Data-driven Approaches for Texture and Motion

by

Alexei A. Efros

B.S. (University of Utah) 1997
M.S. (University of California, Berkeley) 1999

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Jitendra Malik, Chair
Professor David A. Forsyth
Professor Stephen Palmer

Fall 2003

The dissertation of Alexei A. Efros is approved:

Chair Date

Date

Date

University of California, Berkeley

Fall 2003

Data-driven Approaches for Texture and Motion

Abstract

Data-driven Approaches for Texture and Motion

by

Alexei A. Efros

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Jitendra Malik, Chair

Access to vast amounts of visual information and the increase in computing power are facilitating the emergence of a new class of simple data-driven approaches for image analysis and synthesis. Instead of building an explicit parametric model of a phenomenon, the data-driven techniques rely on the underlying data to serve as its own representation. This dissertation presents work in two different domains, visual texture and human motion, where data-driven approaches have been particularly successful.

Part I describes two algorithms for texture synthesis, the problem of synthesizing visual texture (e.g. grass, bricks, pebbles) from an example image. The goal is to produce novel samples of a given texture, which, while not identical, will be perceived by humans as the same texture. The proposed texture synthesis process grows a new image outward from an initial seed, one pixel/patch at a time. A Markov random field model is assumed, and the conditional distribution of a pixel/patch given all its neighbors synthesized so far is estimated by querying the sample image and finding all similar neighborhoods. The degree of randomness is controlled by a single perceptually intuitive parameter. The method aims at preserving as much local structure as possible and produces good results for a wide variety of synthetic and real-world textures. One discussed application of the method is *texture transfer*, a novel technique that allows texture from one object to be "painted" onto a different object.

Part II presents an algorithm for the analysis and synthesis of human motion from video. Instead of reconstructing a 3D model of the human figure (which is hard), the idea is to "explain" a novel motion sequence with bits and pieces from a large collection of stored

video data. This simple method can be used for both action recognition as well as motion transfer – synthesizing a novel person imitating the actions of another person ("Do as I Do" synthesis) or performing actions according to the specified action labels ("Do as I Say" synthesis).

Professor Jitendra Malik
Dissertation Committee Chair

To my chair,
for supporting me all these years.

Contents

List of Figures

Acknowledgments

Once upon a time, at a village high up in the majestic mountains of Georgia there lived a strange eagle. Every day, the eagle flew high up in the skies and every day it threw itself down on to the sharp rocks with its bare bottom. One day, the village elders went up to the eagle:
– O Dear Eagle, why is it that every day you fly high up in the skies and then throw yourself down on to the sharp rocks with your bare bottom?
And the eagle said:
– O Dear People, it must be difficult for you to understand why every day, I fly high up in the skies and then throw myself down on to the sharp rocks with my bare bottom.
It is because I have no friends.

So let us drink to friendship!

A Georgian Toast
as told by Piotr Gibas

There are so many people who have helped me in one way or another and to whom I owe deep gratitude. It would be impossible to mention all of them here, but I will try anyway.

First, I would like to thank my academic adviser, Jitendra Malik, for taking me off the street as a young and foolish "I wanna build a seeing robot" dilettante and turning me into an old, seasoned academic (though foolish all the same – I still want to build that seeing robot someday). Jitendara's encyclopedic erudition, his fearlessness in the face of hard research problems, his laconic aphorisms and sudden, deep insights, his daily visits to the lab and long chats at Nefeli's are all parts of what makes him not only a great adviser but also a very fun person. I sincerely appreciate all his support, generosity, and, above all, patience.

It has been a pleasure to learn from David Forsyth; his clever comments and witty criticisms have many a time saved me from embarking on a wrong path and delivered me "home and dry". Steve Palmer, who has been kind enough to be my outside member, has been my main source for learning about visual perception, first through his book and later from his wonderful lectures. I was fortunate enough to have had Bill Freeman as my summer mentor (back in his care-free MERL days); working with him was very fun and easy, and I learned a lot. I am also indebted to my undergraduate mentors, Tom Henderson, Bill

Thompson and Pete Shirley, for all their time and help. My father, Alexei L. Efros, has certainly been my biggest influence, starting me on the path that would become my career. Not only did he get me interested in computers at a very early age, but his children's stories about psychology, perception, intelligent machines, and that very hard problem in physics called *object recognition*, largely determined my future field of study.

The legendary atmosphere of 545 Soda Hall and its environs has had the most profound impact on shaping my scientific thinking. Being surrounded by so many intelligent and creative people, always ready to talk, argue, and shoot down ideas, was certainly the best thing about my entire graduate school experience. I would like to sincerely thank the Good Old Guard: Thomas Leung, Jianbo Shi, Serge Belongie, Chris Bregler, Yizhou Yu, Paul Debevec and Yair Weiss; as well as "The Next Generation": Andrea Frome, Laura Walker, Stella Yu, Okan Arikan, Alex Berg, Andras Ferencz, Charless Fowlkes, David Martin, Erik Miller, Greg Mori, Deva Ramanan, Xiaofeng Ren and Hao Zhang. Thomas Leung, in particular, has spent an enormous amount of time and energy with me, patiently explaining and carefully piecing together the intricate world of computer vision for me and, in effect, becoming my unofficial mentor. It is under his guidance that the beginnings of this thesis have emerged and I am very honored to have him as the co-author of my first real paper. The help of my other co-authors – Bill Freeman, Alex Berg, Greg Mori, and Jitendra Malik – is also greatly appreciated.

The most wonderful thing about my years in Berkeley is the friends that I was so incredibly lucky to make here. For the first time in my nomadic life, there are people who are always happy to see me; there are phone numbers that I can always call; there are homes where I can just show up, unannounced, and be warmly welcomed, fed, and offered a bed for the night. First, I want to thank my two wonderful friends, Jianbo Shi and Marco Zambon. The phrase "this work would not have been possible without (blah-blah-blah)..." is an old cliché but in my case it's completely true – without these two, not only would my thesis never see the light of day, but my whole life would have taken a much grimmer turn. Jianbo, singlehandedly, got me out of my "funk" and set me onto the right path, while Marco was there to make sure that I don't stray from it. Their dedication and friendship, though thoroughly undeserved, are whole-heartedly appreciated. I have many many other friends whom I'd like to thank: Andras Ferencz (for suggesting that I move into I-House), Anne-Celine Lambotte (for being the best museum guide), Chenchang Zhu (for Pushkin), Chrissy Chum (for putting my life back in order; for chick-magnet glasses,

for pottery, and, of course, for Betsy), Dave Hart (for beautiful guitar playing), Deanna Kang and Dana Wulfe (for claiming I am not too old), Eli Lebow (for the best jokes), Elizabeth Pollock (for the wine/vine rhyme), Elizabeth Zacharias (for the Shadow), Gabor Pete and Kati Palinkas (for imagination and the beautiful smell of apples which grow on the shores of Lake Balaton), Hermann Cuntz (for Malbec and the long evening talks), Joaquin Rosales (for Cortazar, for Rosales, for La Llorona and for Tequila), Kelly Then (for all the good advice), Kristof Madarasz (for showing me Richter), Lela Samniashvili (for the poetry), Lillian Chu (for a million cakes and one suit), Medha Pathak (for food, shelter, and a million other things), Mike Downes (for Borges), Noam Berger, Karen Lehmann, Ben Reynolds and Xiaoyue Zhao (for The Meershluck Club, the only organization whose member I am proud to be) Pere Rovira (for being the most romantic postmodernist I know), Pinar Duygulu-Sahin (for listening), Piotrec Gibas (for being Piotrec), Polina Dimova (for knowing me too well), Profesora Maria Del Mar Rubio Varas, Ph.D (for the wisdom), Ria Steiger (for the train trip), Tamas Kalman and Yanlei Diao (for all the parties in their cozy place), Tim Strunkov (for the kayak), Tolya Grinshpan and Tanya Ioudovina (for lake Anza and the good old times), Von Bing and Anne Yap (for not abandoning me during my rough days), Yoshie Yamaji (for the pomegranate as big as her head) and, of course, Betsy (for never once flying away). Many of the people mentioned above also assisted me in the research for Appendix A, with Tolya and Von Bing being especially enthusiastic researchers. In addition, I would like to thank Eli for carefully proofreading (with a map in his hand, as only Eli can) the appendix and Medha and Polina for proofreading other parts of this manuscript.

I would like to thank Liza Levina for being my closest friend for many many years, for teaching me pretty much everything I know, and for putting up with me for as long as she did. I also have a number of ex-friends who, at one time, have been very good and kind to me and I thank them for that, hoping that better days still lie ahead. Finally, I would like to thank my fickle, long-gone muse for that faint afterglow which still persists, adding a touch of warmth and color to an otherwise grey world.

Chapter 1

Introduction

I speak and speak, ... but the listener retains only the words he is expecting. ...It is not the voice that commands the story: it is the ear.

Italo Calvino
Invisible Cities

When we look out into the world, the amount of information that we *think* we are seeing at any given moment is enormous compared with what is actually getting delivered to the photoreceptors in our eyes. From sparse, noisy, ambiguous, hopelessly inconsistent data the brain manages to create a complete, coherent visual experience, a comprehensible universe that is often wrong but always rich and vivid. It appears that the brain routinely "hallucinates" a large portion of the visual world based on past memory (the Lateral Geniculate Nucleus (LGN), a relay station on the path to the visual cortex, gets only about 20% of its input from the eyes, the rest comes from other parts of the brain [59]). That's why it sometimes takes days for your friends to realize that you've got a new haircut – they know you so well that they hardly ever *look* at you anymore, relying instead on their memories. This is also why painters as well as graphics programmers are able to use simple tricks that recreate only a few aspects of an object's appearance but still produce images that "look right". And this is also one reason why computer vision is so difficult: the interpretation of pixel values in a given image is not uniquely determined – it often depends on what has been seen in the past.

Representing visual experience in a meaningful and usable way poses a major research challenge. It is desirable to have a representation that captures the essence of the data, the process by which it came about, in a compact and well-understood way.

Unfortunately, constructing such parametric models for visual phenomenon is an extremely difficult task, requiring a great deal of prior knowledge and producing models that are very domain-specific. And while the search continues for a general parametric representation, a very different approach is gaining popularity, a simple technique that does not attempt to construct a model, instead using the data as its own representation.

1.1 Learning by Plagiarism

During the last few years the increase in computer speed, storage capacity, and, especially, the availability of large amounts of information in electronic form have caused a fundamental shift in how knowledge is represented and even defined. Science fiction writers and AI researchers dream of future all-knowing machines that would be able to answer any questions posed to them. But in a way, the future is already here: the Internet search engines, such as GOOGLE, are used by millions of people to find answers on the widest range of topics. Of course, a search engine doesn't "understand" the queries and has no inherent intelligence per se, it's just a clever pattern matcher. What makes the resulting system so powerful is the vastness of the data, the implicit postmodern assumption that all questions have already been answered, many times and in many ways. And although no generalization is being done, inference is still possible in this framework as demonstrated, for example, by the automatic spelling-correction in some search engines. Instead of relying on a fixed dictionary, the data itself is used to correct the spelling of the query (note the importance of context: "clime" in "clime the stairs" is corrected by GOOGLE into "climb", but in "clime and punishment" becomes "crime"). In effect, a search engine is an example of what psychologists term *associative memory*: a small fragment of information triggers the recall of the appropriate memory which is associated with the fragment.

Associative memory[1] belongs to a family of example-based or *data-driven* approaches. It is a very powerful mechanism for working with data which is either poorly understood or too complex to be modeled explicitly (which is the case for much of visual data). In this dissertation, I have applied the data-driven paradigm in two domains, texture and human action, where parametric approaches have not been particularly successful.

[1]Here the term *associative memory* refers to the original psychological concept, and not necessarily to the way it is being used in the neural network literature.

1.2 Overview

Part I is devoted to Texture. Visual texture refers to spatially repeating patterns on a surface, such as grass, a brick wall, or pebbles on a beach. Texture provides a good model for testing the data-driven approaches because a single texture image contains a very large number of different examples of that texture. The goal of *texture synthesis* is to produce novel samples of a given texture, which, while not identical, will be perceived by humans to be the same texture. It is used in computer graphics to populate virtual environments with real textures, filling-in holes in stereo reconstructions, and texture mapping non-developable surfaces. In Chapter 2, we given a brief introduction to texture and texture synthesis. Chapter 3 describes a data-driven pixel-based texture synthesis approach. In this framework, texture is "grown" one pixel at a time from an initial seed. For each new pixel, its partially-filled neighborhood is used to query the database (in this case, the input texture image) to find all similar neighborhoods, which provide us with the value for the pixel in question. In Chapter 4, the approach is generalized to operate on patches of texture, using a method we called *Image Quilting*. Chapter 5 introduces texture transfer. We note that the use of partial context for querying is not limited to finding instances *containing* the query, but, more generally, can be applied to retrieving instances that are *associated* with it in some way. If such associations are established across domains, either empirically or functionally, high-level structure within one domain can induce relationships within another domain, recasting the content of the former into the language of the latter. Applied to texture synthesis, this allows us to re-render an image using texture taken from a completely different image. We call this process *texture transfer*.

Part II deals with the analysis, classification, and synthesis of human actions. This is a richer, more diverse, and consequently more difficult domain than texture, comprising video sequences, as opposed to single static images. Nonetheless, this is a good test-bed for data-driven approaches, since the video sequences all have a single subject – people, thus a lot of example data can be obtained from a reasonable amount of video footage. Automatic recognition of human actions in video sequences is an important problem that has recently received much interest. Traditional approaches to this problem attempt to recover high-level features such as limb positions and joint angles and then build a parametric representation for a given activity. However, this process is very sensitive to noise, and generally fails on poor-quality data (e.g. when the person is far away from the camera). In Chapter 6,

we, again, take an example-based, data-driven approach, collecting a large amount of video data for each action, so that a novel action sequence can be classified simply by finding a similar action in the database. Matching two video sequences is done using a spatio-temporal motion descriptor which consists of a set of rough low-level image measurements. Using the same technique we are also able to extend the idea of texture transfer to motion synthesis, e.g. synthesizing a novel video of a person imitating the actions of a different person (Chapter 7).

Part I

Texture Synthesis

Chapter 2

On Texture and its Synthesis

What is Jazz?
If you hafta ask, you ain't never gonna know!
Louis Armstrong

2.1 What is texture?

Texture is a ubiquitous part of our everyday visual experience. And yet, it is difficult to come up with a precise definition for it. The best that I can do is to say *look*:


pebbles


windows


yogurt


rock surface


tomatoes

pebbles on the beach, the facade of a building, the yogurt on your plate – these are all examples of texture. Roughly, texture can be thought of as a collection of similar visual elements (called *texels*) that are too numerous or too ill-defined to be perceived as individual objects. For example, there are so many pebbles on a beach that we don't see each pebble separately, but perceive them all as a single object – a pebble beach. A different example is a rock surface: although it is not at all clear what the individual visual elements are, the

periodic in-between stochastic

Figure 2.1: The spectrum of textures. There are many different kinds of texture, ranging from purely periodic (as us the brick wall) to purely stochastic (such as white noise), to something in-between, like the slice of banana-bread.

image as a whole gives a distinct impression of a rough surface.

Textures also vary greatly depending on the *arrangement* of the visual elements (Figure 2.1). Sometimes the arrangement is quite regular and periodic, as in the brick wall image. In other instances, the regularity is only statistical, or stochastic, as in the image of white noise. Most textures, however, fall somewhere in between these two extremes. The image of a banana bread slice is a good example: although the dimples and the cavities themselves are rather regular, their placement, scale, and orientation are all quite stochastic.

2.2 Texture Synthesis

In the past decade computer graphics experienced a wave of activity in the area of image-based rendering as researchers explored the idea of capturing samples of the real world as images and using them to synthesize novel views rather than recreating the entire physical world from scratch. This, in turn, fueled interest in image-based *texture synthesis* algorithms. Such an algorithm should be able to take a sample of texture (i.e. Figure 2.2a) and generate an unlimited amount of image data which, while not exactly like the original, will be perceived by humans to be the same texture (Figure 2.2b). A naive approach (still practiced by many video-game developers) is to simply tile the input texture, as shown on Figure 2.2c. However, as can be seen from the figure, the tiling produces an unsatisfactory result, with obvious repetition and seams, while the synthesized texture looks much more realistic. The applications of texture synthesis in graphics include populating virtual environments with real textures, filling-in holes in stereo reconstructions, and texture mapping non-developable surfaces.

(a) input texture (b) synthesized texture (c) simple tiling

Figure 2.2: Texture Synthesis vs. Tiling. Given an input texture (a), a texture synthesis algorithm produces a larger sample of the texture, such as (b). Simply tiling the input texture produces a much less appealing result (c).

The problem of texture synthesis can be formulated as follows: let us define texture as some visual pattern on an infinite 2-D plane which, at some scale, has a stationary distribution. Given a finite sample from some texture (an image), the goal is to synthesize other samples from the same texture. Without additional assumptions this problem is clearly ill-posed since a given texture sample could have been drawn from an infinite number of different textures. The usual assumption is that the sample is large enough that it somehow captures the stationarity of the texture and that the (approximate) scale of the texture elements (*texels*) is known.

2.2.1 Previous work

Texture analysis and synthesis has had a long history in psychology, statistics and computer vision. In 1950 Gibson pointed out the importance of texture for visual perception [26], but it was the pioneering work of Bela Julesz on texture discrimination [31] that paved the way for the development of the field. Julesz suggested that two texture images will be perceived by human observers to be the same if some appropriate statistics of these images match. This suggests that the two main tasks in statistical texture synthesis are (1) picking the right set of statistics to match, (2) finding an algorithm that matches them.

Motivated by psychophysical and computational models of human texture discrimination [6, 37], Heeger and Bergen [28] proposed to analyze texture in terms of histograms

of filter responses at multiple scales and orientations. Matching these histograms iteratively was sufficient to produce impressive synthesis results for stochastic textures (Zhu et.al. [69] provide a related approach based on Gibbs sampling). However, since the histograms measure marginal, not joint, statistics they do not capture important relationships across scales and orientations, thus the algorithm fails for more structured textures. By also matching these pairwise statistics, Portilla and Simoncelli [44] were able to substantially improve synthesis results for structured textures at the cost of a more complicated optimization procedure.

In the above approaches, texture is synthesized by taking a random noise image and *coercing* it to have the same relevant statistics as in the input image. An opposite approach is to start with an input image and *randomize* it in such a way that only the statistics to be matched are preserved. De Bonet [8] scrambles the input in a coarse-to-fine fashion, preserving the conditional distribution of filter outputs over multiple scales (jets). This method was demonstrated on a wide range (mostly synthetic) textures although the randomness parameter seems to exhibit perceptually correct behavior only on largely stochastic textures. Another drawback of this method is the way texture images larger than the input are generated. The input texture sample is simply replicated to fill the desired dimensions before the synthesis process, implicitly assuming that all textures are tilable which is clearly not correct. Xu el.al. [62], inspired by the Clone Tool in PhotoShop, propose a much simpler approach yielding similar or better results. The idea is to take random square blocks from the input texture and place them randomly onto the synthesized texture (with alpha blending to avoid edge artifacts). The statistics being preserved here are simply the arrangement of pixels within each block. While this technique will fail for highly structured patterns (e.g. a chess board) due to boundary inconsistencies, for many stochastic textures it works remarkably well. A related method was successfully used by Praun et.al. [45] for semi-automatic texturing of non-developable objects.

The major shortcoming of all the above approaches is that none can generate visually appealing results for a wide range of textures (such as on Figure 2.1), especially for textures with a strong non-stochastic component. In the following chapters, a simple new algorithm will be presented that will address these issues.

Chapter 3

Pixel-based Synthesis Algorithm

First, you must become friends with every pixel.
Jitendra Malik

In his 1948 article, *A Mathematical Theory of Communication* [53], Claude Shannon mentioned an interesting way of producing English-sounding written text using n-grams. The idea is to model language as a generalized Markov chain: a set of n consecutive letters (or words) make up an n-gram and completely determine the probability distribution of the next letter (or word). Using a large sample of the language (e.g., a book) one can build probability tables for each n-gram. One can then repeatedly sample from this Markov chain to produce English-sounding text. This is the basis for an early computer program called Mark V. Shaney, popularized by an article in *Scientific American* [18], and famous for such pearls as: *"I spent an interesting evening recently with a grain of salt"*.

The algorithm in this chapter relates to an earlier work by Popat and Picard [43] in trying to extend this idea to two dimensions. The three main challenges in this endeavor are: 1) how to define a unit of synthesis (a letter) and its context (n-gram) for texture, 2) how to construct a probability distribution, and 3) how to linearize the synthesis process in 2D.

Our algorithm "grows" texture, pixel by pixel, outward from an initial seed. We chose a single pixel p as our unit of synthesis so that our model could capture as much high frequency information as possible. All previously synthesized pixels in a square window

around p (weighted to emphasize local structure) are used as the context. To proceed with synthesis we need probability tables for the distribution of p, given all possible contexts. However, while for text these tables are (usually) of manageable size, in our texture setting constructing them explicitly is out of the question. An approximation can be obtained using various clustering techniques, but we choose not to construct a model at all. Instead, for each new context, the sample image is queried and the distribution of p is constructed as a histogram of all possible values that occurred in the sample image as shown on Figure 3.1. The non-parametric sampling technique, although simple, is very powerful at capturing statistical processes for which a good model hasn't been found. Here it should be noted that a very similar algorithm was independently discovered in 1981 by Garber [23] but, sadly, forgotten due to its then computational intractability.

3.1 The Algorithm

In this work we model texture as a Markov Random Field (MRF). That is, we assume that the probability distribution of brightness values for a pixel given the brightness values of its spatial neighborhood is independent of the rest of the image. The neighborhood of a pixel is modeled as a square window around that pixel. The size of the window is a free parameter that specifies how stochastic the user believes this texture to be. More specifically, if the texture is presumed to be mainly regular at high spatial frequencies and mainly stochastic at low spatial frequencies, the size of the window should be on the scale of the biggest regular feature.

3.1.1 Synthesizing one pixel

Let I be an image that is being synthesized from a texture sample image $I_{smp} \subset I_{real}$ where I_{real} is the real infinite texture. Let $p \in I$ be a pixel and let $\omega(p) \subset I$ be a square image patch of width w centered at p. Let $d(\omega_1, \omega_2)$ denote some perceptual distance between two patches. Let us assume for the moment that all pixels in I except for p are known. To synthesize the value of p we first construct an approximation to the conditional probability distribution $P(p|\omega(p))$ and then sample from it.

