustaria darle gracias a mis padres y hermano. Sin su amor y soporte desde pequeño, nada de esto hubiera sido posible.

Thanks also go to my CPL and GVU colleagues and friends for providing an excellent and stimulating research environment. In particular, I would like to thank Gabriel Brostow, Scott Carter, Vivek Kwatra, Arno Schödl, Drew Steedly and Tee Tanawongsuwan for discussions that ranged from the serious to the ridiculous.

I would also like to thank my advisor, Irfan Essa, for providing the opportunity to do this work. He provided excellent support at all stages of my graduate career, from the very beginning before I had read any papers and was just getting started. In addition, I am grateful to my thesis committee: Aaron Bobick, Sing Bing Kang, Jim Rehg, and Greg Turk, who provided thoughtful criticism and helpful comments.

I also appreciate the support of the AT&T Foundation, which funded the research in this dissertation. Last, but certainly not least, I thank all of my friends in Atlanta for the great times the past few years.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	iii
LIST OF FIGURES	vii
SUMMARY	x
1 INTRODUCTION	1
2 RELATED WORK	4
2.1 Texture synthesis	4
2.2 Video synthesis	5
2.3 Example based processing	5
2.3.1 Example based rendering	6
2.3.2 Example based image processing	6
3 NON-PARAMETRIC EXAMPLE BASED PROCESSING	8
3.1 Generalized example based processing	14
3.2 Approximation algorithm	16
3.3 Results	19
3.3.1 Image processing	21
3.3.2 Video processing	21
3.3.3 3D surface material property transfer	28
3.3.3.1 Motivation	31
3.3.3.2 Image synthesis	31
3.3.3.3 Results	34

4	PROBABILISTIC EXAMPLE BASED PROCESSING	36
4.1	Color normalization	37
4.2	Belief Propagation	39
4.3	Results & Discussion	42
4.3.1	Texture transfer	42
4.3.2	Super-resolution	43
4.3.3	Non-photorealistic rendering	43
5	EXPERIMENTAL EVALUATION	53
5.1	Super-resolution experiments	54
5.2	Emboss experiments	59
5.3	Emboss (textured images) experiment	64
5.4	Multiple training sources experiment	67
5.5	Large number of training sources experiments	69
5.5.1	Differing amounts of training data	71
5.5.2	Test input: photographs	71
5.5.3	Test input: anisotropically blurred	71
6	SYSTEMS ISSUES	80
6.1	Memory usage	80
6.1.1	Clustering	81
6.1.2	Dimensionality reduction	81
6.2	Approximate nearest neighbors	87
6.3	Patch size variation	88
7	3D SURFACE NORMAL SYNTHESIS BY EXAMPLE	91
7.1	Normal Map Capture	93
7.2	Skin Growing	96
7.2.1	Normal map synthesis	96
7.2.2	Skin stitching	97
7.3	Skin Rendering	98

7.3.1	Per-pixel Bump mapping	99
7.3.2	Lafortune Shading of Skin	101
7.4	Results	101
8	CONTRIBUTIONS	105
9	DISCUSSION	107
9.1	Different data types	107
9.2	Different features	108
9.3	Larger datasets	108
9.4	One to many functions	109
9.5	Objects	109
9.6	Conclusion	109
	REFERENCES	111
	VITA	116

LIST OF FIGURES

1	Filter bank used for the example based processing experiments in this section. . . .	9
2	Font reconstruction experiment.	10
3	Six out of 20 faces that were used for the charcoal portrait experiment.	11
4	Charcoal portrait experiment output.	12
5	Six out of 72 shrub images that were used for the pen and ink experiment.	13
6	Pen and ink experiment output.	14
7	Initial matching: nearest neighbor.	16
8	Subsequent matching: coherence matching.	17
9	Comparison with image analogies.	20
10	Overview of video processing pipeline.	21
11	Components of our feature vector for video.	22
12	Video feature vector matching.	23
13	Color correction experiment results.	25
14	Noise removal experiment results.	26
15	Motion blur experiment results.	26
16	Painting style experiment results.	27
17	Training data used in Figure 18.	28
18	Results of non-photorealistic video rendering.	29
19	Rendering from example pipeline.	30
20	Results of the algorithm run on different materials, lighting conditions, and 3D models.	33
21	Renderings of the Stanford bunny 3D model dataset using chocolate and orange as materials under varying illumination.	35
22	Renderings of the Stanford bunny using various materials photographed outdoors. .	35
23	Final results of our algorithm when different color normalizations are used.	37
24	Texture transfer results.	46

25	Super-resolution results.	47
26	<i>Reflection (Self-Portrait)</i> , 1985 by Lucian Freud results.	48
27	<i>Full Fathom Five</i> , 1947 by Jackson Pollock results.	49
28	<i>The Church at Moret</i> , 1894 by Alfred Sisley results.	50
29	<i>Road with Cypress and Star</i> , 1890 by Vincent Van Gogh results.	51
30	<i>Self-Portrait with Bandaged Ear</i> , 1889 by Vincent Van Gogh results.	52
31	Super-resolution experiment: Training images used in experiments.	55
32	Super-resolution experiment: B image errors.	56
33	Super-resolution experiment: Circle image errors.	56
34	Super-resolution experiment: Square image errors.	57
35	Super-resolution experiment: X image errors.	57
36	Emboss experiment: Training images used in experiments.	58
37	Emboss experiment: B image errors.	59
38	Emboss experiment: Circle image errors.	59
39	Emboss experiment: Square image errors.	60
40	Emboss experiment: X image errors.	60
41	Emboss experiment with textured images: Training images used in experiments. . .	62
42	Different outputs using the ‘B’ image as input and other images as training from Figure 41 (a).	63
43	Emboss experiment with textured images: B image errors.	64
44	Emboss experiment with textured images: Circle image errors.	65
45	Emboss experiment with textured images: Square image errors.	66
46	Emboss experiment with textured images: X image errors.	66
47	Six out of 20 faces that were used for the convolution experiment.	67
48	Multiple training sources experiment.	68
49	RMS error versus ground truth for different numbers of randomly picked training images.	68
50	Kernel used for convolution experiments.	69
51	Varying the amount of data in the Monet training set.	70
52	Monet training dataset outputs.	73

53	Additional Monet training dataset outputs.	74
54	Outputs from altered training input photograph.	75
55	Monet leave one out experiment 1.	76
56	Monet leave one out experiment 2.	77
57	Monet leave one out experiment 3.	78
58	Monet leave one out experiment 4.	79
59	Effects of dimensionality reduction on the hay texture transfer dataset.	83
60	RMS error of varying numbers of eigenvectors versus not using SVD for hay texture transfer dataset.	83
61	Effects of dimensionality reduction on non-photorealistic impressionism dataset.	84
62	RMS error of varying numbers of eigenvectors versus not using SVD for non-photorealistic impressionism dataset.	84
63	Effects of dimensionality reduction on super-resolution dataset.	86
64	RMS error of varying numbers of eigenvectors versus not using SVD for super-resolution dataset.	86
65	Varying pixel patch sizes and overlap.	90
66	Silicone mold of skin (about the size of a nickel).	94
67	Cheek and edge of forehead normal maps.	95
68	Middle of forehead and nose normal maps.	95
69	Initial normal map captured from mold and synthesized map.	97
70	Fine scale skin structure blending.	97
71	Per-pixel bumpmapping pipeline.	98
72	Skin rendering results.	103
73	Skin rendering results under varying illumination.	104

SUMMARY

The example based processing problem can be expressed as: “Given an example of an image or video before and after processing, apply a similar processing to a new image or video”. Our thesis is that there are some problems where a single general algorithm can be used to create varieties of outputs, solely by presenting examples of what is desired to the algorithm. This is valuable if the algorithm to produce the output is non-obvious, e.g. an algorithm to emulate an example painting’s style. We limit our investigations to example based processing of images, video, and 3D models as these data types are easy to acquire and experiment with.

We represent this problem first as a texture synthesis influenced sampling problem, where the idea is to form feature vectors representative of the data and then sample them coherently to synthesize a plausible output for the new image or video. Grounding the problem in this manner is useful as both problems involve learning the structure of training data under some assumptions to sample it properly. We then reduce the problem to a labeling problem to perform example based processing in a more generalized and principled manner than earlier techniques. This allows us to perform a different estimation of what the output should be by approximating the optimal (and possibly not known) solution through a different approach.

CHAPTER 1

INTRODUCTION

In this dissertation, we propose methods to perform example based processing. The example based processing problem can be expressed as: “Given an example of an image or video before and after processing, apply a similar processing to a new image or video”. Our thesis is that there are some problems where a single general algorithm can be used to create varieties of outputs, solely by presenting examples of what is desired to the algorithm. This is valuable if the algorithm to produce the output is non-obvious, e.g. an algorithm to emulate an example painting’s style. We limit our investigations to example based processing of images, video, and 3D models as these data types are easy to acquire and experiment with. We represent this problem first as a texture synthesis influenced sampling problem, where the idea is to form feature vectors representative of the data and then sample them coherently to synthesize a plausible output for the new image or video. Grounding the problem in this manner is useful as both problems involve learning the structure of training data under some assumptions to sample it properly. We then reduce the problem to a labeling problem to perform example based processing in a more generalized and principled manner than earlier techniques. This allows us to perform a different estimation of what the output should be by approximating the optimal (and possibly not known) solution through a different approach.

Example based processing is by its nature a very difficult problem since the training data will be too sparse to establish direct input and output mappings of pixel patches. A single exemplar of the type of processing that is to be approximated is not sufficient to represent how the processing algorithm observed in the training data would process a new piece of data. However, if the *exact* output is not required, it is possible to estimate what the processing would do to a new piece of data. The training data itself, with colors and contrast possibly shifted, can be sampled in such a way as to approximate the output of the processing. While more examples could be acquired, it is

very useful to be able to use just a single example as the processing represented in the example pair could be too time-consuming or impossible to repeat numerous times to construct other examples.

We theorize that the only factors that differentiate example based processing algorithms for different problem domains and data types from each other are the choice of features to use for the feature vectors and the technique to match them. By varying the choices of features and matching algorithms, this thesis presents several algorithms to perform example based processing of images, video and 3D models. Its contributions are:

- A novel approach to perform example based processing of images. We present a new framework influenced by probabilistic inference rather than texture synthesis techniques. Our approach estimates output images without copying patches easily visible in the training data.
- A method to process videos by example. An example pair of videos is input showing a video before and after processing. A new video can then be processed similarly with no knowledge of the processing observed. We show that several types of video processing can be represented as a series of purely local pixel neighborhood operations in space and time. Treating the video processing operations in this manner allows us to generalize the processing to new input sequences.
- Synthesis of 3D surface normals from a small sample. A normal field can be modeled as an MRF as nearby locations on the surface are highly correlated, especially if the surface is smoothly / non-varying in curvature. We use this technique to create and render realistic skin microstructure from small molds of skin.
- Realistic rendering of 3D models based on a single photograph of the source material to be used for rendering. Novice users can use this technique to create photorealistic renderings with only a digital camera and a simple user interface. Our work in this area bridges the gap between pure 2D texture synthesis methods and full bidirectional texture function synthesis.

The component that ties all of these approaches together is the treatment of each problem domain as a Markov random field (MRF). An MRF is a multidimensional version of a Markov chain, where nodes modeling a process are arranged in a graph instead of a chain. Nodes have strong dependencies on nodes that are nearby and no or little dependence on nodes that are not. These

relationships are typically given *a priori* in training data/measurements or sometimes estimated to infer higher-level information about each node in the field. MRFs have proven to be a useful representation for images and video as the local neighborhood of each pixel in the input will determine to a large extent what the output should be, and there is much prior work on solving different MRF problems. MRFs are a useful framework for example based processing because of this reason; local state and/or labeling choices make these problems tractable since only small finite neighborhoods need to be considered at a time. The contribution of each approach is in how to cast the problem into this framework. However, as a consequence, the results produced by these algorithms should not be thought of as full solutions to these problems. Rather, these are solutions that *estimate* what the actual process being modeled would synthesize given new data. In some cases, such as art stylization, the process is unknown and as a consequence the results are very subjective.

These algorithms are presented as proof that even though the actual processes could possibly be unknown, satisfactory results can still be achieved. Quantifying these results whether or not ground truth is available is difficult, since in most cases only a single example is used, and for most processing this is simply not enough to model it. As a result, we do not claim in this thesis to ever approximate the processing completely. Rather, these algorithms should be viewed as tools to produce results in a similar *style* as the training, where we define style as being similar in color distribution, global appearance, and temporal continuity (if in the time domain).

We begin by discussing related work in the next chapter. Chapter 3 presents a method for performing non-parametric example based processing. This technique is related to texture synthesis algorithms, and relies on sampling from the training data coherently to synthesize an estimated output for given input data. Chapter 4 shows how the relationship between example based processing and MRF labeling problems allows us to compute results that do not exhibit visibly copied patches from the training data, yet are comprised of pixels from the training data. Chapter 5 presents experiments that more deeply investigate the properties of these algorithms. In Chapter 6, we discuss various issues related to the design and implementation of example based processing algorithms. Chapter 7 discusses how to estimate and then synthesize 3D surface normals for more realistic skin microstructure, applying image synthesis techniques on surfaces.

CHAPTER 2

RELATED WORK

Example based image and video processing is a relatively new research area. However, its roots extend back into work from machine learning, computer vision, and computer graphics as these are the building blocks of the field. It is similar to image based rendering in this respect, although with a heavier emphasis on machine learning.

2.1 Texture synthesis

Texture synthesis is the strongest ancestor of example based processing research because the texture synthesis problem is a variant of the same problem: given an example image, synthesize a larger image that resembles the example. Popat and Picard [47] performed texture synthesis by modeling texture image sources as semiparametric probability mass functions and then sampling pixels using causal neighborhoods in scanline order to perform synthesis. Heeger and Bergen [21] relied on constructing two Laplacian pyramids: one of the source texture and one of white noise. Each pyramid level of the white noise pyramid was then histogram matched with the corresponding level of the texture pyramid. Once the histogram matched white noise pyramid is collapsed, a texture similar to the exemplar is produced. The approach of De Bonet [8] was a non-parametric sampling based approach to the problem. However, this algorithm, like previous approaches, did not work well on a large number of texture images. These approaches can be thought of to be heavily influenced by work in texture analysis and psychophysics.

The idea of treating texture images as Markov random fields (MRFs) instead to aid in synthesis was proposed by Efros and Leung [12]. Treating the texture synthesis problem in this manner produced better results than previous approaches, and allowed for novel applications such as hole filling in an incomplete texture image. Wei and Levoy [65] have a similar MRF-based algorithm

which made use of vector quantization techniques to achieve better results more efficiently. Their work was recently extended to perform image synthesis on 3D surfaces [63, 66]. Ashikhmin [1] was the next to significantly advance work on the 2D texture synthesis problem with the realization that the best texture synthesis results come from copying. This idea marked a departure from treating the texture synthesis problem strictly as an MRF sampling problem, which allowed for better results. The problem can be reduced even further by pasting random square blocks from the input texture and then hiding the seams by blending [38] or using dynamic programming [13]. Patch-based texture synthesis can also be performed by cutting and pasting irregularly shaped blocks from the source texture using graph cuts [33].

2.2 Video synthesis

Video synthesis is relevant because it is also a form of example based processing, where a source video is provided and some cost function guides the creation of new sequences using the example sequence. Bregler *et al.* [5] used phonemic representations to segment an audio track from a video into small sequences that are then morphed to match a new vocal track. This work, like the texture synthesis work of De Bonet [8], was influential in its combination of ideas from machine learning and pattern recognition to address computer graphics problems.

The idea [1] that the most photo-realistic image synthesis results come from directly copying from the training data was independently applied towards video synthesis by [56, 55]. In their work, a cost function is optimized and Q-learning is used to create looped and controllable video sequences. By using whole frames already present in the input video and reducing the problem to figuring out an arbitrarily long proper coherent ordering of these frames, highly photo-realistic results can be synthesized.

2.3 Example based processing

The increasing overlap between computer graphics, vision, and machine learning inspired several researchers to create example based processing algorithms. Processing by example is very valuable since some of these problems, such as non-photorealistic rendering, are very difficult to express

otherwise.

2.3.1 Example based rendering

Example based rendering is a practical domain because rendering algorithms for 3D models could be non-intuitive to invent. In the work of Hamel and Strothotte [19] a model is rendered with a non-photorealistic renderer, and then has its rendering “style” transferred to a new model by matching various 2D representations of both 3D models such as curvature and shadows, as in G-buffers [54]. The work of Freeman *et al.* [15] is also pioneering in its usage of machine learning to create an example based stroke drawing system. In their work, they transfer a line drawing style by forming linear combinations of subsets of exemplar line strokes.

2.3.2 Example based image processing

Leung and Malik [36] did not work on example based image processing directly, but their work was influential in our initial work in this area. In their work, they used textons to form a basis for all images of texture. Textons are long vectors comprised of the response of a pixel neighborhood to a filter bank defined in their work. The vectors are clustered in a high dimensional space to determine the most descriptive exemplars for classification. One very interesting application shown in this work is the synthesis of new texture images under novel lighting conditions. This is done by taking a set of images of an input texture under various known lighting conditions and computing the filter responses to all pixels in the image. Once the responses are known, the closest texton in the data can be identified for those conditions, and the pixel neighborhood the texton describes can be looked up for a new lighting condition. If this is done for every pixel in the input texture, a new texture image can be synthesized of the input texture under a novel lighting condition. In this manner, one can think of generating a novel lighting condition as an image processing algorithm comprised of a (possibly) different convolution kernel for each pixel in the input image.

This led us to experiment with using textons to learn image processing by example (described in Chapter 3). This approach was unsuccessful however because of the curse of dimensionality [2]. The texton filter bank is very large and many images are needed to learn the proper basis for a class of images. Hertzmann *et al.* [26] were able to succeed by not using filter banks of responses

at all, rather just the pixel neighborhoods alone. However, even using pixel neighborhoods alone still suffers from the curse of dimensionality, particularly for a single image example pair as they use. Their algorithm’s major insight is to combine the texture synthesis work of Wei and Levoy [65] with Ashikhmin [1]. The image analogies algorithm computes distances with both of these metrics and switches between the candidate sampling locations based on a user parameter. Just as Ashikhmin’s idea of copying from the training image yielded better results for natural images, using his metric for example based image processing helped Hertzmann *et al.* avoid the curse of dimensionality since they copy as much as possible from the example training images.

Recently, probabilistic methods have gained favor for these problems due to their robustness when faced with very limited training data. In Freeman *et al.* [14], belief propagation [72] is used to construct high frequency image details in low-resolution images using exemplar pairs of low-resolution/high-resolution image patches. This work can be considered to be the first to use maximum likelihood inference to perform example based image processing. Recently, this work has been extended [61] to use natural image statistics along with a technique to generate candidates from the training data for improved results. Belief propagation has also recently been used successfully on other computer vision problems such as stereo [60] and surface reconstruction [45]. Jojic and Frey [30] use probabilistic inference and learning to find the parameters of a model explaining the contents of a sequence and then decompose it into sprites and layers for video synthesis.

CHAPTER 3

NON-PARAMETRIC EXAMPLE BASED PROCESSING

Image processing operations can be very difficult to understand or express algorithmically. For example, how does one make an algorithm to ‘paint like Van Gogh’? Replicating an artist’s style is extremely hard because there are distortions and exaggerations of image features coupled with a rendering style. However, if there are no distortions of image features, the image processing can be treated as a set of local pixel neighborhood operations that must be spatially coherent.

As a result, our first experiments in this area were with learning Adobe Photoshop image filters. These filters were an attractive challenge because they produced image effects that were not possible with simple kernel convolutions, yet ultimately consisted of an unknown series of image processing operations including convolutions with unknown kernels to produce the output image. The texton work of Leung and Malik [36], described in Chapter 2.3.2, is related because of their novel lighting texture experiment. The experiment implied to us that if novel lighting can be predicted by treating the problem as a series of different local convolutions, that the same framework might be useful for learning general (i.e. convolution based) image processing operations.

We implemented the texton work of Leung and Malik with several extensions. The first was to construct Laplacian pyramids [6] for all of the images and to compute the textons at each level separately. This was done to capture large frequency band image transformations. The filter bank we used is shown in Figure 1. One immediate challenge was in figuring out how to weight each filter in the filter bank; it was not obvious whether some filters’ responses should be considered more important than others’. Also, it was not obvious whether Laplacian or Gaussian pyramids should be used. We chose Laplacian pyramids because they reduced the pixel variability needed to match well compared to images alone or Gaussian pyramids. An additional simplification was to work only with grayscale images first since we wanted to perform learning without letting color

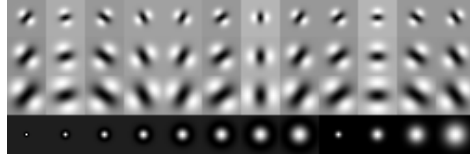


Figure 1: Filter bank used for the example based processing experiments in this section.

complicate things.

Since our algorithm was motivated by the texton work, we used many example pairs of input/output images to attempt to learn the processing that was observed. A large number of input/output pixel neighborhoods were needed to cluster meaningful textons. Laplacian pyramids were constructed for all example image pairs as well as for the input image whose output the user wanted to estimate. Each pixel neighborhood in the training data was convolved with a filter bank, and the responses were concatenated to form long vectors, as in the Leung and Malik work. The major difference is that our vectors were double the length of theirs since we would concatenate the example input and output training image filterbank responses together. These vectors were then clustered using the K-means algorithm [2] to learn the textons, i.e. the cluster centers of these filter responses. After the textons were learned, the vectors were split in half into the responses from the example training input and output images.

To synthesize the output image, a new Laplacian pyramid was constructed, with no values for any pixels. The entries were filled in scanline order starting from the coarsest level of the pyramid and progressing to the finest. For each pixel at each level in the pyramid, we computed the filter responses for the same pixel location in the input image's pyramid at the same level, and found the texton with the smallest $L2$ error. Once the texton was identified, the algorithm would multiply the textons with the pseudoinverse of the filterbank to compute a pixel intensity. This was done because the textons were means of responses and thus had no corresponding pixel values. The algorithm would stop when it reached the last pixel at the finest level, and the output image would be produced by collapsing the pyramid.

One large question in this problem is whether even the largest set of training data will be sufficient to represent a processing operation. To determine whether the problem was tractable, we first made the processing operation be the identity operation; that is, the input and output are equivalent. Images with distinctly different features should be sufficient for learning the identity

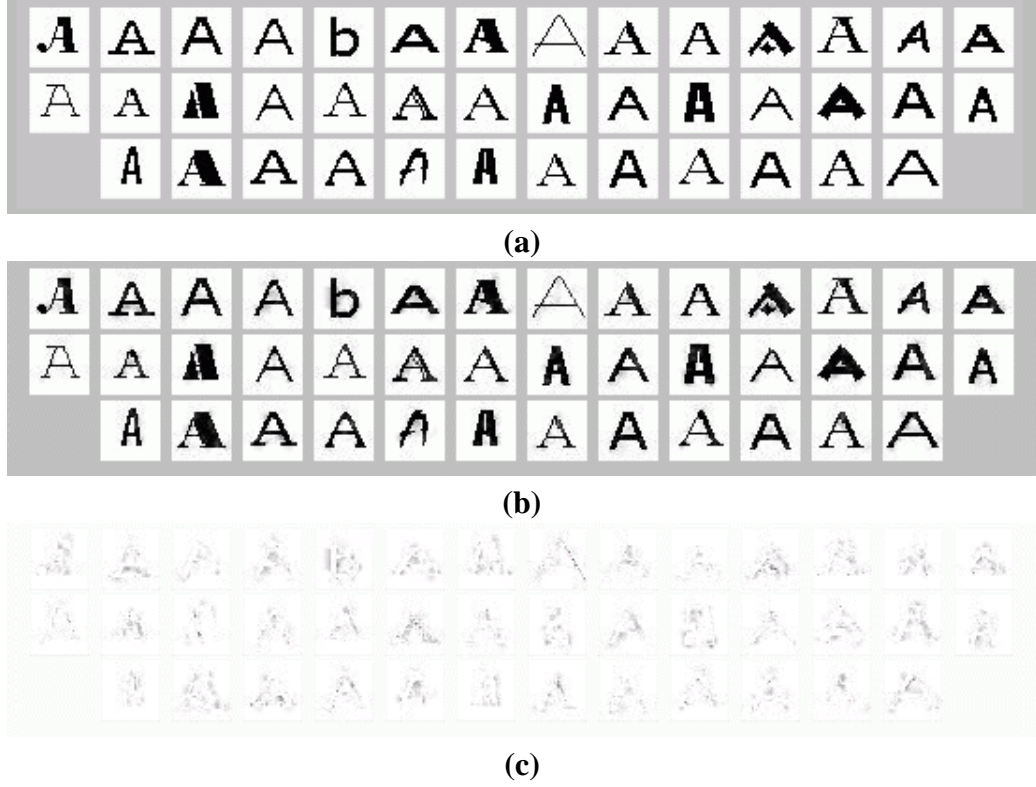


Figure 2: Font reconstruction experiment: (a) Original images, (b) reconstructed images of (a), where the image to be reconstructed is left out of the training data set, (c) reconstruction error.

operation provided that there are enough example input/output neighborhood pairs, and that these pairs are small enough.

We created a collection of small font images and attempted to reconstruct each font using all of the others as training. Each training pair in the training set consisted of a font image as both the training input and output. Figure 2 (a) shows the images used for this experiment. Figure 2 (b) shows the results of the algorithm. The results in Figure 2 (c) had very small errors, only in the inner parts of the letters which were hard to reconstruct. Gray values resulted in the outputs from multiplying the textons with the pseudoinverse of the filterbank to estimate the original pixel values in the Laplacian pyramid. This simple matching criterion worked well even for a letter with more round edges than the others (the letter ‘b’ in the dataset). The round edges were approximated by copying portions of hard edges from the other letters. This experiment implied that some classes of images could indeed be reconstructed accurately from pixel neighborhoods of other images and that the example based image processing problem could be tractable with a large enough dataset.

We then used this algorithm to attempt to learn several Adobe Photoshop image filters. Figure

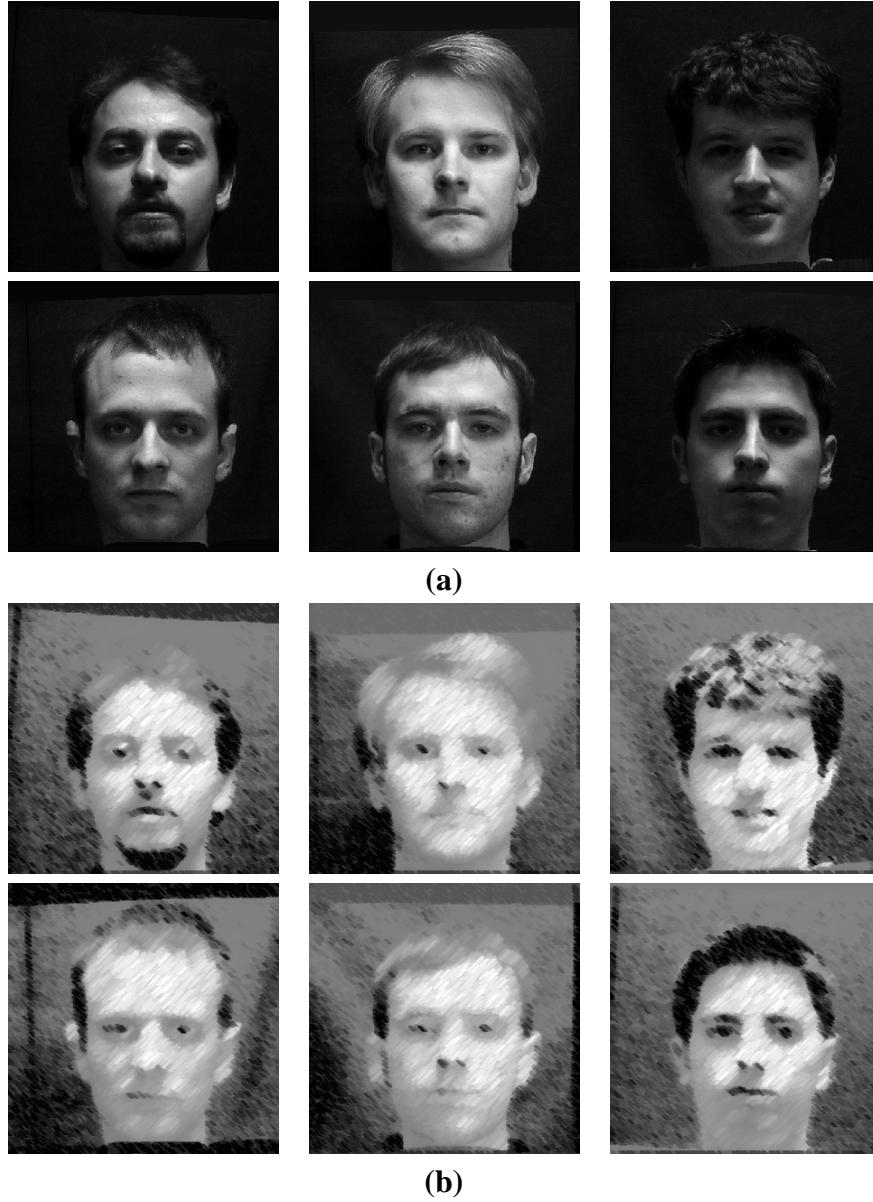


Figure 3: Six out of 20 faces that were used for the charcoal portrait experiment. (a) Original images, warped so that facial features all line up with one another, (b) The result of using Adobe Photoshop's charcoal filter on the images.

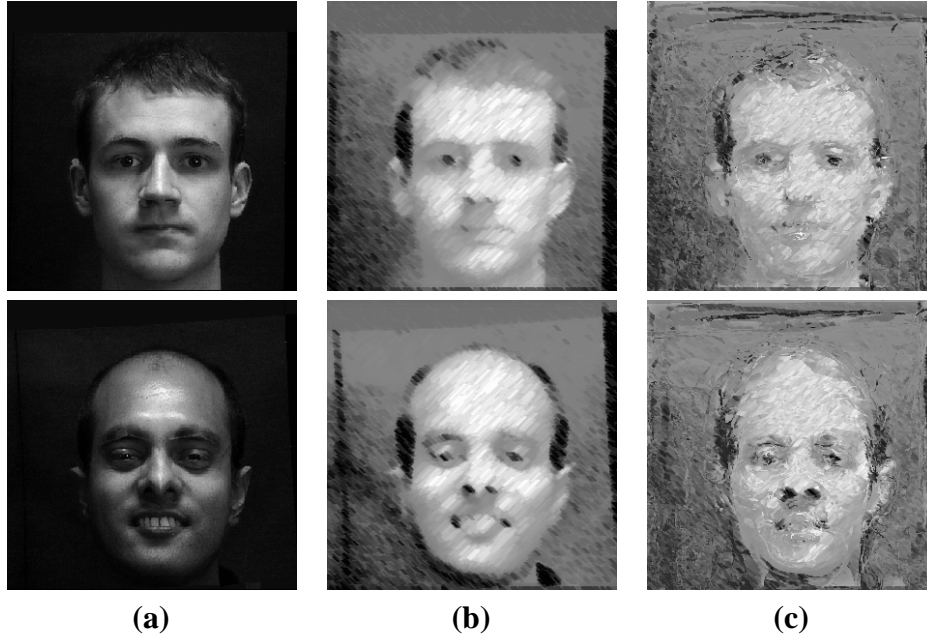


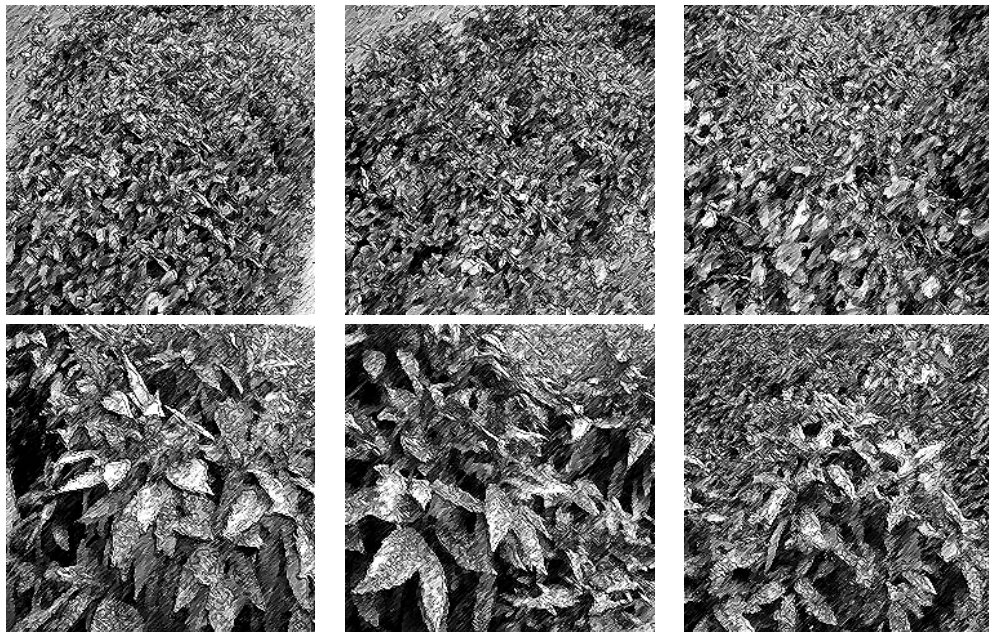
Figure 4: Charcoal portrait experiment output: (a) Original images, (b) Adobe Photoshop charcoal filter of (a), (c) Filter estimate using 19 training images.

3 shows the training data used in the first experiment, to learn the ‘charcoal’ filter. Figure 4 shows the results of using this algorithm to estimate the processing output for two different faces. The results are visibly similar to the output, however, all of the visible brushstrokes are incoherent. Figure 5 shows the training data used in the second experiment, to learn the ‘ink outlines’ filter. Figure 6 shows the algorithm’s output estimating this filter. The results in this figure are worse than those seen for the charcoal experiment because the pen and ink filter is harder to reconstruct since the filter bank only contained a few steerable filters.