Based on our MRF model we assume that p is independent of $I \setminus \omega(p)$ given $\omega(p)$.

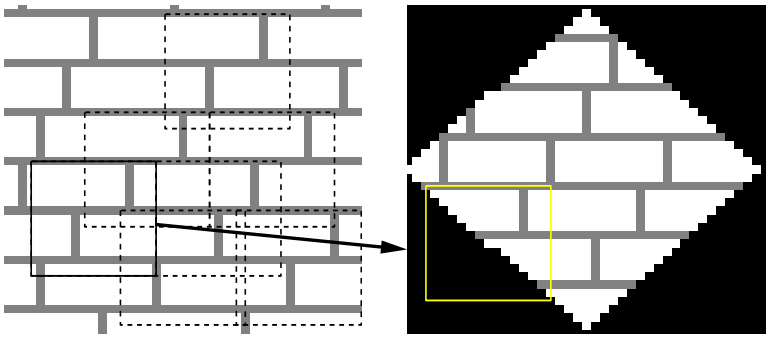


Figure 3.1: Algorithm Overview. Given a sample texture image (left), a new image is being synthesized one pixel at a time (right). To synthesize a pixel, the algorithm first finds all neighborhoods in the sample image (boxes on the left) that are similar to the pixel's neighborhood (box on the right) and then randomly chooses one neighborhood and takes its center to be the newly synthesized pixel.

If we define a set

$$\Omega(p) = \{\omega' \subset I_{real} : d(\omega', \omega(p)) = 0\}$$

containing all occurrences of $\omega(p)$ in I_{real}, then the conditional pdf of p can be estimated with a histogram of all center pixel values in $\Omega(p)$. [1] Unfortunately, we are only given I_{smp}, a finite sample from I_{real}, which means there might not be any matches for $\omega(p)$ in I_{smp}. Thus we must use a heuristic which will let us find a plausible $\Omega'(p) \approx \Omega(p)$ to sample from. In our implementation, a variation of the nearest neighbor technique is used: the closest match $\omega_{best} = argmin_{\omega}\, d(\omega(p), \omega) \subset I_{smp}$ is found, and all image patches ω with $d_(\omega(p), \omega) < (1 + \epsilon) d(\omega(p), \omega_{best})$ are included in $\Omega'(p)$, where $\epsilon = 0.1$ for us. The center pixel values of patches in $\Omega'(p)$ give us a histogram for p, which can then be sampled, either uniformly or weighted by d.

Now it only remains to find a suitable distance d. One choice is a normalized sum of squared differences metric d_{SSD}. However, this metric gives the same weight to any mismatched pixel, whether near the center or at the edge of the window. Since we would like to preserve the local structure of the texture as much as possible, the error for nearby pixels should be greater than for pixels far away. To achieve this effect we set $d = d_{SSD} * G$ where G is a two-dimensional Gaussian kernel.

[1] This is somewhat misleading, since if all pixels in $\Omega(p)$ except p are known, the pdf for p will simply be a delta function for all but highly stochastic textures, since a single pixel can rarely be a feature by itself.

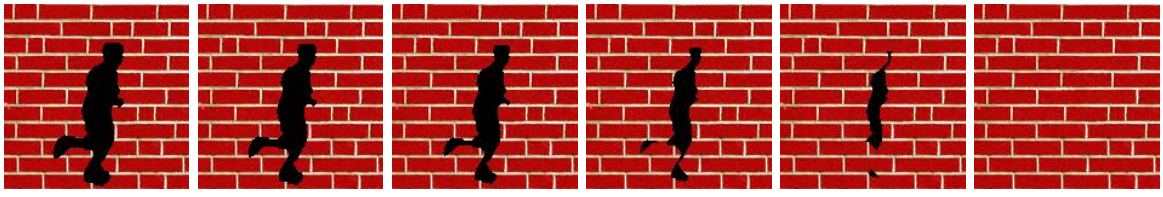


Figure 3.2: Growing Texture. The texture is grown, layer by layer to maximize the number of known neighbors for each synthesized pixel.

3.1.2 Synthesizing texture

In the previous section we have discussed a method of synthesizing a pixel when its neighborhood pixels are already known. Unfortunately, this method cannot be used for synthesizing the entire texture or even for hole-filling (unless the hole is just one pixel) since for any pixel the values of only some of its neighborhood pixels will be known. The correct solution would be to consider the joint probability of all pixels together but this is intractable for images of realistic size.

Instead, a Shannon-inspired heuristic is proposed, where the texture is grown in layers outward from a 3-by-3 seed taken randomly from the sample image (in case of hole filling, the synthesis proceeds from the edges of the hole, as shown on Figure 3.2). Now for any point p to be synthesized only *some* of the pixel values in $\omega(p)$ are known (i.e. have already been synthesized). Thus the pixel synthesis algorithm must be modified to handle unknown neighborhood pixel values. This can be easily done by only matching on the known values in $\omega(p)$ and normalizing the error by the total number of known pixels when computing the conditional pdf for p. This heuristic does not guarantee that the pdf for p will stay valid as the rest of $\omega(p)$ is filled in. However, it appears to be a good approximation in practice. One can also treat this as an initialization step for an iterative approach such as Gibbs sampling. However, our trials have shown that Gibbs sampling produced very little improvement for most textures. This lack of improvement indicates that the heuristic indeed provides a good approximation to the desired conditional pdf.

3.1.3 Algorithm details

Let `SampleImage` contain the image we are sampling from and let `Image` be the mostly empty image that we want to fill in (if synthesizing from scratch, it should contain

```
function GrowImage(SampleImage,Image,WindowSize)
  while Image not filled do
    progress = 0
    PixelList = GetUnfilledNeighbors(Image)
    foreach Pixel in PixelList do
      Template = GetNeighborhoodWindow(Pixel)
      BestMatches = FindMatches(Template, SampleImage)
      BestMatch = RandomPick(BestMatches)
      if (BestMatch.error < MaxErrThreshold) then
        Pixel.value = BestMatch.value
        progress = 1
      end
    end
    if progress == 0
      then MaxErrThreshold = MaxErrThreshold * 1.1
  end
  return Image
end
```

Figure 3.3: The main portion of the pixel-wise texture synthesis algorithm.

a 3-by-3 seed in the center randomly taken from `SampleImage`, for constrained synthesis it should contain all the known pixels). `WindowSize`, the size of the neighborhood window, is the only user-settable parameter. The main portion of the algorithm is presented on Figure 3.3.

Function `GetUnfilledNeighbors()` returns a list of all unfilled pixels that have filled pixels as their neighbors (the image is subtracted from its morphological dilation). The list is randomly permuted and then sorted by decreasing number of filled neighbor pixels. `GetNeigborhoodWindow()` returns a window of size `WindowSize` around a given pixel. `RandomPick()` picks an element randomly from the list. `FindMatches()` is shown on Figure 3.4.

`Gaussian2D()` generates a two-dimensional Gaussian in a window of given a size centered in the center and with a given standard deviation (in pixels). In our implementation the constant were set as follows: `ErrThreshold = 0.1`, `MaxErrThreshold = 0.3`, `Sigma = WindowSize/6.4`.

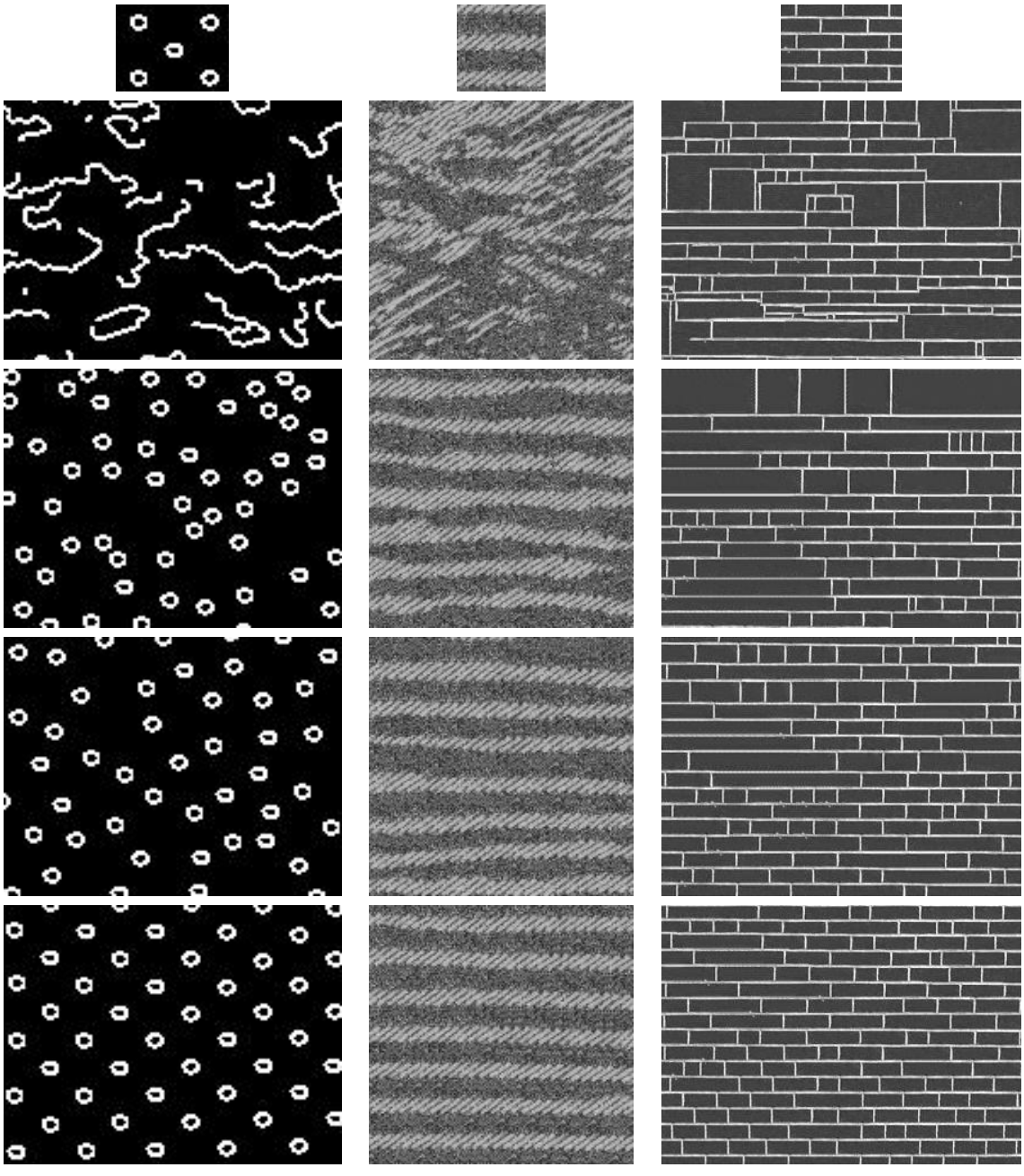
```
function FindMatches(Template,SampleImage)
  ValidMask = 1s where Template is filled, 0s otherwise
  GaussMask = Gaussian2D(WindowSize,Sigma)
  TotWeight = sum i,j GaussiMask(i,j)*ValidMask(i,j)
  for i,j do
    for ii,jj do
      dist = (Template(ii,jj)-SampleImage(i-ii,j-jj))^2
      SSD(i,j) = SSD(i,j) + dist*ValidMask(ii,jj)*GaussMask(ii,jj)
    end
    SSD(i,j) = SSD(i,j) / TotWeight
  end
  PixelList = all pixels (i,j) where SSD(i,j) <= min(SSD)*(1+ErrThreshold)
  return PixelList
end
```

Figure 3.4: Function `FindMatches()` called from `GrowImage()`.

3.2 Results

Our algorithm produces good results for a wide range of textures. The only parameter set by the user is the width w of the context window. This parameter appears to intuitively correspond to the human perception of randomness for most textures. As an example, the image with rings on Figure 3.5a has been synthesized several times while increasing w. In the first synthesized image the context window is not big enough to capture the structure of the ring so only the notion of curved segments is preserved. In the next image, the context captures the whole ring, but knows nothing of inter-ring distances producing a Poisson process pattern. In the third image we see rings getting away from each other (so called Poisson process with repulsion), and finally in the last image the inter-ring structure is within the reach of the window as the pattern becomes almost purely structured.

Figures 3.6 and 3.7 show synthesis examples done on real-world textures. Examples of constrained synthesis are shown on Figure 3.8. The black regions in each image are filled in by sampling from that same image. A comparison with De Bonet [8] at varying randomness settings is shown on Figure 3.9 using texture 161 from his web site.



(a) (b) (c)

Figure 3.5: Results for different neighborhood sizes. Given a sample image (left), the algorithm synthesized four new images with neighborhood windows of width 5, 11, 15, and 23 pixels respectively. Notice how perceptually intuitively the window size corresponds to the degree of randomness in the resulting textures. Input images are: (a) synthetic rings, (b) Brodatz texture D11, (c) brick wall.

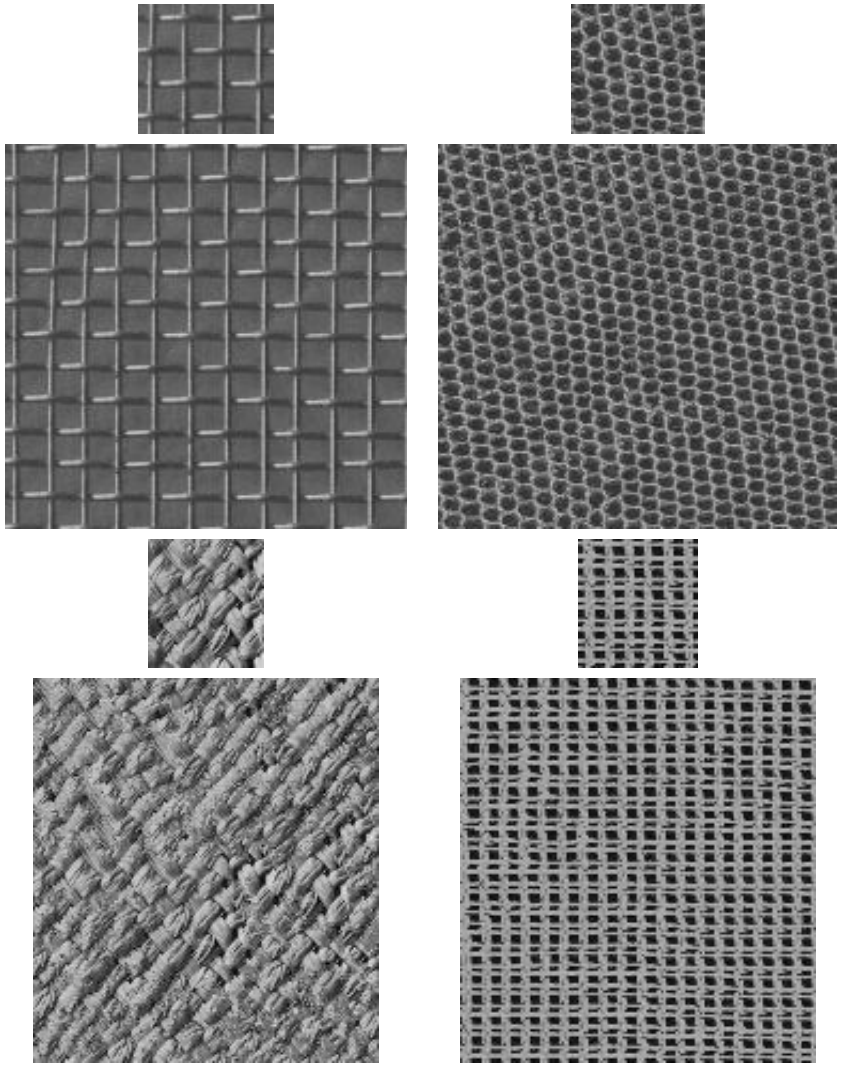


Figure 3.6: Texture synthesis results on real-world textures. Original images (top), and synthesized (bottom). Input images are D1, D3, D18, and D20 from the Brodatz collection [11]

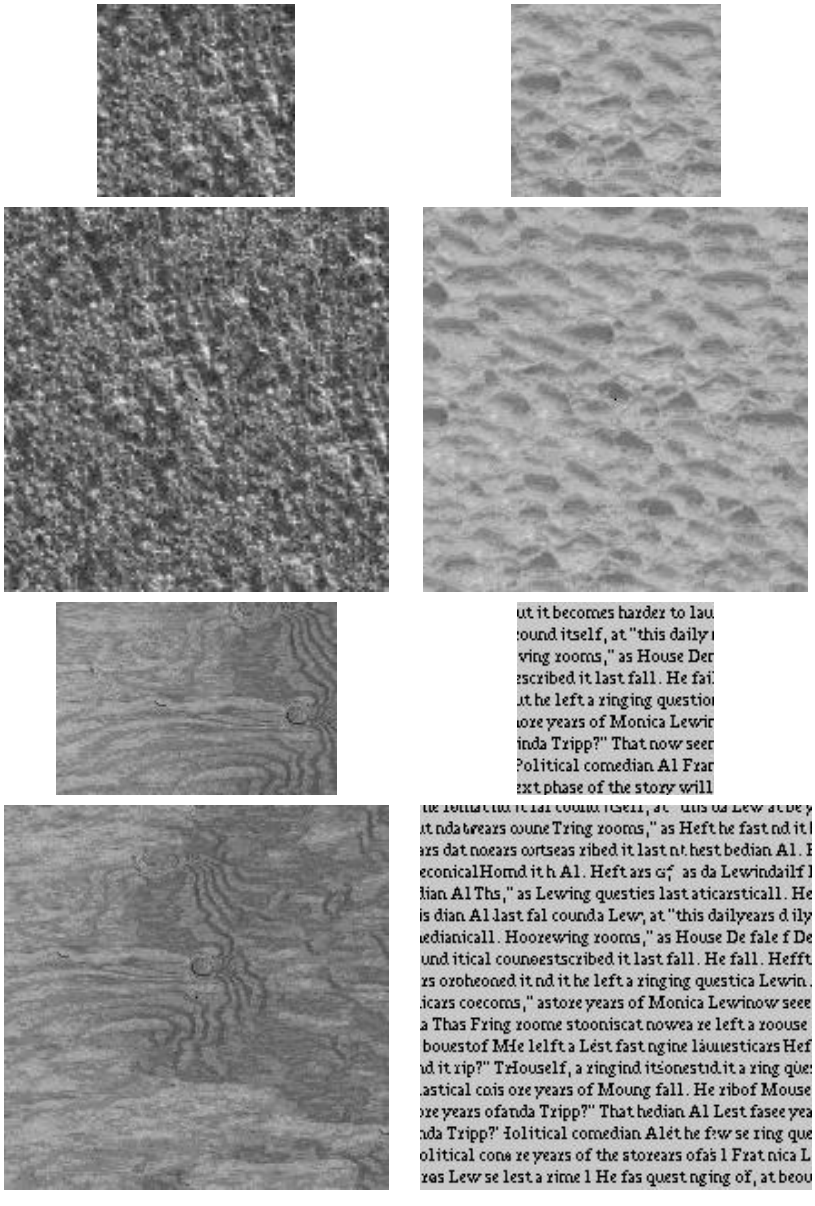


Figure 3.7: More texture synthesis results on real-world textures. Original images (top) and synthesized (bottom). The input images are granite, bread, wood, and text (a homage to Shannon).

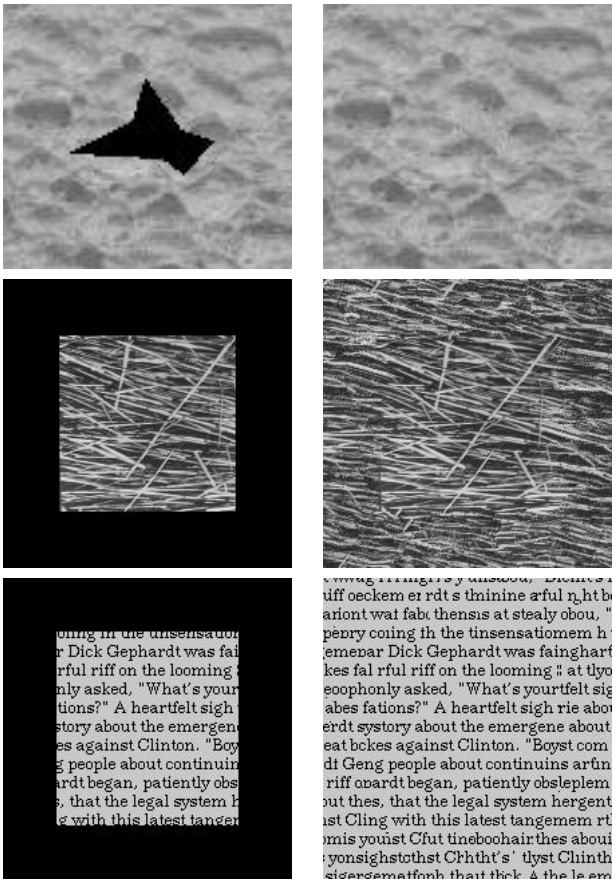


Figure 3.8: Examples of constrained texture synthesis. The synthesis process fills in the black regions.