This algorithm proved that example based processing is possible, but that the curse of dimensionality is a major hurdle to overcome since the training data is extremely sparse. Even with large numbers of training pairs with simple filters and large filter banks, simple nearest neighbor matching would always yield incoherent results. Image analogies [26], while an extremely similar algorithm, succeeded where our algorithm did not by treating the problem as a constrained texture synthesis problem instead of a filter reconstruction problem. In their work, only single example pairs are used, instead of collections of example pairs as in this work. They also did not use filter banks, rather, just the pixel intensities themselves using Gaussian pyramids. This allowed them to keep the dimensionality of their feature vectors small, allowing them to perform more matches



(a)



(b)

Figure 5: Six out of 72 shrub images that were used for the pen and ink experiment. (a) Original images, (b) The result of using Adobe Photoshop's ink outlines filter on the images.

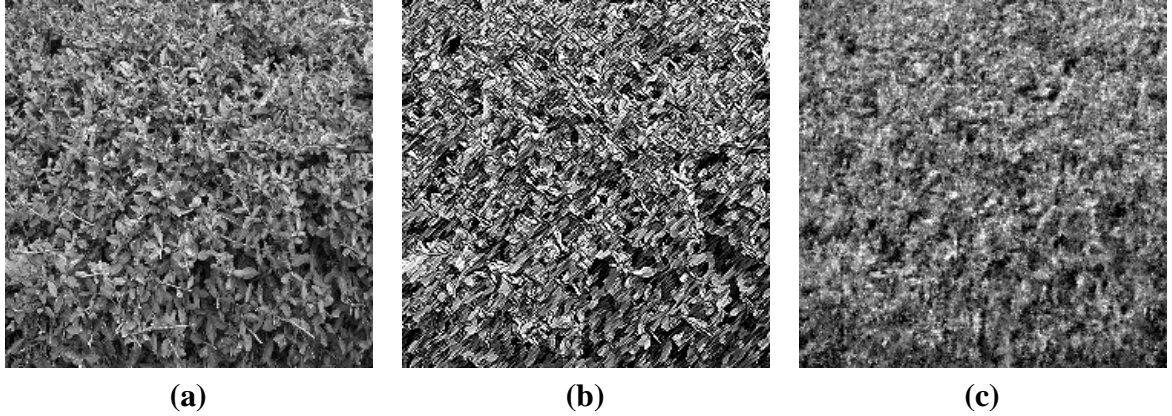


Figure 6: Pen and ink experiment output: (a) Original images, (b) Adobe Photoshop ink outlines filter of (a), (c) Filter estimate using 9 training images.

faster. The final and most critical difference was in their use of Ashikhmin’s metric in addition to the nearest neighbor metric when choosing training feature vectors that matched the input image feature vectors well. This allowed them to retain the coherency present in the training images since they tend to copy brushstrokes, something which our algorithm did not do.

3.1 Generalized example based processing

Image analogies treats the problem of example based image processing as a hybrid of constrained 2D texture synthesis and filter reconstruction instead of a filter reconstruction problem alone. However, the matching algorithm presented in that work was only applied to pixel neighborhood based feature vectors. By generalizing the matching algorithm, we extend it to work with arbitrary feature vectors, where the features are chosen appropriate to the data domain. This is one of the major insights of this thesis; example based processing can be decomposed into two problems. The first is selection of descriptive features to include in the feature vectors. The second is the matching algorithm used to match the feature vectors. In this section, we show how to use different feature vectors to perform various different types of example based processing besides image processing. We use the matching algorithm from image analogies, but different feature vectors for each problem domain. We also discuss why this approach is valid.

The problem of example based processing can be thought of as how to use a sampling of a function to determine plausible function values in areas where there is no training data. The image

analogies matching algorithm approximates a vector valued function by coherently copying from the areas given by the (normalized) training data to construct function values where there may not be any for the input image. This can be generalized even further if we think of the training data as being comprised of points and their corresponding function values. In addition, these training examples can be a very sparse sampling of the function, particularly if it is of high dimensionality as are the problems we present.

The matching algorithm makes some assumptions about the function to be learned from the data, which we refer to as F . We assume: (1) the function is piecewise constant, (2) an operator, $N(x) = C$, is defined on the function such that for any $x \in D$, where D is the domain of F , an ordered set C is returned, where $C = (x_a, x_b, \dots)$.

The first assumption is that the function representing the processing that will be approximated is piecewise constant. This assumption is very important as it is the reason the image analogies matching algorithm produces such satisfying results with scarce training data. For a given 3x3 pixel neighborhood, the matching algorithm implicitly assumes that slight variations in any of the pixels should match to the same 3x3 pixel neighborhood in the processed image. Indeed, the neighborhoods pulled from the images in image analogies are low-pass filtered before constructing feature vectors for the neighborhood to increase the number of constant pieces in the domain.

The second assumption is merely that some neighborhood operator exists for points in F 's domain. The ordering of the points returned by the operator is defined by the position of the members of C relative to x ; that is, x 's neighborhood.

The dimensionality of F 's domain and range will depend on the problem. The domain consists of points in \mathbb{R}^n , and the range consists of points in \mathbb{R}^m , with no restrictions on n or m . Our training data consists of pairs of points, $t_i = (x_i^t, y_i^t)$, where $F(x_i^t) = y_i^t$, and $x^t \in \mathbb{R}^n$, $y^t \in \mathbb{R}^m$. We define T to represent all given training data, $T = (t_0, t_1, \dots, t_p)$, where p is the number of training pairs. Furthermore, we define $X^t = (x_0^t, x_1^t, \dots, x_p^t)$ and $Y^t = (y_0^t, y_1^t, \dots, y_p^t)$.

The algorithm is given T , X^t , Y^t , and a set of points P where we would like to approximate F . We define P as $P = (p_0, p_1, \dots)$ where each p_i is a pair $p_i = (x_i, y_i)$, with $F(x_i) = y_i$, and $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^m$. Initially, each p_i is initialized to be $p_i = (x_i, 0)$, since we do not know the function value for any x_i . As the algorithm progresses, each x_i is assigned a point from Y^t , thus approximating the function.

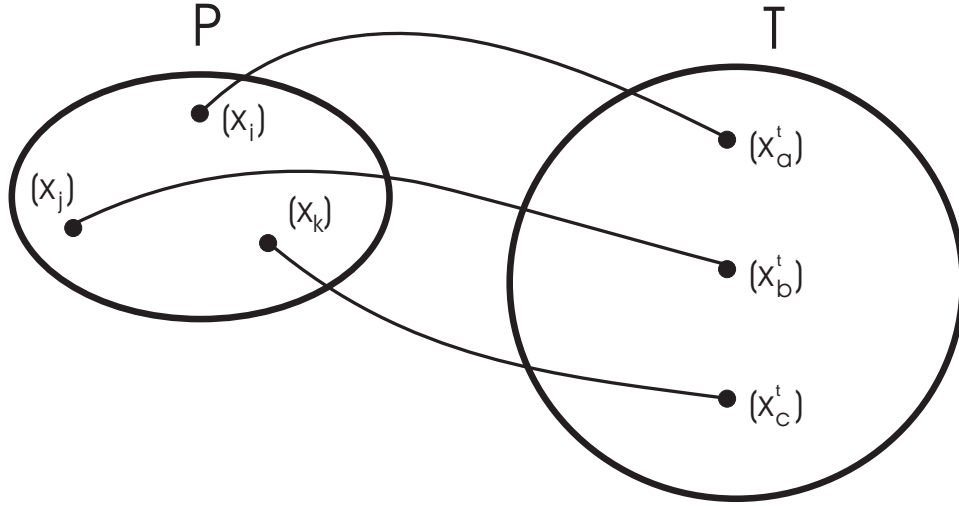


Figure 7: Initial matching: nearest neighbor. The nearest neighbors of points in P are found in T .

The major problem is how to perform these assignments. Obviously, if there is an $x_i = x_j^t$, then $F(x_i) = F(x_j^t) = y_j^t$. However, this is rarely the case, especially when m and n are very large since the x_i 's and y_i 's consist of entire pixel neighborhoods. For instance, in the case of the problems we are interested in they are ≈ 100 . The algorithm must also provide assignments for each x_i such that they are consistent when considering x_i 's neighborhood, $N(x_i)$. That is, the function must be approximated at each point with respect to all of the neighbors at that point, for all of the points in P .

Our assumptions assist us in performing assignments that are similar to the function behavior observed in the training data in regions where there is no training data. Since the neighborhood operator $N()$ yields an enumerated set, we can use it to identify points in the training data with similar neighborhoods to points in P . If the neighborhoods match well, then we can use function values from that area of the sampling to approximate the function where we do not have data. This approach could yield discontinuities if used to approximate continuous functions, but since we only address piecewise constant functions, this is not a concern.

3.2 Approximation algorithm

Our algorithm is a multi-pass algorithm. On the first pass, we approximate the function at all of the points in P very coarsely. On subsequent passes, we refine our approximation so that it matches

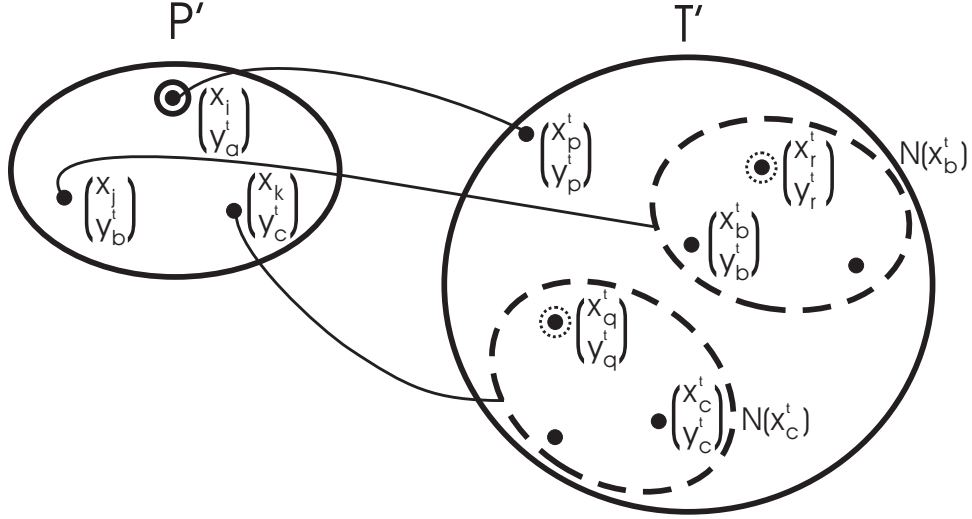


Figure 8: Subsequent matching: coherence matching. The nearest neighbor of the point in P' is added to the candidate set along with the coherent matches. The coherent matches are formed by looking at where each of the P' neighbors were sampled from in T' and then adding the corresponding points in T' with the same neighborhood offset to the candidate set.

the variations observed in the training data T more closely. This algorithm is a generalization of the image analogies matching algorithm using the operators and notation defined above. Generalizing the matching algorithm allows us to use it to match arbitrary feature vectors to approximate other piecewise constant processing functions besides image processing.

On the first pass, we do the following: $\forall p_i \in P, p_i = (x_i, y_i^t)$, such that

$$k = \arg \min_{x_k^t \in X} \text{error}(x_i, x_k^t) \quad (1)$$

$$\text{error}(a, b) = \|a - b\|^2. \quad (2)$$

This is a very coarse approximation because each x_i 's approximated function value is determined independently of its neighbors and vice versa. Figure 7 shows this step. Subsequent passes incorporate neighborhood information to refine the approximated function values.

On subsequent passes, we treat the point pairs in P and T as being concatenated vectors in \mathbb{R}^{n+m} . We define $p'_i = \text{concat}(x_i, y_i)$ and $t'_i = \text{concat}(x_i^t, y_i^t)$, where $c = \text{concat}(a, b)$ is a function that concatenates vectors a and b to form a new vector c such that $\dim(c) = \dim(a) +$

$\dim(b)$. Furthermore, we define P' to be the set of all concatenated p'_i 's and T' to be the set of all concatenated t'_i 's.

Concatenated vectors are used for the subsequent passes so that the matching will be influenced by both the points and their function values. Using concatenated pairs encourages matches that come from contiguous training point neighborhoods, thus matching the training data more closely. Figure 8 shows how the matching is performed in subsequent passes.

Subsequent passes are thus: $\forall p_i \in P, p_i = (x_i, y_k^t)$, such that

$$k = \begin{cases} k_{coh} & \text{if } \alpha \text{error}(p'_i, t'_{k_{coh}}) < \text{error}(p'_i, t'_{k_{nn}}), \\ k_{nn} & \text{otherwise} \end{cases} \quad (3)$$

where α is a user parameter. k_{nn} is the index of the nearest neighbor match of the training vector in T' with p'_i . k_{coh} is the index of the training vector in T' that matches p'_i the best from the subset of training vectors with the same neighborhood relationship as the matches of the neighbors of x_i . That is, for each neighbor of x_i , we look at its matching y^t from the previous iteration and its corresponding x^t . Then, for each x^t corresponding to each y^t match, we find the training point with the same neighbor relationship with it as x_k has with x_i . For each of these related points, we compute the error of its concatenated counterpart with the p_i we are approximating.

Equation 3 is a weighting function that determines whether to emphasize or de-emphasize the t' vector that matches the p' point without considering neighborhoods. Increasing the value of α penalizes the $t'_{k_{nn}}$ vector in favor of the $t'_{k_{coh}}$ vector. This is to encourage the algorithm to match entire neighborhoods of points rather than individual points in the space.

$$k_{nn} = \arg \min_{t'_{k_{nn}} \in T'} \text{error}(p'_i, t'_{k_{nn}}) \quad (4)$$

$$k_{coh} = \arg \min_{x_k \in N(x_i)} \text{error}(p'_i, T'_{related}(\text{Match}(x_k), N_{idx}(x_k, x_i))) \quad (5)$$

where $\text{Match}(x_k) =$ the best y^t that was found for x_k in the previous iteration, and $N_{idx}(a, b) =$ index of b in $N(a)$.

$$T'_{related}(y_b^t, \text{index}) = \text{concat}(X_{related}^t(y_b^t, \text{index}), Y_{val}(X_{related}^t(y_b^t, \text{index}))) \quad (6)$$

$$X_{related}^t(y_b^t, \text{index}) = N_{elem}(X_{val}(y_b^t), \text{index}) \quad (7)$$

where $N_{elem}(a, i) = \text{element } i \text{ of } N(a)$, $X_{val}(y_i^t) = x_i^t$ and $Y_{val}(x_i^t) = y_i^t$. $X_{related}^t()$ and $T'_{related}()$ return the point from X^t and T' respectively which is the index^{th} element of the neighborhood of $x_b^t = X_{val}(y_b^t)$. These two operations allow us to compare neighborhoods of points with those in the training data, allowing us to match entire neighborhoods coherently.

k_{coh} encourages coherent matching of neighborhoods between iterations. It is ultimately what allows for good matches in areas of the function space where no training data exists. Using k_{coh} , the most similar neighborhood from the training data to x_i 's neighbors' matches is used and matches from these areas of the function space are selected. It is important to note that the k_{coh} training point is not always selected. The best results come from choosing an α such that k_{coh} is selected most of the time but k_{nn} is selected as well, just not as often. This is to keep the algorithm from getting stuck copying function values from the same region of the training space. This is undesirable because in image and video synthesis applications excess copying from single regions results in obvious seams when the algorithm switches copying between different regions of the training space unless special care is taken when sampling or the seams are hidden. α should also be chosen such that k_{nn} is not selected always, as the algorithm would then reduce to simple nearest neighbor matching. For the problems in Section 3.3, nearest neighbor matching results in videos that are not temporally coherent and images with banding artifacts. This is because the training data for these problems is too sparse for simple nearest neighbor matching.

3.3 Results

So far, we have shown a vector matching algorithm that can approximate piecewise constant functions. However, the contents of the x and y vectors have not been described. x and y are, in fact, feature vectors with features dependent on the problem. The choice of what features to encode

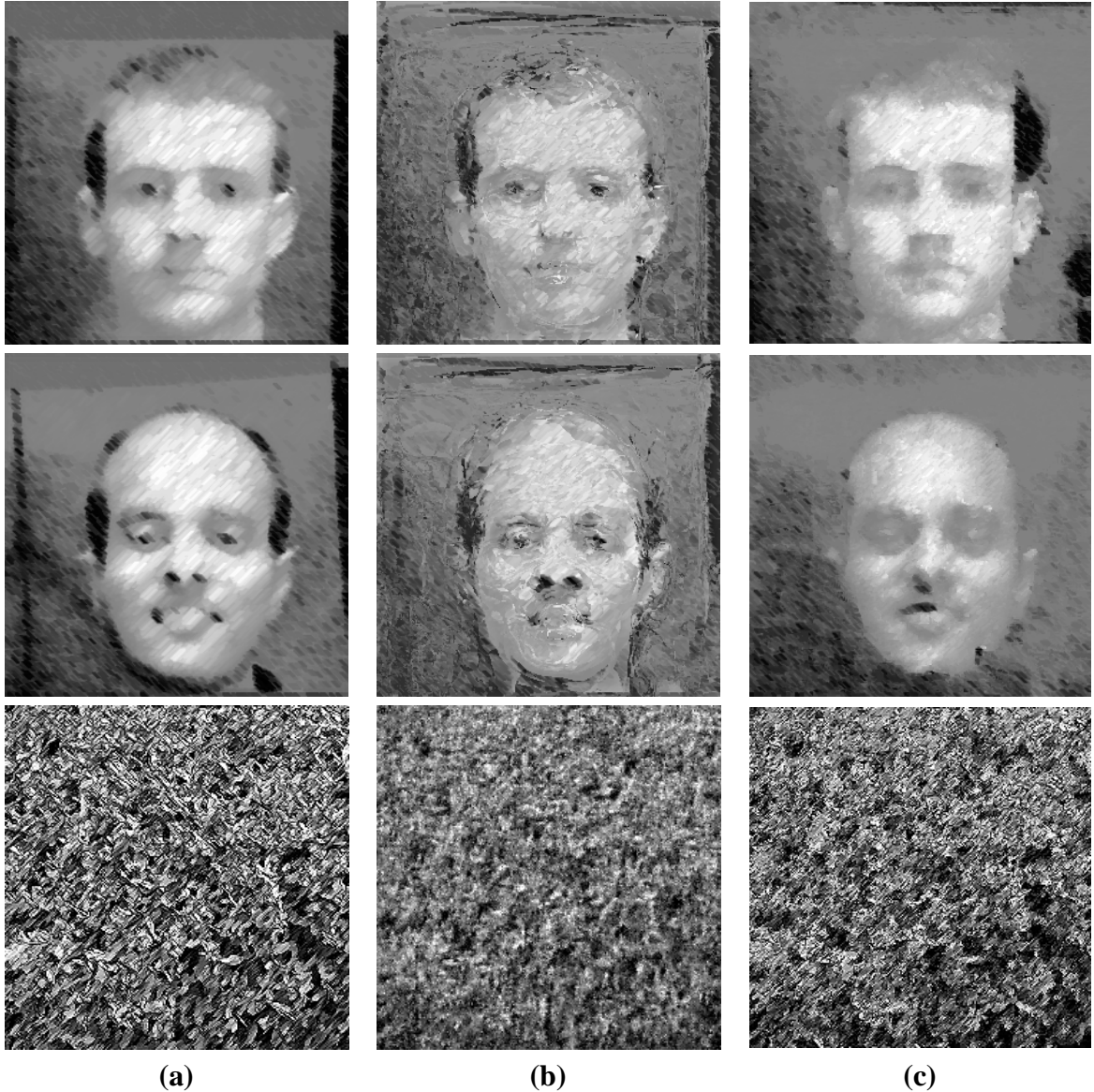


Figure 9: Comparison with image analogies: (a) Ground truth, (b) Results using prior algorithm, (c) image analogies matching results.

in the feature vector is almost as important as the choice to use the vector matching algorithm described above.

In this section, we show results that we were able to produce using this framework. We show various types of video and 3D model applications of our approach that were previously only possible to achieve using different algorithms. Using our framework, it is possible to approximate different types of processing using the same algorithm.

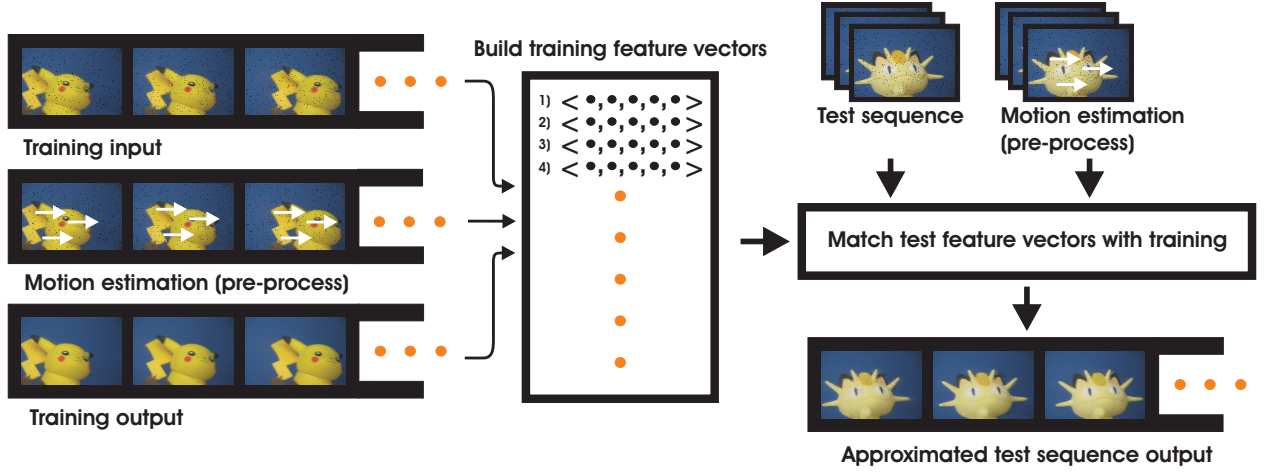


Figure 10: Overview of video processing pipeline.

3.3.1 Image processing

We first revisit some earlier results to give an intuition of the differences between the image analogies based generalized matching algorithm presented in Section 3.1 and the texton-based algorithm from the beginning of this chapter. Figure 9 (a) shows ground truth and Figure 9 (b) the results using the earlier algorithm (from Figures 4 and 5). Figure 9 (c) shows the results from using image analogies based matching. The results using this matching technique do not resemble the ground truth exactly since the processing in these images is non-localized and hard to learn, but they are more coherent than the earlier results. In particular, the pen and ink filter results in the last row are much closer to ground truth since this filter is more localized than the charcoal filter shown in the other two rows.

3.3.2 Video processing

We used our framework to approximate several different types of video processing. We assumed that the processing would not warp the frames or change pixel neighborhood relationships to meet the conditions discussed in Section 3.1. For training, we give the algorithm a pair of videos. This pair consists of a single sequence before and after processing. We are then able to produce a processed version of a new video clip with no prior knowledge of what the processing in the training example was. Figure 10 shows an overview of this problem.

The feature vectors for this problem consist of pixel neighborhoods from frames nearby in time

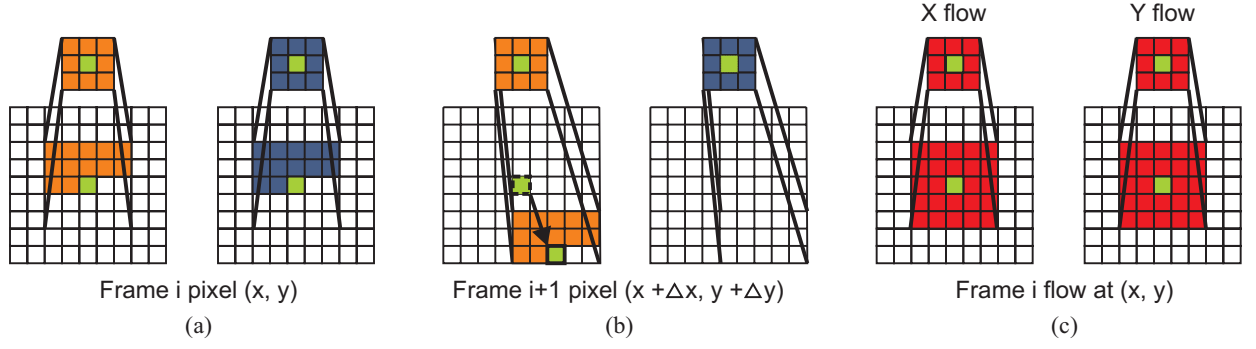


Figure 11: Components of our feature vector. Orange neighborhoods are from the unprocessed sequences, dark blue neighborhoods are from the processed sequences, and red neighborhoods are the X and Y components of flow vectors.

as well as different frequency bands of the neighborhoods, to deal with both large and small scale changes in the video. In addition, we estimate the motion of each pixel in the sequence and add this information to the feature vector as well. In this manner, we are able to match blocks of video to blocks from the training videos that have matching appearance and dynamics. To construct feature vectors, we take both the input and output sequences and convert them to the YIQ color space. We process the color channels independently, though for the results in this section, we only processed the Y channel, since that saved processing time. The training input sequence's histogram is then linearly matched with the test sequence's to ensure that the feature vectors will be in the same area of the feature space when matching.

Once color conversion has been done, we compute optical flow [3] of the entire sequence and construct Gaussian pyramids of the flow [6]. Next, we construct Gaussian pyramids of each frame in the sequence. Then, for a pixel at (x, y) in level l of frame i , we construct a feature vector comprised of:

1. the 3x3 neighborhood of pixel $(x/2, y/2)$ in level $l + 1$, frame i of the input pyramid
2. half of the 5x5 neighborhood of pixel (x, y) in level l , frame i of the input pyramid
3. the 3x3 neighborhood of pixel $(x/2, y/2)$ in level $l + 1$, frame i of the output pyramid
4. half of the 5x5 neighborhood of pixel (x, y) in level l , frame i of the output pyramid
5. the 3x3 neighborhood of pixel $(x'/2, y'/2)$ in level $l + 1$, frame $i + 1$ of the input pyramid

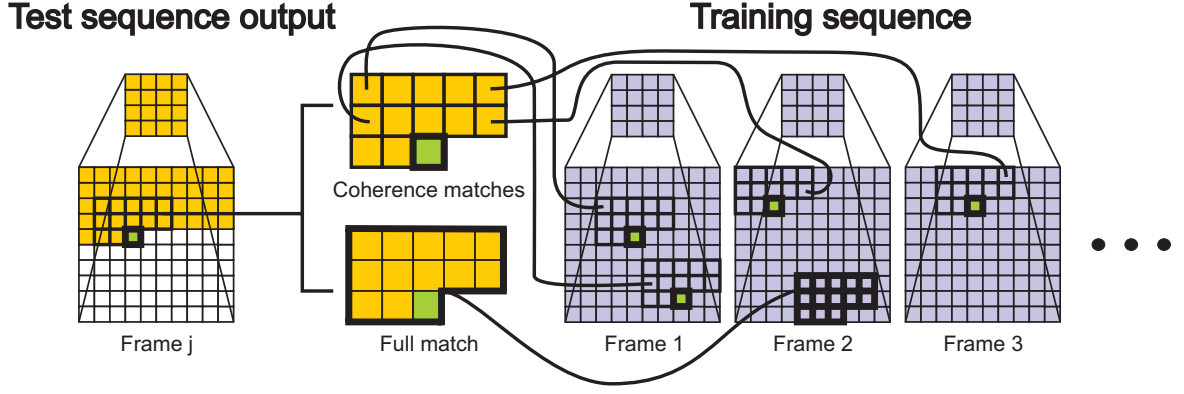


Figure 12: A pixel in the test sequence output is assigned an intensity from the training sequence. A tradeoff is performed between the training vector with the smallest L2 error with the pixel’s feature vector and the training vectors that would match the pixel from copying through the entire video sequence.

6. half of the 5×5 neighborhood of pixel (x', y') in level l , frame $i + 1$ of the input pyramid
7. the 3×3 neighborhood of pixel $(x'/2, y'/2)$ in level $l + 1$, frame $i + 1$ of the output pyramid
8. the 3×3 neighborhood of pixel $(x/2, y/2)$ in level $l + 1$, frame i of the x flow pyramid
9. the 5×5 neighborhood of pixel (x, y) in level l , frame i of the x flow pyramid
10. the 3×3 neighborhood of pixel $(x/2, y/2)$ in level $l + 1$, frame i of the y flow pyramid
11. the 5×5 neighborhood of pixel (x, y) in level l , frame i of the y flow pyramid

where $x' = x + \Delta x$ and $y' = y + \Delta y$, and $(\Delta x, \Delta y)$ are the x, y components of the optical flow for (x, y) . Figure 11 shows these components visually.

This feature vector is similar to the one used in [26], but differs in that it also encodes the temporal properties of the pixel. Components 5-7 of the vector act as a “look ahead” to the next frame to encode how the pixel’s neighborhood changes in appearance. Using the optical flow for these components effectively stabilizes the pixel’s neighborhood between frames n and $n + 1$. Components 8-11 are the flow itself, which aid in learning effects that are more motion rather than texture dependent, such as motion blur.

Gaussian pyramids are used to improve the quality of matches. At the coarser levels of the pyramid, large features in the video affect matching more than small features. Matching at all

levels, from coarse to fine, ensures that small and large scale features affect the matching equally. We use 3x3 neighborhoods for features from the previous pyramid level and 5x5 neighborhoods for features from the current pyramid level because we found that those were the smallest sizes that would still yield good results. This agrees with the neighborhood sizes for the feature vectors in [26] for images. Figure 12 shows the matching process in more detail.

We tried the algorithm on several data sets of different video processing to see how it would perform. The video processing ranged from being strictly spatial (e.g. color correction) to being both spatial and temporal (e.g. motion blur). Our algorithm was able to produce videos that resembled plausible results of the observed processing.

Color correction We took a 22 frame sequence and changed its colors in Adobe Premiere. The original and color modified sequence were then provided as training to our algorithm before giving it a new sequence to process. Figure 13 shows the results. Color correction is a toy problem since it is purely a pixel based operation. The color tone is slightly off, but it is faithful to the tone of the training processed footage, which is what is expected.

Noise removal We added random salt and pepper noise to a 15 frame sequence of a toy being slid left and right. We used the noisy sequence as the training input and the original sequence as the training output. Then, as a test sequence, we used a different sequence that had noise added to it as well. Our algorithm removed the salt and pepper noise and created a temporally coherent video (Figure 14).

Motion blur from CG We took a 3D animation of a person kicking and applied 3D motion blur in Maya (a 3D modeling and animation program) to create a motion blurred version of the animation. We then took the same animation data and applied it to a model with different geometry and texture and created a video sequence that we used as our test sequence (Figure 15). Our algorithm produces some motion blur (evident on the legs), but does not blur as much as Maya does. This is because our 3x3 and 5x5 neighborhoods are too small to capture such large amounts of blur. However, our algorithm was still able to learn to blur despite the fact that the effect relied on *a priori* 3D knowledge and was too large for its neighborhoods.



(a)



(b)



(d)



(e)

Figure 13: Color correction: (a) & (b) training input/output, (c) test sequence, (d) approximated output, (e) ground truth.

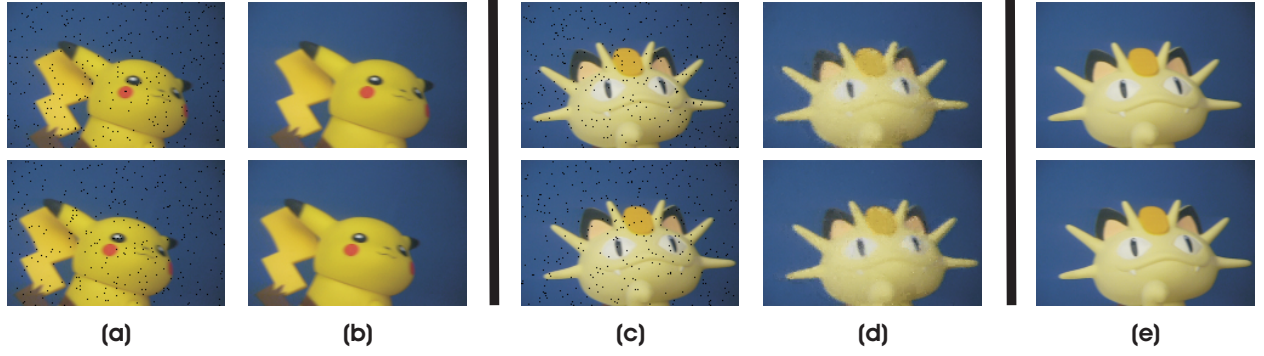


Figure 14: Noise removal: (a) & (b) training input/output, (c) test sequence, (d) approximated output, (e) ground truth.

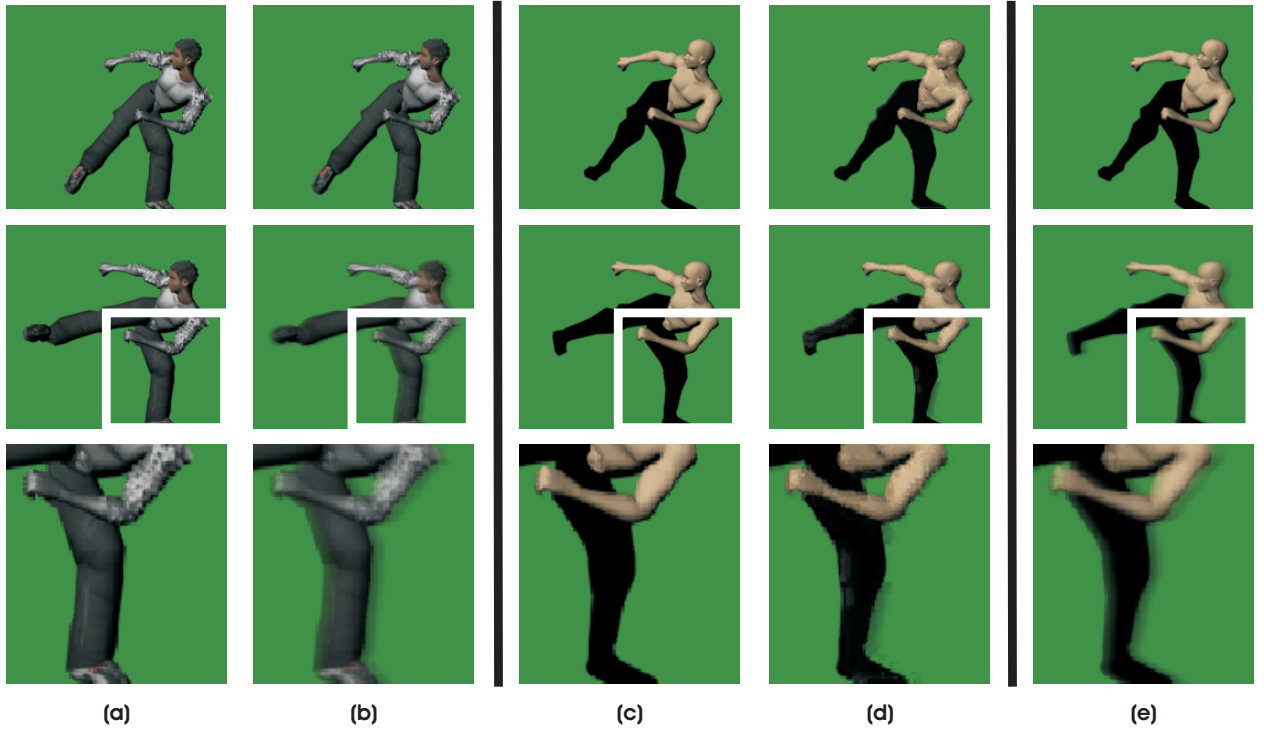


Figure 15: Motion blur: (a) & (b) training input/output, (c) test sequence, (d) approximated output, (e) ground truth.

Non-photorealistic rendering of video Rendering video non-photorealistically is an interesting problem because the mapping between a painting’s *style* to video is non-obvious. Prior work has been based on “pushing” strokes around based on optical flow calculated for the sequence. Temporal coherence is achieved by using optical flow and heuristics to constrain brush strokes [39, 27] or by “relaxing” the brush strokes through a minimization [25]. The main problem with these techniques is that the style that can be used is dependent on the implementation. We are

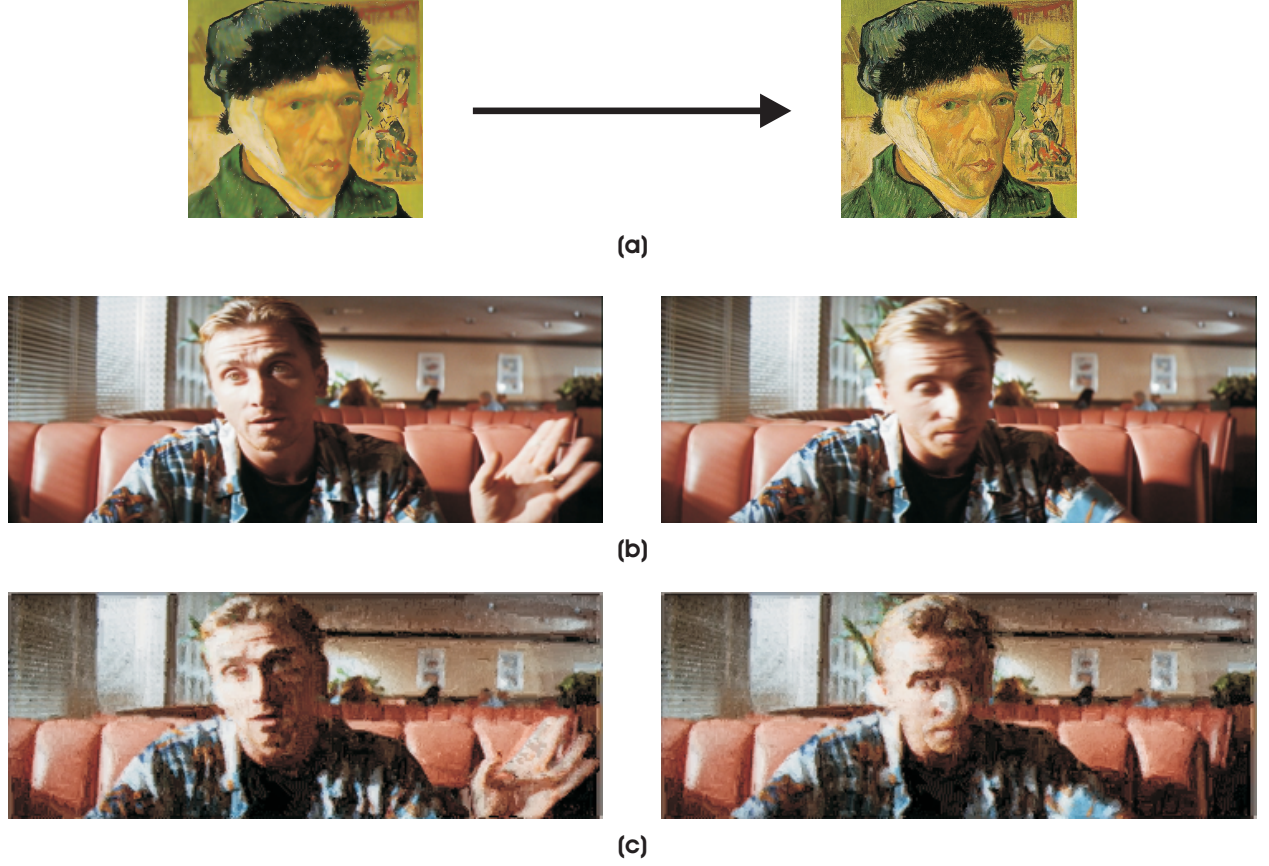


Figure 16: Painting style: (a) training input/output (video), (b) test sequence, (c) approximated output.

able to render video based directly on existing art. Figure 16 shows a Van Gogh painting’s brush strokes used on a video sequence, resulting in temporally coherent brush strokes that come from the painting (treated as a video sequence).

However, the results in Figure 16 were not satisfactory because high-resolution videos require very large training sets. As a result, we could only create non-photorealistically rendered videos of a very low resolution. In addition, the training data consisted of a static video comprised of a single image of a painting in the target style. Consequently, the algorithm was not able to make use of the motion information present in the video to produce better results and high-frequency brush strokes were not visible. The output video contained high-frequency brush strokes from the painting, but the strokes were very small since the training data was extremely limited.

For this application, we approximate the function $F(\text{input video}) = \text{painterly version of video}$. We use the same feature vectors for video as before, but perform additional analysis. First, we

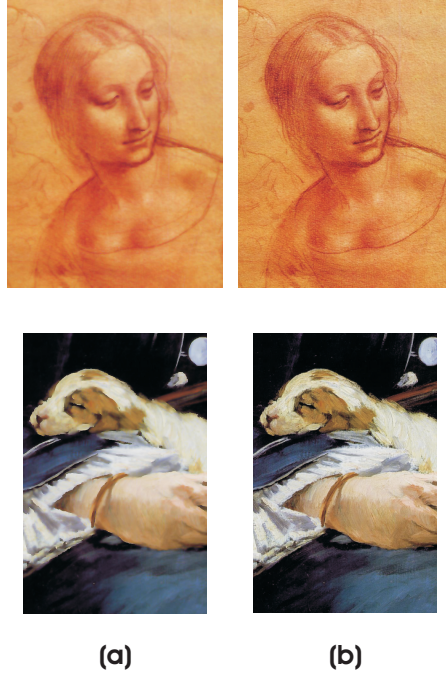


Figure 17: Training data used in Figure 18. Top: *Head of a Woman*, 1510-1511 by Leonardo Da Vinci, Bottom: *Le Chemin de Fer*, 1872-1873 by Edouard Manet. We use (a) anisotropically blurred versions of the paintings as training input, and (b) the paintings as training output. The blurred versions of the paintings are treated as the ‘real’ versions of the paintings. In addition, we rotate the paintings so that we can use motion as a feature.

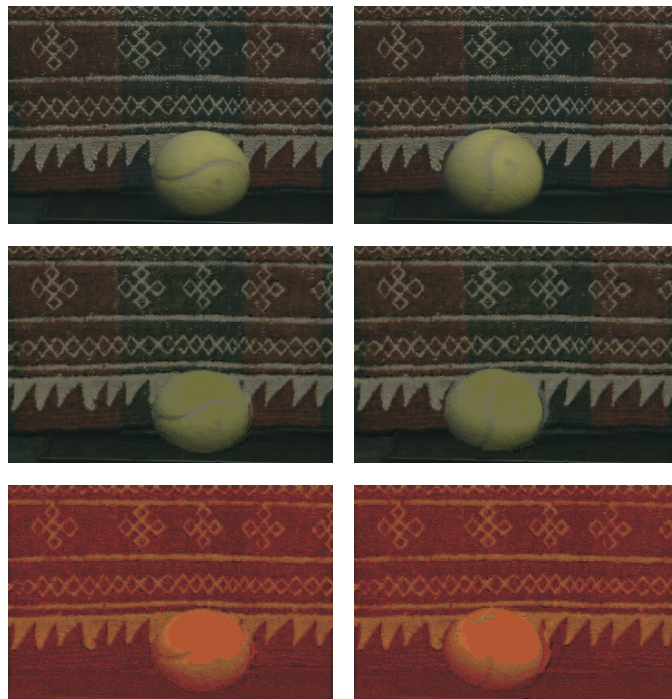
cluster the training points using Lloyd’s algorithm [16]. This allows us to use example paintings of an even higher resolution for improved results. We also rotate the painting about its center, both clockwise and counter-clockwise. This provides more training data about what the painting looks like in motion, thus allowing better matches when objects are moving in the video. Since a video version of the painting’s contents is not available, this is only a rough approximation, and serves just to assist the algorithm in creating a painterly version of the video. Figure 18 shows some examples of applying the technique using the training data in Figure 17. Simple nearest neighbor matching results in videos that are temporally incoherent because the training data is too sparse.

3.3.3 3D surface material property transfer

Many techniques exist for creating photorealistic renderings of 3D models. However, such methods are usually either computationally intensive, hard to implement, or require significant manual parameter tuning. We apply our framework to create single-view photorealistic renderings of 3D



(a)



(b)

Figure 18: Results of non-photorealistic video rendering. Top row: original sequences, middle row: Manet painting style, bottom row: Da Vinci drawing style.

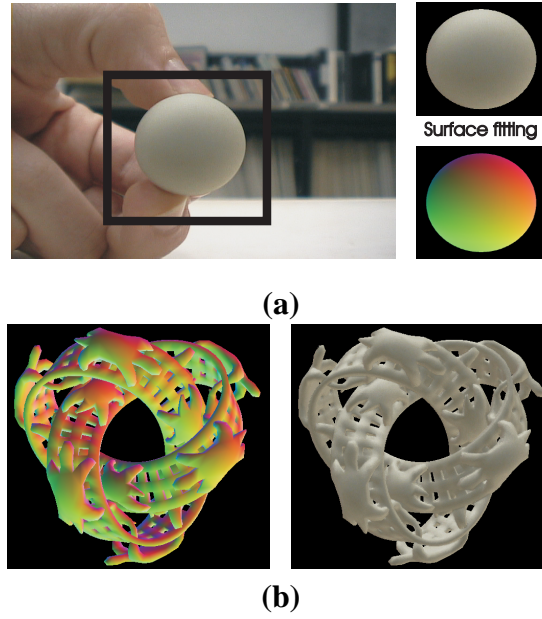


Figure 19: (a) Target material is cropped from photograph, fitted with 3D ellipsoid, and rendered with encoded normals, (b) Input 3D surface, left: encoded normals, right: result.

models devoid of self-shadowing or self-occlusions based solely on photographs of target materials.

Our algorithm uses a digital photograph of a sphere or ellipsoid of the source material for the input 3D model. Our pipeline is shown in Figure 19. Since ellipsoids are such smoothly varying surfaces, we can fit 3D surfaces through the boundaries of the material photograph. The ellipsoids are fit by hand by warping a rendering of a sphere with encoded normals so that the width and height match those of the photograph using a commercial image processing program.

We then encode the normals of the fitted 3D ellipsoid and the 3D model as colors to produce two images; one of the fitted surface material, and one of the 3D model from the desired viewpoint. Finally, we coherently sample from the photograph of the ellipsoid material by using the two encoded normal images.

3.3.3.1 Motivation

The material image used by the system is in essence a single bidirectional texture function (BTF) image. A BTF is a mapping from the 4D space of viewing and lighting directions (each parameterized by a pair of tilt and azimuth angles) to the space of images:

$$\tau : L \times V \rightarrow I \quad (8)$$

where L and V are lighting and viewing directions respectively. Since the photograph provides an exemplar of a material’s appearance under a particular viewing and lighting direction, if it is mapped properly onto a model as texture, photorealistic renderings would be producible. However, such a mapping would cause a warping/loss of detail in areas of high surface curvature, and could cause high frequency albedo in the image to become incoherent. Sampling from a sphere or ellipsoid addresses these issues. The key intuition of our approach is that since these shapes provide coverage of the complete set of unit normals and are assumed to be the same scale as the 3D models, they can be used as a good estimate for how a complex surface would be illuminated if made of the same material.

Our approach works because the lighting, but not the albedo, varies with the surface normal. However, the albedo must be correct in synthesized renderings. For simple untextured material, such as colored plastic without specular highlights, matching normals alone results in plausible renderings and the lit sphere technique [58] would be sufficient. For a textured material, such as an orange, this approach will not yield good results. This is because matching normals alone will cause the high frequencies of the orange albedo to mismatch, producing a result that resembles noise. It is also worth noting the work of Hertzmann and Seitz [24] as it is the inverse version of this problem. They match reflectance from illuminated spheres to infer the 3D shape of a real object from an image whereas we use 3D shape to infer reflectance.

3.3.3.2 Image synthesis

We use normals to match in rendering since a sphere or ellipsoid provides coverage of the complete set of unit normals. In addition, if it is of a similar scale as the 3D model to be rendered, using a sphere or ellipsoid as a ‘stand-in’ for the reflectance of a more complex object is justified.

We call the 3D rendering of a model to be the rendering of its normals color-coded, i.e. (R,G,B) maps to (X,Y,Z) and the range (0, 255) maps to (-1,1). We perform this encoding so that we can perform matching over a 2D surface (the encoded normal image) instead of over the 3D surface itself to avoid having to parameterize the surface.

The normals are matched between the rendered 3D model and the rendered sphere or ellipsoid model. Simple nearest neighbor is not sufficient for photorealistic rendering since the material may contain high-frequency albedo that must be present in the rendering as well. We use small local surface neighborhoods and multi-resolution pyramids in matching. This ensures that we preserve both small and large scale surface and albedo details while matching.

We match the normals of the new 3D model with those of the material ellipsoid while respecting the local albedo that has already been matched at the current and previous pyramid levels while matching from coarse to fine levels. We construct feature vectors consisting of this information for each color channel (using the RGB color space).

Using coherent matching, we can handle textured materials without distorting the albedo when sampling. We also use tree structured vector quantization to speed-up the feature vector matching. In addition, background pixels are skipped in matching and are not part of the training set to speed up computation. The steps of our approach are:

1. Render a 3D model with its normals encoded as colors along with a visibility mask
2. Render a 3D sphere or ellipsoid with its normals also encoded as colors
3. Warp the rendered material surface image so that it lines up with the surface in the photograph of the source material
4. Match all normals in the 3D model rendering with the material's normals coherently in scanline order
5. Use the visibility mask to crop out any regions where the rendering does not match the silhouette of the 3D model

Figure 19 pictorially depicts the pipeline. We use the visibility mask produced by our renderer to crop out regions where image analogies does not follow the rendered object's silhouette. It is

possible for the matching algorithm not to follow the silhouette perfectly because of the coherence matching; applying the mask corrects the model's silhouette and holes if necessary. We can then composite the result onto a photograph of the source material's scene if desired using any image manipulation program (e.g. Adobe Photoshop, Gimp, etc.)

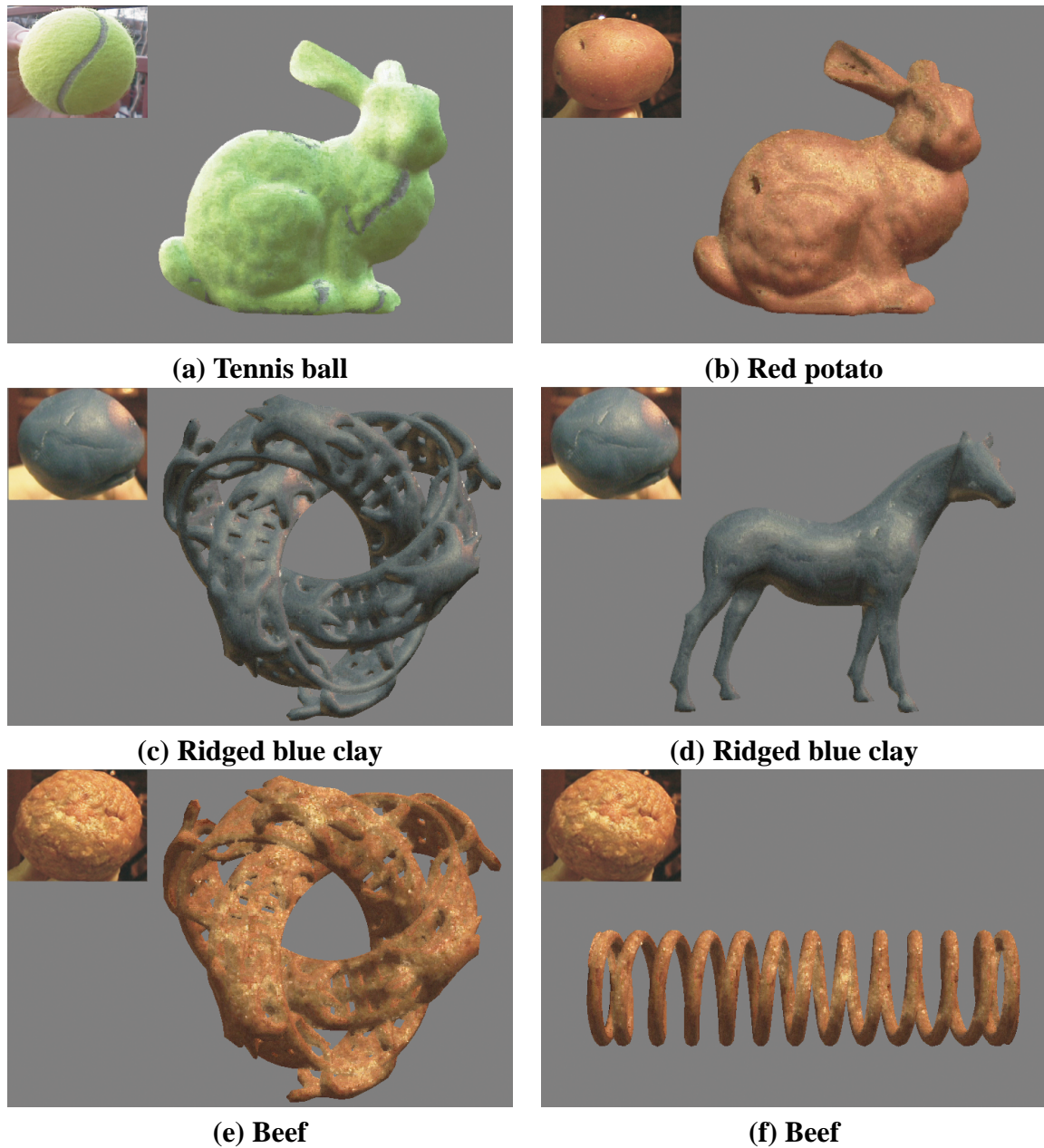


Figure 20: Results of the algorithm run on different materials, lighting conditions, and 3D models.

3.3.3.3 Results

We have run the algorithm on different combinations of lighting conditions, materials, and models. Our approach produces results in about 5-10 minutes on a Intel Xeon 2GHz processor, depending on image sizes. In Figure 20 we show several results with different models, materials, and lighting. The method produces realistic results for these textured target materials. Details such as lumps in the clay and the groove in the tennis ball are preserved. The lumps in Figure 20 (d) are not exact copies of entire lumps; some are combinations of several lumps. This prevents the result from looking like a rearranged version of the source material. The high frequency materials are sampled coherently while matching the varying surface curvature and holes in the models.

Figure 21 demonstrates that the algorithm is able to light the rendered models plausibly without any knowledge of the lights in the scene. In addition, the results are very different for the two materials. The specular highlights are successfully transferred from the source materials onto the model. Coherent matching instead of straight nearest neighbor results in highlights that are similar but not exact copies of those in the source images. We used a stand-in object to produce a shadow that we then composited the renderings over to show the degree of photorealism achievable with this approach. The method produces images which composite very well onto the background plate. In Figure 22 we show results taken from materials with no high frequency content. Our technique is able to produce good renderings of these materials, even when some of the source photographs are out of focus.

These results show that high-frequency albedo is preserved as expected. Our approach does well with a large number of materials, however some, such as the tennis ball material in Figure 20 (a) can be problematic. These materials are difficult because multiple properties must be matched simultaneously on the material object and only a single image is present. As a result, the algorithm trades off between following some high-frequency detail like the tennis ball grooves and following the model's surface. This could be remedied by using multiple reference images of the same object from different orientations but with the same illumination to provide additional samples for each normal in matching.

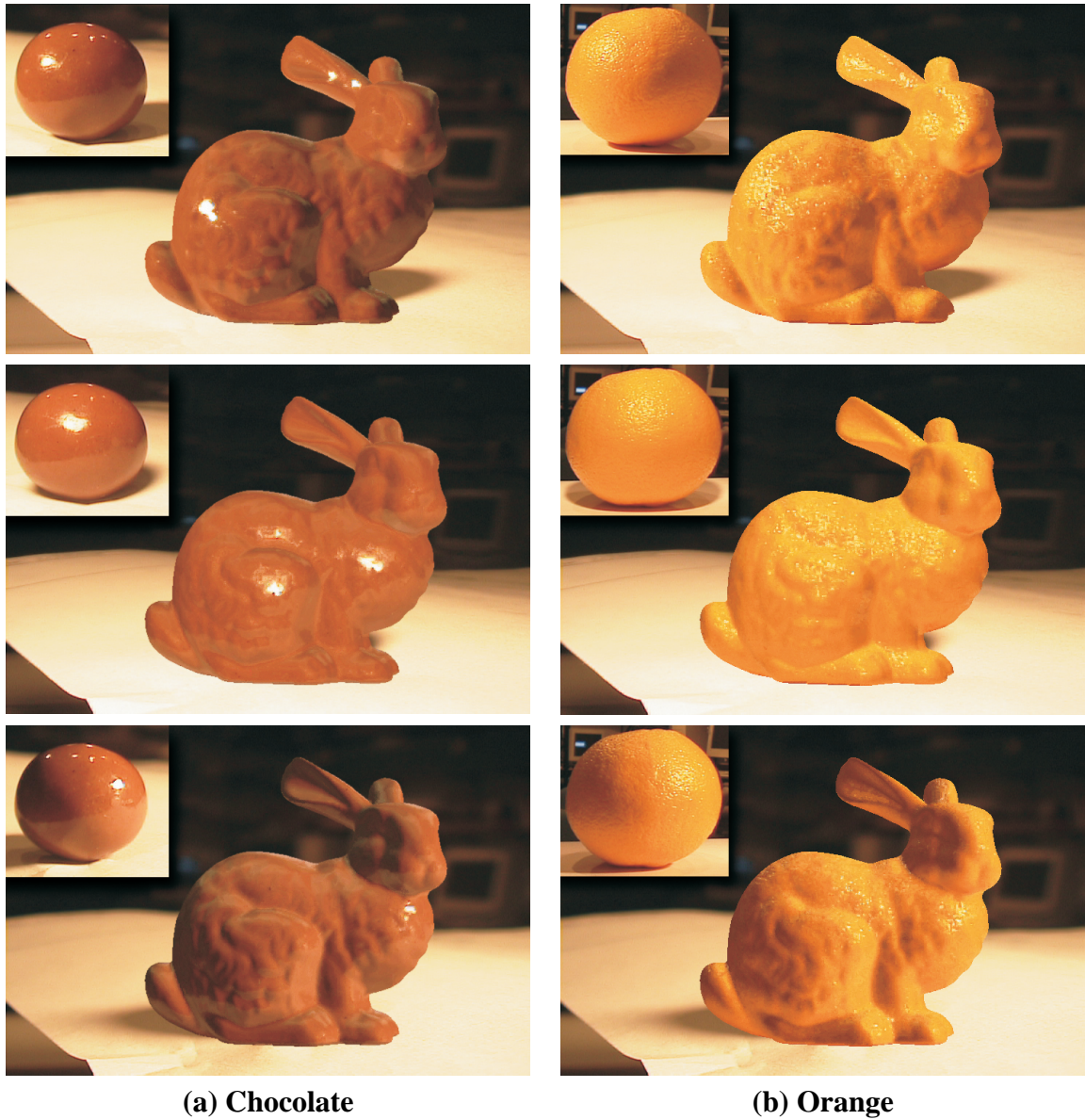


Figure 21: Renderings of the Stanford bunny 3D model dataset using chocolate and orange as materials under varying illumination. The shadow is of the original object in the scene, not of the model. Inset in each image is the material photograph that was sampled to produce the rendering.

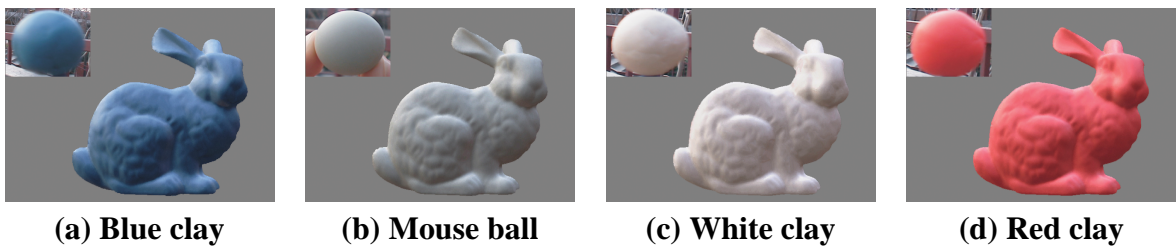


Figure 22: Renderings of the Stanford bunny using various materials photographed outdoors.

CHAPTER 4

PROBABILISTIC EXAMPLE BASED PROCESSING

In the prior chapter, we described how to perform various types of example based processing in a non-parametric manner, heavily influenced by work from texture synthesis. One major disadvantage of the framework shown in the previous chapter is exactly its relation to texture synthesis. Since the algorithm is encouraged to copy from regions in the training data, sometimes large patches can be seen in the output and in the example training. This is suitable for some applications, but for others, such as texture transfer, super-resolution, and non-photorealistic rendering of abstract art, it is insufficient.

The approach described in this chapter treats the problem of example based image processing as a general MRF labeling problem. By treating it in such a general manner, it is able to synthesize results comprised of pixels from the training images without copying large patches from the processed example training image. This algorithm makes use of a color space, created by perception researchers [50], to assist in matching color transformations more closely. By working in a color space strongly suited to our goal and by representing the learning problem as a general MRF labeling problem, we are able to synthesize results comprised of pixels from the processed example training images without copying large patches from it.

Belief propagation (BP) allows us to estimate the maximum *a posteriori* (MAP) labeling closely and computationally efficiently. Its power lies in its message passing; labeling decisions in areas of high confidence propagate outward more strongly than low confidence labelings. Over time, high confidence labelings improve the labelings in nearby regions, which in turn affect their neighbors as well. This suppresses any low confidence labelings. As such, BP is well suited for our problem as labelings in some regions may be tough to assign properly using only local support. For example, in the case of non-photorealistic rendering, brush strokes should be continuous. If

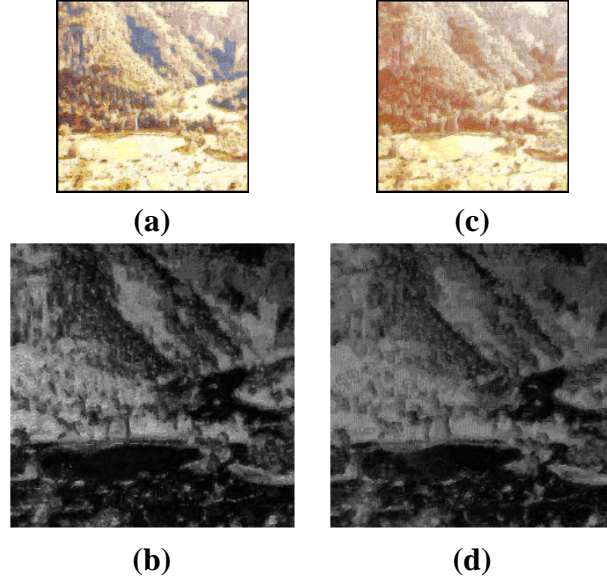


Figure 23: Final results of our algorithm when different color normalizations are used. $l\alpha\beta$ normalization: (a) in color, (b) inverted (grayscale) to show fine detail, RGB normalization: (c) in color, (d) inverted (grayscale) to show fine detail. $l\alpha\beta$ normalization yields a higher contrast output.

the labelings are assigned purely locally with small support, brush strokes could end up being discontinuous in the output image. This is the major insight of the image analogies work; an explicit constraint is used while matching to ensure coherence if a labeling is ambiguous. In our work, there is no explicit constraint, however, the labelings will be globally coherent because of our use of BP. Using BP results in output that is comprised of copied patches from training images, yet these patches are very hard to see in the output image.

4.1 Color normalization

Colors must be normalized between the unprocessed example training image and the input image because they could be substantially different in appearance. Color normalization is important because it keeps neighborhood matches from locking onto specific colors, thus matching image features more closely.

The choice of color space to work in is also very significant. In image analogies, the YIQ color space is used because it decomposes RGB colors into a luminance channel, Y , and two

channels encoding color information, I and Q . The Y channels of the unprocessed training image and the input image are then histogram matched prior to neighborhood matching. The I and Q channels are left alone if the input image’s colors are desired, or are overwritten with the I and Q intensity information from the corresponding matched processed training image pixel if colors corresponding to the processed training image are desired instead.

Using the YIQ space in this manner yields good results, as evidenced by the image analogies work, however, better matching can be achieved by using the all of the color information, instead of the luminance channel alone. Histogram matching the I and Q channels independently is incorrect because these two channels contain interrelated information, which would be lost by performing such matching. The image analogies authors found unsatisfactory results matching RGB information, where each pixel was treated as a 3×1 vector and the histogram matching was approximated as a linear map [23].

Recently, the $l\alpha\beta$ color space was created [50] and used to match color information [49] between images and to colorize grayscale images [68]. The technique proposed in [49] is simply to convert all pixels in both images from RGB to $l\alpha\beta$ using conversion matrices derived in that work, and then histogram match the l , α and β channels individually between the source and target images. Since the l , α and β channels are almost orthogonal, an individual matching is conducted on each channel without destroying color information. After matching, the inverse transformation is applied to the target image to return to the RGB space, yielding an image whose color distribution matches that of the source image. We use this technique to match the unprocessed example training image’s colors with our input image’s colors. Depending on the application, we sometimes match the processed example training image as well, for instance, if we want to use a painting’s brush strokes but not its color scheme.

This histogram matching is non-linear and provides a better mapping between the color ranges in two images. A linear RGB mapping where the mean and covariance of 3×1 color vectors in an image are mapped to a new mean and covariance provides a mapping that has the same color mean and covariance as in a target image. However, matching these statistics exactly might not map the color range correctly since the mapping may be non-linear. In Figure 23 we compare this technique with $l\alpha\beta$ mapping when used in our algorithm. We found that using the $l\alpha\beta$ mapping yields results that are more satisfying visually for our purposes, and with much higher contrast.

This is because using this space yields better color mappings with more normalized training and input feature vectors, improving belief propagation’s performance.

4.2 Belief Propagation

Image analogies, texture transfer, and super-resolution can all be expressed as inference problems: given an arbitrary input image, infer the corresponding output image based on a collection of image patch mappings provided *a priori*. This inference reduces to pixel labeling since the output image patches are matched and then sampled to produce the inferred output image. If we treat these images and patches as MRFs, we can compute the MAP labeling based on the training priors. Reducing these problem domains to MRF labeling allows us to create a single framework that can be used for all of them. We use belief propagation to perform the labeling itself since it provides an excellent estimate of the MAP labeling and is computationally efficient. The labeling produced by belief propagation for this problem is attractive because it minimizes visible seams from sampled image patches while sampling from a high enough number of areas from the training images so that no patches from the training images are readily visible.

Creating the network Our training data consists of observation and scene variables comprised of pixel neighborhoods coming from the example training images. The observation variables are 7x7 pixel neighborhoods from the unprocessed example training image, and the corresponding scene variables are composed of 3x3 pixel neighborhoods from the same locations in the processed example training image. These sizes were chosen since they yield results that are very spatially coherent and high in contrast (see Section 6.3 for comparisons with other settings). We construct these two variables for every possible pixel in the example training images.

Once the training data has been collected, we construct a Markov network representing the input image. We construct a square grid, with a node at every other pixel in the input image. Each node’s observation is the 7x7 pixel neighborhood of the pixel represented by the node. We then use belief propagation to find the scene variables from the training data that provide the MAP labeling for all of the nodes.

Matching with training data We use belief propagation to find the scene variables from the training data that are the best matches to each node in our Markov network, thus labeling the MRF. In this section, we briefly describe the belief propagation algorithm, but we refer readers seeking a deeper explanation to the excellent tutorial by Yedidia *et al.* [72].

In an MRF, the joint probability over the scene variables, x , and observation variables, y , can be expressed by:

$$P(x_1, \dots, x_N, y_1, \dots, y_N) = \prod_{(i,j)} \Psi(x_i, x_j) \prod_k \Phi(x_k, y_k) \quad (9)$$

where Ψ and Φ are compatibility functions which will be described below, and N is the number of nodes in the Markov network.

$$\hat{x}_{j_{MAP}} = \underset{x_j}{\operatorname{argmax}} \Phi(x_j, y_j) \prod_k M_j^k \quad (10)$$

$$M_j^k = \max_{x_k} \Psi(x_j, x_k) \Phi(x_k, y_k) \prod_{l \neq j} \tilde{M}_k^l \quad (11)$$

Each node is initialized with n candidate scene variables from the training data. The n correspond to the n training observation variables with the smallest $L2$ error with the node's observation. Equation 10 selects the candidate from the n that is the MAP estimate. We use the MAP rules because there is a strong local maximum of posterior, even for non-Gaussians, as was found by Weiss and Freeman [67].