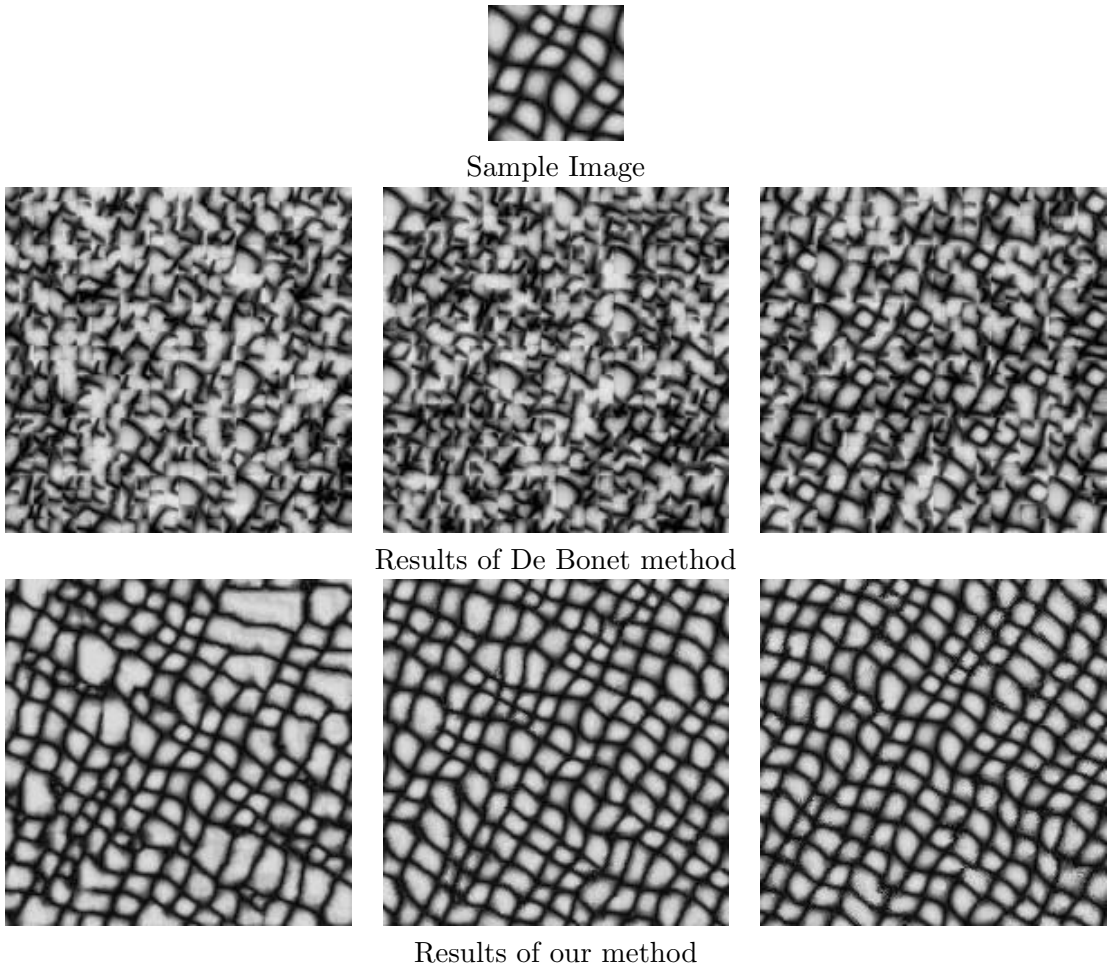


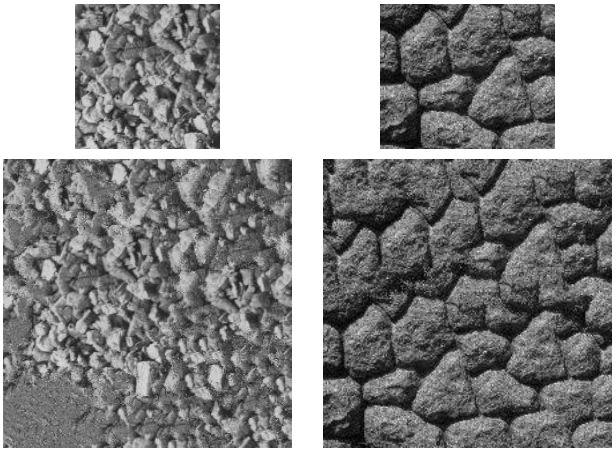
Figure 3.9: Comparison of results. Texture synthesized from sample image (top) with our method (middle) compared to De Bonet [8] (bottom) at decreasing degree of randomness (left to right).

3.3 Limitations

As with most texture synthesis procedures, only frontal-parallel textures are handled. However, it is possible to use Shape-from-Texture techniques [24, 38] to pre-warp an image into frontal-parallel position before synthesis and post-warp afterward.

One problem of our algorithm is its tendency for some textures to occasionally "slip" into a wrong part of the search space and start growing garbage (Figure 3.10a) or get locked onto one place in the sample image and produce verbatim copies of the original (Figure 3.10b). These problems occur when the texture sample contains too many different types of texels (or the same texels but differently illuminated) making it hard to find close matches for the neighborhood context window. These problems can usually be eliminated by providing a bigger sample image. We have also used growing with limited backtracking as a solution.

A major problem of he algorithm is that it is very slow since it needs to search the entire input image each time a pixel is synthesized. One way to speed it up is to use a multiscale approach with smaller window sizes. However, in our trials, the quality of the resulting textures deteriorated drastically because the high-frequency shift information was largely being lost in the multiscale representation (this is especially apparent on the high-frequency textures such as the text image). We also attempted to speed-up the search itself by performing it in the Fourier domain. In the image domain, the neighborhood search is typically computed using two convolution operations (the exact formula can be easily derived by expanding the SSD metric $\sum_{\omega}(A-B)^2$). But note that a convolution in the image domain is equivalent to multiplication in the Fourier domain. Therefore, the two convolutions can be done by two multiplications plus 6 FFT operations (and since the input image doesn't change, only 4 FFTs are actually required at each iteration). In our experiments, however, the FFT-based search performed worse than the original two convolutions approach. The reason for this probably lies in the fact that the convolution kernel (the neighborhood patch) is typically much smaller than the input image. Since the FFT procedure pads the kernel to be the size of the input image, the data overhead can become quite large, compared to a well-optimized convolution routine.



(a) (b)

Figure 3.10: Failure examples. Sometimes the growing algorithm "slips" into a wrong part of the search space and starts growing garbage (a), or gets stuck at a particular place in the sample image and starts verbatim copying (b).

3.4 Applications

Apart from letting us gain a better understanding of texture models, texture synthesis can also be used as a tool for solving several practical problems in computer vision, graphics, and image processing. Our method is particularly versatile because it does not place any constraints on the shape of the synthesis region or the sampling region, making it ideal for constrained texture synthesis such as hole-filling. Moreover, our method is designed to preserve local image structure, such as continuing straight lines, so there are no visual discontinuities between the original hole outline and the newly synthesized patch.

For example, capturing a 3D scene from several camera views will likely result in some regions being occluded from all cameras [17]. Instead of letting them appear as black holes in a reconstruction, a localized constrained texture synthesis can be performed to fill in the missing information from the surrounding region. As another example, consider the problem of boundary handling when performing a convolution on an image. Several methods exist, such as zero-fill, tiling and reflection, however all of them may introduce discontinuities not present in the original image. In many cases, texture synthesis can be used to extrapolate the image by sampling from itself as shown on Figure 3.11.

The constrained synthesis process can be further enhanced by using image segmentation to find the exact sampling region boundaries. A small patch of each region can

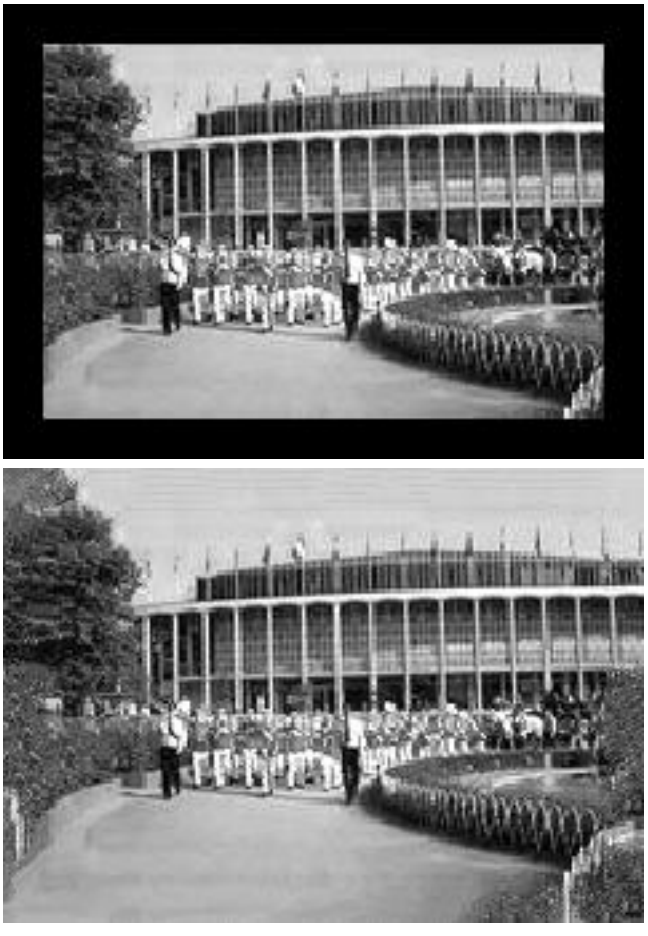


Figure 3.11: Image Extrapolation. The texture synthesis algorithm is applied to a real image (top) extrapolating it using itself as a model, to result in a larger image (bottom) that, for this particular image, looks quite plausible. This technique can be used in convolutions to extend filter support at image boundaries.

then be stored together with region boundaries as a lossy compression technique, with texture synthesis being used to restore each region separately. If a figure/ground segmentation is possible and the background is texture-like, then foreground removal can be done by synthesizing the background into the foreground segment.

Our algorithm can also easily be applied to motion synthesis such as ocean waves, rolling clouds, or burning fire by a trivial extension to 3D.

3.5 Follow-up Work

When first published in 1999, this algorithm generated considerable enthusiasm in the computer vision and graphics communities. Since then, several researchers have proposed optimizations and extensions to the algorithm. Here is an attempt to list some of them.

The original algorithm can be excruciatingly slow when synthesizing large textures (a full search of the input image is required to synthesize every pixel!). Several researchers have proposed optimizations to the basic pixel search method including a pyramid-based multiscale framework [60], neighborhood clustering (using TSVQ [60] or approximate-nearest-neighbor [19]) and neighborhood coherence [2, 57]. Note, however, that none of the above techniques can guarantee to find the true nearest neighbors for a patch (and some can't even give an ϵ bound). While this is not a huge problem for simple textures (such as the synthetic pattern on Figure 3.9), on more complex textures this can substantially degrade the quality of the results (see [60] and Figure 4.9).

Harrison [27] proposed an interesting method for finding the optimal ordering of pixels to be synthesized based on Information Theory. In a recent elegant paper, Criminisi el al. [13] showed that an ordering based on prioritizing strong edges is enough to successfully fill in holes in arbitrary (not just texture!) images. Zhang et al. [66] provide a method of synthesizing progressively-variant textures by utilizing a user-specified texton map. Finally, in the field of theoretical statistics, Levina [34] proved consistency of our algorithm under certain assumptions.

Chapter 4

Image Quilting: Patch-based Synthesis Algorithm

...but even what we imagine is but a crazy-quilt of bits snipped off from actual experiences.

David Hume
Law of Nature

One curious fact about one-pixel-at-a-time synthesis algorithms presented in the previous chapter is that for most complex textures very few pixels actually have a choice of values that can be assigned to them. That is, during the synthesis process most pixels have their values totally determined by what has been synthesized so far. As a simple example, let us take a pattern of circles on a plane. Once the algorithm has started synthesizing a particular circle, all the remaining pixels of that circle (plus some surrounding ones) are completely determined! In this extreme case, the circle would be called the texture element (*texel*), but this same effect persists to a lesser extent even when the texture is more stochastic and there are no obvious texels. This means that a lot of searching work is waisted on pixels that already "know their fate". It seems then, that the unit of synthesis should be something more than a single pixel, a "patch" perhaps. Then the process of texture synthesis would be akin to putting together a jigsaw puzzle, quilting together the patches, making sure they all fit together. Determining precisely what are the patches for a

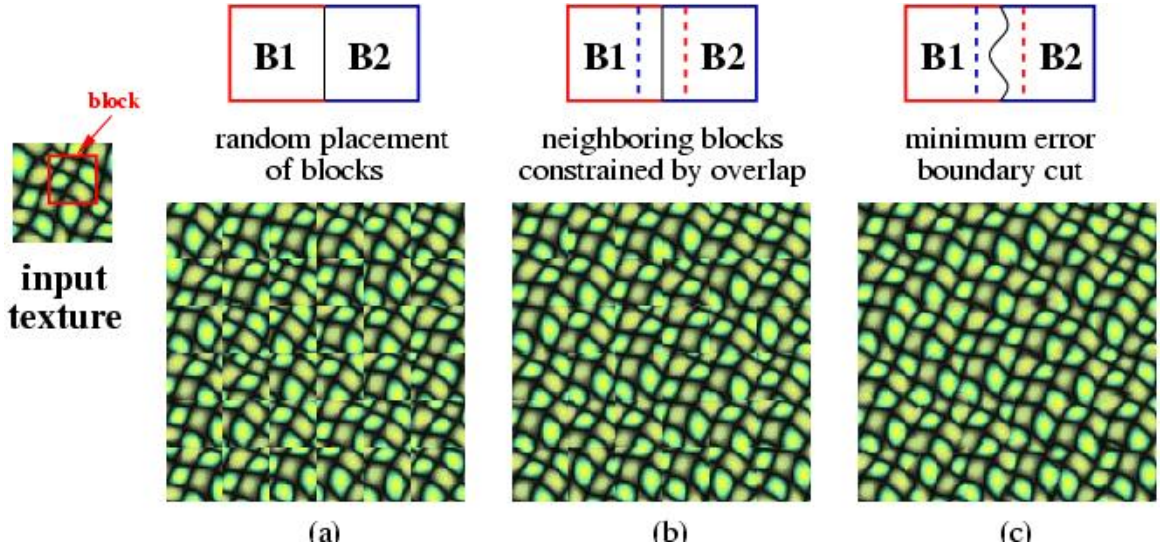


Figure 4.1: Quilting texture. Square blocks from the input texture are patched together to synthesize a new texture sample: (a) blocks are chosen randomly (similar to [62, 45]), (b) the blocks overlap and each new block is chosen so as to "agree" with its neighbors in the region of overlap, (c) to reduce blockiness the boundary between blocks is computed as a minimum cost path through the error surface at the overlap.

given texture and how they are put together is still an open problem. Here we will present an very naive version of stitching together patches of texture to form the output image. We call this method "image quilting".

4.1 Image Quilting

In this section we will develop our patch-based texture synthesis procedure. Let us define the unit of synthesis B_i to be a square block of user-specified size from the set S_B of all such overlapping blocks in the input texture image. To synthesize a new texture image, as a first step let us simply tile it with blocks taken randomly from S_B. The result shown on Figure 4.1(a) already looks somewhat reasonable and for some textures will perform no worse than many previous complicated algorithms as demonstrated by [62, 45]. Still, the result is not satisfying, for no matter how much smoothing is done across the edges, for most structured textures it will be quite obvious that the blocks do not match.

As the next step, let us introduce some overlap in the placement of blocks onto the new image. Now, instead of picking a random block, we will search S_B for such a block that

by some measure agrees with its neighbors along the region of overlap. Figure 4.1(b) shows a clear improvement in the structure of the resulting texture, however the edges between the blocks are still quite noticeable. Once again, smoothing across the edges will lessen this problem but we will attempt to solve it in a more principled way.

Finally, we will let the blocks have ragged edges which will allow them to better approximate the features in the texture. Now, before placing a chosen block into the texture we will look at the error in the overlap region between it and the other blocks. We find a minimum cost path through that error surface and declare that to be the boundary of the new block. Figure 4.1(c) shows the results of this simple modification.

4.1.1 Minimum Error Boundary Cut

We want to make the cut between two overlapping blocks on the pixels where the two textures match best (that is, where the overlap error is low). This can easily be done with dynamic programming (Dijkstra's algorithm can also be used [16], as well as a more sophisticated segmentation technique [32]).

The minimal cost path through the error surface is computed in the following manner. If B_1 and B_2 are two blocks that overlap along their vertical edge (Figure 4.1c) with the regions of overlap B_1^{ov} and B_2^{ov}, respectively, then the error surface is defined as $e = (B_1^{ov} - B_2^{ov})^2$. To find the minimal vertical cut through this surface we traverse e ($i = 2..N$) and compute the cumulative minimum error E for all paths:

$$E_{i,j} = e_{i,j} + \min(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}). \tag{4.1}$$

and what these paths are:

$$MinPaths_{i,j} = \text{argmin}(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}). \tag{4.2}$$

In the end, the minimum value of the last row in E will indicate the end of the minimal vertical path though the surface and one can trace back and find the path of the best cut. Similar procedure can be applied to horizontal overlaps. When there is both a vertical and a horizontal overlap, the minimal paths meet in the middle and the overall minimum is chosen for the cut. That is, one first finds a path in the vertical region and the horizontal region separately, going *into* the corner. At the corner the two paths meet and one needs to find the best place for them to intersect. One simple way is to look at the

dynamic programming values on the *diagonal* of the corner for both paths. Adding the two diagonals together and finding the maximum will indicate the place where the two paths should meet.

4.1.2 The Image Quilting Algorithm

The complete quilting algorithm is as follows:

- Go through the image to be synthesized in raster scan order in steps of one block (minus the overlap).
- For every location, search the input texture for a set of blocks that satisfy the overlap constraints (above and left) within some error tolerance. Randomly pick one such block.
- Compute the error surface between the newly chosen block and the old blocks at the overlap region. Find the minimum cost path along this surface and make that the boundary of the new block. Paste the block onto the texture. Repeat.

The size of the block is the only parameter controlled by the user and it depends on the properties of a given texture; the block must be big enough to capture the relevant structures in the texture, but small enough so that the interaction between these structures is left up to the algorithm.

In all of our experiments the width of the overlap edge (on one side) was 1/6 of the size of the block. The error was computed using the $L2$ norm on pixel values. The error tolerance was set to be within 0.1 times the error of the best matching block.

4.1.3 Synthesis Results

The results of the synthesis process for a wide range of input textures are shown on Figures 4.2-4.8. While the algorithm is particularly effective for semi-structured textures (which were always the hardest for statistical texture synthesis), the performance is quite good on stochastic textures as well. The two most typical problems are excessive repetition (e.g. the berries image), and mismatched or distorted boundaries (e.g. the mutant olives image). Both are mostly due to the input texture not containing enough variability. Figure 4.9 shows a comparison of quilting with other texture synthesis algorithms.

The algorithm is not only trivial to implement but is also quite fast: the unoptimized MATLAB code used to generate these results ran for between 15 seconds and several minutes per image depending on the sizes of the input and output and the block size used. Because the constraint region is always the same it's very easy to optimize the search process without compromising the quality of the results (see also Liang et.al. [35] who report real-time performance using a very similar, independently developed approach).

4.2 Conclusion

In this chapter we have introduced *image quilting*, a method of synthesizing a new image by stitching together small patches of existing images. Despite its simplicity, this method works remarkably well when applied to texture synthesis, producing results that are equal or better than the pixel-based algorithm of the previous chapter but with improved stability (less chance of "growing garbage") and at a fraction of the computational cost.

Besides many practical applications, we think this work also provides new insights into the way texture is perceived. The fact that crude patches of texture stitched together with a few constraints work just as well as meticulously synthesizing each pixel by itself is a very non-trivial observation. It clearly shows the enormous importance of local structure for successful modeling of visual phenomena.

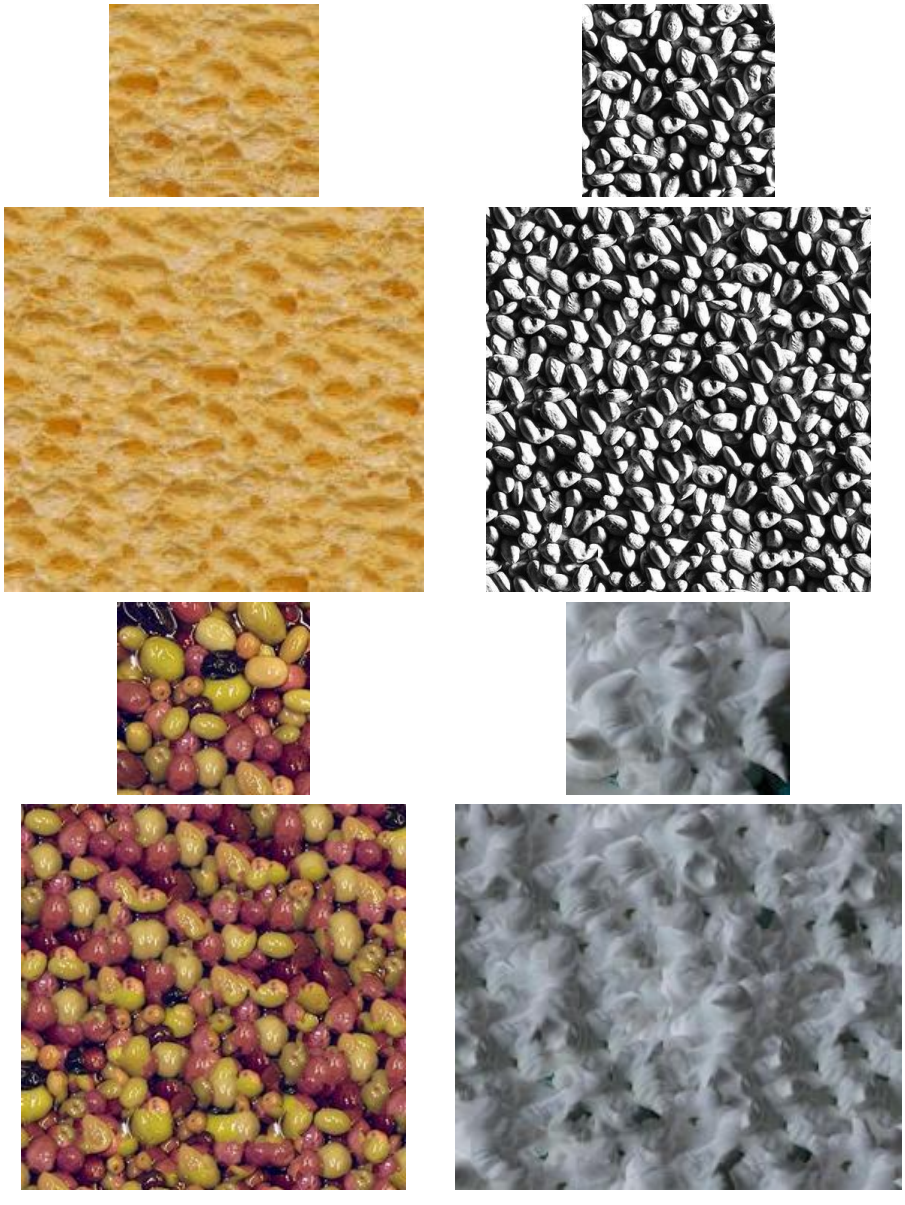


Figure 4.2: Image quilting synthesis results for a wide range of textures.. The resulting texture (top) is synthesized at twice the size of the original (bottom).