In belief propagation, messages are passed between all of the nodes, representing their 'beliefs' about the scene variable candidates, taking into account the candidates of the neighbors. In this notation, M_j^k is an n dimensional vector with each element of the vector representing the compatibility of each scene variable candidate of node j with node k (one of possibly many neighbors of j). Equation 11 can be considered to summarize the computations at neighboring nodes, and results in an increased or decreased weighting of each candidate of j , considering all of the neighbors of k . \tilde{M} represents a message from a previous iteration.

$$\Psi(x_k^l, x_j^m) = \exp(-|d_{jk}^l - d_{kj}^m|^2 / 2\sigma_s^2) \quad (12)$$

$$\Phi(x_k^l, y_k) = \exp(-|y_k^l - y_o|^2 / 2\sigma_i^2) \quad (13)$$

Since the scene patches overlap, we compute the error of the overlaps to compute the compatibility matrix Ψ . Thus, equation 12 can be used to construct a matrix expressing the compatibility between all of the candidates of node k and node j with each other. The rows of the compatibility matrix are indexed by the candidate ids (l) of node k , and the columns indexed by the candidate ids (m) of node j . d_{jk} and d_{kj} represent the pixels that overlap between the 3x3 pixel neighborhood used as a scene variable of nodes j and k between j and k (d_{jk}) and k and j (d_{kj}). Equation 13 is simply a measurement of how different the observation variable corresponding to each scene candidate for node k (indexed by l) is from the actual neighborhood corresponding to node k . This equation is used with the compatibility matrix to weight up/down the different scene candidates with respect to the candidates at neighboring nodes, and their neighbors in equation 11.

The compatibility functions Ψ and Φ can be learned from the training data or approximated. We use the same approximations as in [14] shown in equations 12 and 13. These simple compatibility functions yield satisfactory results because our problem is a more general version of super-resolution. However, the descriptiveness of the features is possibly more important. Figure 23 illustrates this point; depending on how the histograms are matched in our images, the same compatibility functions can yield very different labelings. Likewise, using the features used in [14] would yield excellent super-resolution results with these functions, but would produce unsatisfactory labelings for the more general image mappings we are interested in.

On the first iteration, we find all of the scene candidates for each node in the network by choosing the n (typically 10) whose corresponding observation variables are the approximate nearest neighbors to each node's observation. All messages are initialized to be column vectors of 1's, and we compute the MAP scene estimates at each node using equation 10. On subsequent iterations, we update all of the messages using equation 11, and recompute the MAP scene estimates. We typically get stable results in about 5 to 10 iterations, depending on the values of σ_s and σ_i , which we treat as user parameters. Lower values for σ_s penalize scene candidates that differ in overlap from neighboring scene candidates. Similarly, lower values for σ_i penalize scene candidates whose corresponding observation variables differ from the observation at the node they are candidates for.

Once we are done iterating, we produce an output image by averaging the overlap regions at pixels where there were no nodes in the Markov network. Averaging sometimes yields grid artifacts in the output images, but these tend not to be visible or present if the images have high contrast and significant high frequency detail.

4.3 Results & Discussion

We have used our framework to learn several types of image processing operations by example. For each experiment, the same training image pairs and input image were used with our system and with image analogies (downloaded from <http://mrl.nyu.edu/projects/image-analogies/>). We also compare our results with our implementation of the quilting texture transfer algorithm presented in [13] since it can be used to perform general image mappings as well. In the quilting runs, each iteration of the quilting algorithm reduces the patch and overlap size by a third.

Our algorithm produces results that resemble both the training images and the input images without strictly copying from the training images. There is no constraint on copying in contrast to image analogies which has an explicit copying constraint and quilting which has an implicit ('use large enough patches') constraint. Our algorithm typically takes 8 minutes to initialize and 1.5 minutes per iteration thereafter on an Intel Xeon 2GHz processor for 250x250 images. 5-7 iterations usually produce a satisfactory labeling.

4.3.1 Texture transfer

Texture transfer is an interesting problem because the goal is to synthesize an image that resembles the input image in organization and the training input in appearance. Our algorithm synthesizes an image that looks like both the input image and the training image while following the image features. It can do this because belief propagation encourages matching of large scale features globally. This is because the dimensionality of the observation variables is higher than that of the scene variables. It is hard to provide a quantitative comparison for this problem because there is no correct solution. The image analogies results (Figure 24 (e)-(f)) are very coherent and tend to resemble the input texture more than the photograph. We used image quilting to do the transfer as well (Figure 24 (g)). Quilting also tends to favor the input texture. By adjusting the contrast

weights, it is possible to favor the input texture more than the input image, or vice versa. This was as close as we were able to adjust the parameters to perform the texture transfer.

4.3.2 Super-resolution

We used super-resolution as a test case because it was a problem that was originally addressed well with belief propagation and image analogies. We also wanted to see how well the algorithms would perform if color were not an issue at all. We did not try using bandpassed images as in [14], because we wanted our framework to work for different varieties of image mappings. This experiment was purely to see how our algorithm and the others would perform on a task that did not require color to provide a fair basis for comparison.

In this experiment, our algorithm does a much better job of keeping the whiskers, eyes and nose intact than either algorithm. This is due to the BP message passing; BP ensures that the high resolution patches chosen at each node resemble their corresponding blurry observations while forming a coherent output image. Figure 25 shows that while the image analogies algorithm could be easily modified to do a better job matching by using the $l\alpha\beta$ color space instead of the YIQ space, there are some set limitations due to the scanline nature of the algorithm. The quilting texture transfer algorithm suffers because of its reliance on patches. We experimented with different patch sizes, number of iterations, and different weights on the matching constraints. However, due to its reliance on patches, the quilting algorithm is, not surprisingly, ill-suited for this application.

4.3.3 Non-photorealistic rendering

Non-photorealistic rendering (NPR) is an interesting application because it allows for the automation of many of the mechanical details of painting creation. NPR can also be useful to create media that would otherwise be prohibitively time expensive, such as a video comprised of a changing painting. It would be very attractive for users to be able to specify a style by providing an example painting rather than setting a collection of parameters in an NPR program.

Providing a good comparison between the algorithms is hard for this application because these results are highly subjective. Our algorithm produces the type of results we expected for this problem. Our output images simultaneously look like the input image and the training painting

while not necessarily resembling either exactly. In addition, it is hard to identify the sampling locations of the brush strokes in the training images as was our intention. This produces an output that has a similar look to the painting, yet does not appear to be a rearranged version of the source image.

Large brush-stroke painting In this experiment, we used a painting included with the image analogies distribution. Figure 26 shows our results for this experiment. Our algorithm produces results with matching colors and strokes similar to the input painting. When the source image color is used, our results and the image analogies results are similar, though not identical as our result has more visible ‘flecks’ of paint. Our output resembles the quilting result when the painting color is used. However, both results differ in their composition.

Abstract painting In this experiment, we used an abstract painting consisting of globs of paint on a canvas. Our algorithm synthesizes an image that resembles the painting, yet contains no visibly copied patches from the painting. This data set presents a challenge for image analogies because there are only large scale features to copy from, and switching from one glob of paint to another could create visible seams and other artifacts. Image analogies and quilting produce interesting results that differ from ours and from each other. As a result, abstract painting rendering appears to be a problem domain where having all three algorithms as an option would be valuable to a user.

Impressionist painting Figure 28 shows results run on an impressionist painting with pastel colors. Our algorithm synthesizes an image that has similar brush strokes to the painting, but loses some of the detail at the bottom of the image (e.g. the houses). When we use the painting’s original colors, our synthesized image looks like a mix of the training painting and the input image.

Post-impressionist painting 1 Figure 29 shows results run on a post-impressionist painting. Our algorithm synthesizes an image that has similar brush strokes to the painting, particularly in the shrubbery to the left and especially the right of the door. These strokes are very similar to those in the tree/plant areas of the painting. Similarly, the strokes used for the path along the bottom leading up to the door resemble the much smaller strokes used on the road in the painting.

Post-impressionist painting 2 Figure 30 shows results run on a post-impressionist painting. The results bear similarities to the source painting in several areas. The bushy tail of the squirrel resembles the brim of Van Gogh's hat. The squirrel's fur resembles portions of the jacket and door in the painting. These results are due to the fact that these corresponding areas are texturally similar to each other in the training and input images. Likewise, since there is no texture and hardly any pixel value variation in the area to the right of the squirrel's head, the algorithm has a hard time selecting which training samples to use in this area of the output image.

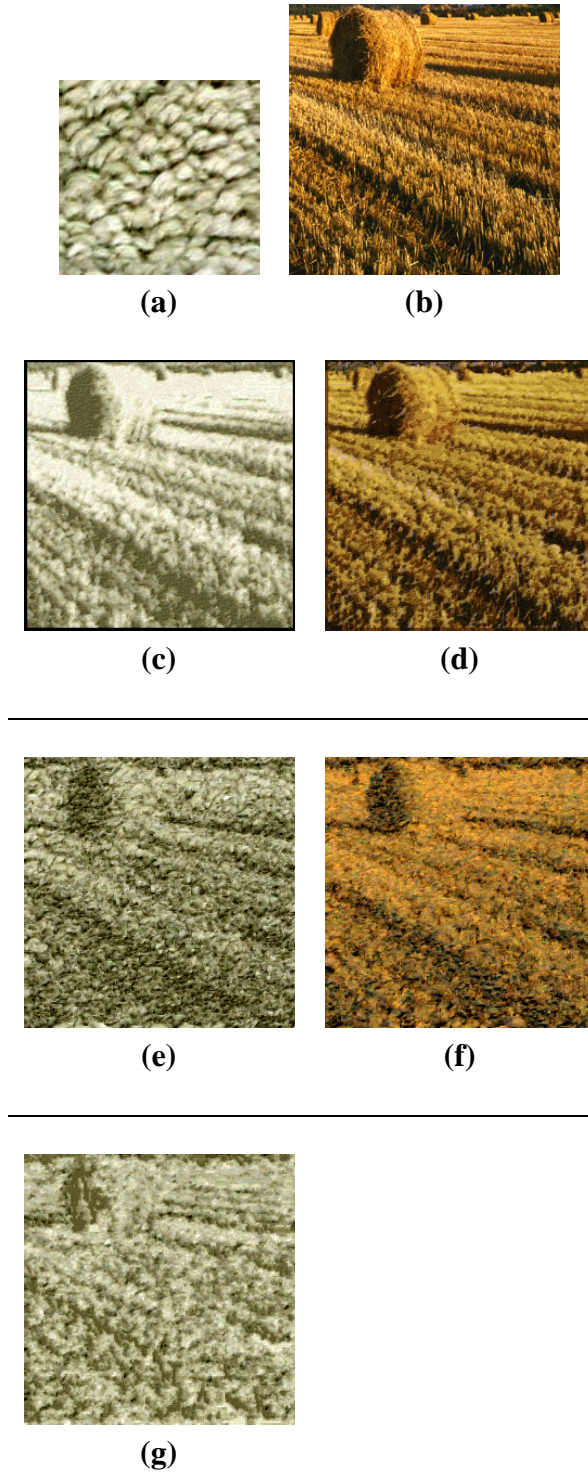


Figure 24: Texture transfer, (a) target image (used as both training input and output), (b) input image, (c)-(d) output of our algorithm with $\sigma_i = 10$, $\sigma_s = 0.5$ and texture/input colors, (e)-(f) , image analogies output with $\kappa = 2$ and texture/input colors, (g) quilting texture transfer result, 30x30 patches with 10x10 overlap, 3 iterations.

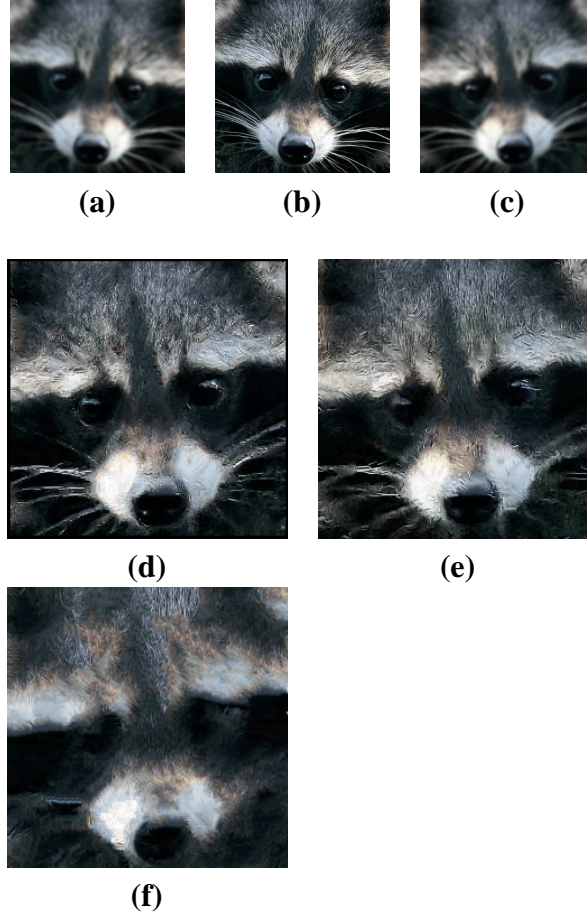


Figure 25: Super-resolution, (a) training example input: blurred image, (b) training example output: original image, (c) input image (mirrored version of (a)), (d) output of our algorithm with $\sigma_i = 1$, $\sigma_s = 0.5$, (e) image analogies output with $\kappa = 2$, (f) quilting texture transfer result, 30x30 patches with 15x15 overlap, 2 iterations.

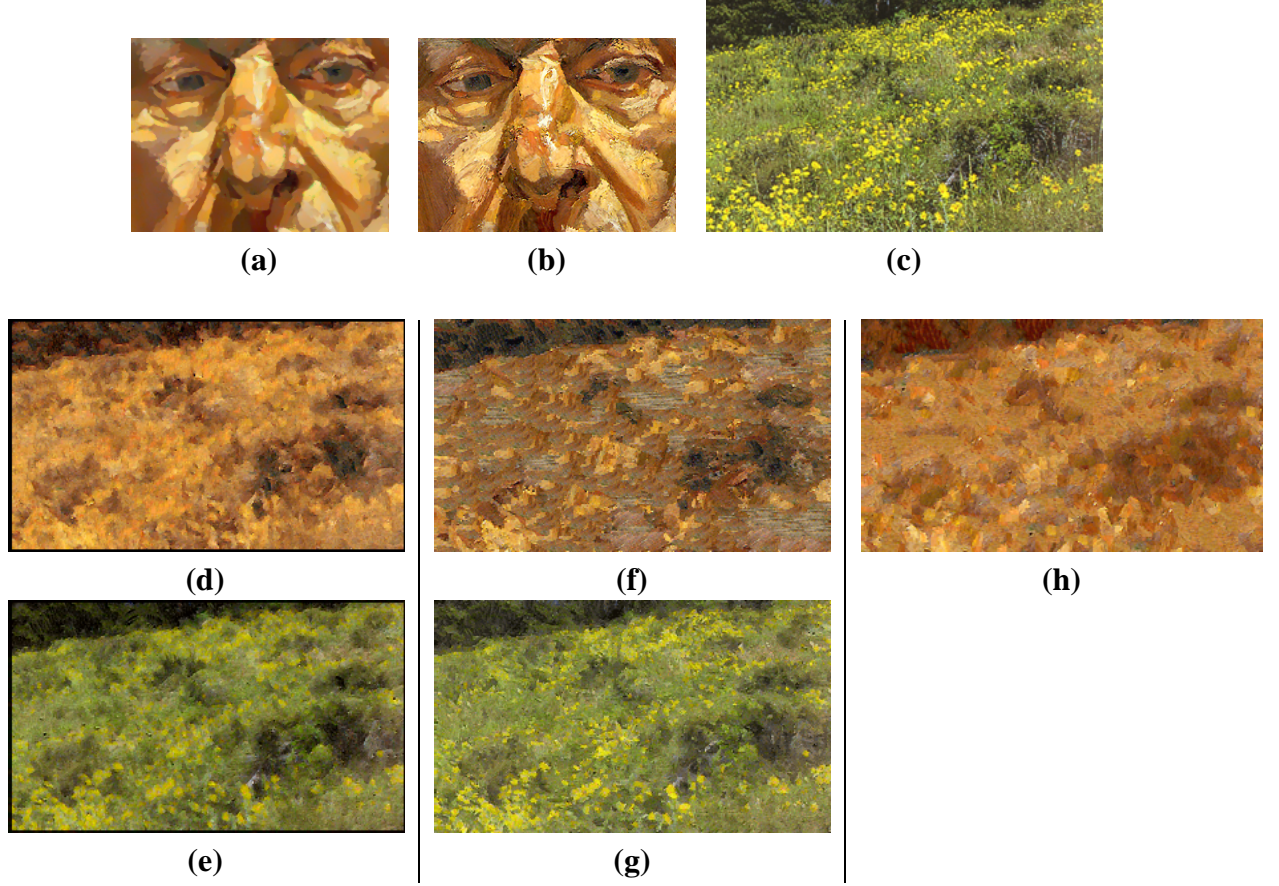


Figure 26: *Reflection (Self-Portrait)*, 1985 by Lucian Freud, (a) training example input: anisotropically blurred painting, (b) training example output: original painting, (c) input image, (d)-(e) output of our algorithm with $\sigma_i = 1$, $\sigma_s = 0.5$ and painting/input colors, (f)-(g) image analogies output with $\kappa = 2$ and painting/input colors, (h) quilting texture transfer result, 30x30 patches with 10x10 overlap, 2 iterations.

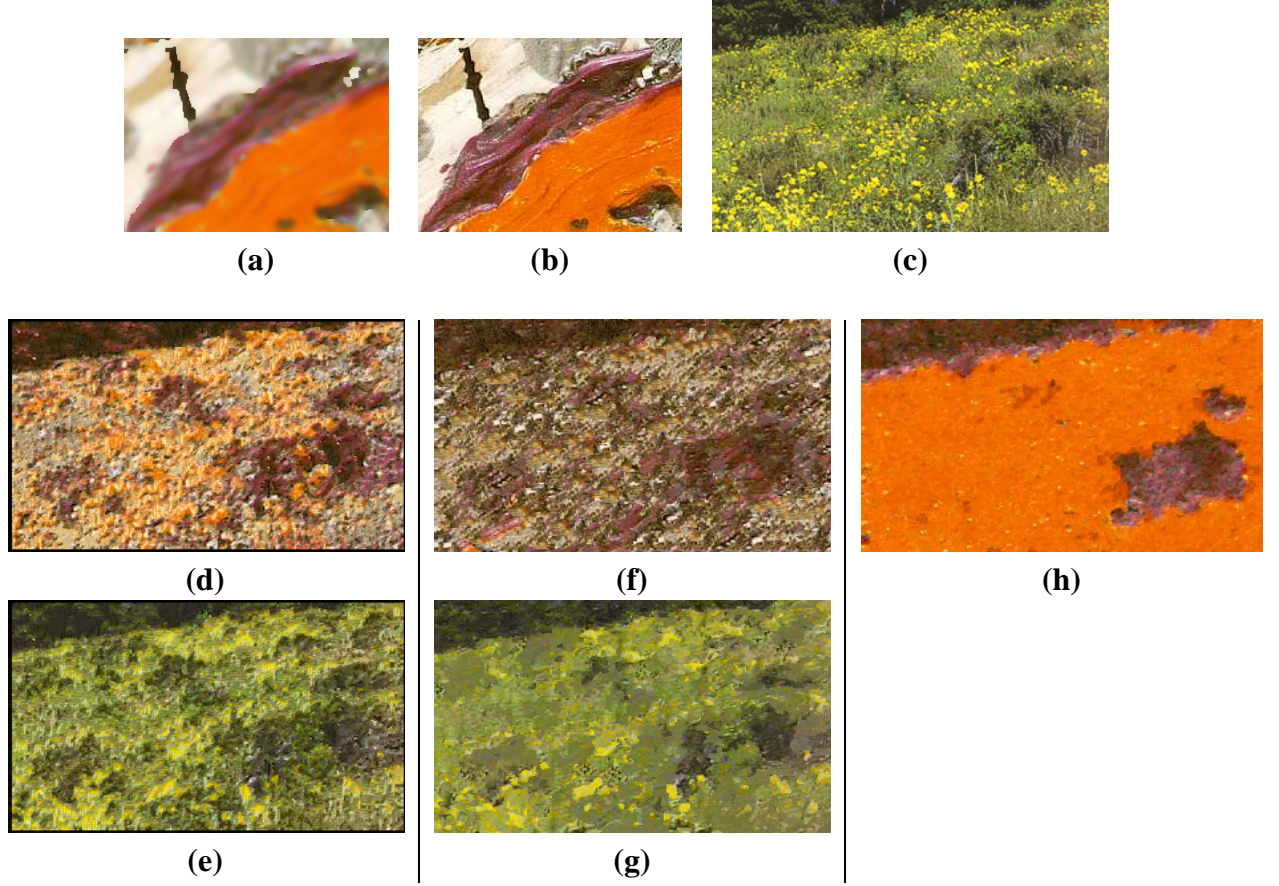


Figure 27: *Full Fathom Five*, 1947 by Jackson Pollock, (a) training example input: anisotropically blurred painting, (b) training example output: original painting, (c) input image, (d)-(e) output of our algorithm with $\sigma_i = 1$, $\sigma_s = 0.5$ and painting/input colors, (f)-(g) image analogies output with $\kappa = 2$ and painting/input colors, (h) quilting texture transfer result, 40x40 patches with 20x20 overlap, 3 iterations.

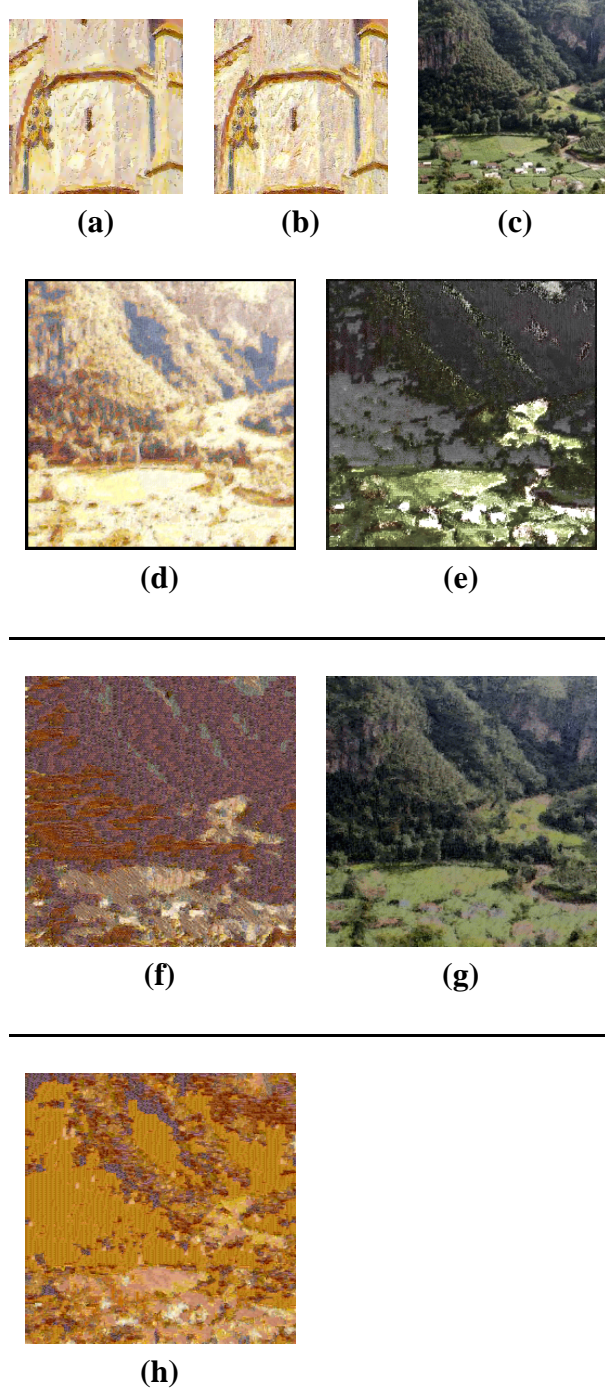


Figure 28: *The Church at Moret*, 1894 by Alfred Sisley, (a) training example input: anisotropically blurred painting, (b) training example output: original painting, (c) input image, (d)-(e) output of our algorithm with $\sigma_i = 1$, $\sigma_s = 0.5$ and painting/input colors, (f)-(g) image analogies output with $\kappa = 2$ and painting/input colors, (h) quilting texture transfer result, 30x30 patches with 10x10 overlap, 3 iterations.

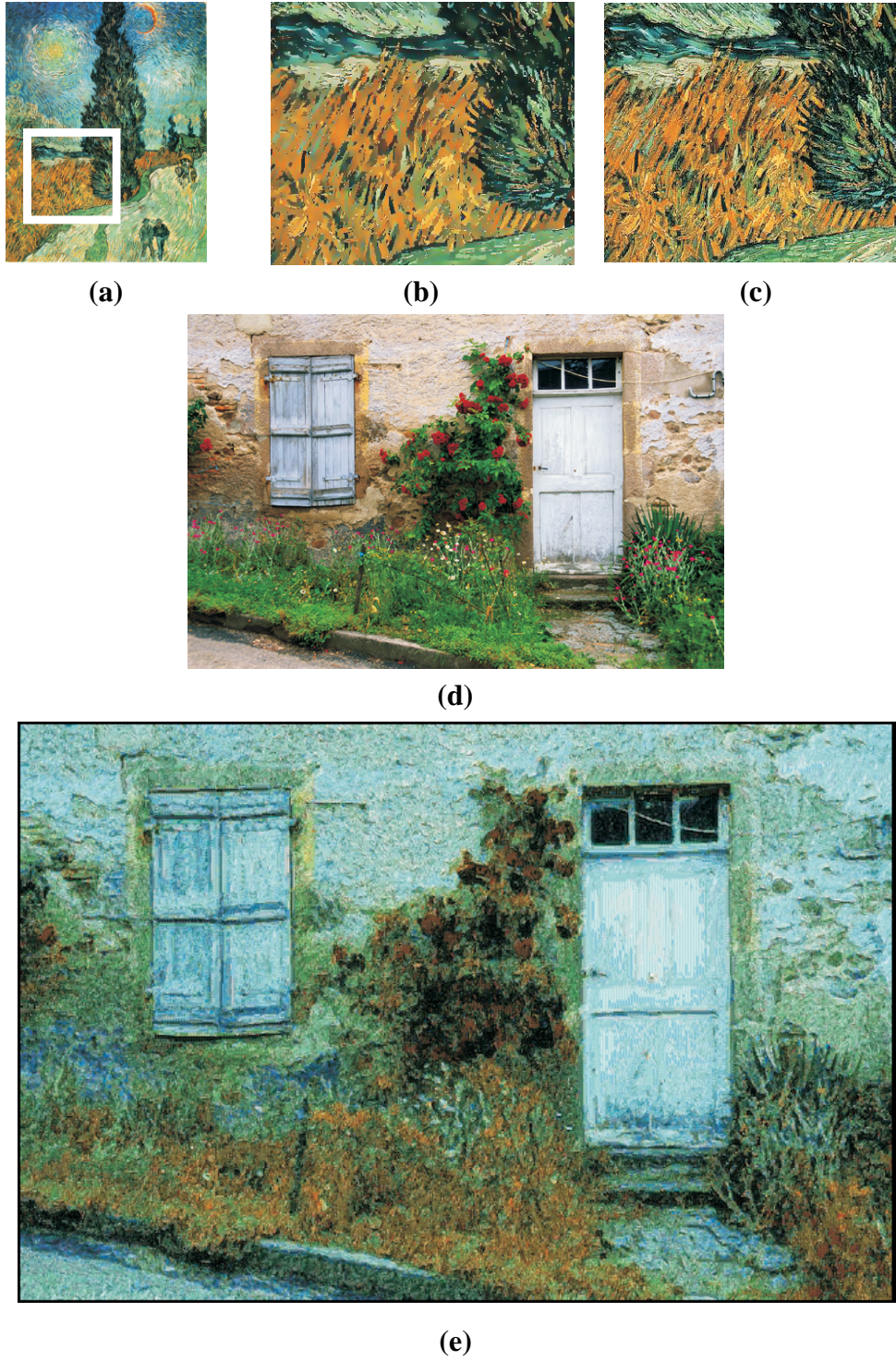


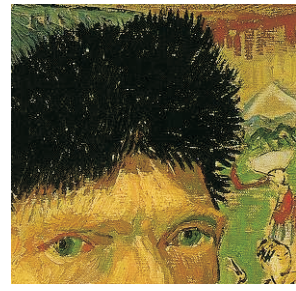
Figure 29: *Road with Cypress and Star*, 1890 by Vincent Van Gogh, (a) entire painting used, shrunk to show composition, (b) portion of training example input: anisotropically blurred painting, (c) portion of training example output: original painting, (d) input image, (e) output of our algorithm with $\sigma_i = 1$, $\sigma_s = 0.5$ and painting's colors.



(a)



(b)



(c)



(d)



(e)

Figure 30: *Self-Portrait with Bandaged Ear*, 1889 by Vincent Van Gogh, (a) entire painting used, shrunken to show composition, (b) portion of training example input: anisotropically blurred painting, (c) portion of training example output: original painting, (d) input image, (e) output of our algorithm with $\sigma_i = 1$, $\sigma_s = 0.5$ and input image's colors.

CHAPTER 5

EXPERIMENTAL EVALUATION

So far, we have presented various example based processing algorithms which were non-parametric and closely related to texture synthesis (Chapter 3) and a probabilistic algorithm related to MRF labeling (Chapter 4). The latter algorithm is not related to texture synthesis algorithms, yet has an MRF framework in common with those approaches. In this chapter, we present various experiments to contrast this probabilistic approach with texture synthesis related example based image processing algorithms to illustrate their differences.

We created four different images to use as training and input against each other. In each experiment, we tried every combination of the four images in both roles. The four images were created so as to have very different edges to test how well the algorithms generalized the training data given. The images consist of an X, a circle, a square, and a letter B rotated thirty degrees to the right. The circle was chosen for its lack of hard edges, the X for its diagonal edges, the square for its hard edges, and the B for having traits in common with each of the other images to see if the algorithms could take advantage of this fact. For the experiments in Sections 5.1 and 5.2, we compare our algorithm against image analogies alone since image quilting cannot cope with untextured images. We compare against quilting as well in the experiment in Section 5.3 since it uses textured versions of these images. For each error measurement between an algorithm output and ground truth, several trials with differing rotations and translations of the training pair were used to compute the error's mean and standard deviation. Twelve trials were run consisting of eight rotations spaced by 45 degrees, and four translations in both the x and y directions, with each transformation applied to the training pair.

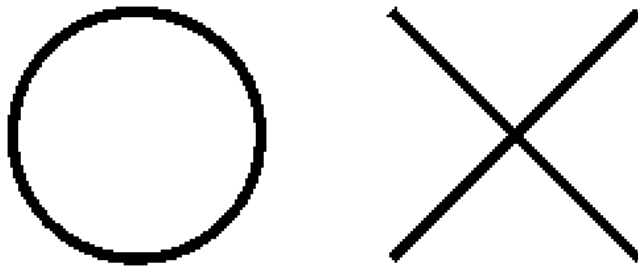
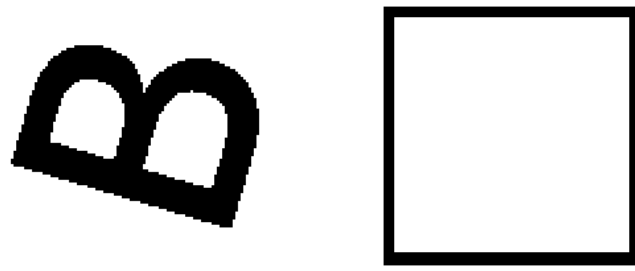
5.1 Super-resolution experiments

To create the input images, we took the image set and resized them to be half their initial size. We then resized the reduced images to their original sizes to produce the inputs. No interpolation was performed when down/upsampling so that the interpolation would not affect the results.

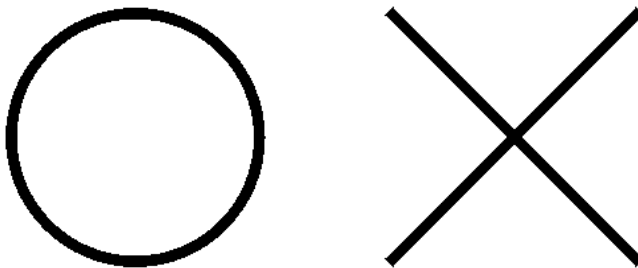
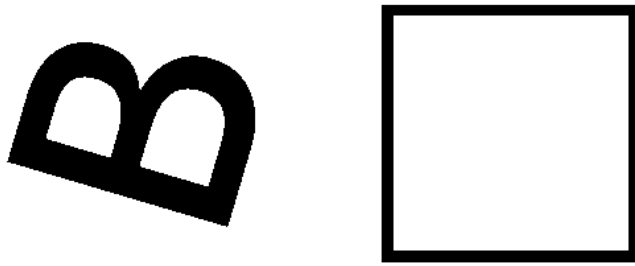
Figure 31 shows all of the images used in these experiments. Figures 32-35 show the mean and standard deviation of rms error across the trials compared to ground truth. Image analogies and belief propagation perform very similarly in this experiment. There are slight differences depending on which input/training image combinations are used, but the errors are similar. The lack of texture reduces the variability in intensity values, and the small patch sizes used by the algorithms in matching ensure that reasonable matches are selected. In Figure 34 our algorithm and image analogies do better/worse than each other when the circle and X images are used for training. Belief propagation does worse when the circle is used because the jaggedness of the circle training input image causes suboptimal patches to be chosen for some pixels. One interesting difference in the two results is that ours is a complete square, but the image analogies result has one edge that is jagged, which causes a higher visual error but a lower rms error. This suggests that human perception based error metrics may yield additional insights that pixel-based metrics may miss.

Figures 34 and 35 show that both algorithms have higher error variability when the square and X images are used as test input. This is because these images consist of single dominant features: horizontal edges in the square image, and diagonal edges in the X image. Consequently, transforming the training data causes the error to change significantly since the images used for training contain these features only at certain orientations.