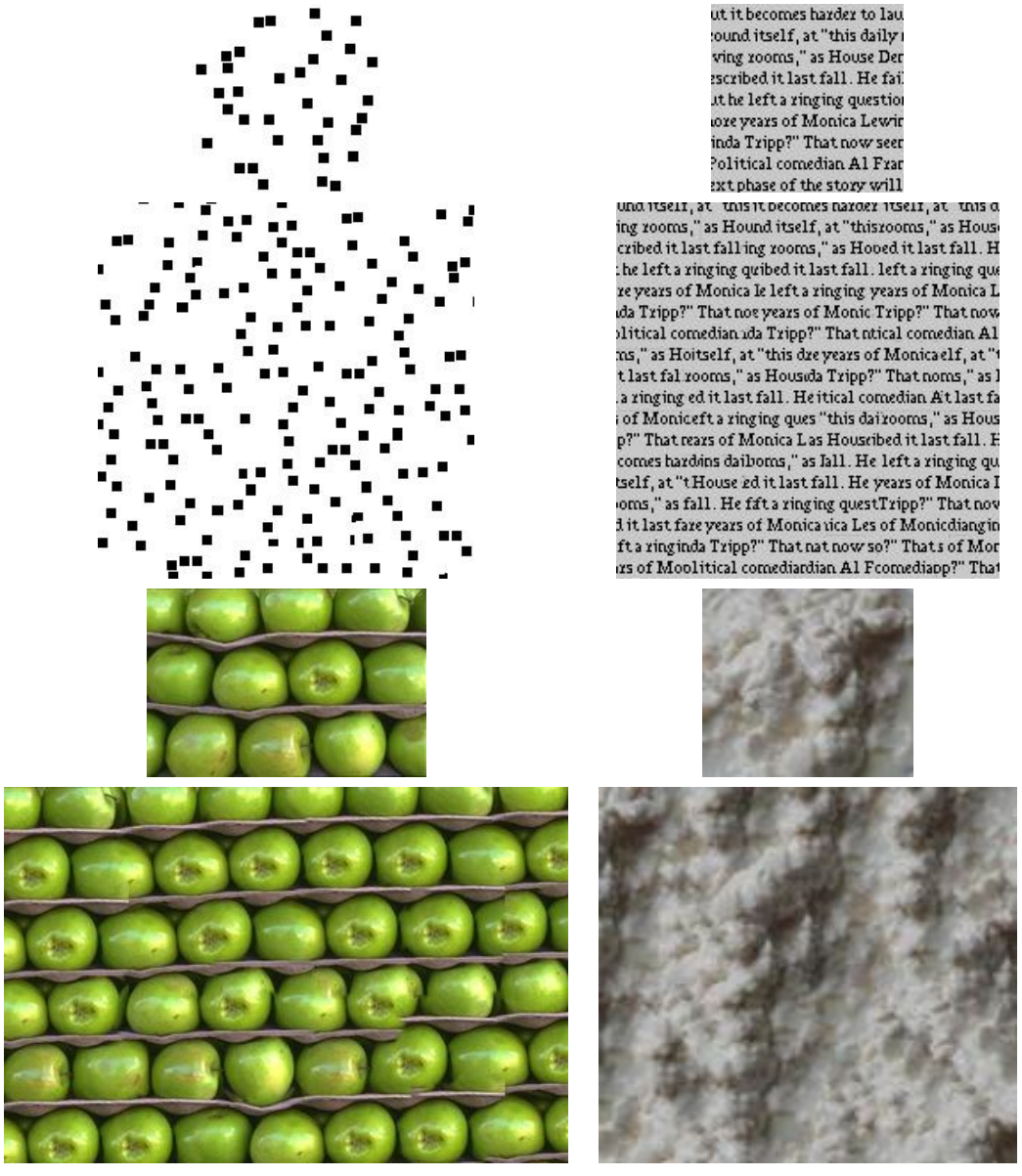


Figure 4.3: More Image Quilting results

Figure 4.4: More Image Quilting results

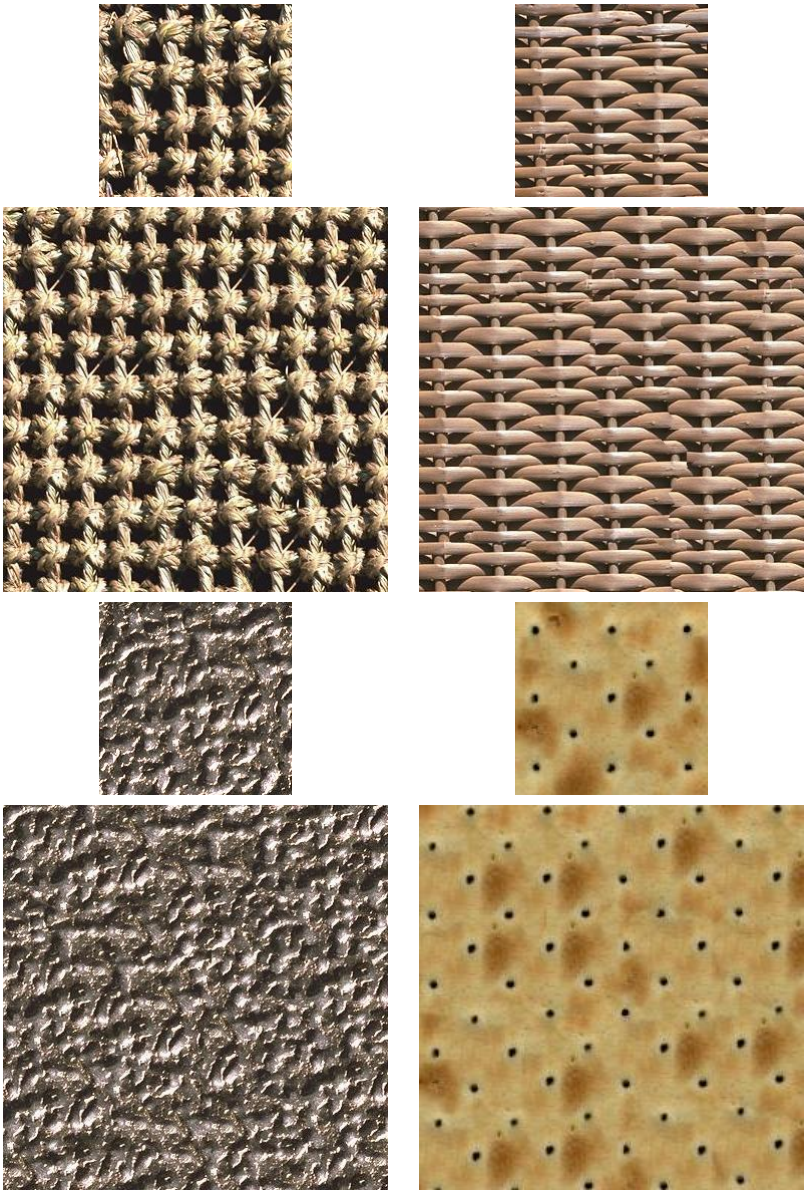


Figure 4.5: More Image Quilting results

Figure 4.6: More Image Quilting results

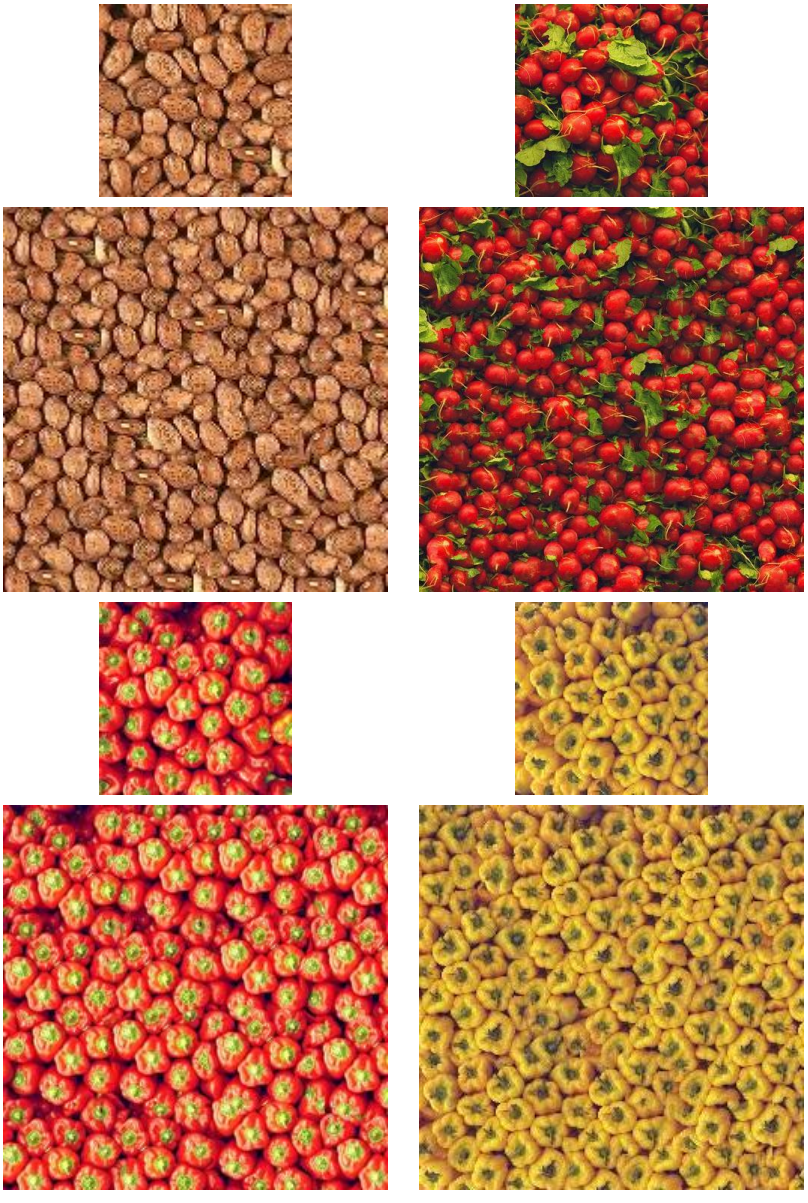


Figure 4.7: More Image Quilting results

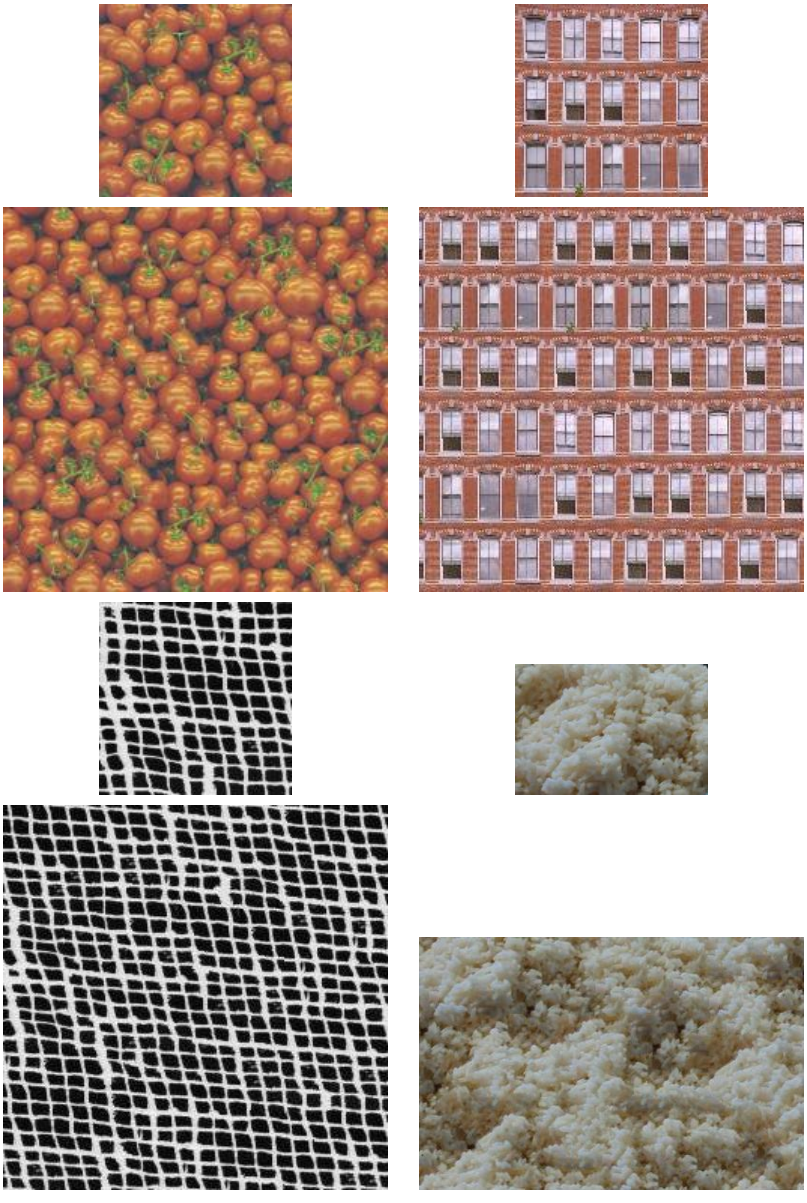


Figure 4.8: More Image Quilting results

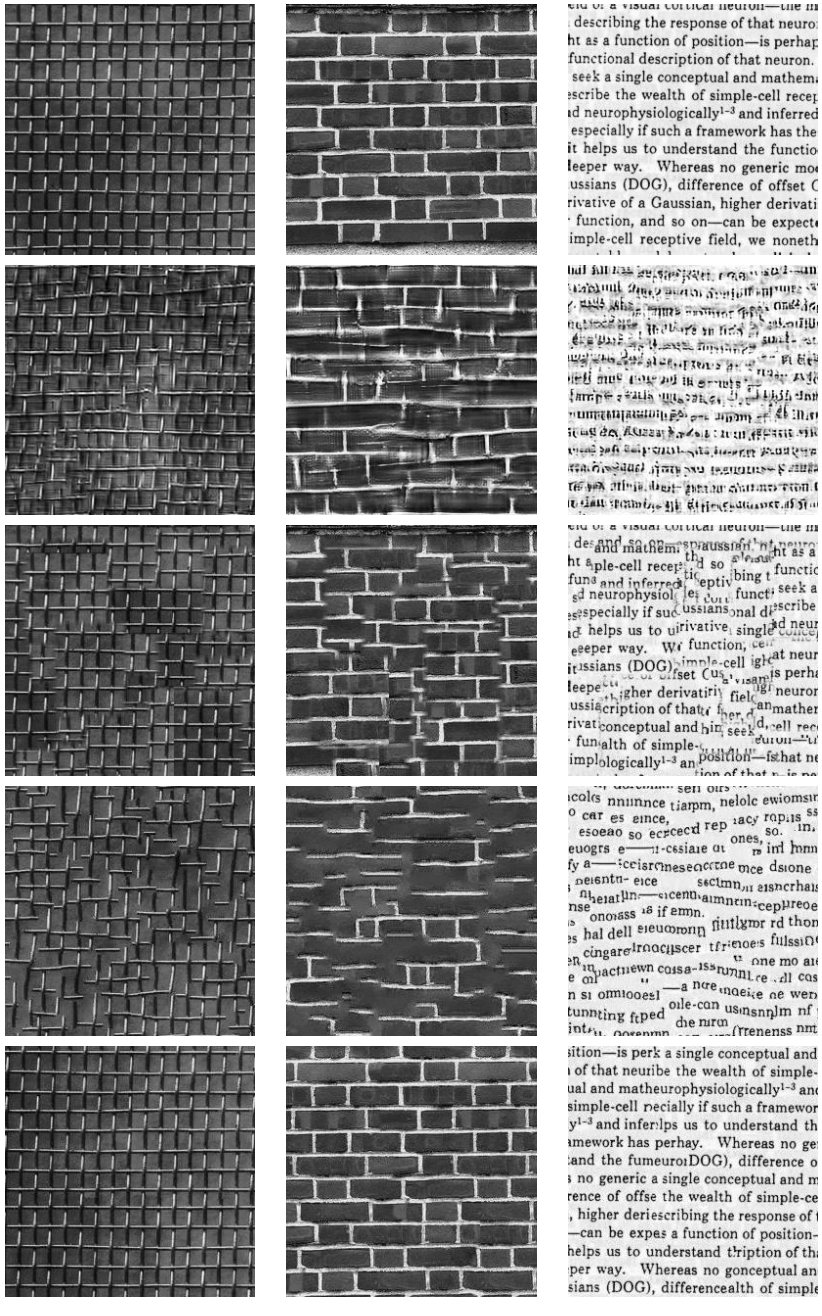


Figure 4.9: Comparison of various texture synthesis methods on structured textures.. Image Quilting results are virtually the same as Efros & Leung [20] (not shown) but at a much smaller computational cost.

Chapter 5

Application: Texture Transfer

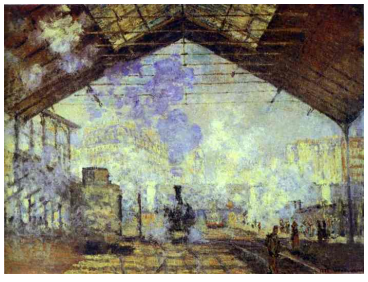
It is not well known but Nielsen used the same notes as Bach, but with some considerable modification to their ordering.

John Butt
(in the program notes
for a Bach and Neilsen recital)

An object can be though of as a product of two processes – *style* and *content*. Style is a generic property that the object shares with all other objects of its class, while content describes what is unique about this particular object. For example, Monet's *Gare St. Lazare* has an impressionist style that it shares with other Monet paintings, while its content is the depiction of the train station in Paris (Figure 5.1). Many natural phenomena can be thought of in a similar way: a potato's style is its skin, dimples, color – all the properties that define its "potatoness", while its particular shape is the content.

The problem of discovering and disentangling style and content in visual imagery has been studied in computer vision (e.g. [22]). While some success has been reported on a few examples, including the letter/font problem proposed by Hofstadter [30], the general solution remains illusive. Here we will not attempt to solve this problem. Instead, we will focus on providing a partial solution to a simpler problem – synthesizing an image of an object in the style of another object.

We begin with a simple observation: for a large class of images, the process we defined as "style" is largely local (on a scale of small image patch) with "content" referring


Gare St. Lazare by Monet


style


content

Figure 5.1: Style and Content. Monet's *Gare St. Lazare* (left) has an impressionist style that it shares with other Monet paintings (e.g. middle), while its content is the depiction of the St. Lazare train station in Paris (right)

primarily to the overall impression from the entire image. Our idea, then, is to *treat style as texture* and to modify our texture synthesis algorithm to transfer texture from one image onto another (see Figure 5.2). We call this process *texture transfer*.

Of course, equating "style" with texture is highly inaccurate, even absurd from the artistic point of view. Indeed, looking at Monet's painting on Figure 5.1 it is clear that his style is much more than just the famous impressionistic brush-strokes, encompassing mood, light, and the overall feeling of the scene. Therefore, we would like to explicitly restrict the applicability of our method to such cases where the style = texture assumption is justified.

Methods have been developed in particular rendering domains which capture the spirit of our goals in texture transfer. Our goal is like that of work in non-photorealistic rendering (e.g. [14, 48, 41]). A key distinction is that we seek to characterize the output rendering style by sampling from the real world. This allows for a richness of rendering styles, characterized by samples from photographs or drawings.

Three concurrent papers, all developed independently, are closely related to our work [27, 2, 19]. All three propose the idea of texture transfer based on variations of our pixel-based synthesis algorithm (in particular, see the elegant paper by Hertzmann et.al. [19]). In Section 5.1.1 we will discuss the strengths and weaknesses of our method in relation to theirs.

input images quilting results

Figure 5.2: Image Quilting for texture synthesis and texture transfer. Using the rice texture image (upper left), we can synthesize more such texture (upper right). We can also transfer the rice texture onto another image (lower left) for a strikingly different result.

5.1 Texture Transfer

For texture transfer, we have chosen to use our image quilting algorithm (Chapter 4) over the pixel-based algorithm (Chapter 3). Because the "style" we are attempting to transfer from one object on to another can only loosely be called texture, a patch-based approach will work better at preserving the local appearance of the image (see a comparison with a pixel-based method in Section 5.1.1). We augment the synthesis algorithm by requiring that each patch satisfy a desired *correspondence map*, $\vec{C}$, as well as satisfy the texture synthesis requirements. The correspondence map is a spatial map of some corresponding quantity over both the texture source image and a controlling target image. That quantity could include image intensity, blurred image intensity, local image orientation angles, or other derived quantities.

An example of texture transfer is shown in Figure 5.2. Here, the correspondence map are the (luminance) image intensities of the man's face. That is, bright patches of face and bright patches of rice are defined to have a low correspondence error. The synthesized rice texture conforms to this second constraint, yielding a rendered image where the face image appears to be rendered in rice.

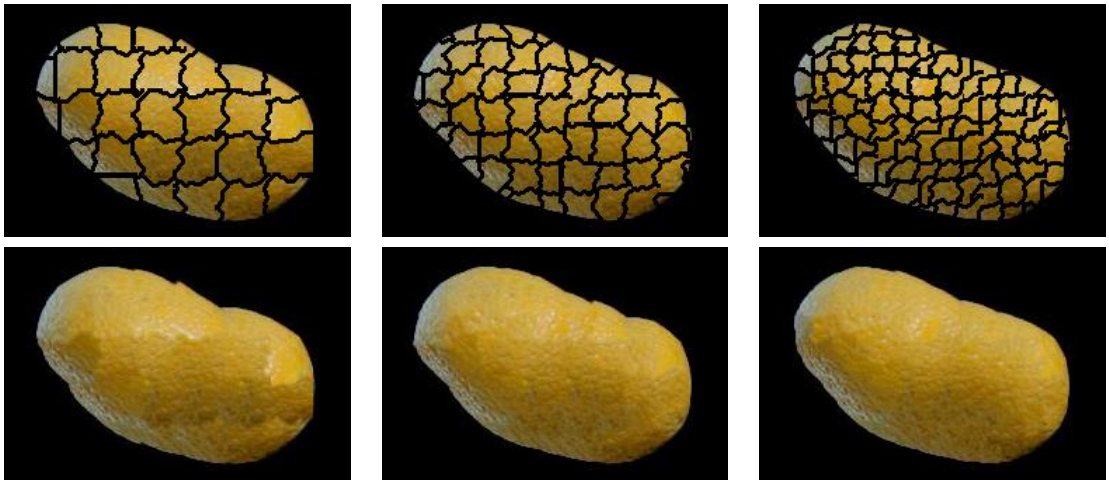


Figure 5.3: Iteration for Texture Transfer. Sometimes, one pass is not enough. Here the texture from an Orange is being transfered on to a potato. The potato is figure drawn with large orange patches, and then the patch size is gradually decreased. Top row shows the selected patched, and bottom row shows the result after quilting

For texture transfer, image being synthesized must respect two independent constraints: (a) the output are legitimate, synthesized examples of the source texture, and (b) that the correspondence image mapping is respected. We modify the error term of the image quilting algorithm to be the weighted sum, α times the block overlap matching error plus $(1 - \alpha)$ times the squared error between the correspondence map pixels within the source texture block and those at the current target image position. The parameter α determines the trade-off between the texture synthesis and the fidelity to the target image correspondence map.