Figure 35 shows the fundamental differences in the algorithms when the circle and square images are used as training with the X image as test input. Image analogies' errors do not change much as the circle is transformed, but do vary when the square is transformed. This is because of its reliance on copying. The circle does not change much under rotations or translations, so copying from similar locations in the image results in similar errors under different transformations. The square does change significantly, so the errors increase. Belief propagation does better on the square training set, but worse on the circle training set. The reasons for this are opposite to the



(a)



(b)

Figure 31: Super-resolution experiment: Training images used in experiments, (a) inputs, (b) outputs.

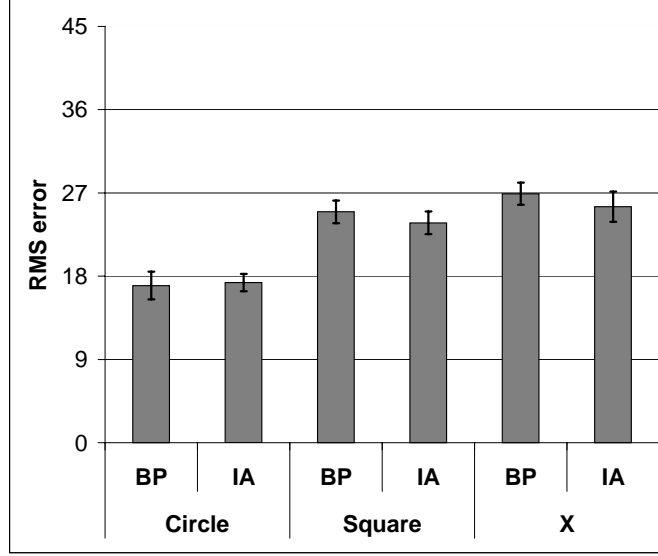


Figure 32: Super-resolution experiment: B image. *BP*: belief propagation, *IA*: image analogies.

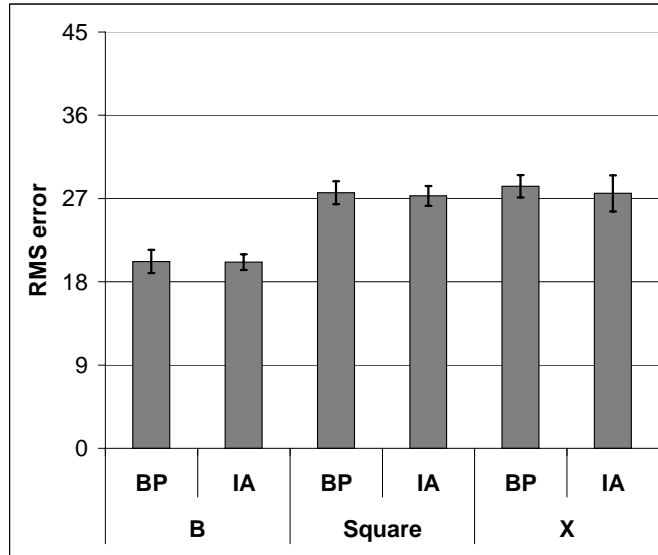


Figure 33: Super-resolution experiment: Circle image. *BP*: belief propagation, *IA*: image analogies.

reasons why image analogies' errors varied. The circle changes locally since it is aliased and this affects the result since different input/output pixel patch pairs will be produced under different transformations. When the square is transformed, its appearance changes as well, however, the differences between the training input and output are localized since strong edges are present.

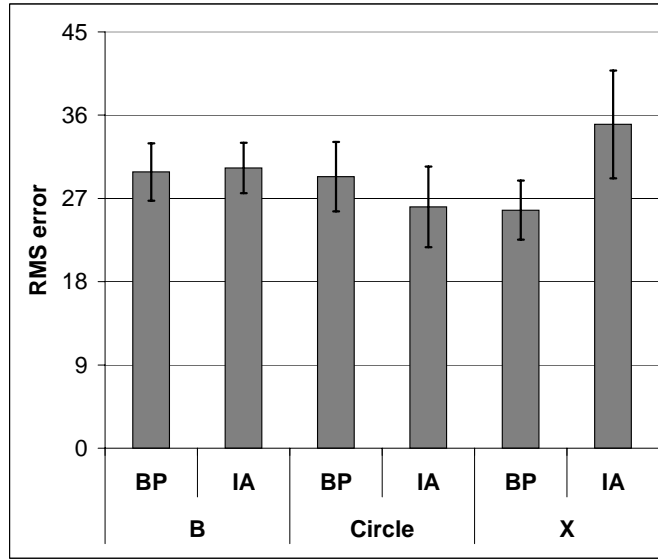


Figure 34: Super-resolution experiment: Square image. *BP*: belief propagation, *IA*: image analogies.

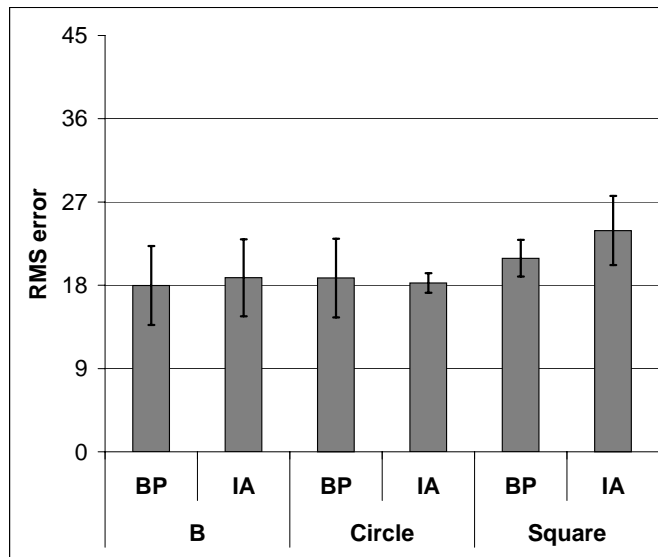
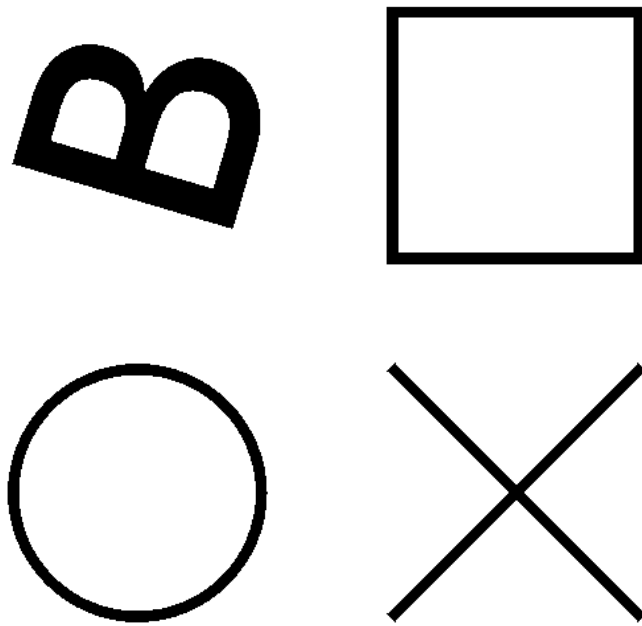
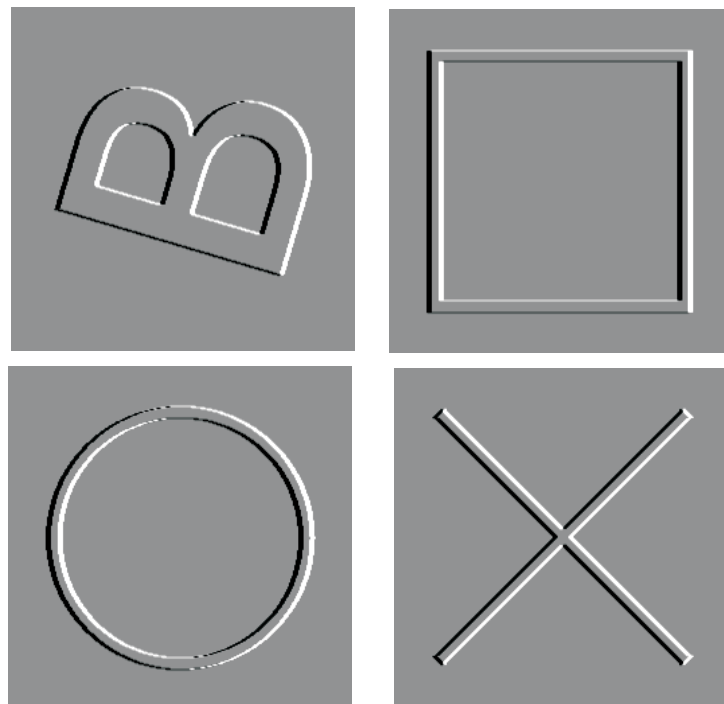


Figure 35: Super-resolution experiment: X image. *BP*: belief propagation, *IA*: image analogies.



(a)



(b)

Figure 36: Emboss experiment: Training images used in experiments, (a) inputs, (b) outputs.

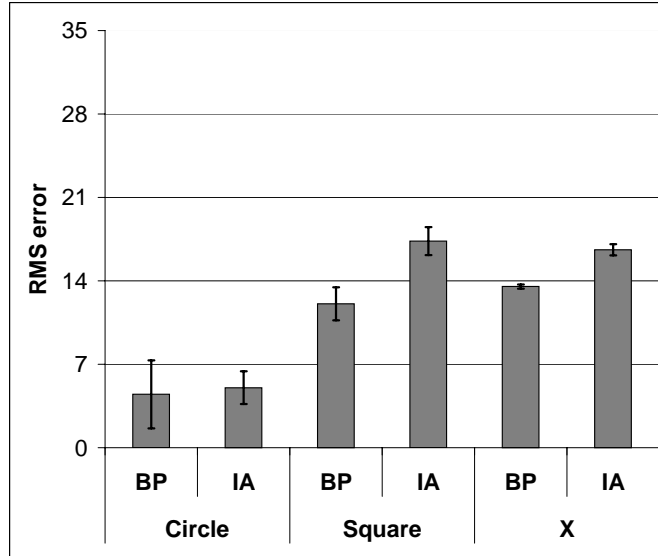


Figure 37: Emboss experiment: B image. *BP*: belief propagation, *IA*: image analogies.

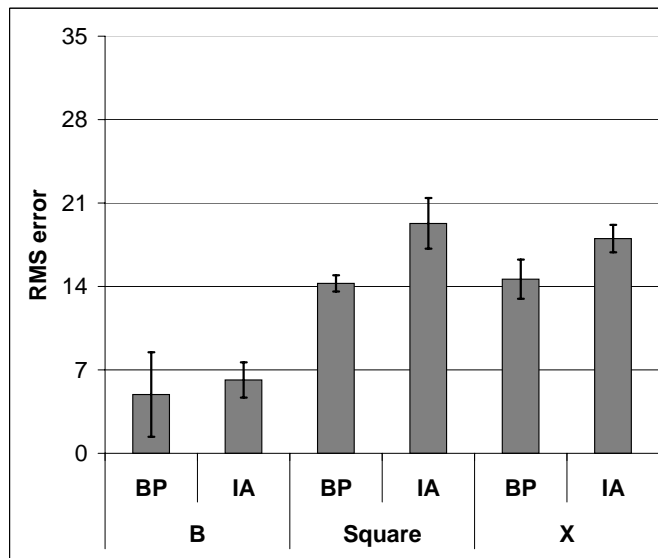


Figure 38: Emboss experiment: Circle image. *BP*: belief propagation, *IA*: image analogies.

5.2 Emboss experiments

We took the input images and applied the Adobe Photoshop emboss filter. Figure 36 shows all of the images used for these experiments. Figures 37-40 show the mean rms error compared to ground truth. In this set of experiments, belief propagation almost always has a lower error than image

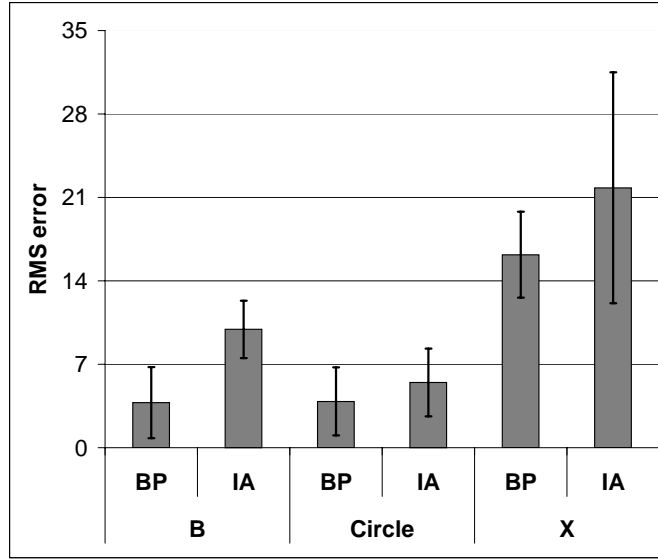


Figure 39: Emboss experiment: Square image. *BP*: belief propagation, *IA*: image analogies.

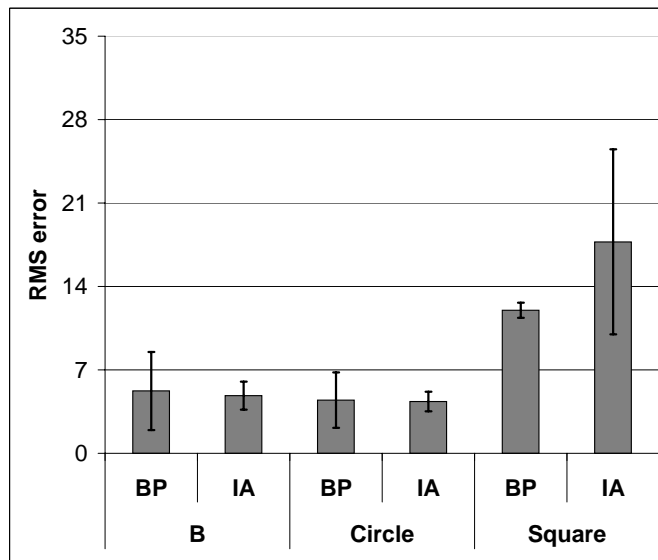


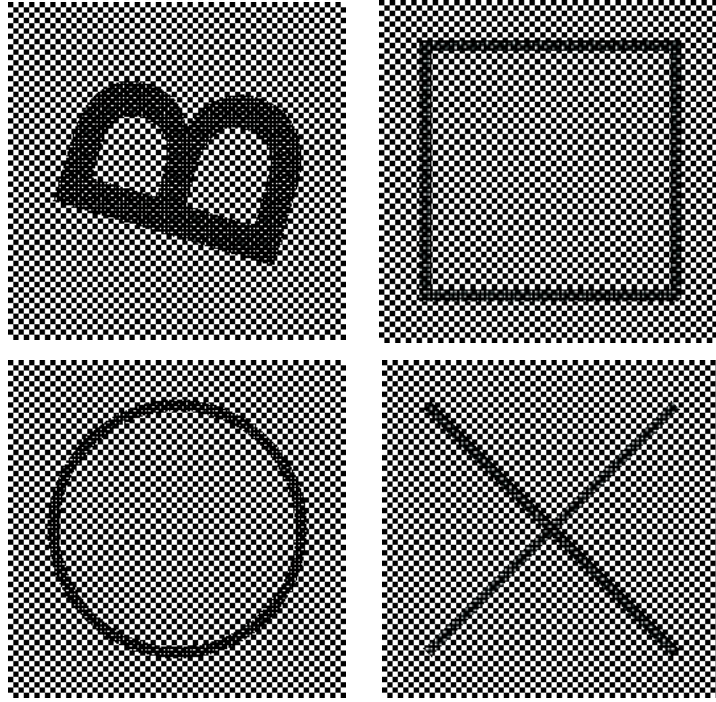
Figure 40: Emboss experiment: X image. *BP*: belief propagation, *IA*: image analogies.

analogies, sometimes by a large amount. This is due to the combination of textureless images and the use of training output that is significantly different than the training input. The lack of texture causes image analogies to get stuck coherently copying along the inside of an edge, but then once the edge's contour is reached the image regions being matched drastically change in appearance causing a bad match. These sudden changes occur because of the major edge differences between

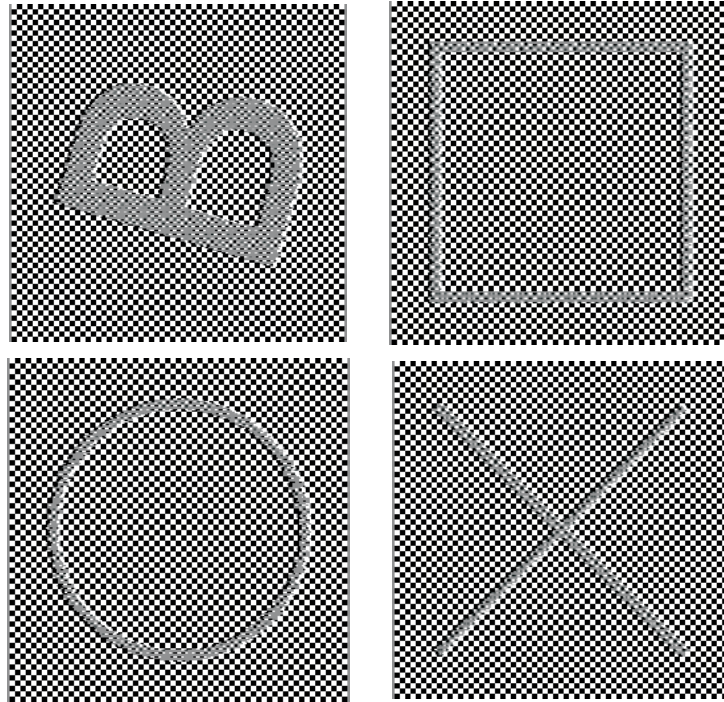
all four images.

However, the majority of the image analogies outputs are visually plausible results; they only suffer from minor incoherence once the algorithm starts copying from badly matching parts of the training image. The only visually incoherent results occur when using the square and X images with each other. These are the results with the largest rms and visual errors. The outputs look like dotted line versions of what the output should be because the algorithm gets confused by the staircase pattern along the edges of the X in both cases. As it begins to follow the contours while copying, they either jut inward or outward, forcing the algorithm to choose the nearest neighbor instead of the coherent match at every other pixel. Belief propagation has problems with the B image as test input and training. This is because the available pixel input/output patch pairs for matching and training changes significantly due to the asymmetry of the B image. This difference in pixel patches over transformations causes variation in the error. Both algorithms have large error variations whenever the X and square images are used as test or training against each other for the reasons previously discussed.

In all of these experiments, belief propagation does a better job at generalizing the very different images to synthesize output with low error. It also does a good job dealing with untextured regions, however, as we found in experiments on real world images, it is dependent on image contrast. This dataset consists of very high contrast images which is optimal for the algorithm.



(a)



(b)

Figure 41: Emboss experiment with textured images: Training images used in experiments, (a) inputs, (b) outputs.

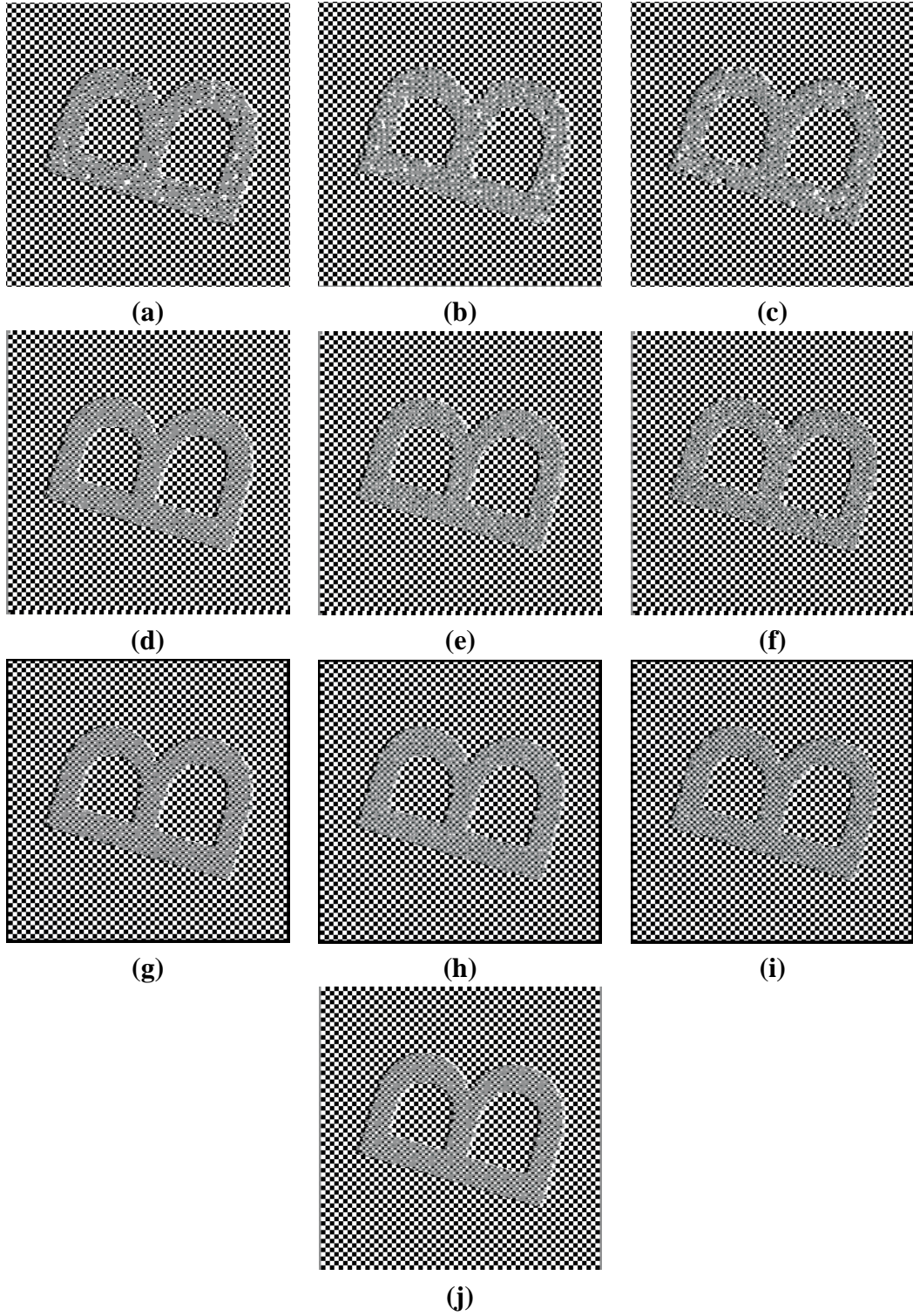


Figure 42: Different outputs using the 'B' image as input and other images as training from Figure 41 (a). Quilting: (a) using 'Circle', (b) using 'Square', (c) using 'X'; Image analogies: (d) using 'Circle', (e) using 'Square', (f) using 'X'; Belief propagation: (g) using 'Circle', (h) using 'Square', (i) using 'X'; (j) ground truth.

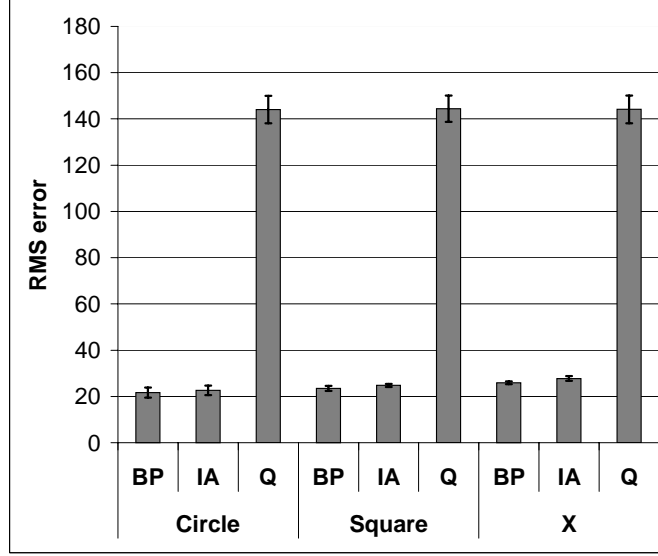


Figure 43: Emboss experiment with textured images: B image. *BP*: belief propagation, *IA*: image analogies, *Q*: image quilting.

5.3 Emboss (textured images) experiment

We took the input images from the previous experiments and added texture to them. This was done so that we could compare our algorithm with image quilting as well, since that algorithm requires textured images to determine where to place patches. We replaced the white backgrounds with a black and white checkerboard pattern. We also added a checkerboard-like pattern to the letters themselves. After applying texture to the background and foreground, we again applied the Adobe Photoshop emboss filter. Figure 41 shows all of the images used for these experiments. Figures 43-46 show the mean rms error compared to ground truth.

Image quilting has large rms errors because the checkerboard pattern in its synthesized outputs do not line up exactly with those in the input image. This is because the patch size is smaller than the size of a black or white box on the checkerboard. We experimented with larger block sizes, but then ended up with visually incoherent results. We show one complete set of results for the B image with all three algorithms in Figure 42. Figure 42 (a)-(c) show the quilting results. While the error is high, these results are visually coherent and visually similar to ground truth. The image analogies results shown in (d)-(f) are similar to the belief propagation results in (g)-(i).

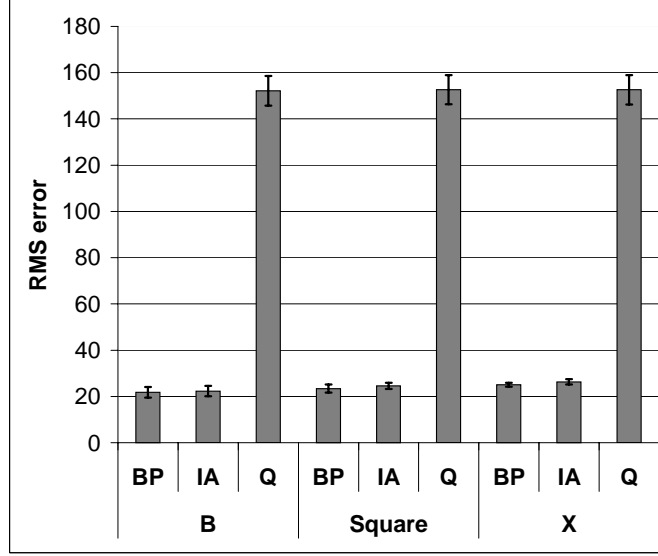


Figure 44: Emboss experiment with textured images: Circle image. *BP*: belief propagation, *IA*: image analogies, *Q*: image quilting.

In this set of experiments the results and errors generated by image analogies and belief propagation are comparable. Belief propagation generates results that are slightly more spatially coherent with smaller rms error. The quilting results capture the checkerboard pattern correctly but do not capture the interior texture pattern on the letters properly. This is due to the nature of the algorithm; quilting is better suited for texture synthesis or unknown example based processing such as non-photorealistic rendering due to its use of large patches in synthesis.

The presence of texture prevents image analogies from getting stuck coherently copying to the point of making bad matches (as in the experiments of Section 5.2) since texture ensures that the coherent matches selected will be suitable.

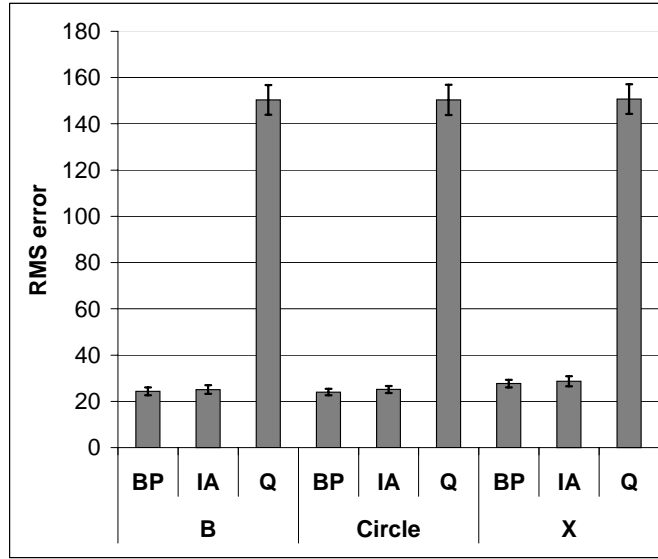


Figure 45: Emboss experiment with textured images: Square image. *BP*: belief propagation, *IA*: image analogies, *Q*: image quilting.

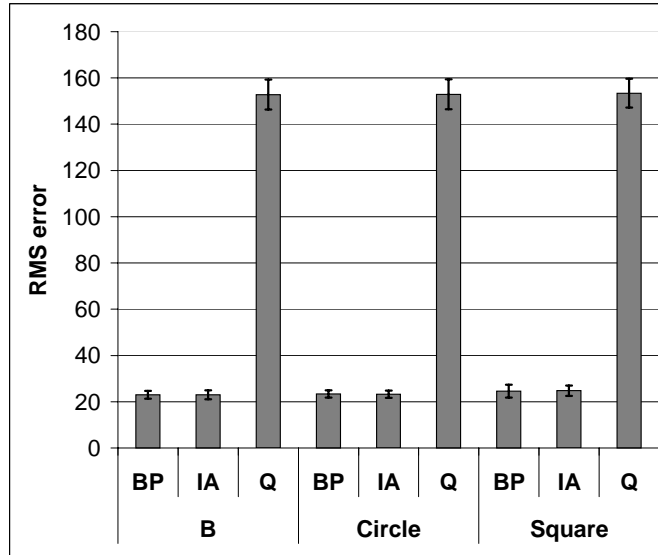


Figure 46: Emboss experiment with textured images: X image. *BP*: belief propagation, *IA*: image analogies, *Q*: image quilting.

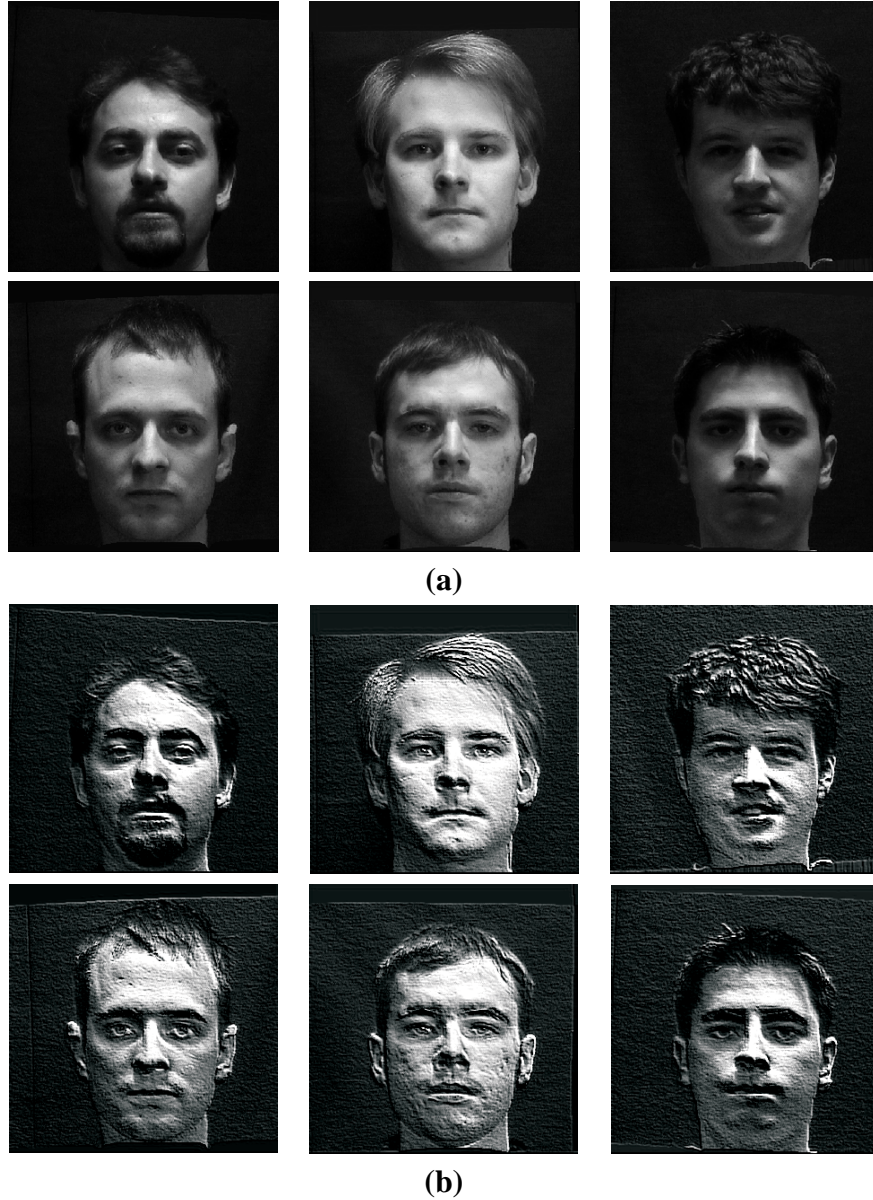


Figure 47: Six out of 20 faces that were used for the convolution experiment. (a) Original images, warped so that facial features all line up with one another, (b) The result of convolving with a 3x3 filter on the images.

5.4 Multiple training sources experiment

In this experiment, we wanted to see how much the error would reduce when adding additional images to the training set. The training dataset is comprised of facial images before and after convolution (Figure 47) with a 3x3 kernel (Figure 50). Four different images of faces were used

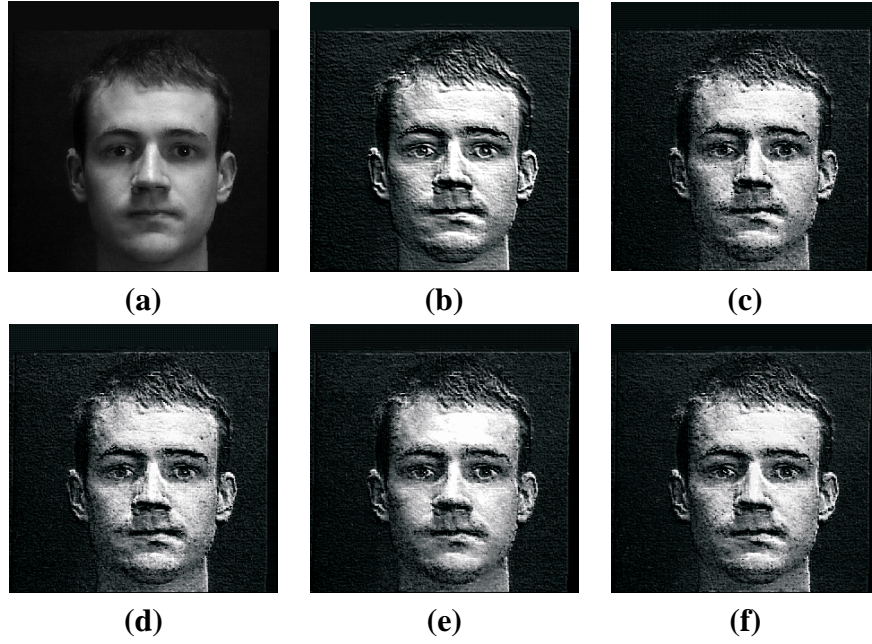


Figure 48: Multiple training sources experiment: (a) test input, (b) ground truth, (c) 1 image in training set, (d) 7 images in training set, (e) 13 images in training set, (f) 19 images in training set.