More precisely, Let $\vec{l}(B_i)$ be a vector of the top and left boundary pixels of source texture patch B_i. Let $\vec{L}$ be the corresponding patch overlap pixels of the image created so far at the output patch. $\vec{C}$ is a vector of the correspondence map elements at this same output patch position, and $\vec{c}(B_i)$ is a vector of the elements of the correspondence map for source texture patch B_i. Then for the output patch of the texture transfer algorithm, we seek to minimize the weighted error, Err:

$$Err = \alpha|\vec{l}(B_i) - \vec{L}|^2 + (1 - \alpha)|\vec{c}(B_i) - \vec{C}|^2, \tag{5.1}$$

where α is a parameter that determines the trade-off between the texture synthesis and the fidelity of the synthesized image to the target correspondence map.

Because of the added constraint, sometimes one synthesis pass through the image

is not enough to produce a visually pleasing result. In such cases, we iterate over the synthesized image several times, reducing the block size with each iteration (see Figure 5.3). The only change from the non-iterative version is that in satisfying the local texture constraint the blocks are matched not just with their neighbor blocks on the overlap regions, but also with whatever was synthesized at this block in the previous iteration. This iterative scheme works surprisingly well: it starts out using large blocks to roughly assign where everything will go and then uses smaller blocks to make sure the different textures fit well together. In our tests, we used $N = 3$ to $N = 5$ iterations, reducing the block size by a third each time, and setting α at the ith iteration to be $\alpha_i = 0.8 * \frac{i-1}{N-1} + 0.1$.

Our texture transfer method can be applied to render a photograph using the line drawing texture of a particular source drawing (Figure 5.5); or to transfer material surface texture onto a new image (see Figure 5.4). For the orange texture the correspondence maps are the source and target image luminance values; for the line drawing the correspondence maps are the blurred luminance values.

5.1.1 Comparison to Pixel-based Approaches

Concurrent with our work, three other sets of researcher have independently developed the notion of texture transfer [27, 2, 19]. All of them are based on variations of our pixel-based synthesis algorithm (Chapter 3), whereas we have used the Image Quilting algorithm (Chapter 4) instead. Of these, the work of Hertzmann et al. [19]), termed Image Analogies, is the most developed and comprehensive system. Figure 5.6 shows a comparison between our method (left) and Image Analogies (right). As can be seen, the Image Analogies result looks closer to the target photograph but does not reproduce the source texture very faithfully. Our method, on the other hand, sticks close to the source texture, at the cost of looking less like the target photograph. This clearly demonstrates the trade-off of the two approaches: patch-based methods are better at preserving the original structure of the source texture, while the pixel-based methods are able to better adapt the texture to the target image.

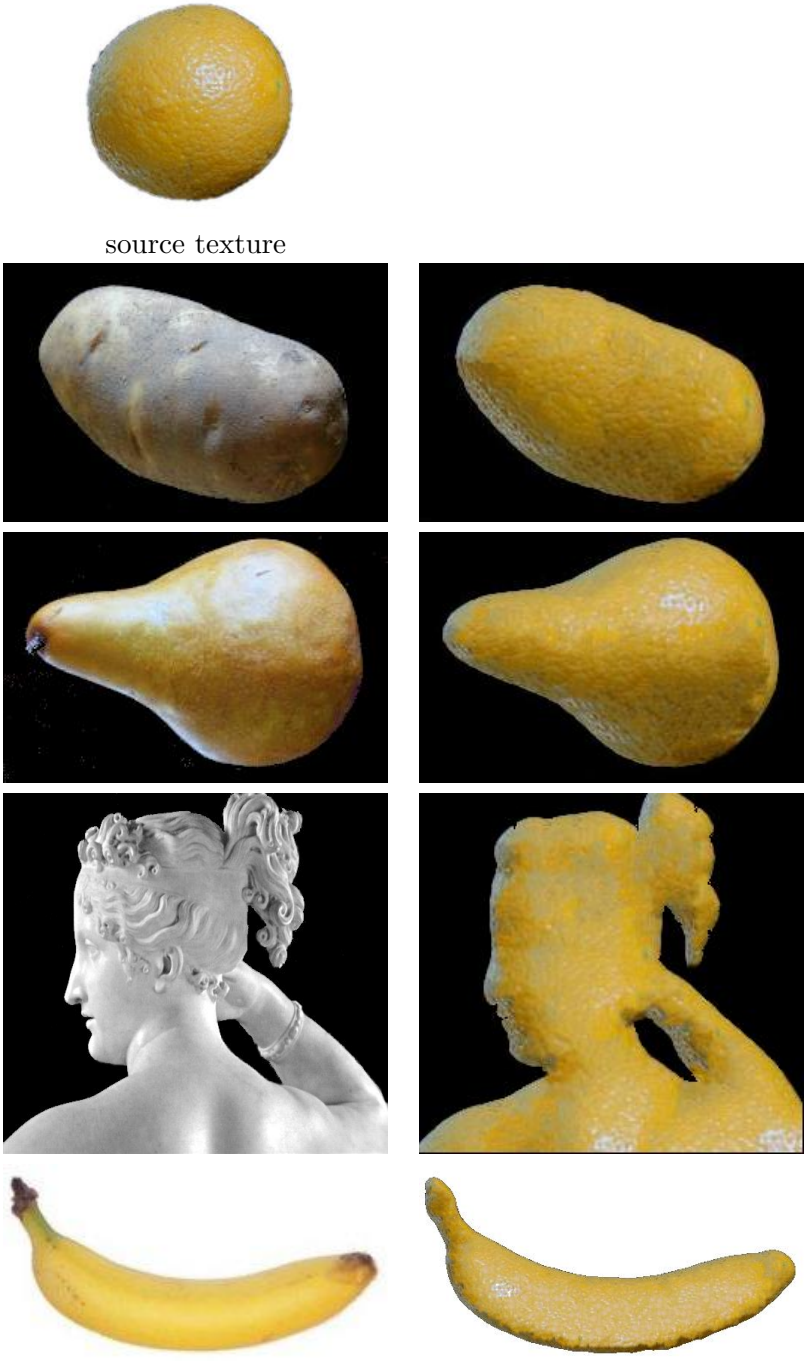
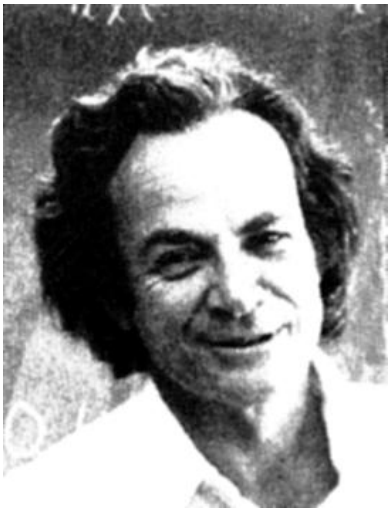


Figure 5.4: Texture transfer. Here, we take the texture from a orange and transfer it onto different objects. The result has the texture of the source image and the correspondence map values of the target image.

source texture



target image

correspondence maps

texture transfer result

Figure 5.5: Texture Transfer with different correspondence map. Here a Picasso drawing is taken as a source texture (upper left) and applied to re-render a photograph of Richard Feynman (lower left) in a line-drawing style. A low-pass filtered version of the drawing is taken as the source correspondence map. The Feynman photo is used as its own correspondence map. The result (lower right) definitely shows the influence of Picasso – here Dr. Feynman is no longer smiling.

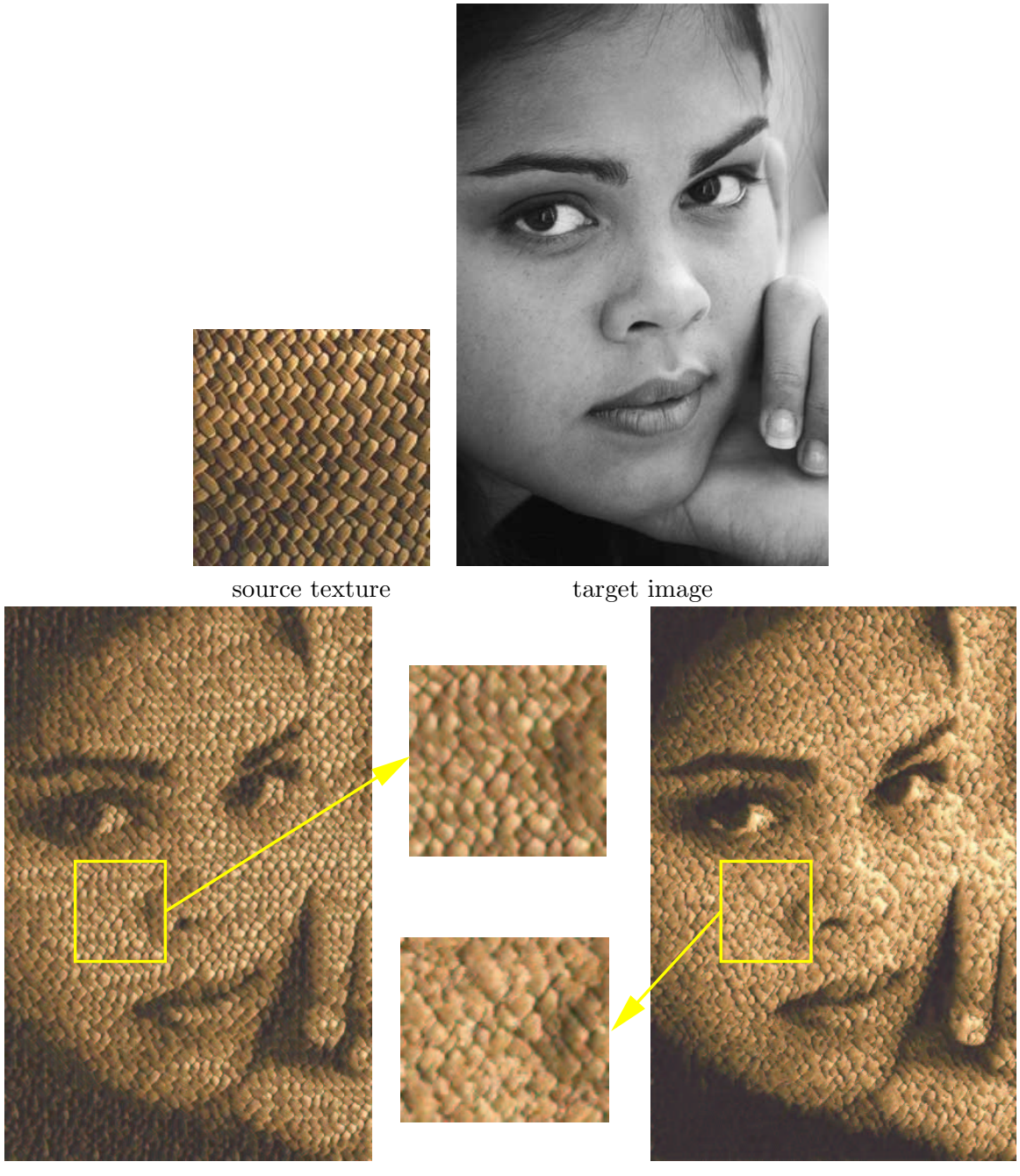


Figure 5.6: Comparison to Image Analogies. Given a source texture and a target image above, the results of using IMAGE QUILTING (left) and IMAGE ANALOGIES (right) are presented below. Note the zoomed-in patterns of texture in the center.

5.2 Other Applications

The popularity of the texture synthesis approaches described in the previous chapters inspired several other applications in computer graphics and computer vision. Here we will mention a few of them.

As mentioned in Chapter 3, texture synthesis can be used for "growing" texture onto non-developable 3D objects. A number of researchers have since pursued this idea based on variants of our pixel-based algorithm [64, 58, 61, 57, 66] as well as Image Quilting [54]. Several authors proposed using the same methods for synthesizing things other than visual textures, including: Bidirectional Texture Functions (BTFs) [57], displacement and transparency maps [64], bump maps [33] and curves [29]. In computer vision similar methods have been applied to texture classification and segmentation [1, 9].

Part II

Human Actions

Chapter 6

Action Recognition at a Distance

It irritated him that the "dog" of 3:14 in the afternoon, seen in profile, should be indicated by the same noun as the dog of 3:15, seen frontally.

"My memory, sir, is like a garbage heap"

Jorge Luis Borges
Fumes, his memory

Consider video such as the wide angle shot of a football field seen in Figure 6.1. People can easily track individual players and recognize actions such as running, kicking, jumping etc. This is possible in spite of the fact that the resolution is not high – each player might be, say, just 30 pixels tall. How do we develop computer programs that can replicate this impressive human ability?

It is useful to contrast this medium resolution regime with two others: ones where the figures are an order of magnitude taller ("near" field), or an order of magnitude shorter ("far" field). In near field (Figure 6.2a), we may have 300 pixel tall figures, and there is reasonable hope of being able to segment and label parts such as the limbs, torso, and head, and thus mark out a stick figure. Strategies such as [63, 47, 46] work best when we have data that support figures of this resolution. On the other hand, in far field (Figure 6.2b), we might have only 3 pixel tall figures – in this case the best we can do is to track the figure as a "blob" without the ability to articulate the separate movements of the different locations in it. Blob tracking is good enough for applications such as measuring pedestrian

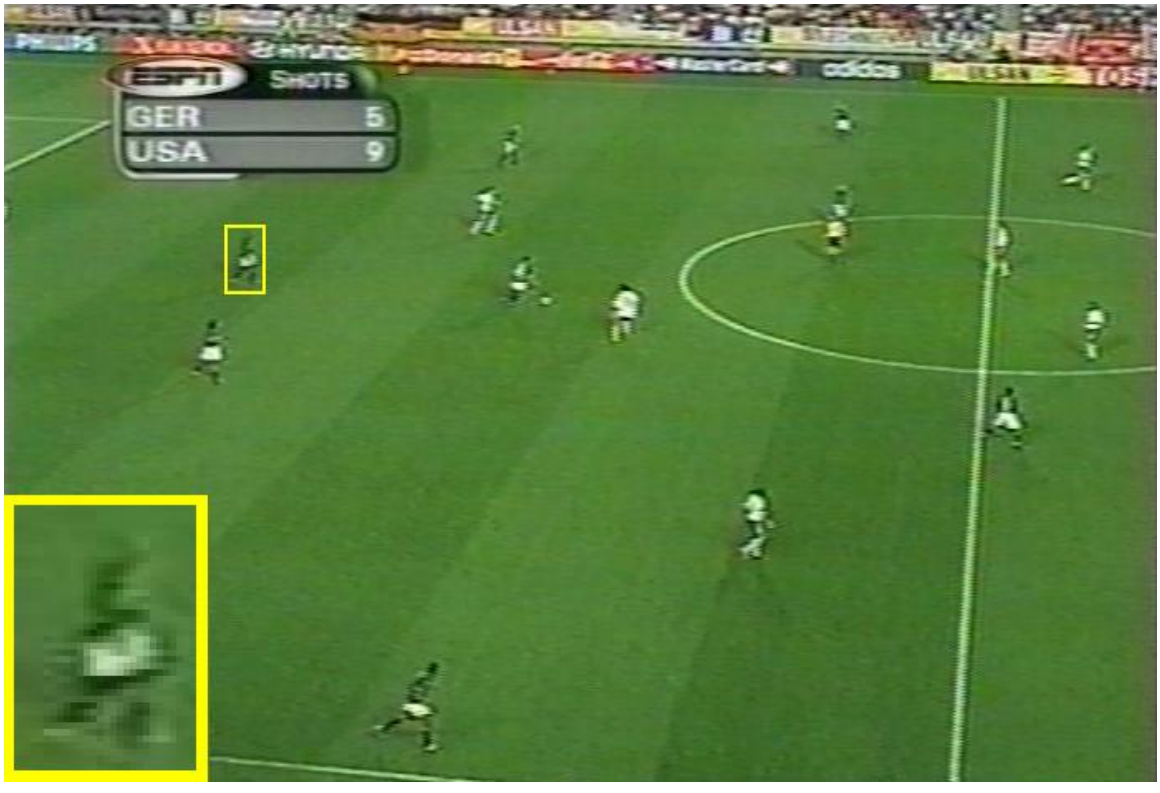


Figure 6.1: Human figures at medium resolution.. A typical frame from the NTSC World Cup broadcast used as our data. Humans are extremely good at recognizing the actions of the football players, despite the low resolution (each figure is about 30 pixels tall; see the zoomed in player at the lower left corner).

traffic, but given that the only descriptor we can extract is the translation of the blob as a whole, we cannot expect to discriminate among too many action categories.

In this paper, we develop a general approach to recognizing actions in "medium" field. Figure 6.3 shows a flow diagram. We start by tracking and stabilizing each human figure – conceptually this corresponds to perfect smooth pursuit movements in human vision or a skillful panning movement by a camera operator who keeps the moving figure in the center of the field of view. Any residual motion within the spatio-temporal volume is due to the relative motions of different body parts: limbs, head, torso etc. We will characterize this motion by a descriptor based on computing the optical flow, projecting it onto a number of motion channels, and blurring. Recognition is performed in a nearest neighbor framework. We have a stored database of previously seen (and labeled) action fragments, and by computing a spatio-temporal cross correlation we can find the one most similar to

(a) Near Field (b) Far Field

Figure 6.2: Typical examples of figures at near and far field.. In the near field (a), there is a reasonable hope of being able to segment and label parts such as the limbs, torso, and head. In the far field (b), the best one can do is to track the whole figure as a "blob". We are interested in the medium resolution regime that lies between these two extremes.

the motion descriptor of the query action fragment. The retrieved nearest neighbor(s) can be used for other applications than action recognition – we can transfer attached attributes such as appearance or 2D/3D skeletons from the action fragment in the database to the one in the query video sequence.

Note that we do *not* use the movement of the figure as a whole – the stabilization step intentionally throws away this information. In far field, this would in fact be the *only* information available for a moving figure blob, and it would certainly make sense for an integrated system for action recognition to capitalize on this cue. Our motivation is scientific – we want to understand the "extra" information available corresponding to relative motions among the different locations of the figure, just as one might ignore color to better understand the role of shape in object recognition. It may also be worth remarking that there are situations such as a person on a treadmill, or when the camera pans to keep an actor in the field of view, when the overall motion of the figure blob is unavailable or misleading.

This chapter is organized as follows. Next section reviews related work. In Section 6.1, we develop the motion descriptor. This is the core of the paper – it is well known that optical flow measurements are noisy, so to be able to use them in a robust way for action matching is a fundamental contribution. Given the descriptor and matching tech-

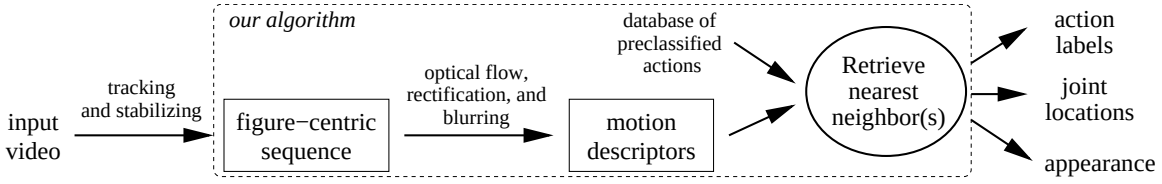


Figure 6.3: Data flow for our algorithm.. Starting with a stabilized figure-centric motion sequence, we compute the spatio-temporal motion descriptor centered at each frame. The descriptors are then matched to a database of preclassified actions using the k-nearest-neighbor framework. The retrieved matches can be used to obtain the correct classification label, as well as other associated information.

nique, in Section 3 we show classification results on a variety of datasets – ballet, tennis, football. In the following chapter, we show how the process of extracting best matching action fragments from the database has other side benefits. We are able to perform "skeleton transfer" on to the input figure sequence, as well as synthesize novel video sequences in two ways we call "Do as I do" or "Do as I say".

6.0.1 Related Work

This work addresses action recognition in "medium field" based on analyzing motion channels. As discussed above, most work in human tracking and activity recognition is only appropriate for "near field" with higher resolution figures. Shah and Jain [52] review the previous work on activity recognition, much of which involves tracking at the level of body parts. Gavrila and Davis' survey paper [25] provides a thorough review of the tracking literature, but it is largely inapplicable for the type of data we are considering in this work.

Another class of methods analyze motion periodicity [42, 51, 15, 12]. Of particular relevance is the work of Cutler and Davis [15], which is one of a few attempts at analyzing poor quality, non-stationary camera footage. Their approach is based on modeling the structure of the appearance self-similarity matrix and can handle very small objects. They report classification results on three categories: "person", "dog", "other". Unfortunately, methods based on periodicity are restricted to periodic motion.

Action classification can be performed in a nearest neighbor framework. Here the main challenge is to find the right representation for comparing novel data with stored examples. Bobick and Davis [7] derive the Temporal Template representation from background subtracted images. They present results on a variety of choreographed actions across

different subjects and views, but require two stationary cameras with known angular interval, a stationary background, and a reasonably high-resolution video. Song et al. [55] demonstrate detection of walking and biking people using the spatial arrangement of moving point features. Freeman et al. [21] use image moments and orientation histograms of image gradients for interactive control in video games. Developing this theme, Zelnik-Manor and Irani [65] use marginal histograms of spatio-temporal gradients at several temporal scales to cluster and recognize video events. Despite its simplicity, this representation is surprisingly powerful. The paper reports promising action similarity results on three different datasets with 3-4 classes, assuming a single actor and a static background. However, since only the marginal information is being collected over each frame, the classes of actions that can be discriminated must have substantially different motion speed and orientation profiles.