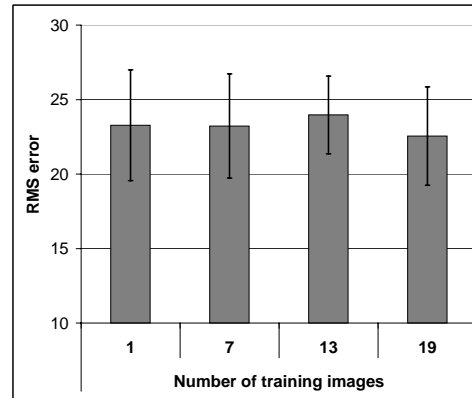


Figure 49: RMS error versus ground truth for different numbers of randomly picked training images.

as test input with varying amounts of training data, not including each face used as test input, to compute the mean and standard deviation of the error. Figure 48 shows one of the input images used as well as the outputs generated by the system when different numbers of training images are used. It is hard to see any difference in the outputs, however the error is indeed going down for the most part as can be seen in Figure 49. The mean error increases slightly when 13 training images are used. This is probably due to some patches in the additional training images being

$$\begin{bmatrix} -1 & -1 & -2 \\ 1 & 1 & -1 \\ 2 & 1 & 1 \end{bmatrix}$$

Figure 50: Kernel used for convolution experiments.

better matches to patches in the input images in terms of improving spatial coherence. That is, adding more training images will result in increased spatial coherence, but may also increase the error if a specific filter is the approximation goal, as it does in this case.

5.5 Large number of training sources experiments

When using example based processing algorithms to emulate painting styles, it can be difficult to choose a good training example pair. There may be an example pair that resemble the input image in color/contrast, but whose colors are very different or which have brush strokes that are unappealing. While color normalization can be used to allow for any training image to be used for a particular input image, a better output may result from choosing an image that is closer in appearance to begin with.

In this experiment, we address this problem by using an extremely large dataset of training examples of a single painting style. Using more training data of a single style should produce results that are truer to the style than choosing a random example training image in that style. We chose to train on the impressionist painting style since this is a style that is more localized than others. Indeed, this is why it was one of the first emulated by non-photorealistic algorithms. Since it is a more localized style, using a couple hundred training images should be sufficient to get a good coverage of the feature space for this style. For non-localized styles, thousands of training examples may be required, which our algorithm implementation cannot handle for reasons discussed in Chapter 6.

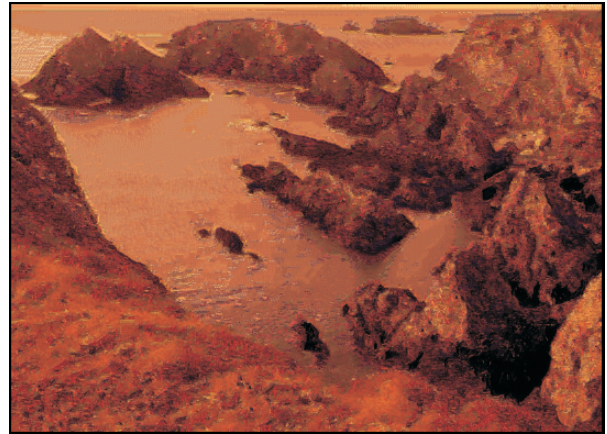
We used over 260 paintings by Claude Monet in the impressionist style as training data for the experiments in this section. These paintings were scanned from various art books [29, 59, 35, 53] at 400dpi and were not compressed. The paintings used for the experiments in other sections came from the web and had visible compression artifacts, which we wanted to decorrelate. Our goal is

to see what the differences are between using random single training images and many at one time.

In this section, we do not compare our results with other algorithms because others cannot be extended to handle such large training sets or would handle them poorly. Image analogies requires nearest neighbor calculations with the entire training set in scanline order, which is computationally impractical. Conversely, our algorithm performs nearest neighbor calculations incrementally over the patches in the image since there is no scanline dependence. Scanline independence allows the algorithm to spread the incremental matching over multiple processors as well, for additional gains in computational speed. Image quilting is patch based like our algorithm, so it could be extended. However, the results may be incoherent or have repetitive patches since dynamic programming may not work well if patches come from different locations in different images.



(a)



(b)



(c)



(d)

Figure 51: Varying the amount of data in the training set. (a) Test input photograph, (b) 50 training pairs, (c) 100 training pairs, (d) Full training set.

5.5.1 Differing amounts of training data

We used the same test input photograph with differing numbers of randomly selected training data. The results of this experiment are in Figure 51. We found that there is a clear correlation between the training set and the colors that are used in the output. In addition, using more training pairs results in synthesized outputs that are more highly textured.

5.5.2 Test input: photographs

To determine how well the Monet training set generalized, we used various different photographs as test input. Some of the photographs and their corresponding synthesized outputs appear in Figures 52 and 53. In these experiments, the full training set was used and only the test input was varied. The Monet training set generalizes well across photographs with very different colors and gradients. Each synthesized output image contains different high-frequency detail not present in the others, demonstrating that different training images are sampled according to the test input as expected.

5.5.3 Test input: anisotropically blurred

The training inputs in the dataset were anisotropically blurred to emulate the ‘photograph’ version of the corresponding output (original paintings). We performed two sets of experiments where the test input was anisotropically blurred; one for photographs, and one for paintings that were left out of the training set. Figure 54 shows how the synthesized outputs changed when input photographs were used as test input with and without blurring. In general, the outputs do not have as much high-frequency detail since most was removed by blurring the input. The lack of high-frequency detail results in a more painterly appearance, at the expense of some artifacts being introduced due to the addition of regions with low contrast or missing texture because of blurring.

We next performed experiments where one of the paintings was left out of the training set and an anisotropically blurred version of it was used as the test input. We chose four paintings at random and left them out of the training set. For each experiment, we tried using the blurred painting as test input as well as a horizontally flipped version of it. The flipped version was tried to determine the dependency of the outputs on the ordering of pixels in neighborhoods. In addition,

we experimented with varying the amount of training data. We tried using the full dataset minus the painting chosen for the experiment, as well as a single random painting from the dataset to see how the blurring and flipping would affect the outputs. For the single training painting experiments, the same painting was used for all of the experiments.

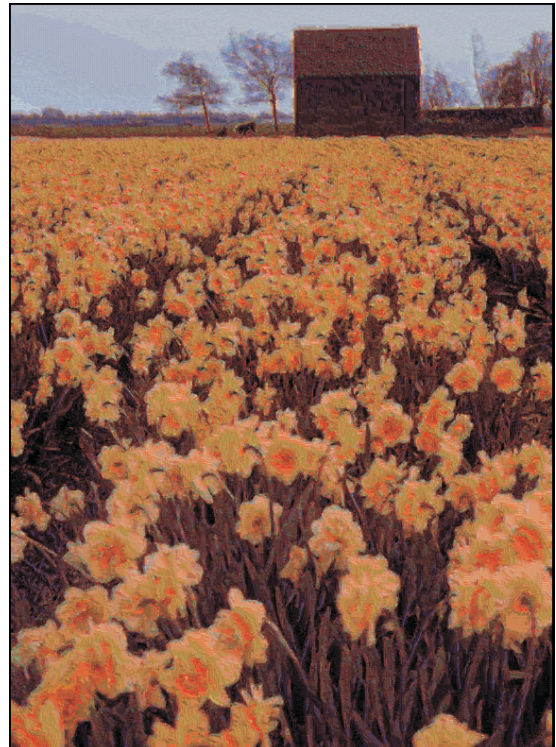
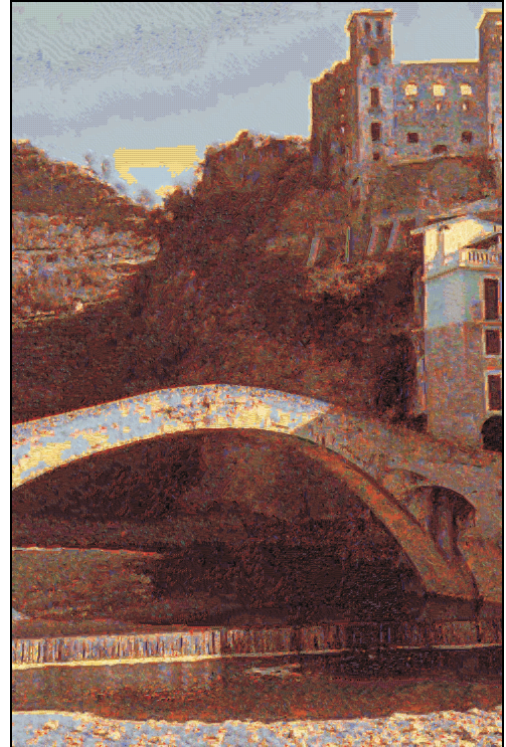
Figures 55-58 show the results of these experiments for four different paintings. We also show close up regions in color and in grayscale for comparison. The results from the horizontally flipped test input versions are flipped back for better comparison. In these results, horizontally flipping the training input does not affect the synthesized output much. Using more training data results in outputs that are closer to the ground truth, as expected.



(a)

(b)

Figure 52: (a) Input photograph, (b) output using full training set.



(a)

(b)

Figure 53: (a) Input photograph, (b) output using full training set.



(a)

(b)

Figure 54: Outputs from altered training input photograph: (a) original output, (b) output from anisotropically blurred photograph as test input.

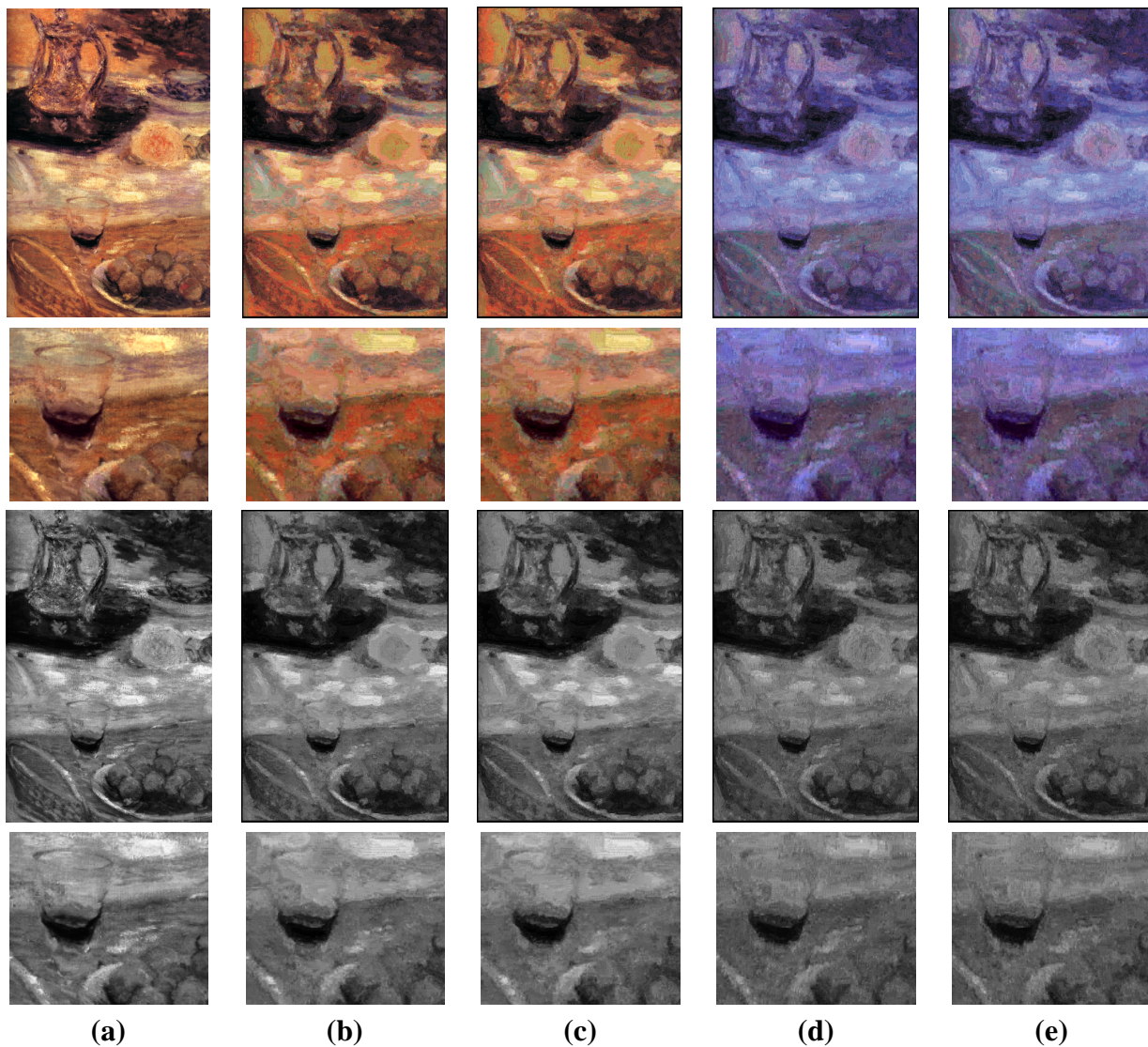


Figure 55: Leave one out experiment: (a) Original painting, (b) output using full training set, (c) output using horizontally flipped input and full training set, (d) output using single randomly picked training pair, (e) output using horizontally flipped input and single randomly picked training pair.

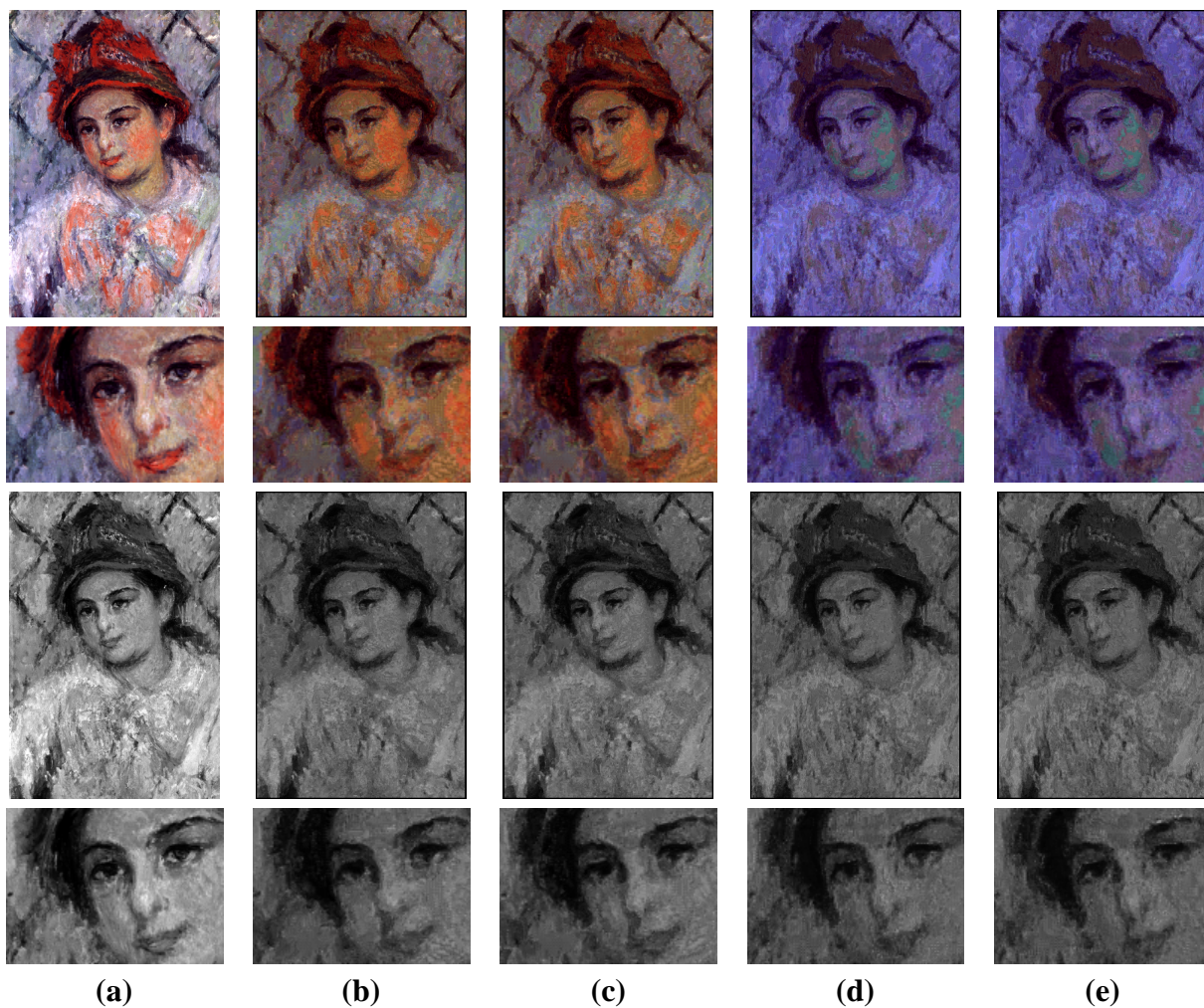


Figure 56: Leave one out experiment: (a) Original painting, (b) output using full training set, (c) output using horizontally flipped input and full training set, (d) output using single randomly picked training pair, (e) output using horizontally flipped input and single randomly picked training pair.

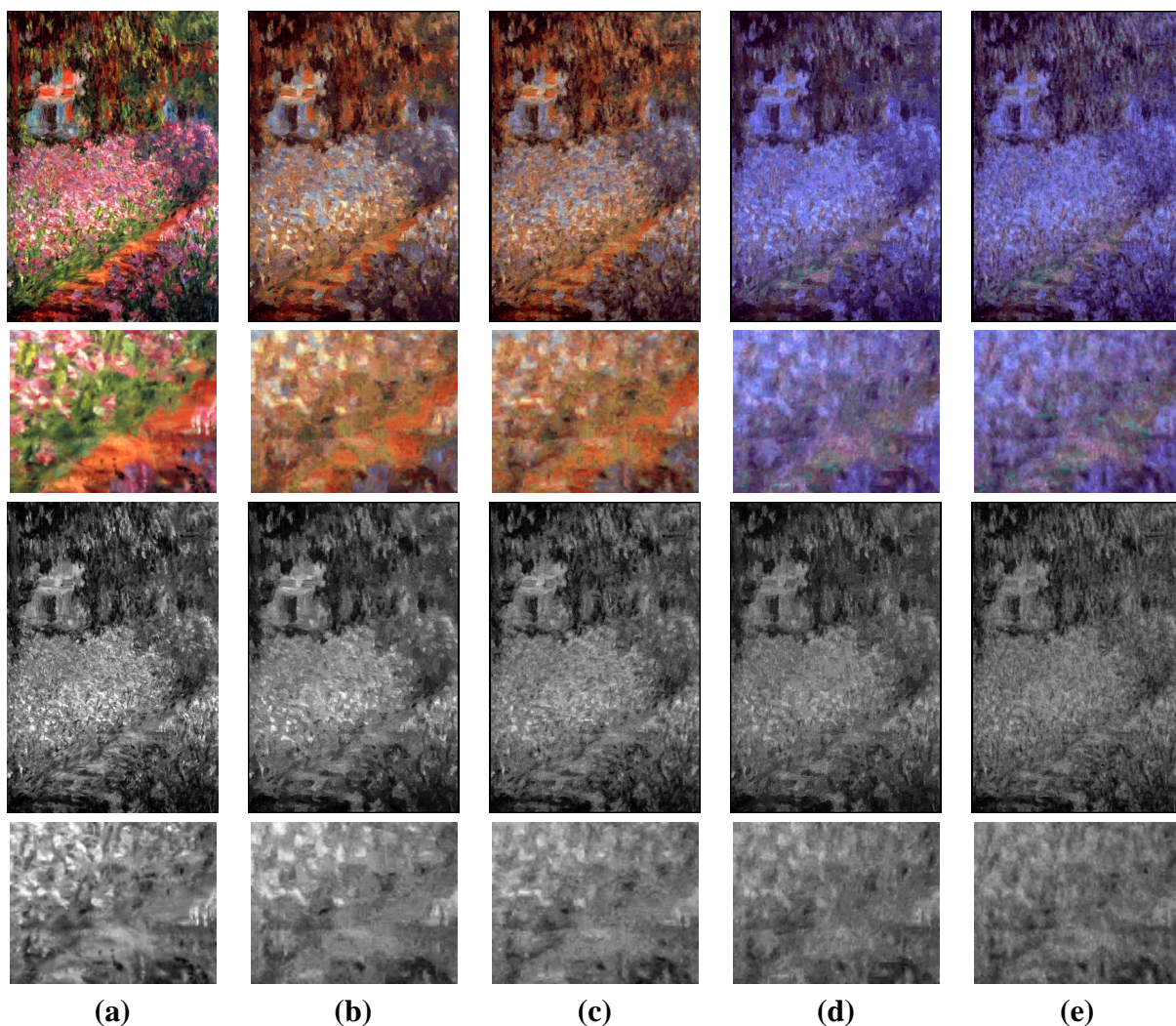


Figure 57: Leave one out experiment: (a) Original painting, (b) output using full training set, (c) output using horizontally flipped input and full training set, (d) output using single randomly picked training pair, (e) output using horizontally flipped input and single randomly picked training pair.

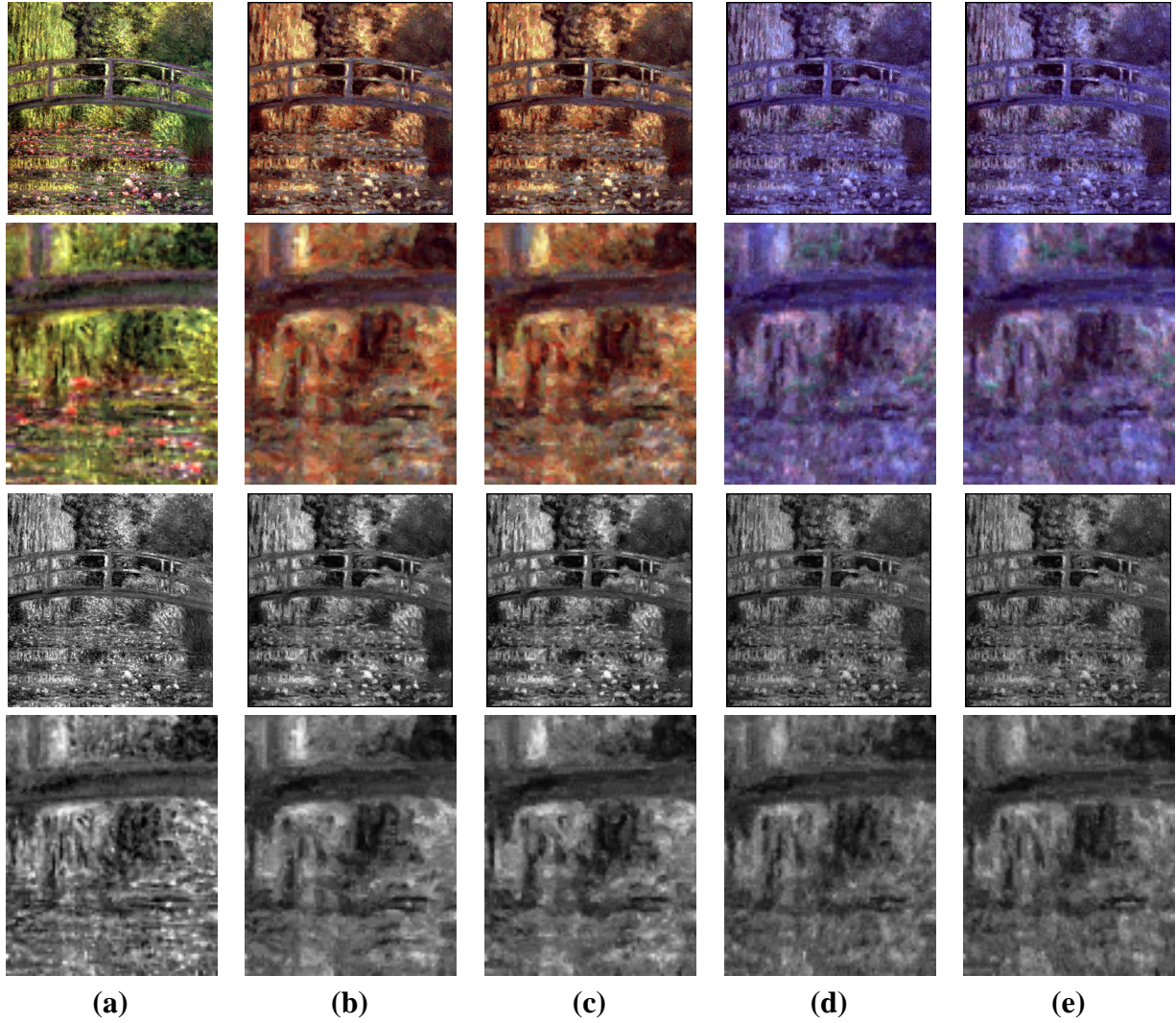


Figure 58: Leave one out experiment: (a) Original painting, (b) output using full training set, (c) output using horizontally flipped input and full training set, (d) output using single randomly picked training pair, (e) output using horizontally flipped input and single randomly picked training pair.

CHAPTER 6

SYSTEMS ISSUES

One of the biggest challenges in performing example based image processing is simply managing the large amount of data. In Chapter 5, we demonstrated that adding more training examples can improve the quality of the results. However, there are several prices to pay for adding significant amounts of training data. One is that memory consumption will be very large. Another is that the run-time of the algorithm will be increased significantly. In this chapter, we discuss these issues in-depth and present several techniques to allow example based processing algorithms to be able to use extremely large datasets and run several orders of magnitude faster that would be possible otherwise. Images are used as the data type to illustrate these points, however this discussion applies to other data types as well.

6.1 Memory usage

Training data is usually very high-dimensional and hard to store completely in memory. For example, the training vectors used in Chapter 4 are 294 dimensional floating point vectors. This means that each vector requires 1176 bytes to store. As of this writing, memory storage is limited to 4GB on personal computers. Clever programming can allow for the vectors to be stored partly on disk as only half of the vectors need to be in memory for nearest neighbor (Section 6.2) calculations. Allowing for that and for a negligible amount of memory usage for the rest of the program, over 2 million training vectors can be stored in main memory. While this is a large amount of training data, for some applications this may not be enough. For algorithms that are purely dependent on training data, any limitation on the amount of training data that can be used is problematic. In this section, we discuss various techniques to bring the memory size of the training data down significantly so that additional experimentation with even larger datasets than have been discussed in this

thesis can be attempted.

6.1.1 Clustering

Training vectors can be clustered using techniques from machine learning [11] and/or vector quantization [16]. Vector quantization has already proven to be useful in reducing the number of training vectors for the texture synthesis problem [65], so it should be useful for learning more general example based processing. One caveat with any clustering or quantization technique however is that it is possible to get rid of too much of the training data in the interest of bringing memory consumption down. With extremely large amounts of training data, this becomes less of a problem, and for most learning problems, clear clusters do become apparent. For the work presented in this thesis, clustering was not used since we wanted to avoid having our results be correlated to a particular data reduction technique. However, for any practical implementation of any of the algorithms described, clustering/quantization should prove helpful in reducing memory consumption.

6.1.2 Dimensionality reduction

Dimensionality reduction is another commonly used technique in machine learning to make training data more manageable. However, most techniques, such as principal component analysis (PCA) are rooted in performing a singular value decomposition (SVD) of the data. In these techniques, the training vectors are used as columns in a matrix that is then decomposed. Problems arise when the matrix becomes too big to store even in memory. There have been many techniques proposed to compute an SVD incrementally or to approximate it, but they usually suffer from problems [7, 37, 74]. These techniques usually can either only approximate the eigenvectors or can result in eigenvectors that are not completely orthogonal, and thus non-optimal spans of the training data feature space.

Recently, Brand [4] has proposed a new technique for computing an SVD incrementally resulting in orthogonal eigenvectors that are equivalent to those that would be computed using a traditional SVD algorithm. We briefly describe the algorithm, but refer readers to the paper itself for additional important details regarding numerical stability and eigenvector orthogonality preservation. We begin with a matrix M whose columns are some training vectors. We then have a

new matrix C , whose columns are also training vectors. We would like to compute the singular value decomposition of the combined matrix: $U \text{diag}(s) V^T \xleftarrow{\text{svd}} [MC]$, where U and V are orthogonal matrices whose columns give a linear basis for the rows and columns of $[MC]$ respectively, s are the eigenvalues of the matrix, and $\xleftarrow{\text{svd}} [MC]$ means an SVD of matrix $[MC]$. However, if the concatenated matrix $[MC]$ is too large to compute an SVD directly, we can first compute $U_1 \text{diag}(s_1) V_1^T \xleftarrow{\text{svd}} [M]$ and then update U_1 and V_1 to represent the vectors in C as well. First, we project C onto U_1 , $L = U_1^T C$. Then we compute the projection, K , of C onto the subspace orthogonal to U_1 . We can do this by first computing $H = C - U_1 L$ and then $JK \xleftarrow{\text{qr}} H$, where $\xleftarrow{\text{qr}} H$ means a QR-decomposition of H . The following identity then holds true:

$$[U_1 J] \begin{bmatrix} \text{diag}(s_1) & L \\ 0 & K \end{bmatrix} \begin{bmatrix} V_1 & 0 \\ 0 & I \end{bmatrix}^T = [MC] \quad (14)$$

Brand's observation is that since the left and right matrices (to the left of the equal sign) are orthogonal and unitary, all that needs to be done to update the SVD is to diagonalize the middle matrix. The steps to diagonalize the middle matrix are then:

$$U_2 \text{diag}(s_2) V_2^T \xleftarrow{\text{svd}} \begin{bmatrix} \text{diag}(s_1) & L \\ 0 & K \end{bmatrix} \quad (15)$$

$$U = [U_1 J] U_2; \quad s = s_2; \quad V = \begin{bmatrix} V_1 & 0 \\ 0 & I \end{bmatrix} V_2 \quad (16)$$

$$(17)$$

Resulting in the updated SVD:

$$U \text{diag}(s) V^T = [U_1 \text{diag}(s_1) V_1^T C] = [MC] \quad (18)$$

This approach allows for the computation of the SVD of arbitrarily large matrices. We used this technique to compute the eigenvectors of training data for various example based image processing datasets using the algorithm described in Chapter 4. We then used the computed eigenvectors to reduce the dimensionality of the training and input vectors using PCA while matching to speed up the processing and lower memory usage. Processing was sped up by about a minute per belief

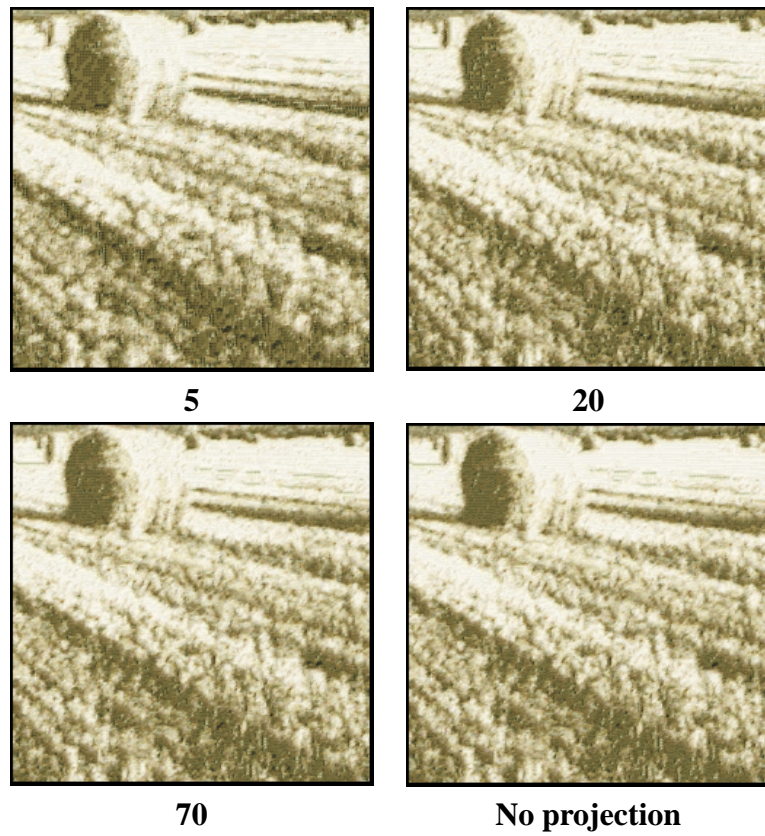


Figure 59: Effects of dimensionality reduction on the hay texture transfer dataset, with varying numbers of eigenvectors compared to not using SVD at all.

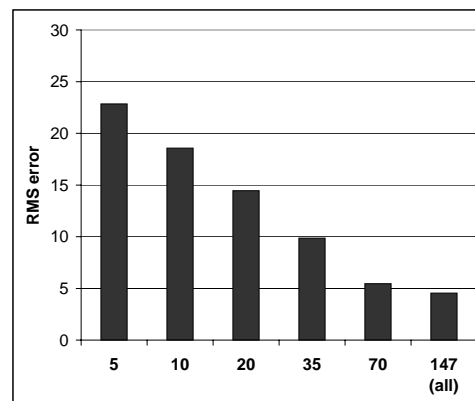


Figure 60: RMS error of varying numbers of eigenvectors versus not using SVD for hay texture transfer dataset.

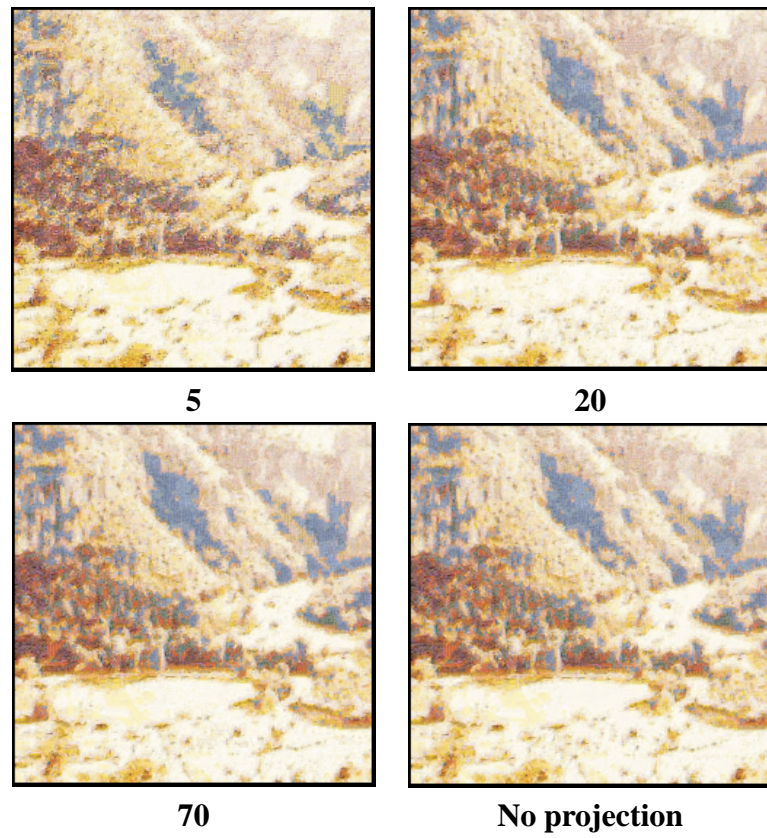


Figure 61: Effects of dimensionality reduction on non-photorealistic impressionism dataset, with varying numbers of eigenvectors compared to not using SVD at all.

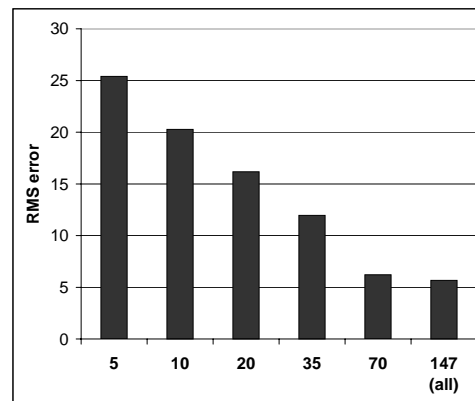


Figure 62: RMS error of varying numbers of eigenvectors versus not using SVD for non-photorealistic impressionism dataset.

propagation iteration, for typical total savings of about 5-10 minutes for small images, and several hours for larger images. We would not have been able to apply PCA to these datasets without using this incremental technique since the training data matrix would be too large to fit in memory. For each experiment, we used several different numbers of eigenvectors ranging from 5 to all. For each number of eigenvectors retained, we ran the algorithm and computed the rms error against the original output which did not use PCA.

Texture transfer, hay dataset: We used the same training data from Figure 24; Figure 59 shows the results and Figure 60 shows the rms error plots. The error is highest when using only 5 eigenvectors as expected, yet the output is still spatially coherent. There is a noticeable lack of high-frequency detail when using 5 eigenvectors, resulting in the large error. The rest of the results do not appear to have differences with ground truth, yet there is some error.

Impressionism, valley dataset: We used the same training data from Figure 28; Figure 61 shows the results and Figure 62 shows the rms error plots. When only 5 eigenvectors are used, the output looks somewhat more painterly. This is probably due to the descriptiveness of the first few eigenvectors; patches that would normally have large errors while matching now have smaller errors since their projections in the subspace are now closer. This causes a loss of detail in the output which could be desirable for non-photorealistic datasets. As more eigenvectors are used, the error quickly drops, stabilizing around 70.

Super-resolution, raccoon dataset: We used the same training data from Figure 25; Figure 63 shows the results and Figure 64 shows the rms error plots. The error quickly drops once we use 10 eigenvectors, unlike the other experiments. This is because the variability of the training input patches is low since they do not contain high-frequency content. As a result, super-resolution is the problem domain that benefits most from PCA-based dimensionality reduction.

Interestingly, in all of the experiments the error with half the eigenvectors is close to that when they are all retained. Reducing the dimensionality of the training vectors by half also speeds up the matching. In these experiments the difference in processing time visibly dropped. However, since the implementation was not completely optimized, it is hard to say how much of a speedup

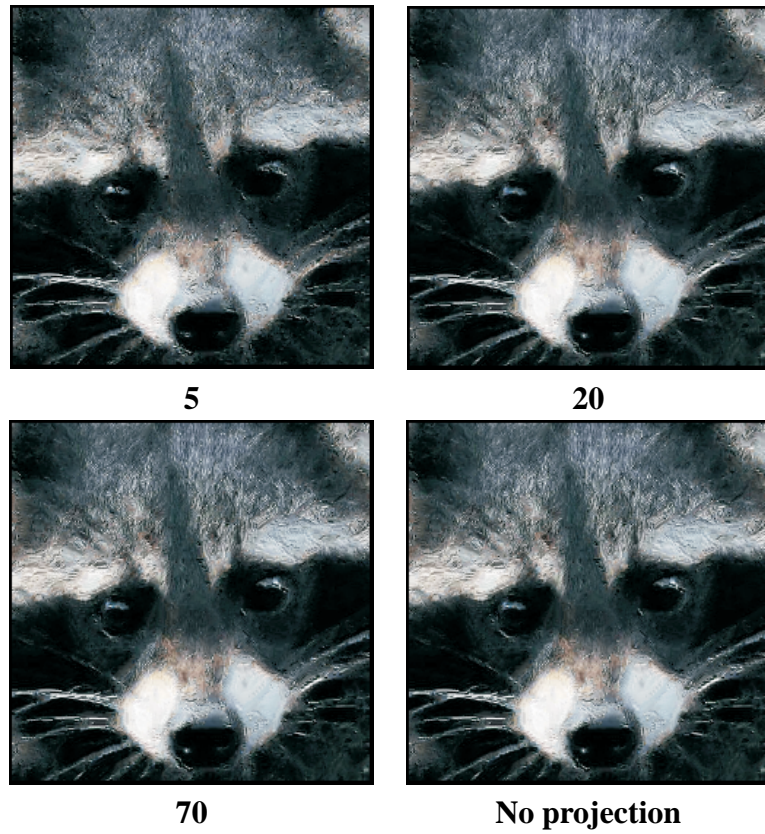


Figure 63: Effects of dimensionality reduction on super-resolution dataset, with varying numbers of eigenvectors compared to not using SVD at all.

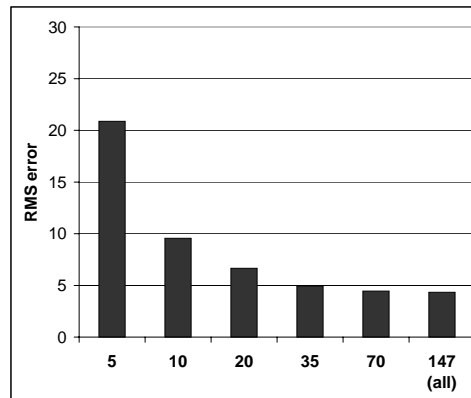


Figure 64: RMS error of varying numbers of eigenvectors versus not using SVD for super-resolution dataset.

is ultimately possible using this approach. With further optimizations, dimensionality reduction using PCA should have major effects on the running time of example based processing algorithms.

6.2 Approximate nearest neighbors

Most example based processing algorithms to date require large numbers of nearest neighbor calculations between input and training vectors. The most common approach to performing these calculations is to compute the approximate nearest neighbors instead using kd-trees [43]. This approach works well for most problems, however, the majority of publicly available software libraries to compute approximate nearest neighbors assume that all training data can be stored in memory at one time [42]. In all of the algorithms presented in this thesis, we have been able to fit all training vectors in memory to make use of these libraries, but have had to limit the amount of training data used as a result. The approaches previously described to reduce memory consumption of training data allow for more training vectors to be used, however there is ultimately a memory limit.

There has been much research in the systems field on external memory algorithms, which are algorithms whose goal is to manage and search large datasets. Applications where these algorithms are commonly used include geographic information systems (GIS) range search and data indexing of astronomical data. An excellent survey of external memory algorithms is Vitter [64]. Here, we limit our focus to one external memory algorithm/data structure, R-trees [18], since they were created specifically for nearest neighbor computations for use in computer aided design (CAD) and geo-data applications. For these problem domains, kd-trees are not useful since they do not take paging of memory into account. For this reason, even a disk based implementation of kd-trees would not be useful for these problems. R-trees have not, to our knowledge, been used for example based processing, but we believe that they could be used in this domain for disk-based nearest neighbor searches of as much training data as can be fit on a disk.

R-trees are height-balanced trees with index records in leaf nodes pointing to spatial data objects and efficient node insert/delete/searching operations. While similar to B-trees, R-trees provide guarantees on minimized paging, which makes them very attractive for disk based point searches as we would like to perform. Each node corresponds to a page in memory or on disk, and

the search algorithm used to traverse the tree is designed to minimize the number of nodes visited. Data objects in an R-tree are comprised of n -dimensional tuples forming a bounding box of the data. Since the data are bounding boxes, a leaf node may consist of several data objects, all in the same area of the space.

Although R-trees were designed for spatial objects such as plots of land, we believe they can be extended to index vectors as well. To use R-trees for approximate nearest neighbor computations of n -dimensional vectors, $2n$ -dimensional data object tuples would be created for each training vector representing a bounding box of width ϵ around the point, where ϵ would be a small user-defined parameter. For each component v_i of training vector v with data tuple d in the R-tree, we set $d_{2i} = v_i - \epsilon$ and $d_{2i+1} = v_i + \epsilon$. Using R-trees in this manner could allow for example based processing algorithms to be extended to attempt to perform learning of processing that would be impossible without large datasets.

6.3 Patch size variation

Since the training vectors are dependent on patch sizes, another method of controlling memory usage is to change the patch sizes themselves. Patch sizes play a major role in texture synthesis algorithms as they usually have to be at least the size of the smallest texture element. For non-texture synthesis example based processing, choosing good patch sizes is harder and is dependent on the training data and input used with the algorithm. The algorithm's implementation details are also important when choosing patch sizes as it may be more advantageous to have many small dimensional training vectors compared to fewer large dimensional vectors.

We experimented with different patch sizes using the algorithm described in Chapter 4. We used the dataset from Figure 28 and tried different patch sizes for the observation and hidden variables. Figure 65 shows the results of this experiment. The size of the observation variable's patch size is always set to be larger than the hidden variable's. This is because the observation variable dominates belief propagation's inference at each pixel and determines which samples to try at each pixel from the training data. In addition, we varied the overlap size (used to help determine spatial coherence in the algorithm) since for larger patch sizes this may also vary.

When using large patch sizes (11x11) for either the hidden or observation variables and small

overlaps, we found that the outputs were not very spatially coherent. This is because only a small number of pixels in neighboring patches need to match well compared to the patch sizes themselves. However, increasing the overlap size for large patches too much results in output that is very blurry. This is due to the overlaps between the large patches being averaged when creating the output image. This could be alleviated by computing a graph cut to merge the two patches instead as in [33]. This would allow the algorithm to choose two compatible neighboring patches and still retain the high-frequency detail in them when synthesizing the output image.

Using the smallest (3x3) patch size for both the observation and hidden variable at the same time results in an output that matches the input image too closely. This is unattractive for a non-photorealistic application, but is useful when a filter is to be emulated exactly since it encourages the patches from the training input and input image to match very closely. For such an application, setting the patch sizes to be the size of the convolution kernel would yield the best results.

The remaining patch size combinations of 9x9 and 7x7 with themselves and each other yield outputs that vary in blurriness and approach the appearance of the combination of 7x7 for the observation variable and 3x3 for the hidden variable, which is what was used for all of the results in Chapter 4. For some types of non-photorealistic datasets, combinations of 9x9 and 7x7 with each other could be useful to create outputs that look painterly to different degrees.

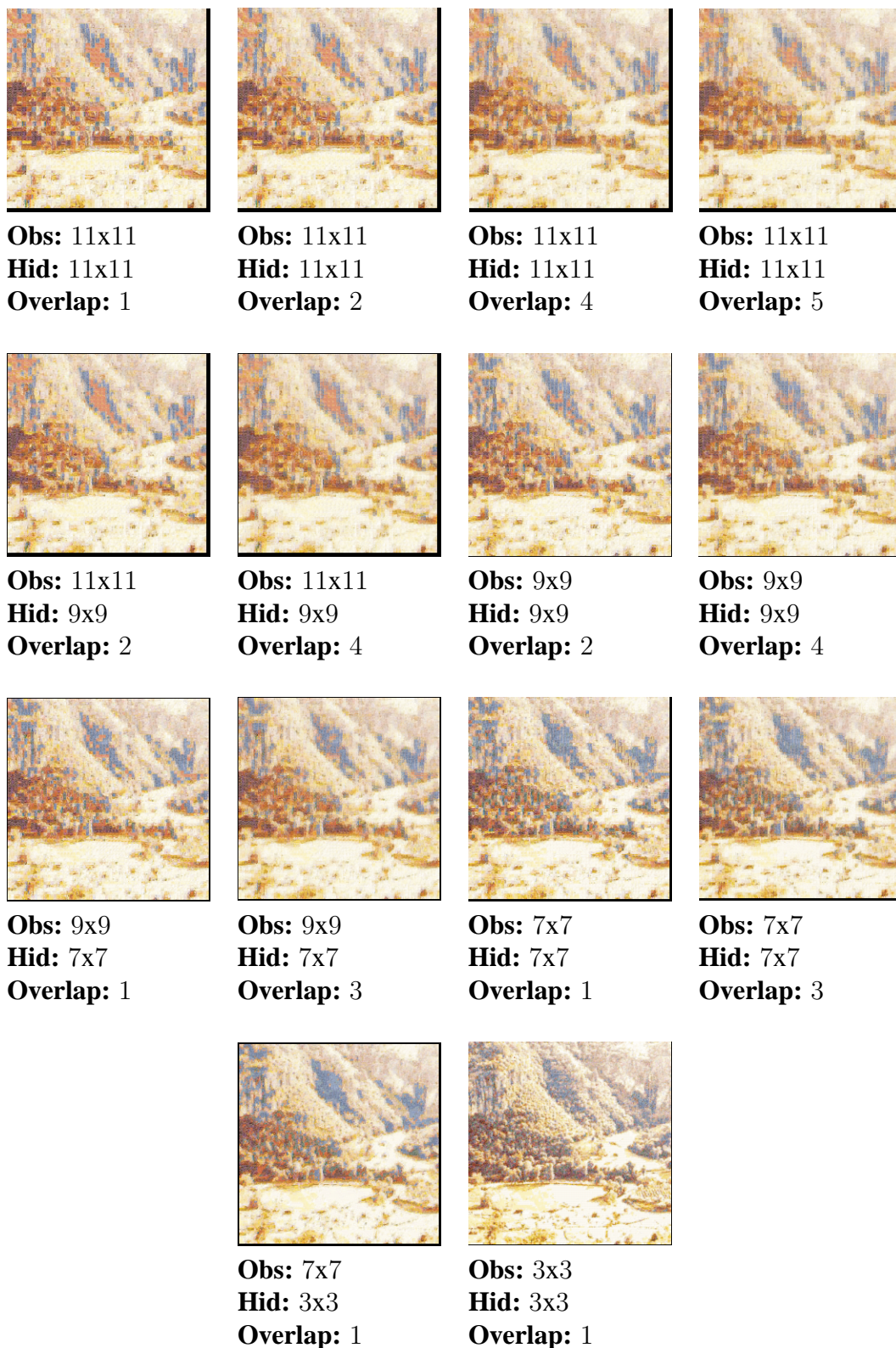


Figure 65: Varying pixel patch sizes and overlap. *Obs*: observation variable patch size, *Hid*: hidden variable patch size, *Overlap*: number of pixels overlapped by neighboring node patches.

CHAPTER 7

3D SURFACE NORMAL SYNTHESIS BY EXAMPLE

In the previous chapters, we have discussed example based image processing. We showed how image based techniques could be extended to emulate the rendering of 3D models in Section 3.3.3 by encoding normal information as color. However, although 3D data was matched, the result was still an image. In this chapter, we show how these techniques can be combined with rendering and computer vision techniques to create synthesized output on a 3D surface instead of an image.

Bumpmaps are very useful to add detail to models or to improve their photorealism. They are usually created by artists, but could be difficult and time-consuming to create by hand for data that is very detailed. In this regard, this is a problem domain that benefits greatly from example based techniques. This chapter presents a method to synthesize 3D normals for use in bumpmapping by example. It turns out that a normal field can be modeled as an MRF as nearby locations on the surface are highly correlated, and the problem reduces to texture synthesis. We use the synthesized normal fields automatically learned from captured skin molds to create and render realistic skin microstructure. The microstructure that is created shares the same appearance and features as the example, yet does not seem to repeat.

Photorealistic skin is very important in a film or a video game and is hard to render for various reasons. One reason is that humans are experts when it comes to skin realism. Another is that skin has a complicated BRDF, one that depends on many factors such as pigmentation, oiliness and dryness. Skin also possesses very fine and complicated detail. The appearance of fine scale skin structure varies smoothly across the face yet each region of skin on the human body has a very distinct appearance unique to its location. For example, forehead fine scale skin structure is distinctly different from nose fine scale structure in Figure 68. Skin that lacks fine scale structure looks unrealistic since fine scale structure is such an integral part of its appearance. In this chapter,

we capture fine scale structure samples, build models of fine scale structure production, and then render this detail using a measured skin BRDF. Our results show that the addition of fine scale structure adds significantly to photorealism of facial models.

We address several problems: (1) capture of fine scale skin structure from actual skin samples, (2) approximation of the stochastic processes that generate fine scale skin structure patterns, and (3) rendering of photorealistic skin in real-time. To capture the fine scale skin structure, we employ a material¹ used in cosmetological science to create skin imprints and measure pore size. In that field, the imprints are laser-scanned to create a range map. The resulting range map is very noisy because the samples are very small (about the size of a nickel), and hence, unsuitable for rendering, but not for the type of analysis they do. We use shape from shading [28] to get a much more accurate range map that is significantly less noisy than laser-scanning can provide. We experimented with various scanners and scanning services and could not find a method to compute a range map that would be adequate for rendering.

Once we produce range maps for imprints from different areas of the face, we approximate the stochastic processes that “generated” each sample. We encode the range maps as images so that this problem reduces to texture synthesis. Since the range map for each skin sample comes from a different area of the face, and hence, a different stochastic process, separate instances of texture synthesis are started on several points on the face. Fine scale structure is “grown” in 3D until full coverage is attained. We use the result as a normal map and use it to do per-pixel bump mapping. We also approximate the BRDF of the skin, so the result approaches photorealism.

Traditionally, when fine scale structure is rendered, it is either drawn/produced by artists by hand, or is rendered by using layers of specialized shaders. While realistic fine scale structure can be attained by these methods, the creation of additional fine scale structure is time intensive since an artist has to manually create new fine scale structure for each desired face. If specialized shaders are used, it might not be possible to render in real-time. In both cases, some underlying properties of the stochastic process are lost since the fine scale structure might only be visually realistic at certain distances/orientations. Since we compute models of the stochastic processes that generate the skin directly from actual skin data, we can render skin that looks realistic in different lighting conditions and at different scales, without any recomputation/resynthesis.

¹Silflo, manufactured by Flexico

There is much work that has been done in the area of realistic facial rendering, however, very little work has been published on capturing or synthesizing fine scale skin structure. The closest work to ours is that of Wu *et al.* [69]. In that work, fine scale skin structure is dependent on its underlying geometry. A Delaunay triangulation is done on each desired skin region to generate triangles in texture space. The edges of the triangles are then raised/lowered depending on user parameters to create a height-field which is then used as a bump map. The resulting bump map does not look very realistic because some Delaunay triangulations will not yield realistic fine scale structure. Also, a user would have to spend a lot of time selecting proper basis functions, rejecting bad triangulations, and ensuring proper fine scale structure blending from one region of the face to another in order to get fine scale structure that approaches realism. In contrast, our method implicitly takes the stochastic properties and variance across the face into account, so we achieve more realistic results with minimal user input. Nahas *et al.* [44] captured skin detail with a laser range scanner, but at a low resolution (256x256 samples). However, this resolution is too low to capture the rich detail we are able to capture using skin molds and photometric stereo.

Recent work done to capture skin reflectance properties is also worth noting. Hanrahan and Krueger [20] simulated sub-surface scattering using Monte Carlo simulations to render skin. Marschner *et al.* have done work on capturing BRDFs from images that come from a calibrated camera with subjects wearing a pattern [40, 41]. Debevec *et al.* [9] use a more complex system to measure skin reflectance as well as surface normals and are also able to incorporate ambient lighting from different environments. All of these approaches yield high-quality renderings, however, none incorporate fine scale skin structure. Recent facial animation work has also resulted in realistic results [17, 46]. However, since the face texture comes from images that are texture blended, high-frequency detail such as fine scale structure is lost. We use measured skin reflectance models in conjunction with fine scale structure for added visual realism.

7.1 Normal Map Capture

We use shape from shading to capture convincing fine scale skin structure. First we make several samples of real skin texture (Figure 66) from various regions of the face using a silicone mold material. Then we apply shape from shading [28] to the silicone mold to recover the normals

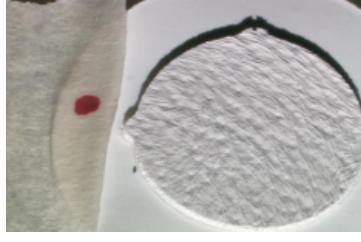


Figure 66: Silicone mold of skin (about the size of a nickel).

of the surface. Our technique is similar to the work done by others [57, 73, 51, 52] except that we do not require any camera calibration or structured light. In addition, we can deal with non-Lambertian surfaces while using all of the pixels from each captured photograph of each sample. Throwing away pixels due to specular reflections would yield suboptimal maps due to our very simple experimental setup.

The shape from shading algorithm requires the BRDF of the surface being measured to be Lambertian. We approximate this by placing polarizing filters in front of the light source and the camera and rotating them with respect to each other to eliminate all specular reflections. The remaining reflection is approximately Lambertian. The illuminator we used to capture all our skin normal data, consisting of a halogen light mounted on a lazy susan, was built for \$40 using materials available at any hardware store.

We take 8 photographs of the silicone mold illuminated by a point light source. The mold remains stationary, but the light position is changed in every frame by rotating the light on the lazy susan. The resulting set of images yields a set of simultaneous linear equations at each (x, y) pixel location that can be solved for the surface normal at that pixel:

$$\rho_{x,y} n_{x,y} \cdot L_i = I_{x,y} \quad (19)$$

$\rho_{x,y}$ is the diffuse albedo of the surface at pixel (x, y) , $n_{x,y}$ is the normal at the pixel, and $I_{x,y}$ is the measured intensity at the pixel. L_i is the light vector for image i . $\rho_{x,y}$ is nearly constant across the silicone mold so it needs to be measured only once per image rather than at each pixel. The light intensity and position with respect to the surface is also approximately uniform over the sample so it also needs to be measured only once. This set of equations is solved at each pixel using least

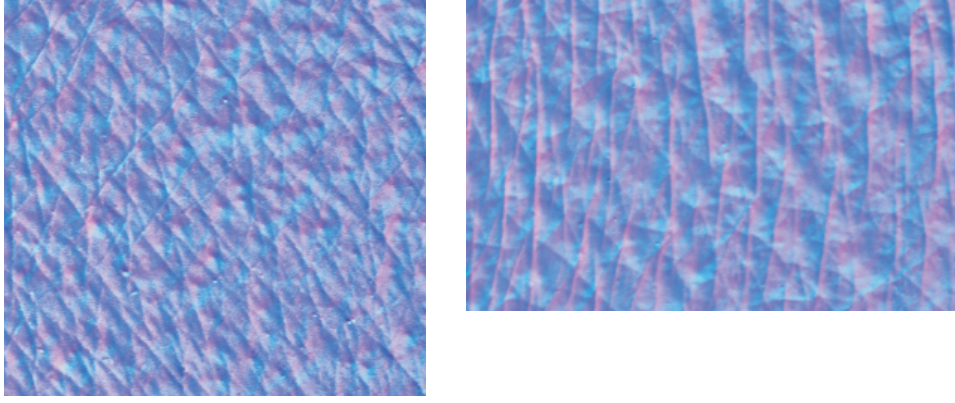


Figure 67: *Left*: Cheek normal map, *Right*: Edge of forehead normal map.

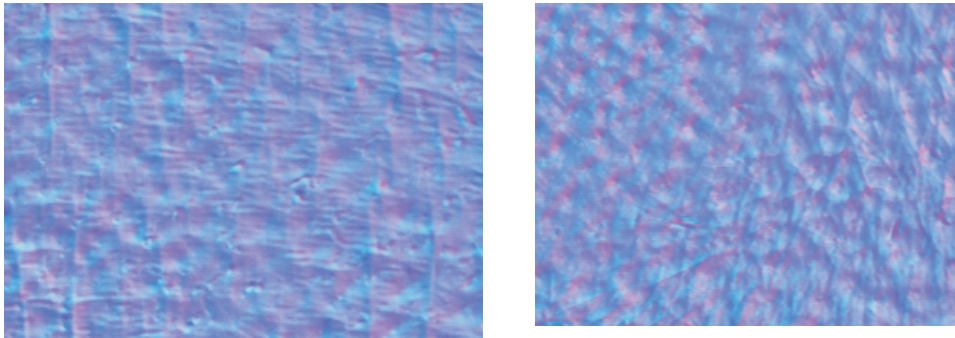


Figure 68: *Left*: Middle of forehead normal map, *Right*: Nose normal map.

squares to give a normal map.

The silicone molds are not always flat, especially those taken in the nose region. This introduces a low frequency change in the surface normals that is inappropriate for bump mapping, where only the high frequency details are of interest and normals need to be perpendicular to the plane. To eliminate the low frequency rotational component we compute the average normal over a 50×50 pixel block centered at each pixel, compute the rotation that maps this normal back into the perpendicular to the normal map plane, and then apply this rotation to the normal computed from shape from shading. The result of this operation is that each normal will be forced to be perpendicular to the surface plane. Low pass filtering alone would not guarantee that the normals would indeed lie in the plane orthogonal to the sample.

Figures 67 and 68 show some example skin textures that we have captured from different regions of the face. The normal maps are encoded versions of the normals where the X, Y, Z axes map to the R, G, B channels, respectively, and the range $\langle -1.0, 1.0 \rangle$ maps to $\langle 0, 255 \rangle$.

7.2 Skin Growing

The normal maps we capture in Section 7.1 are of high detail, but are quite small. The molds are about 21mm in diameter. One simple way to get complete facial coverage would be to take a large amount of these molds and then to interpolate between the gaps. Samples taken from curved areas would be distorted and the process would be very inconvenient.

Another approach involves taking a small number of samples and then learning how to produce more. Fine scale skin structure varies significantly across the face (Figures 67 and 68), which is one reason it is hard to model. Since fine scale skin structure can be thought of as a stochastic process, some ideas from texture synthesis can be applied. This does not address the issue of gaps between different skin types; since skin varies very significantly across the face, it is not adequate to interpolate or blur the edges. Doing so would result in high-frequency discontinuities, which would be very obvious when rendered. We present our solution to this problem in Section 7.2.2.

7.2.1 Normal map synthesis

Since skin can be thought of as a stochastic process, we can apply ideas from texture synthesis to create larger regions than what we have. To create larger patches of skin, we take our normal maps that we acquired through our capture process, and encode them as images as described in Section 7.1. These encoded normal maps can then be thought of as small pieces of texture that we would like to synthesize.

We use the technique by Wei and Levoy [65] to synthesize larger patches. The main idea in that work was to treat texture as a Markov random field. That is, as a stochastic process that is both local and stationary, meaning that a pixel's neighborhood characterizes the process and that this characterization is the same for all pixels in the texture.

First, a histogram equalized (to the input texture) color noise image is generated for the desired dimensions needed for the skin region. Then, for each pixel in the noise image in scanline order, we assign the color of the pixel from the input texture with the most similar causal neighborhood. This is done on each level of a Gaussian pyramid starting with the topmost to ensure that the pixel that is chosen from the input texture is similar at the current and previous frequency bands. Finally, the bottom level of the pyramid is copied into the desired facial region when synthesis is complete.

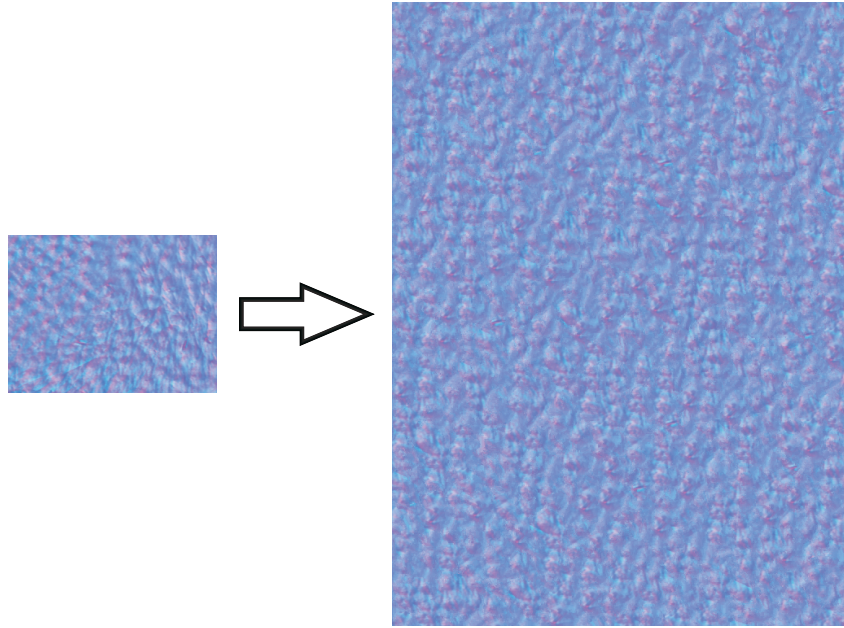


Figure 69: Initial normal map captured from mold and synthesized map.



Figure 70: (a) Each color represents a different type of fine scale skin structure to synthesize, (b) Multi-resolution splining along curves hides the boundaries between different patches.

Results of “growing” skin can be found in Figure 69. Recently, several texture synthesis papers [1, 71] have proposed copying from the source texture for some classes of texture. These approaches could yield even higher quality skin patches since larger contiguous regions from the input normal maps will exist in the synthesized skin.

7.2.2 Skin stitching

Since fine scale skin structure varies across the face, we start a separate instance of texture synthesis at each different region. The regions are defined by a user by selecting a few points (7) on the

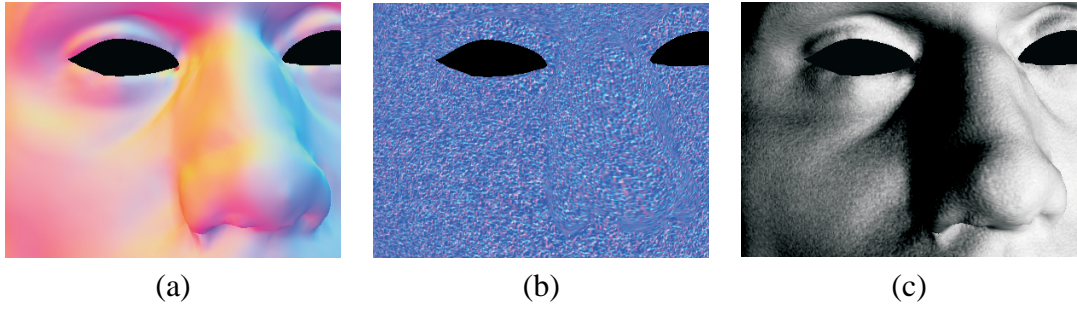


Figure 71: (a) Encoded and transformed light vector (interpolated), (b) Normal map treated as texture (contrast enhanced), (c) $(N \cdot L')$ computed per-pixel.

albedo map (Section 7.3.2), specifying the forehead, nose, and cheeks as in Figure 70(a). This step is the only manual step in our pipeline, and can be done quickly.

A separate instance of texture synthesis is started at each region. When the skin in each region has grown too large for its region, the boundaries between it and its neighbors are hidden by using a modified version of multi-resolution splines [6]. Instead of performing the multi-resolution splining along a straight line along the boundary, we compute a random curve through the boundary instead, seen in Figure 70(b). We have found this looks more like the natural transitions between one type of fine scale structure to another on human skin. One major reason for this is that the boundaries are hidden more effectively along the curve; curvaceous boundaries throw the human eye off and hide seams well [48]. We are able to generate fine scale structure for different faces rapidly with minimal user interaction using this approach.

7.3 Skin Rendering

To render skin realistically, we perform per-pixel bump mapping using the normal maps we have “grown”. Since our mesh topologies are low ($< 15k$ vertices), storing only the normals at vertex locations loses the majority of the detail that we gain through our capture step. Instead, we store our normals in a normal map and access them on a per-pixel basis as we render the mesh. In addition, we approximate the BRDF of skin to achieve greater realism. We will discuss that further in Section 7.3.2.

7.3.1 Per-pixel Bump mapping

Our bumpmapping technique [10] is similar to the work of others [62, 22]. The major difference between the techniques is that the per-pixel lighting calculations are done in texture space instead of object space. We perform the per-pixel lighting calculations in texture space since we would have to warp our normal map so that each normal in our map is in object space otherwise. Warping is undesirable because distortions can occur in the warped normal map, especially with lower resolution meshes. Also, if there are hard edges, the interpolated texture to object space transformation could be incorrect in areas between vertices.

Furthermore, if the mesh is animated, the normal map has to be re-warped each frame, which is very expensive. Working in texture space is more appealing since we only have to update the object to texture space transformation once per frame at each vertex and we have far fewer vertices than normal map pixels.