Our present work can be thought of as building on the Zelnik-Manor and Irani approach by adding the descriptive power of a joint spatio-temporal distribution. This allows us to differentiate between motion in different areas of the image, making our classifier more discriminative. The approach we present works on poor, low-resolution data, does not rely on motion periodicity or a stationary camera, and demonstrates good performance on tasks with relatively large numbers of classes.

Our approach bears strong similarity to the recent tracking work of Zhao et al. [68, 67]. Their motion template is similar to our motion descriptor, but instead of a stabilized spatio-temporal volume, they use 3D motion capture data in a semi-parametric framework to do their classification.

6.1 Measuring Motion Similarity

Our algorithm (Figure 6.3) starts by computing a figure-centric spatio-temporal volume for each person. Such a representation can be obtained by tracking the human figure and then constructing a window in each frame centered at the figure (see Figure 6.4). Any of a number of trackers are appropriate; in our experiments, we used a simple normalized-correlation based tracker, either on raw video or on regions of interest selected by thresholding the temporal difference image. The main requirement is that the tracking be consistent – a person in a particular body configuration should always map to approximately the same stabilized image.

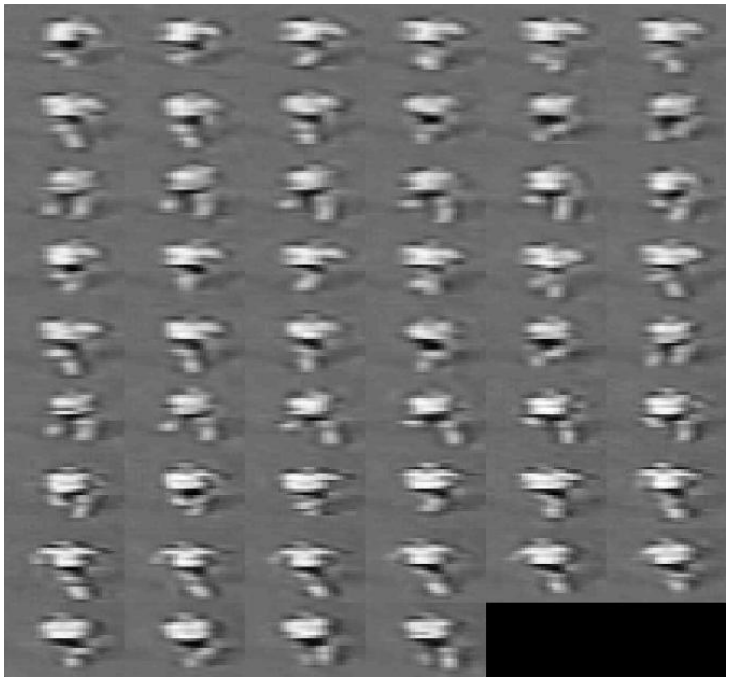


Figure 6.4: Spatio-temporal Volume. We track each player and recover a stabilized spatio-temporal volume, which is the only data used by our algorithm.

Once the motion sequences are stabilized it becomes possible to directly compare them in order to find correspondences. Finding similarity between different motions requires both spatial and temporal information. This leads to the notion of the *spatio-temporal motion descriptor*, an aggregate set of features sampled in space and time, that describe the motion over a local time period. Computing such motion descriptors centered at each frame will enable us to compare frames from different sequences based on local motion characteristics.

The important question is what are appropriate features to put into the motion descriptor. Encoding the actual image appearance by storing the pixel values directly is one possibility, which has been successfully used by Schödl el al. [50] to find similarity between parts of *the same* video sequence. However, appearance is not necessarily preserved across different sequences (e.g. people wearing different clothing). The same is true for spatial image gradients which depend linearly on image values. Temporal gradient is another useful feature [21, 65], but it shares the problems of spatial gradients in being a linear function of appearance. For example, temporal gradients exhibit contrast reversal: a light-dark edge moving right is indistinguishable from a dark-light edge moving left (taking the absolute

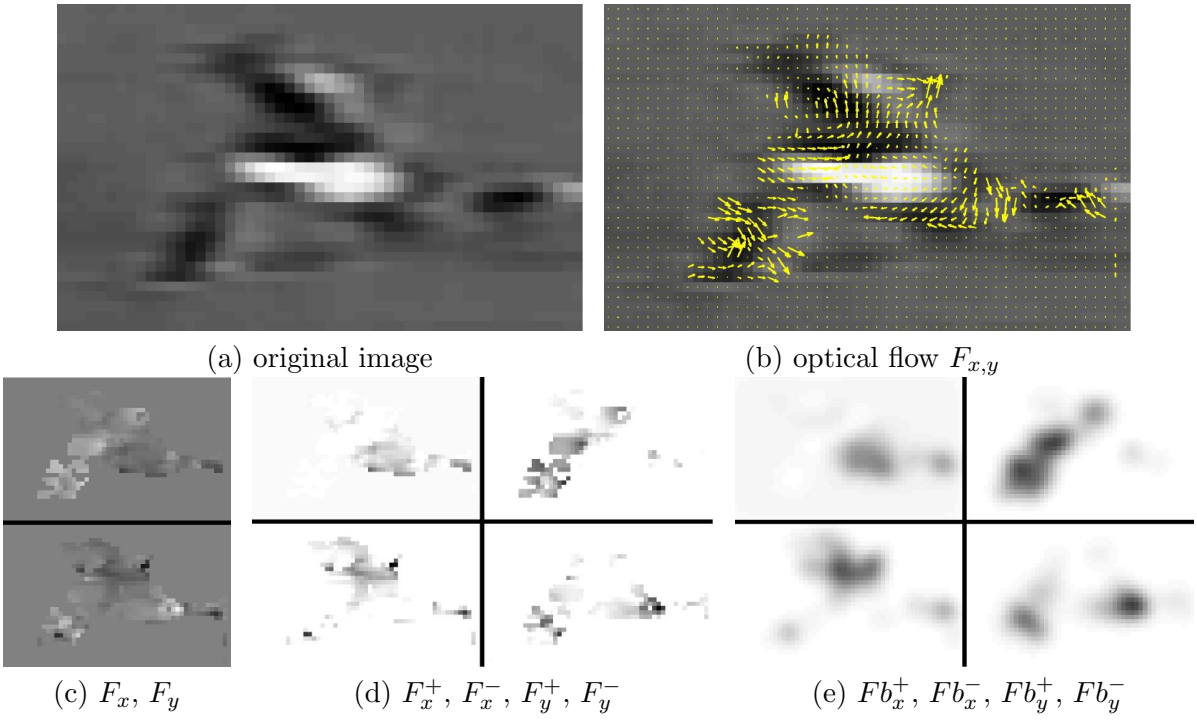
value of the gradient will fix this but it will also remove all information about the direction of motion). One way to make appearance robust to illumination is by detecting edges in the image. Unfortunately, most of our images are far too small and noisy for edge detection to produce reasonable results. Silhouette extraction, a popular technique for near-field action recognition, also won't work here for similar reasons.

We base our features on pixel-wise optical flow as the most natural technique for capturing motion independent of appearance. In biological vision, neurons sensitive to direction and speed of retinal motion have been found in many different species. On the other hand, computer vision experience suggests that computation of optical flow is not very accurate, particularly on coarse and noisy data, such as typical NTSC video footage. Our insight is to treat these optical flow vectors not as precise pixel displacements at points, but simply as a spatial pattern of noisy measurements which are aggregated using our motion descriptor. We think of the spatial arrangement of optical flow vectors as a template that is to be matched in a robust way.

The motion descriptor must perform reliably with features that are noisy, and moreover, be able to deal with input data that are not perfectly aligned either temporally or spatially. Matching under noise and positional uncertainty is often done using histograms of features over image regions [65, 49, 4]. Interestingly, a very similar effect can be obtained by simply blurring the input signal in the correct way [5]. This is a very simple yet powerful technique of capturing only the essential positional information while disregarding minor variations. However, one must be careful that important information in the signal is not lost due to blurring together of positive and negative components. In order to deal with this potential loss of discriminative information we use half-wave rectification, separating the signal into sparse, positive-only channels before it is blurred. In the primate visual system, one can think of each of these *blurred motion channels* as corresponding to a family of complex, direction selective cells tuned to roughly the same direction of retinal motion.

6.1.1 Computing Motion Descriptors

Given a stabilized figure-centric sequence, we first compute optical flow at each frame using the Lucas-Kanade [36] algorithm (see Figure 6.5(a,b)). The optical flow vector field $\mathbf{F}$ is first split into two scalar fields corresponding to the horizontal and vertical components of the flow, F_x and F_y, each of which is then half-wave rectified into four non-negative



(a) original image (b) optical flow $F_{x,y}$

(c) F_x, F_y (d) F_x^+, F_x^-, F_y^+, F_y^- (e) Fb_x^+, Fb_x^-, Fb_y^+, Fb_y^-

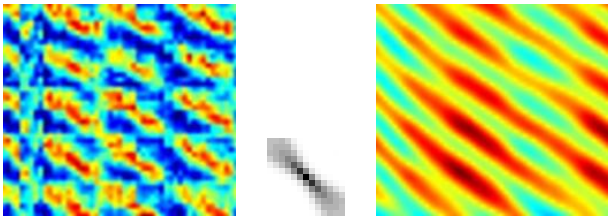
Figure 6.5: Constructing the motion descriptor. (a) Original image, (b) Optical flow, (c) Separating the x and y components of optical flow vectors, (d) Half-wave rectification of each component to produce 4 separate channels, (e) Final *blurry motion channels*

channels F_x^+, F_x^-, F_y^+, F_y^-, so that $F_x = F_x^+ - F_x^-$ and $F_y = F_y^+ - F_y^-$ (see Figure 6.5(c,d)). These are each blurred with a Gaussian and normalized to obtain the final four channels, $\hat{Fb}_x^+$, $\hat{Fb}_x^-$, $\hat{Fb}_y^+$, $\hat{Fb}_y^-$, of the motion descriptor for each frame (see Figure 6.5(e)). Alternative implementations of the basic idea could use more than 4 motion channels – the key aspect is that each channel be sparse and non-negative.

The spatio-temporal motion descriptors are compared using a version of normalized correlation. If the four motion channels for frame i of sequence A are a_1^i,a_2^i,a_3^i, and a_4^i, and similarly for frame j of sequence B then the similarity between motion descriptors centered at frames i and j is:

$$S(i,j) = \sum_{t \in T} \sum_{c=1}^{4} \sum_{x,y \in I} a_c^{i+t}(x,y) b_c^{j+t}(x,y) \tag{6.1}$$

where T and I are the temporal and spatial extents of the motion descriptor respectively. To compare two sequences A and B, the similarity computation will need to be done for every frame of A and B so Eq. 6.1 can be optimized in the following way. First, a frame-to-



(a) (b) (c)

Figure 6.6: Similarity Matrices. (a) A typical frame-to-frame similarity matrix S_{ff} for running, (b) the "Blurry I" kernel K (not shown to scale) used for aggregating temporal information within the similarity matrix, (c) the resulting motion-to-motion similarity matrix S.

Figure 6.7: Representative frames from 16 ballet actions used for our experiments. The actions are (left to right): 1) 2^{nd} pos. plies, 2) 1^{st} pos. plies, 3) releve, 4) down from releve, 5) point toe and step right, 6) point toe and step left, 7) arms 1^{st} pos. to 2^{nd} pos., 8) rotate arms in 2^{nd} pos., 9) degage, 10) arms 1^{st} pos. forward and out to 2^{nd} pos., 11) arms circle, 12) arms 2^{nd} to high fifth, 13) arms high fifth to 1^{st}, 14) port de dras, 15) right arm from high fifth to right, 16) port de bra flowy arms

frame similarity matrix of the blurry motion channels (the inner sums of the equation) is computed between each frame of A and B. Let us define matrix A_1 as the concatenation of a_1's for each frame stringed as column vectors, and similarly for the other 3 channels. Then the frame-to-frame similarity matrix $S_{ff} = A_1^T B_1 + A_2^T B_2 + A_3^T B_3 + A_4^T B_4$. To obtain the final motion-to-motion similarity matrix S, we sum up the frame-to-frame similarities over a T temporal window by convolution with a $T \times T$ identity matrix, thus $S = S_{ff} \star I_T$.

If motions are similar, but occur at slightly different rates then the strong frame to frame similarities will occur along directions close to diagonal but somewhat slanted (note the angle of bands in Fig. 6.6a). In order to take advantage of this fact, we look for strong responses along directions close to diagonal in the frame to frame similarity matrix between A and B. In practice this is achieved by convolving the frame to frame similarity matrix

S_{ff} with the kernel shown in Figure 6.6(b) instead of the identity matrix to obtain the final similarity matrix. The kernel is a weighted sum of near diagonal lines, with more weight put closer to the diagonal.

$$K(i,j) = \sum_{r \in R} w(r)\chi(i,rj) \tag{6.2}$$

where $w(r)$ weights values of r near one relatively more, and R is the range of rates. (Note that we set $\chi(i,rj)$ to one if i and rj round to the same value and zero otherwise). The similarity between two sequences centered at two particular frames can be read from the corresponding entry in the final similarity matrix.

6.2 Classifying Actions

Given a novel sequence to be classified and a database of labeled example actions, we first construct a motion similarity matrix as outlined above. For each frame of the novel sequence, the maximum score in the corresponding row of this matrix will indicate the best match to the motion descriptor centered at this frame (see Figure 6.8). Now, classifying this frame using a k-nearest-neighbor classifier is simple: find the k best matches from labeled data and take the majority label.

We show results on three different domains:

Ballet: choreographed actions, stationary camera (Figure 6.9). Clips of motions were digitized from an instructional video for ballet showing professional dancers, two men and two women, performing mostly standard ballet moves. The motion descriptors were computed with 51 frames of temporal extent.

Tennis: real actions, stationary camera (Figure 6.10). For this experiment, we shot footage of two amateur tennis players outdoors. Each player was video-taped on different days in different locations with slightly different camera positions. Motion descriptors were computed with 7 frames of temporal extent.

Football: real actions, moving camera (Figure 6.11). We digitized several minutes of a World Cup football game (called *soccer* in the U.S.) from an NTSC video tape. We used wide-angle shots of the playing field, which have substantial camera motion and zoom (Figure 6.1). We take only the odd field of the interlaced video in grayscale, yielding, on average, about 30-by-30 noisy pixels per human figure. All motion descriptors were computed with 13 frames of temporal extent.

Figure 6.8 shows the best matches for each frame of some example sequences while Figures 6.9-6.11 shows the quantitative classification results in the form of confusion matrices.

6.3 Conclusion

In this chapter, we presented a technique that can compare motions based on very noisy optical flow. It seems clear that optical flow contains the right information for measuring motion similarity, if only it could be made more robust. Our insight is to treat optical flow not as a precise measurement of pixel displacement, but rather as a set of noisy features which are carefully smoothed and aggregated using our spatio-temporal motion descriptor. We have demonstrated the use of the approach for classifying actions in three different, complex domains. In the next chapter, we will show how the same methodology can be applied for action synthesis as well as skeleton transfer.

Figure 6.8: Best matches for classification (ballet, tennis, football). The top row of each set shows a sequence of input frames, the bottom row shows the best match for each of the frames. Our method is able to match between frames of people performing the same action yet with substantial difference in appearance.

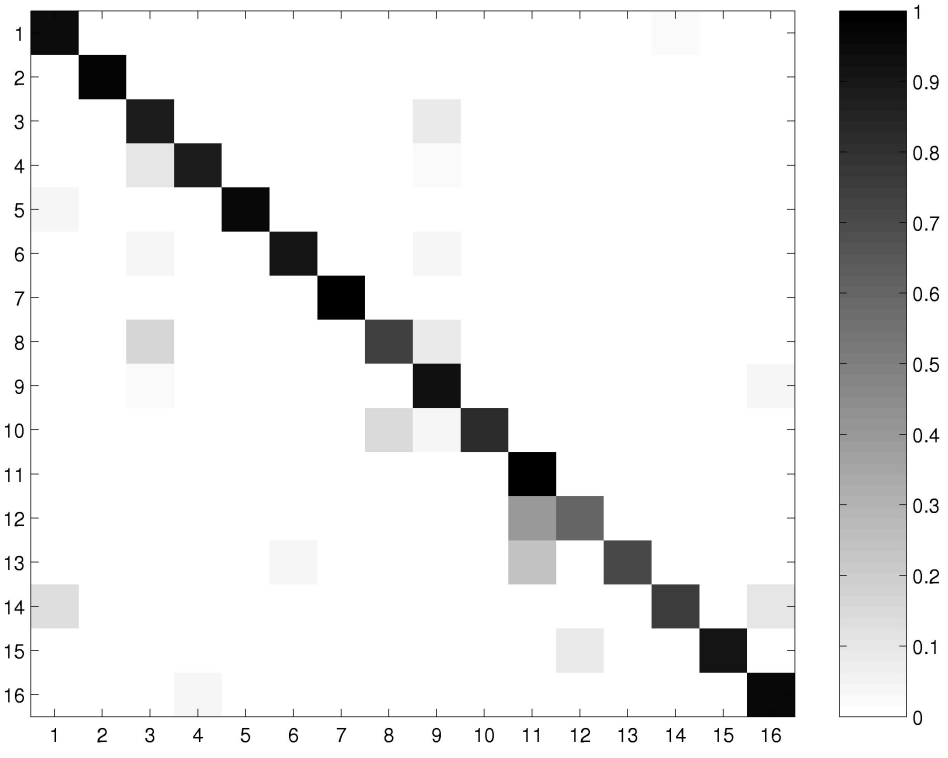


Figure 6.9: Classification: confusion matrix for ballet dataset. Each row represents the probabilities of that class being confused with all the other classes. Ballet databset has 24800 frames. The 16 classes are defined in Figure 6.7. Video of the male dancers was used to classify the video of the female dancers and vice versa. Classification used 5-nearest-neighbors. The main diagonal shows the fraction of frames correctly classified for each class and is as follows: [.94 .97 .88 .88 .97 .91 1 .74 .92 .82 .99 .62 .71 .76 .92 .96]. The algorithm performs quite well on the ballet actions, matching or exceeding previous work in this area. However, the highly controlled, choreographed nature of the actions make this a relatively easy test.

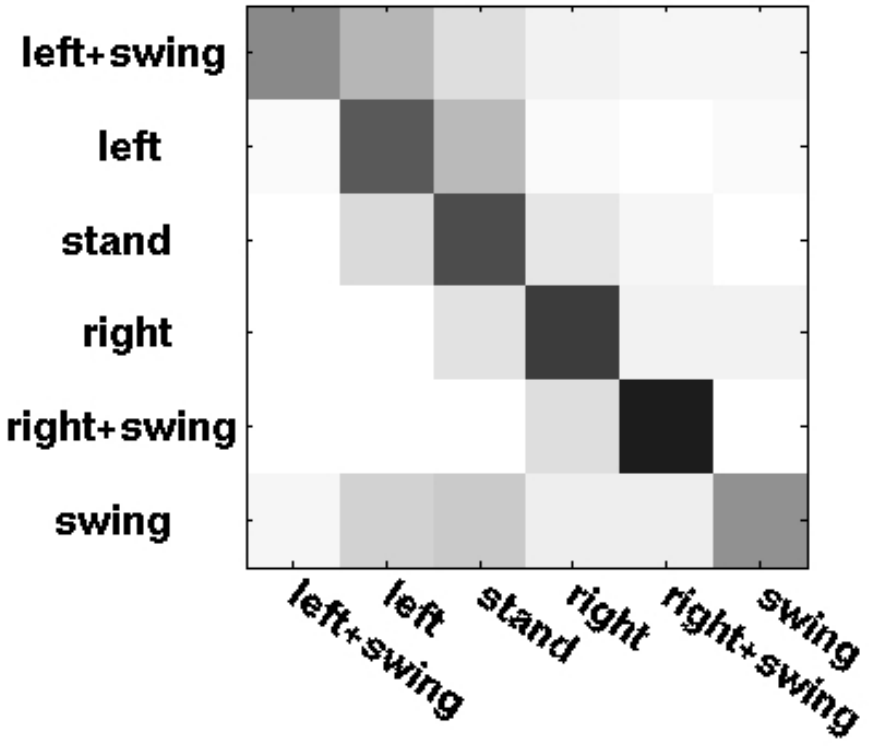


Figure 6.10: Classification: confusion matrix for tennis dataset. The video was subsampled by a factor of four, rendering the figures approximately 50 pixels tall. Actions were hand-labeled with six labels: "swing", "move left", "move right", "move left and swing", "move right and swing", "stand". Video of the female tennis player (4610 frames) was used to classify the video of the male player (1805 frames). Classification used 5-nearest-neighbors. The main diagonal is: [.46 .64 .7 .76 .88 .42]. While the classification is not as good as in the previous experiment, the confusions make sense. For example, the "go left and swing" class gets confused with "go left". In addition some of the swing sequences are misclassified because optical flow occasionally misses the low contrast, motion blurred tennis racket.

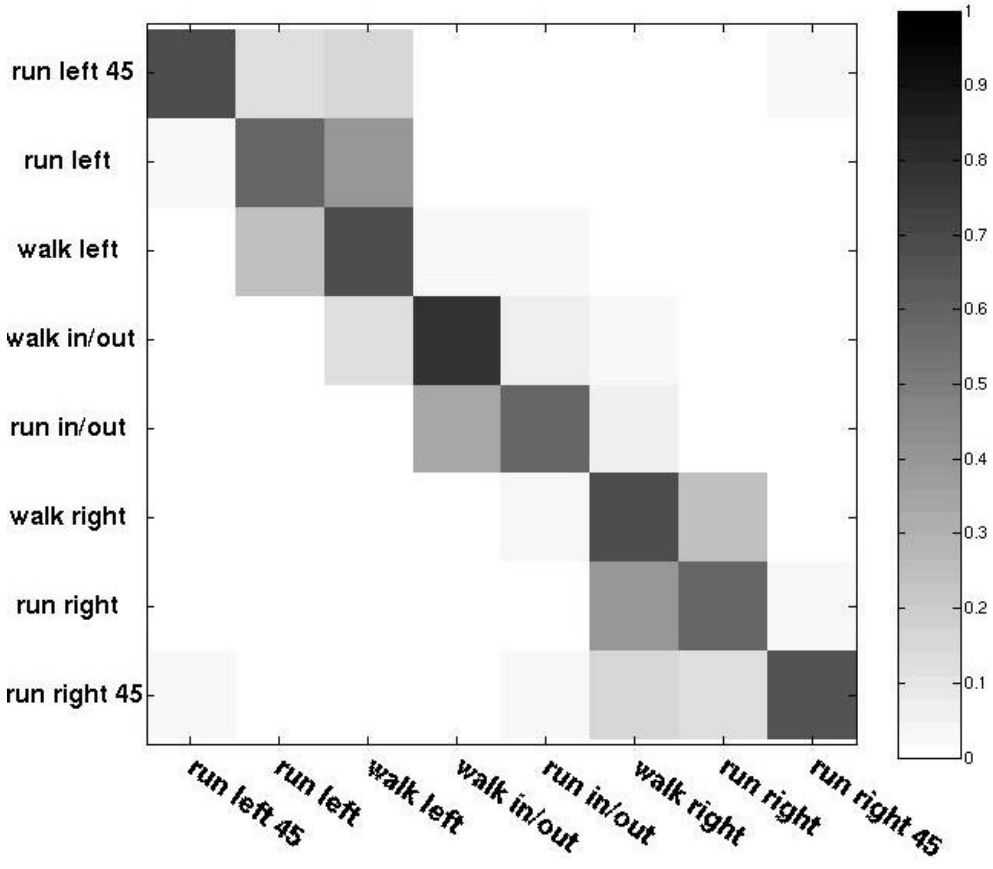


Figure 6.11: Classification: confusion matrix for football dataset. 4500 frames, taken from 72 tracked sequences, supplemented by mirror flipping some of the sequences). We hand-labeled subsequences with one of 8 actions: "run left 45°", "run left", "walk left", "walk in/out", "run in/out", "walk right", "run right", and "run right 45°". The classification used a 1-nearest-neighbor classifier on the entire data set with a leave-one-sequence-out testing scheme. The main diagonal is: [.67 .58 .68 .79 .59 .68 .58 .66]. The classes are sorted according to the direction of motion – confusion occurs mainly between very similar classes where inconsistent ground truth labeling occurs. There is virtually no confusion between very different classes, such as moving left, moving straight, and moving right. Here as with the tennis example the player's direction of motion is successfully recovered even though the algorithm uses *no translational information at all*. This means that the method correctly interprets the movement of human limbs without explicitly tracking them. The results are particularly impressive considering the very poor quality of the input data. Figure 6.8 shows nine consecutive frames from a "run right" sequence (top row) together with the best matching frames from the rest of the database (bottom row). Note that while the best matches come from different players with different appearance and scale, the motion is matched very well.

Chapter 7

Querying the Action Database

It's a poor sort of memory that only works backward.

Lewis Carroll

The classification procedure described in the previous chapter can be thought of as a particular type of *database query*. Given a database of example sequences annotated with action labels, the classifier uses the motion descriptor as a key to query this database. The result of such a query is to retrieve the action label of the database entry with the most similar motion descriptor. But it is also possible to annotate the database entries with other useful information, such as appearance, 2D or 3D joint positions, person identification for the different actors. In this chapter we will discuss how querying and retrieval of this expanded database can facilitate several other useful applications in vision and graphics (see Figure 7.1).

7.1 Skeleton Transfer

Recovering joint positions (i.e. the skeleton) of a human figure from video is an important and difficult problem. Most approaches rely on a person's appearance in each frame to identify limbs or other salient features (e.g. [39]). This will not work for our data – the figures are usually much too small to have identifiable parts at any given frame. Here, again, our solution is to rely on motion instead of appearance. The idea is simple: we

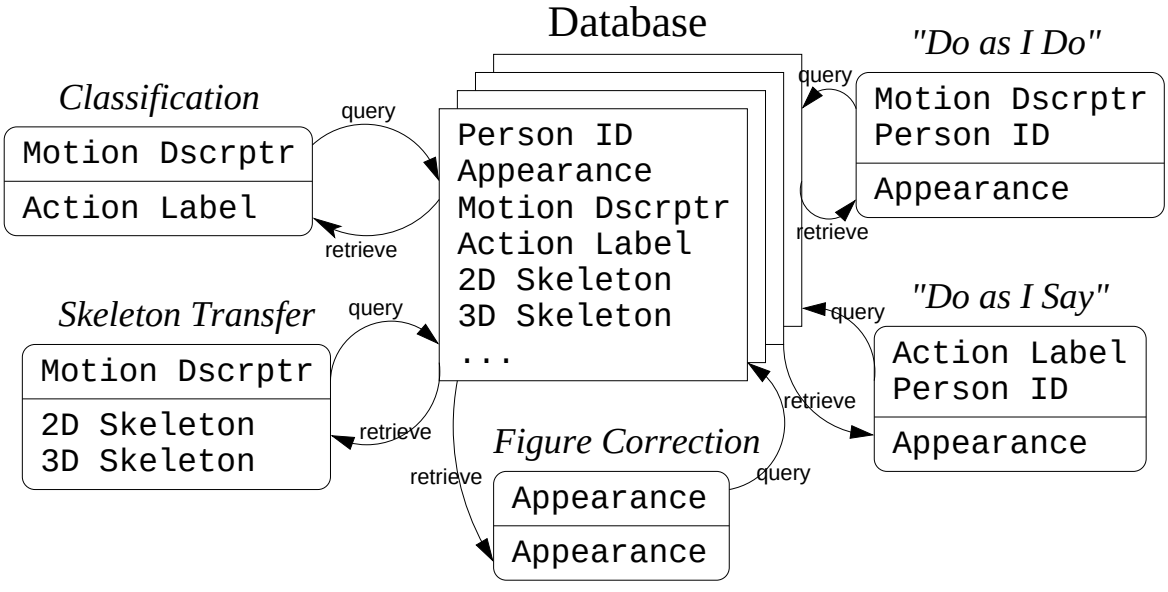


Figure 7.1: Action Database. Our classification algorithm can be interpreted as a database query: given a motion descriptor, retrieve the best matching action label. Other similar queries are possible, resulting in a number of useful applications, such as skeleton transfer, action synthesis, and figure correction. (This diagram does not show the temporal aspect of our action synthesis method.)

annotate each frame in our database with hand-marked joint locations. This means that a novel sequence can now be automatically labeled with joint position markers, essentially transferring a 2D skeleton from the stored example onto the novel sequence (see Figure 7.2, second row). Note that since the motion descriptor is designed to be robust to misalignment, the skeleton transfered in this way may not be placed precisely on the figure. Hence we use a simple refinement step to better align the two sequences by searching for the scale and shift that maximizes the motion descriptor matching score.

An alternative to hand-marking the joint locations is to use available 3D motion capture data (produced in a lab using special markers) to *generate* a suitable database. We can render the MoCap data (using a stick figure) from several viewing directions to create a database of synthetic 2D motion sequences, fully annotated with the original 3D joint locations. Figure 7.2 shows how, given a video sequence (first row), we are able to recover a 3D skeleton (third row). Alternatively we could go to the 3D skeleton from the 2D skeleton, as in [56]. While lifting a 2D figure into 3D is clearly ambiguous (e.g. in side view, the left and right legs often get confused), nonetheless we believe that the information obtained

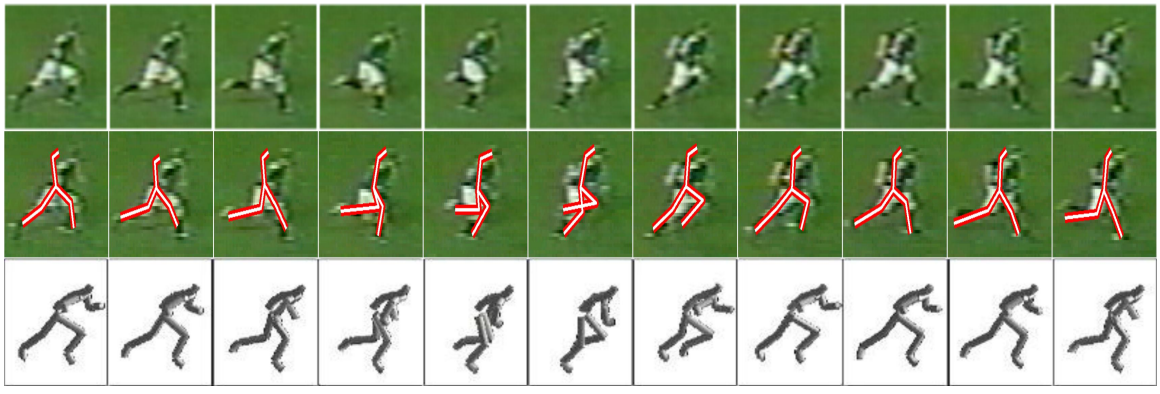


Figure 7.2: Skeleton Transfer. Given an input sequence (top row) we are able to recover rough joint locations by querying the action database and retrieving the best-matching motion with the associated 2D/3D skeleton. Second row shows a 2D skeleton transferred from a hand-marked database of joint locations. Third row demonstrates 3D skeleton transfer, which utilizes Motion Capture data rendered from different viewing directions using a stick figure.

this way is quite valuable.

7.2 Action Synthesis

The visual quality of our motion descriptor matching (see Figure 6.8) suggests that the method could be used in graphics for *action synthesis*, creating a novel video sequence of an actor by assembling frames of existing footage. The idea is in the spirit of Video Textures [50], except that we would like to have control over the actions that are being synthesized. The ultimate goal would be to collect a large database of, say, Charlie Chaplin footage and then be able to "direct" him in a new movie.

"Do as I Do" Synthesis. Given a "target" actor database T and a "driver" actor sequence D, the goal is to create a synthetic sequence S that contains the actor from T performing actions described by D. This problem can be posed as simple query: retrieve the frames from T associated with motion descriptors best matching those from D. However, this process alone will produce a video that is too jerky, since no smoothness constraint is present. In practice, the synthesized motion sequence S must satisfy two criteria: the actions in S must match the actions in the "driver" sequence D, and the "target" actor must appear natural when performing the sequence S. We pose this as an optimization

Figure 7.3: "Do as I Do" Action Synthesis. The top row is a sequence of a "driver" actor, the bottom row is the synthesized sequence of the "target" actor (one of the authors) performing the action of the "driver".

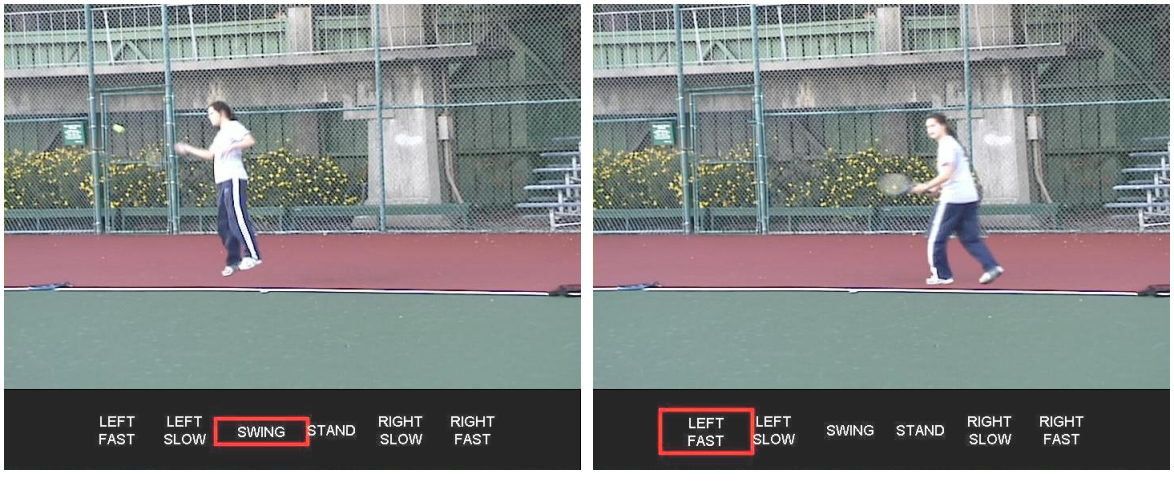


Figure 7.4: "Do As I Say" Action Synthesis. Shown are two frames from a synthesized video of a tennis player performing actions as specified by the commands (at the bottom). For the full video, visit our website.

problem.

Let $W_{act}(u, v)$ contain the motion descriptor similarity between frame u of D and frame v of T. A second matrix W_s is used to enforce the smoothness of the synthesized sequence. Let $W_s(u, v)$ hold the similarity in appearance (frame-to-frame normalized correlation) and in motion (motion descriptors) of frames u and v, both from the target database T. Since we are comparing frames from the *same* actor, we are able to use actual pixel values in the computation of the appearance term. We define the following cost function on S, a sequence of frames $\{\pi_1, \pi_2, ..., \pi_n\}$ picked from T:

$$C(S) = \sum_{i=1}^{n} \alpha_{act} W_{act}(i, \pi_i) + \sum_{i=1}^{n-1} \alpha_s W_s(\pi_{i+1}, \text{succ}(\pi_i)),$$

where $\text{succ}(\pi_i)$ is the frame that follows π_i in T. The cost function has only local terms, and therefore lends itself to being optimized using dynamic programming. A sequence of length n can be chosen from m frames in T in $O(nm^2)$ time. Figure 7.3 shows a few frames from our "Do as I Do" results. See the web page for our video results.

"Do as I Say" Synthesis. We can also synthesize a novel "target" actor sequence by simply issuing commands, or action labels, instead of using the "driver" actor. For example, one can imagine a video game where pressing the control buttons will make the real-life actor on the screen move in the appropriate way. The first step is to classify the "target" data T using our classification algorithm. Now the same approach can be used as in the previous section, except W_{act} now stores the similarity between the desired commands and the frames of T. Figure 7.4 shows two frames from a sequence where the tennis player is being controlled by user commands (shown at the bottom). Note that since dynamic programming is an off-line algorithm, this approach would not directly work for interactive applications, although there are several ways to remedy this.

7.3 Context-based Figure Correction

Another interesting use of the action database is to "clean up" human action sequences of artifacts such as occlusion and background clutter (see top row of Figure 7.5). The main idea, inspired by the dictionary-less spelling correction in search engines like Google, is to use the power of the data as a whole to correct imperfections in each particular sample. In our case, for each frame, we retrieve the k closest frames from the rest of the

Figure 7.5: Context-based Figure Correction. We use the power of our data to correct imperfections in each individual sample. The input frames (top row) are automatically corrected to produce cleaned up figures (bottom row).

database (excluding the few neighboring frames that will always be similar). These k frames are used to compute a median image which becomes the new estimate for the current frame. The idea is that, given enough data, the common part among the nearest neighbors will be the figure, while the variations will be mostly due to noise and occlusions. The median filter averages out the variations thus removing most occluders and background clutter as shown on the bottom row of Figure 7.5.

7.4 Application: Actor Replacement

With the tools developed in this chapter, we are now ready to develop a practical system for replacing an actor in a movie with a different actor. Such a system can have a wide range of applications in the film industry, e.g. replacing stunt doubles or placing long-dead actors in new movies. It could also be used (as we will show here) to help ordinary people realize their life-long dreams of being world-famous football players.

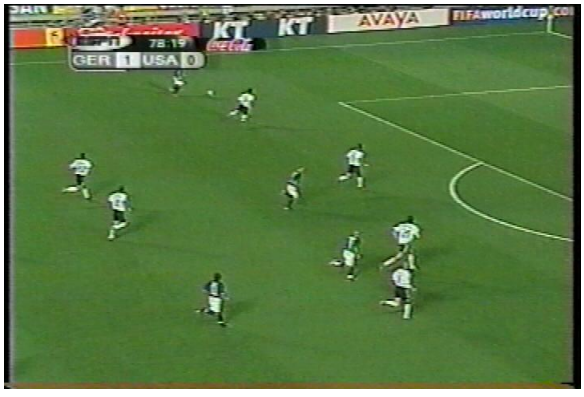
In this experiment we take video broadcast footage of a football game and synthesize a novel person (my co-author and a big football fan) in place of an existing player performing the same motion. This requires 1) tracking the characters and isolating them from the background, 2) retargeting the motion onto the novel character, 3) erasing the existing player from the video, and 4) compositing in the new synthesized character while preserving correct occlusion relationships. What makes this problem particularly challenging is that the camera is not stationary, the scenes are shot from far field, and the interlaced NTSC video is of very poor quality.

The characters in the video are tracked using a simple normalized-correlation tracker, manually initialized by clicking on the person in the first frame. For footage containing separate shots or characters leaving and reentering the shot, the tracker must be roughly told the new search area each time the character reappears. Each tracked figure is cropped and stabilized. Context-based image correction, presented earlier, is used in each frame to get rid of background clutter and occluders. Background subtraction is then used to find the foreground mask for the figure in each frame. The new synthesized sequence is computed exactly as discussed in the "Do as I Do" section, using the existing player as the "driver" and the novel character as the "target". But before we can composite in the new synthesized character, we must erase the figure that we are replacing. We do it by simply filling in its foreground mask with the averaged background color. Now the new sequence can be composited into the original video using the new character's foreground mask, placed at the tracked coordinates of the original player.

In compositing it is important to handle occlusions correctly. Objects occluded by the character pose a problem only when the new character is significantly smaller than the player we are replacing. In this case, a more sophisticated hole-filling algorithm (for example, a variant of our Context-based Image Correction using neighboring frames as background source data) can be used when removing the original figure. However, in this application, we did not find it to be a very significant problem. A more difficult issue is handling objects that occlude the character. In general this problem is very difficult, but in our case, the only occluders are other players. Our simple solution is to track all the relevant players on the field, erase them all, and then repaint them in the top-to-bottom order, based on the y coordinate. This produces correct results since, in the special case of object on the ground plane view from above, the top-to-bottom ordering is equivalent to the standard back-to-front ordering of the painter's algorithm.

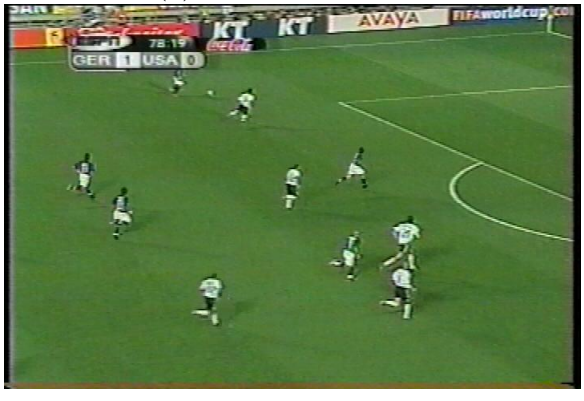
For placing a novel character into the football game video, we have collected several minutes of footage of a person running. We were able to successfully replace several players in the game with our new actor, running in different styles and different directions. We were also able to replace players using other players from the same video (e.g. switching a player from one team to the other). Figure 7.6 shows a few screen shots from the resulting sequences. The reader is referred to the accompanying video [1] for complete results.

[1] `http://www.cs.berkeley.edu/~efros/action/mov/GregWorldCup.avi`



(a) input video frame

(b) sample frame from database



(c) swap teams

(d) novel player

Figure 7.6: Actor Replacement. Given the input image (a), we can replace some players with others from the opposite team (c), or with a novel character (d) taken from a different video footage (b).

7.5 Conclusion

In this chapter we have shown that, with the abundance of visual data, synthesis can often be recast as simply a *query* problem – finding the right images in your data, instead of creating them from scratch. This idea has been realized before, in particular in the work on Video Rewrite [10] and Video Textures [50], among others. The significant generalization provided here is that similarity is defined in the pattern of underlying motion rather than raw appearance. This is what enabled us to exhibit robust behavior under variations in character appearance, scale, and lighting.

Bibliography

[1] Sameer Agarwal and Serge Belongie. Segmentation by example. Technical report, UC San Diego, CS Department, 2003.

[2] M. Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, 2001.

[3] The Berkeley Path Wanderers Association. *A Map of Berkeley's Pathways*. Wilderness Press, 2002.

[4] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. PAMI*, 24(4):509–522, April 2002.

[5] A. C. Berg and J. Malik. Geometric blur for template matching. In *Computer Vision and Pattern Recognition*, pages 607–614, 2001.

[6] J. Bergen and E. Adelson. Early vision and texture perception. *Nature*, 333:363–364, 1988.

[7] A. Bobick and J. Davis. The recognition of human movement using temporal templates. *IEEE Trans. PAMI*, 23(3):257–267, 2001.

[8] J. S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH 97*, pages 361–368, 1997.

[9] E. Borenstein and S. Ullman. Class-specific, top-down segmentation. In *European Conference on Computer Vision*, May 2002.

[10] C Bregler, M Covell, and M Slaney. Video rewrite: driving visual speech with audio. In *SIGGRAPH '97*, pages 353–360, 1997.

[11] P. Brodatz. *Textures.* Dover, New York, 1966.

[12] Robert Collins, Ralph Gross, and Jianbo Shi. Silhouette-based human identification from body shape and gait. In *5th Intl. Conf. on Automatic Face and Gesture Recognition*, 2002.

[13] A. Criminisi, P.Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, June 2003.

[14] C. J. Curtis, S. E. Anderson, J. E. Seims, Kurt W. Fleisher, and D. H. Salsin. Computer-generated watercolor. In *SIGGRAPH 97*, pages 421–430, 1997.

[15] Ross Cutler and Larry Davis. Robust real-time periodic motion detection, analysis, and applications. *IEEE Trans. PAMI*, 22(8), August 2000.

[16] J. Davis. Mosaics of scenes with moving objects. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.*, 1998.

[17] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96*, pages 11–20, August 1996.

[18] A.K. Dewdney. A potpourri of programmed prose and prosody. *Scientific American*, 122-TK, June 1989.

[19] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, 2001.

[20] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999.

[21] W. T. Freeman, K. Tanaka, J. Olita, and K. Kyuma. Computer vision for computer games. In *IEEE 2nd Intl. Conf. on Automatic Face and Gesture Recognition*, 1996.

[22] W. T. Freeman and J. B. Tenenbaum. Learning bilinear models for two-factor problems in vision. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, 97.

[23] D. D. Garber. *Computational Models for Texture Analysis and Texture Synthesis.* PhD thesis, University of Southern California, Image Processing Institute, 1981.

[24] J. Garding. Surface orientation and curvature from differential texture distortion. *ICCV*, pages 733–739, 1995.

[25] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding: CVIU*, 73(1):82–98, 1999.

[26] J. J. Gibson. *The Perception of the Visual World*. Houghton Mifflin, Boston, Massachusetts, 1950.

[27] P. Harrison. A non-hierarchical procedure for re-synthesis of complex textures. In *WSCG '2001 Conference proceedings*, pages 190–197, 2001. See also http://www.csse.monash.edu.au/~pfh/resynthesizer/.