We compute transformations from object space into each vertex's texture space. At each vertex v with position $(v(x), v(y), v(z))$ and texture coordinates $(v(u), v(v))$ we form a coordinate system to perform per-pixel lighting in texture space. This coordinate system is comprised of $\delta U = (\delta u/\delta x, \delta u/\delta y, \delta u/\delta z)$, $\delta V = (\delta v/\delta x, \delta v/\delta y, \delta v/\delta z)$, and $N_{uv} = \delta U \times \delta V$, the texture space normal. The partials and N_{uv} are summed at each vertex, for all triangles, and then normalized. For a triangle T , with vertices v_0, v_1, v_2 the partials are:

$$\begin{aligned} \delta u/\delta x &= -B_x v_0(u)/(A_x v_0(x)), & \delta v/\delta x &= -C_x v_0(v)/(A_x v_0(x)) \\ \delta u/\delta y &= -B_y v_0(u)/(A_y v_0(y)), & \delta v/\delta y &= -C_y v_0(v)/(A_y v_0(y)) \\ \delta u/\delta z &= -B_z v_0(u)/(A_z v_0(z)), & \delta v/\delta z &= -C_z v_0(v)/(A_z v_0(z)) \end{aligned} \quad (20)$$

where $\langle A_x, B_x, C_x \rangle$, $\langle A_y, B_y, C_y \rangle$, $\langle A_z, B_z, C_z \rangle$ are the normals to the xuv , yuv , and zuv planes at vertex v , respectively, and are computed as:

$$\begin{aligned} v1_x &= (v1(x) - v0(x), v1(u) - v0(u), v1(v) - v0(v)) \\ v2_x &= (v2(x) - v0(x), v2(u) - v0(u), v2(v) - v0(v)) \\ \langle A_x, B_x, C_x \rangle &= v1_x \times v2_x \end{aligned} \quad (21)$$

$$\begin{aligned}
v1_y &= (v1(y) - v0(y), v1(u) - v0(u), v1(v) - v0(v)) \\
v2_y &= (v2(y) - v0(y), v2(u) - v0(u), v2(v) - v0(v)) \\
\langle A_y, B_y, C_y \rangle &= v1_y \times v2_y
\end{aligned} \tag{22}$$

$$\begin{aligned}
v1_z &= (v1(z) - v0(z), v1(u) - v0(u), v1(v) - v0(v)) \\
v2_z &= (v2(z) - v0(z), v2(u) - v0(u), v2(v) - v0(v)) \\
\langle A_z, B_z, C_z \rangle &= v1_z \times v2_z
\end{aligned} \tag{23}$$

The δU , δV , and N_{uv} vectors form a transformation matrix at each vertex from object space into texture space:

$$M_T = \begin{bmatrix} \delta u / \delta x & \delta v / \delta x & N_{uv}(x) \\ \delta u / \delta y & \delta v / \delta y & N_{uv}(y) \\ \delta u / \delta z & \delta v / \delta z & N_{uv}(z) \end{bmatrix} \tag{24}$$

The current light position is multiplied by the inverse of the current object to world transformation matrix to bring it into object space. The light vector is then computed at each vertex in object space. It is then transformed into each vertex's texture space by using the M_T for that vertex. After the light vector has been put into texture space at a vertex, it is encoded as a color the same way normals are encoded in the normal map (Section 7.1) and stored as the vertex's diffuse color (Figure 71(a)).

The result of doing this is that light vectors will be linearly interpolated at all points on the mesh by the graphics card. The normal map can then be treated like a texture map (Figure 71(b)) and dotted with the encoded and interpolated light vectors, yielding per-pixel bump mapping (Figure 71(c)). This operation is equivalent to performing a per-pixel $(N \cdot L')$ where N comes from the normal map and is applied on a per-pixel instead of a per-vertex basis, and L' is the light vector in texture space at that vertex.

While the light vectors that result from this operation are normalized, the interpolated light vectors inside the triangle might not be. A solution to this problem is to use the interpolated light vectors as texture coordinates into a cube map, where each entry in the cube map is the normalized version of the index [32].

7.3.2 Lafortune Shading of Skin

Skin has a complicated BRDF that simple Gouraud or Phong shading cannot capture. We approximate the BRDF of skin with a Lafortune *et al.* [34] shading model. The Lafortune model approximates the BRDF of a surface as a weighted sum of generalized cosine lobes. Each cosine lobe is parameterized by three parameters that control scaling of the dot product of the incident and exitant direction vectors along the x , y , and z directions. These parameters and the weight on each lobe comprise a non-linear approximation to the reflectance function. We use three lobes in our renderer.

We measure the parameters of each of the three specular lobes using the technique proposed by Marschner *et al.* [41]. This technique results in the parameters for each lobe as well as the albedo map, that is, the diffuse component of the skin reflectance. In practice, we use a modified version of the Lafortune model. We use the normals from the normal map and the texture space light vectors to compute the diffuse component of the model, and the normals at the vertices to compute the specular component of the model. We use the original normals instead because we do not have access to the interpolated normals at each pixel. Since most graphics cards do not support pixel shaders, we would have to render each specular lobe off-screen as many times as its corresponding exponent if we wanted to compute this per-pixel. This operation is costly since one of the lobes has a very high exponent. We have found that interpolating the specular highlights across triangles looks realistic, but with more complicated lighting, the highlights may appear slightly triangulated. There are techniques for calculating specular reflections from approximated BRDFs efficiently [70], and for approximating the BRDF calculations with hardware [31, 70], which we did not do. However, the imminent adoption of pixel shaders in upcoming consumer-end graphics cards will allow BRDFs to be calculated directly.

7.4 Results

Our renderer runs in real-time on a Pentium III 1GHZ with a Geforce II graphics card at 13-14 frames per second. Figure 72 shows pairs of heads shaded using the Lafortune model discussed in Section 7.3.2 with and without fine scale structure rendering.

The faces with fine scale structure look considerably more realistic than those without. Specular highlights such as those on the forehead and left cheek look more realistic when there is fine detail under the highlight. The skin on the faces with no fine structure simply looks too smooth, although it is shaded realistically.

All results were rendered with a high-resolution (4096x2048 pixel) bumpmap. We found that detail was still retained at 2048x1024 pixels, while most was lost at 1024x512. However, the bumpmap can be quantized as in Tarini *et al.* [62] or indexed more efficiently to lower the memory requirement.

These results show that fine scale structure adds significantly to the realism of rendered faces, even when the BRDF is approximated using measured lobes in a Lafortune shading model. Skin rendered without an approximated BRDF tends to look like plastic, but even with a BRDF, it looks unrealistic. Fine scale structure is something we take for granted when looking at faces, yet is extremely important for visual realism.

Rendering fine scale structure is a good first step in improving the realism of skin, but there are more things that can be done and example based techniques could be helpful towards this end. Currently, low-frequency bumps of the face are not captured since the normals are forced to be perpendicular to the skin sample's plane. Different filtering of the captured sample could extract different properties that could then be synthesized by example using similar techniques. In addition, it would be interesting to extend these ideas to capture other skin properties such as rashes or scars. For these applications, different or additional features may prove more useful in synthesis than using normals alone.

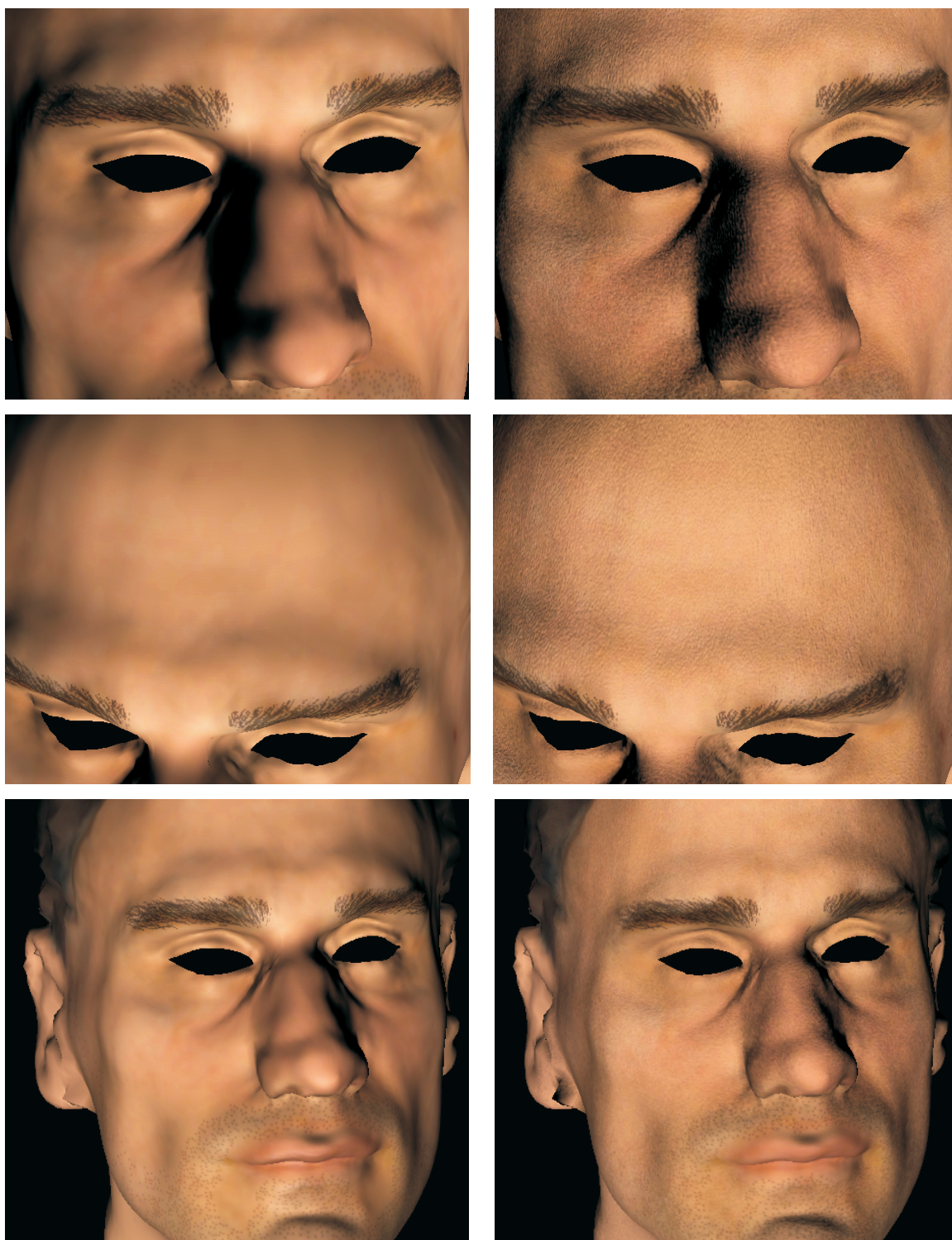


Figure 72: *Left*: Skin rendered without fine scale structure, *Right*: Skin rendered with fine scale structure.

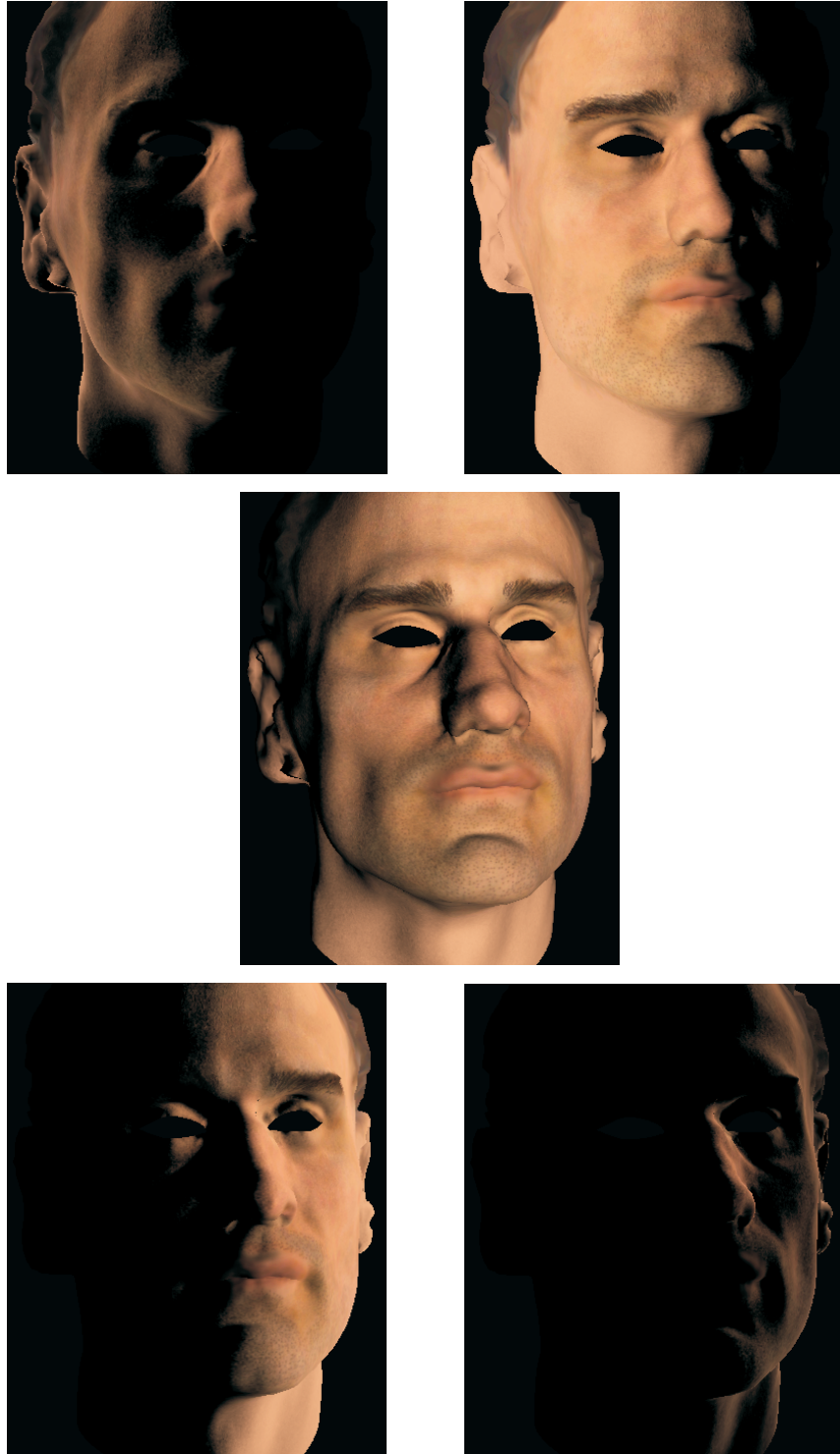


Figure 73: Effects of light position on fine scale structure.

CHAPTER 8

CONTRIBUTIONS

The work presented in this thesis bridges the gap between texture analysis/synthesis, example based image processing, and image inference problems by observing that these differ only in which feature space is used and how matches between the training and input are performed. We have shown how varying feature spaces and matching algorithms allows for previously unaddressed problem domains and data types to be processed by example.

A general framework to perform processing of videos and 3D models using a nonparametric sampling algorithm was presented. This algorithm allows users to perform complicated video processing and produce photorealistic renderings without knowledge of computer vision or graphics. These applications illustrate how general algorithms that learn from example can be used to address several problems under a single framework.

A probabilistic framework for performing example based image processing was described that significantly differs from previous approaches in this area. Casting image processing as an MRF labeling problem allowed us to generate results that were significantly different than what previous approaches were capable of achieving. For subjective problems such as non-photorealistic renderings or those where no quantitative comparison is possible, like texture transfer, an additional choice in example based image processing algorithms is very valuable to users. We also performed experiments to contrast this technique with existing algorithms to illustrate their differences and show which situations each is useful under. In addition, various systems and design issues that future example based processing algorithms will need to take into account to perform learning with arbitrarily large training data sets were presented and discussed.

This thesis also showed how example based image processing could be combined with rendering techniques to automate the creation of imagery that would otherwise be very time consuming.

Skin microstructure properties were measured instead of drawn by an artist to increase photo-realism and to save the artist's time. Skin molds were taken and then turned into 3D surfaces using photometric stereo. We then learned the microstructure surface deformations by example to create structure on the entire face from just a few samples that can be lit realistically on a 3D facial model. This technique can be extended to measure and render other skin detail such as rashes and scars.

CHAPTER 9

DISCUSSION

This thesis has explored example based processing algorithms for various problem domains and data types. Various different features were discussed for use in the creation of feature vectors for training and matching. In this chapter, we discuss possible extensions to the work presented along with a discussion of the limitations of current techniques.

9.1 Different data types

The framework can be applied to other data domains provided that the proper feature vectors and matching algorithms are used. For a new problem domain, spatial and/or temporal properties for data need to be estimated or observed to model local effects. For instance, in the data domain of sound, features consisting of overlapping windows in time along the waveform may prove useful. The matching algorithm used will also depend on the data domain. For problems where there is a directly observable or mathematical relationship between the input and output, the non-parametric algorithm of Chapter 3 may produce the best results. Similarly, for problems where there is no clear relationship between the input and output, the parametric algorithm of Chapter 4 may produce the best results.

To more deeply understand the relationship between the choices of features and matching algorithms, additional experimentation across more data domains is required. The framework may work well on other Markovian data such as text and sound, and may allow for the estimation of interesting processing that would be hard to express algorithmically. For example, applying the framework in the text data domain may allow for the translation of one person's writing style onto another's. Similarly, a recording of one person's voice may be transformed to sound like another person based on training data of both saying similar sentences. Musical styles may be possible to

learn by example as well, to do things like make a song sound like it was recorded in a different room or apply a person's playing style onto a recording.

9.2 Different features

Various different features have been presented in this thesis for use in the feature vectors. However, there are many other types of features that have not been experimented with that could prove useful or superior to those presented. Since the framework is feature vector based, the selection of features is extremely important.

Collections of outputs such as the results of filter banks or the use of wavelets could capture high-level information about pixel patches such as edge orientation or scale properties. Filter banks were used in the texon-based algorithm presented in Chapter 3, but they were not used for the other algorithms. While collections of outputs from filter banks could cause sampling problems due to the increase in dimensionality, using large training sets in conjunction with clustering could prevent such problems.

The choice of color space to use when using color-based features was shown to be extremely important as well. The $l\alpha\beta$ color space is very good for normalization and matching, but a better normalization may be possible if there is prior knowledge of the processing or of the colors in the training and test images.

9.3 Larger datasets

Some results using large datasets were presented in Chapter 5, however additional experimentation is required. Using external memory algorithms and data structures, it may be possible to have even larger datasets than those presented in this thesis. Instead of a training dataset representing a single artist, it may be possible for a training set to represent different artists and artistic styles. Larger datasets may also prove advantageous to learn more complex processing algorithms from example. As was shown in this thesis, additional training data improves the results for most problems, so it may be possible to achieve results of a much higher level of quality with extremely large training sets.

9.4 One to many functions

The example based processing works up to and including this thesis have been successfully applied towards estimation of processing algorithms that are approximately one to one functions. That is, it was assumed that by finding a training pixel patch similar to a test input pixel patch, the output would be approximately the same. However, there are some types of processing where this assumption would be violated. For example, colorizing grayscale images given a set of gray and color pixel patch pairs; a particular gray pixel patch may map to multiple color patches.

It should be possible to estimate one to many functions using the framework as well, but large amounts of training data or *a priori* knowledge may be required. In the colorization problem for example, texture recognition along with edge constraints could be employed to successfully colorize portions of the image, which may then allow for a reduction in possibilities in unknown regions.

9.5 Objects

All of the features discussed so far in the thesis are features computed on pixel patches. It may be possible to learn more advanced processing if higher-level features, such as objects, are used instead. This may allow the framework presented to be used to learn to perform object recognition under certain lighting environments for example. Another interesting experiment would be to use an already existing object recognition system to identify objects in the scene to collect training samples to build an exemplar based appearance model. Moving away from pixels and towards higher-level models may also make it possible to have more robust example based processing systems than can be currently created due to pixel noise/variation.

9.6 Conclusion

This thesis has presented a generalized framework for performing example based processing in different data domains. The framework unifies work from the areas of texture/video synthesis, image inference, and example based processing by treating all problems as MRF labeling problems. We

demonstrated that some processing algorithms can be estimated through pairs of input and output feature vectors, where the features are dependent on the data domain. The framework was then used across different multidimensional data domains including images, video, and 3D models to estimate processing in each domain.

REFERENCES

- [1] ASHIKHMIN, M., “Synthesizing natural textures,” *2001 ACM Symposium on Interactive 3D Graphics*, pp. 217–226, March 2001.
- [2] BISHOP, C. M., *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.
- [3] BLACK, M. and ANANDAN, P., “The robust estimation of multiple motions: Parametric and piecewise-smooth flow-fields,” *CVIU*, vol. 63, pp. 75–104, January 1996.
- [4] BRAND, M. E., “Incremental singular value decomposition of uncertain data with missing values,” in *ECCV02*, pp. 707–720, 2002.
- [5] BREGLER, C., COVELL, M., and SLANEY, M., “Video rewrite: Driving visual speech with audio,” *Proceedings of SIGGRAPH 97*, pp. 353–360, August 1997.
- [6] BURT, P. and ADELSON, E., “The laplacian pyramid as a compact image code,” in *IEEE Transactions on Communications*, pp. 532–540, 1983.
- [7] CHANDRASEKARAN, S., MANJUNATH, B. S., WANG, Y. F., WINKELER, J., and ZHANG, H., “An eigenspace update algorithm for image analysis,” *Graphical models and image processing: GMIP*, vol. 59, no. 5, pp. 321–332, 1997.
- [8] DE BONET, J., “Multiresolution sampling procedure for analysis and synthesis of texture images,” *Proceedings of SIGGRAPH 97*, pp. 361–368, August 1997.
- [9] DEBEVEC, P., HAWKINS, T., TCHOU, C., DUIKER, H., SAROKIN, W., and SAGAR, M., “Acquiring the reflectance field of a human face,” *Proceedings of SIGGRAPH 2000*, pp. 145–156, 2000.
- [10] DIETRICH, S., “Texture space bumpmaps.” Available from <http://www.nvidia.com/>, 2000.
- [11] DUDA, R. O., HART, P. E., and STORK, D. G., *Pattern Classification*. John Wiley and Sons Inc., 2001.
- [12] EFROS, A. and LEUNG, T., “Texture synthesis by non-parametric sampling,” in *ICCV99*, pp. 1033–1038, 1999.
- [13] EFROS, A. A. and FREEMAN, W. T., “Image quilting for texture synthesis and transfer,” *Proceedings of SIGGRAPH 2001*, pp. 341–346, August 2001.
- [14] FREEMAN, W., PASZTOR, E., and CARMICHAEL, O., “Learning low-level vision,” *IJCV*, vol. 40, pp. 25–47, October 2000.

- [15] FREEMAN, W., TENENBAUM, J., and PASZTOR, E., “An example-based approach to style translation for line drawings,” Tech. Rep. MERL-TR99-11, Mitsubishi Electric Research Lab, 1999.
- [16] GERSHO, A. and GRAY, R., *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.
- [17] GUENTER, B., GRIMM, C., WOOD, D., MALVAR, H., and PIGHIN, F., “Making faces,” *Proceedings of SIGGRAPH 1998*, pp. 55–66, 1998.
- [18] GUTTMAN, A., “R-trees: a dynamic index structure for spatial searching,” *SIGMOD '84, Proceedings of Annual Meeting (ACM Special Interest Group on Management of Data)*, vol. 14, no. 2, pp. 47–57, 1984.
- [19] HAMEL, J. and STROTHOTTE, T., “Capturing and re-using rendition styles for non-photorealistic rendering,” *Computer Graphics Forum*, vol. 18, pp. 173–182, September 1999.
- [20] HANRAHAN, P. and KRUEGER, W., “Reflection from layered surfaces due to subsurface scattering,” *Proceedings of SIGGRAPH 1993*, pp. 165–174, 1993.
- [21] HEEGER, D. J. and BERGEN, J. R., “Pyramid-based texture analysis/synthesis,” *Proceedings of SIGGRAPH 95*, pp. 229–238, August 1995.
- [22] HEIDRICH, W. and SEIDEL, H. P., “Realistic, hardware-accelerated shading and lighting,” *Proceedings of SIGGRAPH 1999*, pp. 171–178, 1999.
- [23] HERTZMANN, A., *Algorithms for Rendering in Artistic Styles*. PhD thesis, NYU, 2001.
- [24] HERTZMANN, A. and SEITZ, S. M., “Shape and materials by example: A photometric stereo approach,” in *CVPR03*, pp. 533–540, 2003.
- [25] HERTZMANN, A., “Paint by relaxation,” *Proceedings of Computer Graphics International*, pp. 47–54, July 2001.
- [26] HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., and SALESIN, D. H., “Image analogies,” *Proceedings of SIGGRAPH 2001*, pp. 327–340, August 2001.
- [27] HERTZMANN, A. and PERLIN, K., “Painterly rendering for video and interaction,” *NPAR 2000: First International Symposium on Non Photorealistic Animation and Rendering*, pp. 7–12, June 2000.
- [28] HORN, B. K. P. and BROOKS, M. J., *Shape from Shading*. MIT Press, 1989.
- [29] HOUSE, J., *Monet - Nature Into Art*. Yale University Press, 1986.
- [30] JOJIC, N. and FREY, B., “Learning flexible sprites in video layers,” in *CVPR01*, pp. I:199–206, 2001.
- [31] KAUTZ, J. and MCCOOL, M. D., “Interactive rendering with arbitrary brdfs using separable approximations,” in *Proceedings of 10th Eurographics Workshop on Rendering*, pp. 281–292, 1999.

- [32] KILGARD, M., “A practical and robust bump-mapping technique for today’s gpus.” Available from <http://www.nvidia.com/>, 2000.
- [33] KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., and BOBICK, A., “Image and video synthesis using graph cuts,” *Proceedings of SIGGRAPH 2003*, pp. 277–286, 2003.
- [34] LAFORTUNE, E., FOO, S., TORRANCE, K., and GREENBERG, D., “Non-linear approximation of reflectance functions,” *Proceedings of SIGGRAPH 1997*, pp. 117–126, 1997.
- [35] LALLEMAND, H., *Monet - Impressions of Light*. Todtri Book Publishers, 1994.
- [36] LEUNG, T. and MALIK, J., “Recognizing surfaces using three-dimensional textons,” in *ICCV99*, pp. 1010–1017, 1999.
- [37] LEVY, A. and LINDENBAUM, M., “Sequential karhunen-loeve basis extraction and its application to images,” Tech. Rep. CIS9809, Technion, 1998.
- [38] LIANG, L., LIU, C., XU, Y.-Q., GUO, B., and SHUM, H.-Y., “Real-time texture synthesis by patch-based sampling,” *ACM Transactions on Graphics*, vol. Vol. 20, No. 3, pp. 127–150, July 2001.
- [39] LITWINOWICZ, P., “Processing images and video for an impressionist effect,” *Proceedings of SIGGRAPH 97*, pp. 407–414, August 1997.
- [40] MARSCHNER, S., *Inverse Rendering for Computer Graphics*. PhD thesis, Cornell University, 1998.
- [41] MARSCHNER, S., GUENTER, B., and RAGHUPATHY, S., “Modeling and rendering for realistic facial animation,” in *Proceedings of 11th Eurographics Workshop on Rendering*, pp. 231–242, 2000.
- [42] MOORE, A. W., “Personal communicaton,” 2003.
- [43] MOUNT, D. and ARYA, S., “Ann: A library for approximate nearest neighbor searching,” 1997.
- [44] NAHAS, M., HUITRIC, H., RIOUX, M., and DOMEY, J., “Facial image synthesis using skin texture recording,” *The Visual Computer*, vol. 6, no. 6, pp. 337–343, 1990.
- [45] PETROVIC, N., COHEN, I., FREY, B., KOETTER, R., and HUANG, T., “Enforcing integrability for surface reconstruction algorithms using belief propagation in graphical models,” in *CVPR01*, pp. I:743–748, 2001.
- [46] PIGHIN, F., HECKER, J., LISCHINSKI, D., SZELISKI, R., and SALESIN, D., “Synthesizing realistic facial expressions from photographs,” *Proceedings of SIGGRAPH 1998*, pp. 75–84, 1998.
- [47] POPAT, K. and PICARD, R., “Novel cluster-based probability model for texture synthesis, classification, and compression,” in *Proceedings SPIE Visual Communications and Image Processing*, 1993.

- [48] PRAUN, E., FINKELSTEIN, A., and HOPPE, H., “Lapped textures,” *Proceedings of SIGGRAPH 1998*, pp. 465–470, 1998.
- [49] REINHARD, E., ASHIKHMIN, M., GOOCH, B., and SHIRLEY, P., “Color transfer between images,” *IEEE Computer Graphics and Applications*, vol. 21, pp. 34–41, Sept./Oct. 2001.
- [50] RUDERMAN, D., CRONIN, T., and CHIAO, C., “Statistics of cone responses to natural images: Implications for visual coding,” *J. Optical Soc. of America*, vol. 15, no. 8, pp. 2036–2045, 1998.
- [51] RUSHMEIER, H. and BERNARDINI, F., “Computing consistent normals and colors from photometric data,” in *2nd International Conference on 3D Digital Imaging and Modeling (3DIM)*, pp. 99–108, 1999.
- [52] RUSHMEIER, H., BERNARDINI, F., MITTLEMAN, J., and TAUBIN, G., “Acquiring input for rendering at appropriate levels of detail: Digitizing a pieta,” in *Rendering Techniques*, pp. 81–92, 1998.
- [53] RUSSELL, V., *Monet’s Landscapes*. Bulfinch Press, 2000.
- [54] SAITO, T. and TAKAHASHI, T., “Comprehensible rendering of 3-d shapes,” *Proceedings of SIGGRAPH 90*, pp. 197–206, August 1990.
- [55] SCHÖDL, A. and ESSA, I., “Machine learning for video-based rendering,” in *Advances in Neural Information Processing Systems*, vol. 13, pp. 1002–1008, 2001.
- [56] SCHÖDL, A., SZELISKI, R., SALESIN, D. H., and ESSA, I., “Video textures,” *Proceedings of SIGGRAPH 2000*, pp. 489–498, July 2000.
- [57] SCOPIGNO, R., PINGI, P., ROCCHINI, C., CIGNONI, P., and MONTANI, C., “3d scanning and rendering cultural heritage artifacts on a low budget,” in *European Workshop on ‘High Performance Graphics Systems and Applications’*, pp. 16–17, 2000.
- [58] SLOAN, P.-P. J., MARTIN, W., GOOCH, A., and GOOCH, B., “The lit sphere: A model for capturing npr shading from art,” in *Graphics Interface 2001*, pp. 143–150, June 2001.
- [59] SPATE, V., *Claude Monet - The Color Of Time*. Thames and Hudson, Inc., 1992.
- [60] SUN, J., SHUM, H., and ZHENG, N., “Stereo matching using belief propagation,” in *ECCV02*, p. II: 510 ff., 2002.
- [61] TAPPEN, M. F., RUSSELL, B. C., and FREEMAN, W. T. F., “Exploiting the sparse derivative prior for super-resolution and image demosaicing,” in *3rd Intl. Workshop on Statistical and Computational Theories of Vision (associated with Intl. Conf. on Computer Vision)*, 2003.
- [62] TARINI, M., CIGNONI, P., ROCCHINI, C., and SCOPIGNO, R., “Real time, accurate, multi-featured rendering of bump mapped surfaces,” in *Eurographics*, pp. 119–130, 2000.
- [63] TURK, G., “Texture synthesis on surfaces,” *Proceedings of SIGGRAPH 2001*, pp. 347–354, August 2001.

- [64] VITTER, J. S., “External memory algorithms and data structures: dealing with massive data,” *ACM Computing Surveys*, vol. 33, no. 2, pp. 209–271, 2001.
- [65] WEI, L. and LEVOY, M., “Fast texture synthesis using tree-structured vector quantization,” *Proceedings of SIGGRAPH 2000*, pp. 479–488, July 2000.
- [66] WEI, L.-Y. and LEVOY, M., “Texture synthesis over arbitrary manifold surfaces,” *Proceedings of SIGGRAPH 2001*, pp. 355–360, August 2001.
- [67] WEISS, Y. and FREEMAN, W., “Correctness of belief propagation in gaussian graphical models of arbitrary topology,” Tech. Rep. UCB.CSD-99-1046, Berkeley Computer Science Dept., 1999.
- [68] WELSH, T., ASHIKHMIN, M., and MUELLER, K., “Transferring color to greyscale images,” *ACM Transactions on Graphics*, vol. 21, pp. 277–280, July 2002.
- [69] WU, Y., KALRA, P., and MAGNENAT-THALMANN, N., “Physically-based wrinkle simulation and skin rendering,” in *Eurographics Workshop on Computer Animation and Simulation*, pp. 69–79, 1997.
- [70] WYNN, C., “Real-time brdf-based lighting using cube-maps.” Available from <http://www.nvidia.com/>, 2000.
- [71] XU., Y., GUO, B., and SHUM, H., “Chaos mosaic: Fast and memory efficient texture synthesis,” Tech. Rep. MSR-TR-2000-32, Microsoft Research, 2000.
- [72] YEDIDIA, J., FREEMAN, W., and WEISS, Y., “Understanding belief propagation and its generalizations,” Tech. Rep. TR-2001-22, Mitsubishi Electric Research Lab, 2001.
- [73] YU, Y. and MALIK, J., “Recovering photometric properties of architectural scenes from photographs,” *Proceedings of SIGGRAPH 1998*, pp. 207–217, 1998.
- [74] ZHA, H. and SIMON, H. D., “On updating problems in latent semantic indexing,” *SIAM Journal on Scientific Computing*, vol. 21, no. 2, pp. 782–791, 1999.

VITA

Antonio Haro was born in 1976 in Wayne, New Jersey. He started the undergraduate computer science program at the New Jersey Institute of Technology in 1993. He received his Bachelor of Science degree in 1997 with honors and a minor in Applied Mathematics. He began the Ph.D. program at Georgia Tech's College of Computing in 1997. He worked at Bell Labs from the summer of 1996 to May 1997. From May until August 1997, he interned at AT&T Labs and was awarded a fellowship from AT&T for the duration of his graduate career. The following summers of 1998 and 1999 were spent interning at IBM's TJ Watson Labs and Almaden Research Center, respectively. In the summer and fall of 2000, he interned at Microsoft Research.