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

[29] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven Seitz. Curve analogies. In *Eurographics Workshop on Rendering*, June 2002.

[30] D.R. Hofstadter. *Fluid Concepts and Creative Analogies*. Basic Books, NY, 1985.

[31] Bela Julesz. Visual pattern discrimination. *IRE Transactions on Information Theory*, 8(2):84–92, 1962.

[32] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, July 2003.

[33] Laurent Lefebvre and Pierre Poulin. Extraction and synthesis of bump maps from photograph. In *Graphics Interface Posters Proceedings*, pages 12–13, May 2000.

[34] Elizaveta Levina. *Statistical Issues in Texture Analysis*. PhD thesis, Statistics Department, UC Berkeley, Berkeley, CA, 2002.

[35] L. Liang, C. Liu, , Y. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3), July 2001.

[36] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (darpa). In *DARPA Image Understanding Workshop*, pages 121–130, April 1981.

[37] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanism. *JOSA-A*, 5(5):923–932, May 1990.

[38] J. Malik and R. Rosenholtz. Computing local surface orientation and shape from texture for curved surfaces. *International Journal of Computer Vision*, 23(2):149–168, 1997.

[39] G. Mori and J. Malik. Estimating human body configurations using shape context matching. In *European Conference on Computer Vision LNCS 2352*, volume 3, pages 666–680, 2002.

[40] Gerald Olmsted. *A Rambler's Guide to the Trails of the East Bay Hills/Map: Northern Section.* Olmsted and Brothers Map Company, 1992.

[41] V. Ostromoukhov and R. D. Hersch. Multi-color and artistic dithering. In *SIGGRAPH 99*, pages 425–432, 1999.

[42] Ramprasad Polana and Randal C. Nelson. Detection and recognition of periodic, non-rigid motion. *Int. Journal of Computer Vision*, 23(3):261–282, 1997.

[43] Kris Popat and Rosalind W. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In *Proc. SPIE Visual Comm. and Image Processing*, 1993.

[44] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–71, December 2000.

[45] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *SIGGRAPH 00*, pages 465–470, 2000.

[46] D. Ramanan and D. A. Forsyth. Automatic annotation of everyday movements. Technical report, Tech. Rep. No. UCB//CSD-03-1262, Berkeley, CA, July 2003.

[47] C. Rao and M. Shah. View-invariance in action recognition. *CVPR*, 2:316–321, 2001.

[48] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97*, 1997.

[49] Henry Schneiderman and Takeo Kanade. A statistical method for 3d object detection applied to faces and cars. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.*, volume 1, pages 746–751, 2000.

[50] Arno Schodl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *Proceedings of SIGGRAPH '00*, pages 489–498, 2000.

[51] S. M. Seitz and C. R. Dyer. View-invariant analysis of cyclic motion. *Int. Journal of Computer Vision*, 25(3), 1997.

[52] Mubarak Shah and Ramesh Jain. *Motion-Based Recognition*. Computational Imaging and Vision Series. Kluwer Academic Publishers, 1997.

[53] Claude E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. Journal*, 27, 1948.

[54] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. In *SIGGRAPH '02*, July 2002.

[55] Y. Song, L. Goncalves, and P. Perona. Unupervised learning of human motion. *IEEE Trans. PAMI*, 25(7):814–827, 2003.

[56] C. J. Taylor. Reconstruction of articulated objects from point correspondences in a single uncalibrated image. *CVIU*, 80:349–363, 2000.

[57] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH '02*, pages 665–672, 2002.

[58] Greg Turk. Texture synthesis on surfaces. In *SIGGRAPH 01*, pages 347–354, 2001.

[59] R De Valois and K De Valois. *Spatial Vision*. Oxford University Press, New York, 1990.

[60] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH 00*, pages 479–488, 2000.

[61] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH 01*, 2001.

[62] Y. Xu, B. Guo, and H.-Y. Shum. Chaos mosaic: Fast and memory efficient texture synthesis. Technical Report MSR-TR-2000-32, Microsoft Research, April 2000.

[63] Y. Yacoob and M. J. Black. Parameterized modeling and recognition of activities. *Computer Vision and Image Understanding*, 73(2):232–247, 1999.

[64] L. Ying, A. Hertzmann, H. Biermann, and D. Zorin. Texture and shape synthesis on surfaces. In *Eurographics Workshop on Rendering*, June 2001.

[65] Lihi Zelnik-Manor and Michal Irani. Event-based video analysis. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.*, 2001.

[66] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively variant textures on arbitrary surfaces. In *SIGGRAPH '03*, 2003.

[67] Tao Zhao and Ram Nevatia. 3d tracking of human locomotion: a tracking as recognition approach. In *International Conference on Pattern Recognition*, August 2002.

[68] Tao Zhao, Ram Nevatia, and Fengjun Lv. Segmentation and tracking of multiple humans in complex situations. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, 2001.

[69] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame). *International Journal of Computer Vision*, 27(2):1–20, March/April 1998.

Appendix A

Berkeley's Longest Paths

or, why I took so long to graduate

Один сидит – читает эпос,
Другой сидит и смотрит эрос,
а где-то в Соде некто Эфрос
Сжимая мышь встречает Эос.
Бродил бы меж холмов и хижин,
Но дело есть: computer vision.

Вот день настал, что всем известен:
Один сидит – горланит песни,
Другой сидит и хоть ты тресни,
А некто Эфрос не на месте...
Он бродит меж холмов и хижин –
Он позабыл computer vision!

Tolya Grinshpan

It is with a heavy heart that I prepare to leave Berkeley, which has been my home for the last six years. But before I go, I will pay my last tribute to this wonderful town so neatly tucked into the East Bay hills right across from the Golden Gate Bridge. With its varied terrain, good views, and temperate climate, Berkeley is a perfect place for walking and hiking. Unfortunately, most people (brainwashed by the American car culture) are not aware of all the possibilities when they arrive to Berkeley, and when (as in my case) they finally discover these hidden treasures it is already time to leave.

In this Appendix (which will likely become the most useful part of the present manuscript), I will briefly describe some of the walking and hiking paths in and around Berkeley that I have discovered during my stay here. The list is by no means complete, and I will leave it as an enjoyable exercise to the reader to discover others. Unless otherwise stated, all hikes originate from the Berkeley campus and do not require a car. Distances are given in minutes of not-too-slow walking and I have tried to arrange the hikes in order of increasing distance.

The descriptions should be treated as ideas or suggestions, not detailed instructions; it is assumed that you also have access to a good map of Berkeley (e.g. [3]), maps of the Berkeley hills (the free map of Tilden Park, or better yet, the *Rambler's Guide to the Trails of the East Bay Hills/Map: Northern Section* [40] available at Cody's Books) and/or GOOGLE.

Sunset Walks

> *One day I saw the sun set forty-four times! ... You know, when you are feeling very sad, sunsets are wonderful...*
>
> ANTOINE DE SAINT-EXUPÉRY

Berkeley has numerous wonderful spots to watch the sun set. Here I will describe a few very close to campus, perfect for a quick escape from the lab.

Rose Garden (15 min.): Start at the North Gate (Hearst Ave. and Euclid Ave.) and walk north on Euclid about 8 blocks. Past Bayview Place you will see the Rose Garden on your left. Most of the year, this is not that great a place for sunsets (Mt. Tamalpais blocks part of the view) but the Garden itself is beautiful, especially when bathed in the golden light of the setting sun.

The Bay through the Campanile (15 min.): The walk starts from the lower parking lot of the Lawrence Berkeley Lab, on the east side of campus. To get there, start at the corner of Gayley Rd. and Stadium Rim Way and walk behind the Greek Theatre, where you will find stairs up to the parking lot. Alternatively, you can walk up Hearst Ave. and as it becomes Cyclotron Rd., and turn right into the parking lot.

Walk to the south end of the parking lot and you will see a dirt road beginning right where the parking lot ends. Walk up this road as it turns left and proceeds uphill. When in doubt, always choose the path going up. When the road hits another one, turn right and walk a couple of minutes to find a good place to watch the sunset. There is one spot from where, through the bells of the Campanile, you can see the Bay and a bit of the Golden Gate Bridge.

On The Rock, above it all (20 min.): This is my favorite place to watch a sunset. It is also very convenient if you are on the south-east end of campus (e.g. I-House). First, walk to the east end of the lower Dwight Way (at Fernwald Rd, on the northern edge of the Clark Kerr campus). From there, you will see a trail going uphill. You are now entering the Claremont Canyon Regional Preserve, one of Berkeley's secret jewels. As you walk up, you are suddenly not in Berkeley anymore, but in an idyllic pastoral painting, somewhere in the Swiss Alps, with tall trees, green (if you come in the spring) rolling hills, singing birds, crickets, the smell of quiet serenity... Follow the trail for about 5-10 minutes and you will find a big rock on the side of the hill. Sit on the rock, open a bottle of wine, and *watch*: down below opens a magnificent view of the entire Bay Area – Berkeley, Emeryville, Oakland, Richmond, Marin, San Francisco – all hustling and bustling, little houses everywhere, tiny cars driving over the miniature freeways, speckles of light starting to turn on all over the place as the sun slowly dips into the Pacific Ocean behind the Golden Gate Bridge... while you are sit above it all, sipping your wine in this secret island of untouched wilderness.

Following the trail a bit further, you will find a second, larger rock. Although it's more comfortable and higher up, the view from this rock is less intimate, less magical. You can follow the trail all the way up to Dwight Way (this is the upper continuation of Dwight, disconnected from the lower one), turn left and return to Berkeley through the streets, making a nice loop.

Lawrence Hall of Science (35 min.): The view from the Lawrence Hall of Science is great, but getting there is a bit tricky. One can walk up along Centennial Dr. but that's not very pleasant. One can take the Fire Trail but that's too long. Recently I have found a path that's relatively quick and nice. Start from north campus, the corner of Hearst and LeRoy (i.e. Soda Hall). Walk north on LeRoy (follow the stairs), right on Cedar, left on La Loma, right on Buena Vista, left on Delmar, right on Campus Dr. until it ends on the side of the hill. There a trail will start, which you should follow, going uphill at every fork. The trail will eventually lead to the Hall of Science parking lot, but it might be nicer to just stay there on the hill – the view may not be as panoramic, but more peaceful and quiet.

Taking the Fire Trail back will make this a nice (but long) walk. From the Hall of Science central parking lot, take the stairs all the way up to MSRI. The Fire Trail, a flat serene wooded path good for jogging, starts from there and after an hour deposits you on Panoramic Way (in the area around the previous hike), where you can take the streets and stairs down to campus.

Indian Rock (60 min.) A huge rock, with lots of people and great views, awaits you after an easy (but long) walk along the beautiful streets of North Berkeley. Starting from the corner of Hearst and Oxford (north-west corner of campus), walk north on Oxford for a *long* time until it ends at the Indian Rock Park. Climb the rock (there are stairs at the back for the less adventurous), get our your baguette, Camembert, and a good bottle of Malbec, and you are ready for a proper sun farewell ritual. Afterward, you can walk west along the hidden Indian Rock Path down to Solano Ave. for a good dinner and take a bus back home.

Purpose Hikes

These hikes have a destination, a definite purpose, which prevents them from becoming boring even after many many trips.

Swimming at Lake Anza (1 hr. one way): Lake Anza, a lovely forest lake in the heart of Tilden Park, is perfect for swimming from May to October. In the 1960s, it was a famous midnight skinny-dipping spot and even now you can occasionally spot a naked, badly-aged hippie or two. The best time to swim is in the evening when it's free and there are few people (when the lifeguards are on duty, 9am-6pm, it's often crowded and you will have to pay $3). I like to come right after sundown, when the lake is calm and deserted: the water is perfectly still, reflecting the surrounding forest, steamy mist slowly rising up from the surface illuminated by the moonlight; ducks quacking in the distance. Sometimes the fog rolls in over the lake and is so thick that you have a feeling of swimming in cotton wool, parting it with your hands.

To get to Lake Anza start by following the directions to the Rose Garden above. After a mandatory stop in the garden (to smell the roses, of course!) continue on Euclid until you

find the Tamalpais Path, neatly tucked in at the end of Codornices Park on your right. Go up the steps to Tamalpais Rd, turn left and then left again onto Shasta Rd. Now stay on Shasta all the way to the top (stay alert – Shasta turns a lot and will try to break free of you). Keep your eyes open for various seasonal fruits along the way: plums, blackberries, apples, pears, and figs. When you reach Grizzly Peak Blvd., turn left and after half a block, Shasta will resume on your right. At this point you are at the global maximum and there are several ways to get down to the lake. The most scenic way is to continue on Shasta for a few minutes until you see a sign for Selby Trail on your left. Following the trail will eventually get you down to the lake (having a Tilden map for reference is recommended).

Riding the Steam Train (1.5 hr. one way): Did you know that there is a steam train up in Tilden Park? There are actually two: a small one and a *really* small one, and you can ride both (the first one, run by the Redwood Valley Railway Co. costs $1.75 and is open on weekends and holidays; the other one is free, open sporadically, usually on Sunday afternoons). Yes, you can drive up there, but then you will be riding the little kiddie train feeling stupid, instead of feeling proud of what you have accomplished.

This hike can be started from many different places. Here we will describe just one – the continuation of our sunset hike "The Rock". From the rock, continue uphill on the trail until you emerge on the continuation of Dwight Way. Turn right and walk uphill, along the edge of the Claremont Preserve on your right. When the street turns left continue going up on a trail which (if you are persistent in going up) will eventually emerge on Panoramic Way. Turn right and keep walking up as the street ends (at the gates of a very very fancy house) and becomes a dirt road going up the ridge between the Lawrence Berkeley Labs valley and the Fire Trail on the left, and the Oakland hills on the right. Continue on this road for a while (at forks, choose the road going straight east). Finally, the trail will turn left and after a minute you will emerge on Grizzly Peak Blvd. Turn right and walk 3-5 minutes until you see the Steam Trains sign on your left. You can return the same way, or forge ahead (see Grand Hikes below).

Grand Hikes

The best way to hike is to wake up in the morning and walk, walk, walk until you get somewhere else all tired but very happy and content. No cars, no long drives, no hassle. Here I list a few of my "Grand Hikes", which start right at your doorstep and keep going as long as you want, passing through some of the most beautiful scenery in the Bay Area (especially in springtime).

Berkeley to Orinda BART (3-4 hrs): This is a nice half-day hike, perfect for a warm spring afternoon slowly turning into evening. Officially, you will need an EBMUD hiking permit ($10 for a year including family and 3 guests, by mail or at Tilden Park Nature Area), but it doesn't seem like anyone checks.

First, follow the path to the Steam Trains as discussed above. From there, go to the south end of the parking lot to find the trailhead for the Skyline Trail (also known as Bay Area Ridge Trail). Following the trail for about 15 minutes, you will come to a clearly marked fork. The Skyline Trail will continue to your right (see the Chabot hike below), but you will take the De La Veaga trail to Orinda (left). The trail rolls around beautiful green

hills, with not a soul in sight for miles, eventually coming up to a ridge from where you can see the entire Orinda valley. From there the trail slowly descends and comes out right next to the Orinda BART station (cross the street and walk right along the foot bridge over the freeway to get to the station). From there, it's only 3 stops and you are back home in Berkeley!

Berkeley to Richmond on Skyline Trail (4-5 hrs): The first part of the hike follows the trail to lake Anza (above). From there you can take Wildcat trail (north end of the lake), turn right onto Currant Trail which gets you to Inspiration Point. From there, the well-maintained Nimitz Way heads north along the ridge with breathtaking views of both the Bay and the Orinda Valley. Along the way, you can take a short detour to the Rotary Peace Grove at the Wildcat Peak (wildcat peak trail) for an even better 360-degree view. Back on Nimitz, after a while you will be leaving Tilden Park and entering the neighboring Wildcat Park. Follow Nimitz, eventually taking a left onto the Belgum trail which will bring you down to the Alvarado Staging Area. On weekdays, AC Transit bus 68 will take you to Richmond BART or El Cerrito BART (no bus on weekends). Alternatively, if you are bored of walking on the ridge all the way, you can take other trails in the valley (look at the Tilden and Wildcat maps for ideas). If it's spring, you really can't go wrong, no matter which trail you take – the whole area is amazingly beautiful, with gorgeous views and lots of green open space.

Steam Trains – Inspiration Point – Lake Anza Loop (5-6 hrs): Hike up to the Steam train as described earlier. Go onto the road at the east end of the parking lot; the trailhead will be in a couple of minutes. Either go straight onto the Bayview Trail or go on the Volmer Peak trail, switching to Bayview later (see the free Tilden map). The trail goes along the ridge separating Berkeley from Orinda valley all the way to Inspiration Point. From there, take the Currant Trail down to Lake Anza for a swim, before heading home as described in the Lake Anza section.

For a shorter hike, from the steam train, take the Volmer Peak trail, onto the Lupine trail, then cross the road to the Redwood trail which will bring you to MSRI and the Lawrence Hall of Science (discussed above).

Berkeley to Chabot Observatory on the Skyline Trail (6-7 hrs): The beginning of this hike, again, follows the path to the steam trains. From there, we will take the Skyline Trail going south (trailhead at the south end of the parking lot). Here I will only sketch the trail – the free Skyline Trail map (from the Tilden people) is very useful for details. Note that you don't need the EBMUD permit to walk this trail. The Skyline Trail traverses the rolling spine of the Oakland-Berkeley hills. The scenery is very varied, from fields and pastures, to groves and forests. On the way, you will be passing through several parks. The trail is well-marked and easy to follow. First, you will walk high over the Hwy 24 tunnel and arrive at the Sibley Volcanic Regional Preserve. From there you will walk through Huckleberry Botanic Regional Preserve (yes, there are Huckleberries to be picked, when in season) and into the Skyline Gate of the Redwood Regional Park. The Chabot Observatory is in Redwood park (get a map). From there, there is a bus to take you to the Fruitvale BART station. Also, sometimes in the evenings there are free telescope viewings at the observatory.

San Pablo Reservoir at sunset, as seen from Inspiration Trail

San Pablo Reservoir Loop (7-8 hrs): A wonderful up-and-down-and-up-and-down hike with many lakes and constantly changing scenery[1]. First, follow the trail to lake Anza (above), but continue on Selby trail almost until it ends, then turn right onto Memory trail. Walk on to Jewel Lake (stopping at the Tilden Environmental Education Center if you don't have the EBMUD permit that you will need later on). From the lake, take the Wildcat Peak trail up the hill to the Rotary Peace Grove for one of the best 360 panoramic views of the Bay Area. From there, go north, joining Nimitz Way. Walk on Nimitz for a while, crossing into Wildcat Park, before turning right onto the Eagle's Nest Trail, an unmaintained EBMUD-land path winding its way down the hill toward the San Pablo Reservoir. This is my favorite part of the hike – the slow descent toward this huge lake through the tall sunlit grass fields and (at the end) a thick, dark Eucalyptus forest. The trail ends on San Pablo Dam road. Cross it and go toward the Reservoir (there is a nice park but, alas, no swimming allowed). A pleasant path, called the Old San Pablo Dam road, follows the bank of the Reservoir; follow it south for an hour or so. Eventually you will come back onto the Dam road just in time to catch a trail back home, called the Inspiration trail, taking you to Inspiration Point. From there, take the Currant trail to lake Anza, and proceed as before.

San Francisco and Beyond

Because San Francisco is just a short 20-minute BART ride away, I can't resist briefly sketching a couple great walks in the City. Crossing SF on foot is a perfect way to get to know it (SF is just 7 miles across!). There are many many ways of doing this. One of my favorites: start at Embarcadero BART. Walk through downtown and up Columbus (past the City Lights bookstore and many good Italian bakeries) to Washington Square Park. Take a look at the blatantly socialist-realism murals *a la* Diego Rivera by making a quick detour up to Coit Tower (go east and follow the Filbert Steps, a lovely garden staircase in the Mediterranean tradition). From there, get back to Union St. and walk west all the way to Presidio. Turn right and go down to the Bay. Turn left, walking along the Bay on the Coastal Trail to the Golden Gate Bridge. Continue past the bridge, walking on paths and roads until you hit Land's End – a gorgeous park at the northwest tip of the City. Continue along the coast, turning the curve right into the Pacific Ocean, past Sutro Baths, the Cliff House and finally onto Ocean Beach. If you are just interested in the beach (and sunsets!), go to Fort Funston (MUNI bus #18) and, after spending some time looking at the hang gliders who nest there, walk north along the beach all the way to Land's End. Or, you can

[1] See `http://www.cs.berkeley.edu/~efros/photos/Albums/BerkeleyHillsHike/` for illustrations

take MUNI bus #76 (Sundays only) to Point Bonita Lighthouse across the bay in the Marin Headlands and (after seeing the lighthouse and playing on the beach) follow the Coastal Trail back to SF across the Golden Gate Bridge, pausing on the hill overlooking the bridge on the Marin side, just as the sun sinks into the Pacific, making it look even more golden than usual. Or..., well, the possibilities are endless!

And in case you have a car, going just a bit further (but still staying close the the coast) is certainly worth it. Marin County, just north of SF, is full of great places to hike including the Marin Headlands, Mount Tamalpais, Tennessee Valley, Muir Woods (the hike from Muir Woods to Stinson Beach is wonderful), etc. But the jewel of Marin is **Point Reyes**, a peninsula that extends far into the Pacific, offering a breathtaking coastline, mysterious fog-drenched landscapes, forest lakes, waterfalls, plus plenty of hiking, camping, and kayaking. For a one day trip, hike to Tomales Point and afterward drive to the lighthouse for a magnificent sunset and chance to spot a whale.

Starting just South of Monterey (and expending all the way down to San Luis Obispo) lies **Big Sur**. I won't be even attempt to describe it, except to say that it's an amazing, magical, even spiritual place, especially if you love The Ocean... When you go, Andrew Molera State Park is good for camping and Julia Pfeiffer Burns Park for hiking, plus there is a two-day hike to Sykes Hot Springs (crowded on weekends) where after a 10-mile trek you get to feel like a real nouveau hippie, soaking in a hot tub, naked, surrounded by an ancient forest far far away from civilization).

Finally, the **Lost Coast Trail** in the King Range Wilderness near the California/Oregon border is an amazing 3-6 day hiking experience. Only after spending several days on the thin strip of dry land sandwiched between the ocean on one side and steep, imposing cliffs on the other, can one truly appreciate the power, the willfulness, and the beauty of the Great Pacific.