5-14-2010

The Art of Retargeting (or Retargeting of Art): Fitting a Rectangular Peg into a Square Hole

Wui Ming Aldrich-Gan
Bard College

The Art of Retargeting (or Retargeting of Art): Fitting a Rectangular Peg Into a Square Hole

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Wui Ming Aldrich-Gan

Annandale-on-Hudson, New York
May, 2010

Abstract

Retargeting refers to the general problem of resizing an image or video, possibly changing its aspect ratio, while minimizing the loss and distortion of salient content. Traditionally, an image had to be cropped or stretched in one direction in order to change its aspect ratio, which resulted in lost or distorted content. In this paper, I investigate several retargeting algorithms that were published in recent years, which fall into one of two broad categories—those involving direct pixel removal (also known as seam carving) and those involving a non-uniform remapping of pixels (also known as warping). I then propose two new warping-based algorithms. The first is based on previous work using least-squares, with a new set of constraints that favor global optimization. The second is a recursive, divide-and-conquer algorithm. Real-world applications of image and video retargeting include graphic design and displaying widescreen films on a standard television screen.

Contents

List of Figures

Dedication

So long, and thanks for all the fish.

(Now, how about some squid?)

(Or lobster?)

Acknowledgments

I am forever indebted to the following people:

- my primary senior project adviser, MARY KREMBS, for her extraordinary patience and faith in me,
- the other members of my board, KEITH O'HARA and GREGORY LANDWEBER, for their invaluable advice and support,
- PETER GADSBY, for graciously sorting out this whole joint major thing for me at the last minute,
- ROBERT MARTIN and MELVIN CHEN, for being so friendly and helpful before even I set foot on this campus—or in this country,
- CARLEY GOOLEY, JENNIFER BIENER, STEPHANIE EISS, SARAH WEGENER, DAVID BLOOM, and SEAN COLONNA for being the most wonderful suitemates and floormates ever,
- the Distinguished Scientist Scholars Program and the Bettina Baruch Foundation for their generous scholarships, without which I would not be at Bard,
- every teacher I've ever had at Bard, for being among the most caring and understanding individuals in my life,
- Mom and Dad, for instilling in me a thirst for knowledge that can never be quenched,

and, most of all,

- my wife, MARGARET ALDRICH-GAN, for *everything*. I don't know how I could have made it through these last few weeks if I didn't know you were always here for me in spirit, if not in person. You mean the world to me, and I can't wait for this long-distance nonsense to be over.

1
Introduction

1.1 Motivation

Before widescreen televisions became widely available (and even now that they are), feature films were typically broadcast and released to the home video market in the standard television aspect ratio of $4:3$ (approximately $1.33:1$), regardless of the original aspect ratio in which they were shot and shown in theaters (most commonly $1.85:1$ or $2.35:1$) [8]. This problem of fitting a rectangular peg into an (almost) square hole is not unique to television; graphic designers are constantly faced with the task of fitting images into layouts that cannot accommodate their full width and/or height. The two techniques traditionally used to accomplish this are *cropping* and *scaling*.

1.2 Cropping

1.2.1 Introducing Pan-and-Scan

The industry-standard solution for narrowing the width of widescreen films, known as "pan-and-scan", is based on *cropping*, which is to restrict the visible portion of the image to a smaller rectangular region within the image. As an analogy, one may think of cutting

off rectangular strips from the edges of a photograph in order to reduce its size. During the pan-and-scan conversion process, an engineer moves the cropping window around throughout the duration of the film, in order to fit the most important portion of each frame of the film within the cropping window.

In Section 1.3.1, we will introduce a process called *uniform scaling*, which preserves both the image content and its aspect ratio. With that in mind, we realize that the actual dimensions of the cropping window are inconsequential as long as its aspect ratio matches that of the target dimensions, because the cropped image can later be uniformly scaled to the desired dimensions. It follows, then, that the largest cropping window that will fit within the boundaries of the source image (and thus maximally preserve the image content) is one that has the same height as the source image (since our goal is to produce a narrower image).

Films that have undergone the pan-and-scan process are typically preceded by the following disclaimer [9]:

```
This film has been modified from its original version.
It has been formatted to fit your screen.
```

1.2.2 Limitations of Pan-and-Scan

The pan-and-scan process has two major limitations. First, the target aspect ratio has to be known in advance, since it determines the size of the cropping window. With the recent proliferation of portable media devices of all shapes and sizes, it is becoming more difficult to predict the aspect ratio of the screen on which a film may be displayed. Despite purists' insistence that films be watched only in their original aspect ratio (which results in wasted screen space, as described in Section 1.3.1), there are many who would prefer the image to fill the screen, particularly if the screen is small. The crux of the problem is, of course, that the pan-and-scan process has to be performed manually. If the process

could be automated and implemented within these devices themselves, consumers would be able to watch films at arbitrary aspect ratios, with the conversion performed on the fly.

The second limitation of pan-and-scan is that it relies on the assumption that an optimal cropping window, i.e. a rectangular portion of the original image that represents the entire image, can always be found. Figure 1.2.1 shows a counterexample (one admittedly created by the author of this paper expressly to demonstrate the fallibility of pan-and-scan). This image, in the widescreen aspect ratio of $2.35 : 1$ commonly used in blockbuster films, features two characters—one on the extreme left of the frame, the other on the extreme right. A pan-and-scan cropping window, with the standard television aspect ratio of $1.33 : 1$, is shown with a red outline. Suppose that the two characters are having a conversation with each other. Since it is impossible to fit both characters at once in the cropping window, the pan-and-scan operator must either focus on one character (and have the other speak off-screen) or pan back and forth between them (which may induce motion sickness in the audience). Either way, the original composition of the shot is misrepresented.

Figure 1.2.1. An example where pan-and-scan fails, with cropping window outlined in red. Images of Darth Vader and Voldemort © Lucasfilm and Warner Bros.

1.3 Scaling

1.3.1 Preserving Aspect Ratio

To *scale* an image is to stretch or shrink it along one or both of its axes. Clearly, *uniform scaling*, i.e. applying the same scaling factor to both the horizontal and vertical directions, does not (appreciably) alter the image content. Mathematically, uniform scaling is lossless if images have infinitesimal detail. However, digital images and video are by their very nature discrete; if we were to continuously enlarge an image on a computer, we would quickly reach a point beyond which no additional detail can be recovered, resulting in a blurry or blocky picture. Nevertheless, we will assume that an image that has been uniformly scaled up or down "looks" the same, except larger or smaller. In particular, uniform scaling preserves the *aspect ratio* of an image, which is the ratio of its width to its height.

If we were to watch a widescreen presentation of a film in its original aspect ratio on a standard (non-widescreen) television, we would notice thick "black bars" running horizontally across the top and bottom of the screen. This situation, known as *letterboxing*, arises because the original theatrical frames have simply been scaled uniformly to fit within the dimensions of the television screen. Since the original aspect ratio, which is wider than that of a television screen, is unchanged, the screen is taller than the image, resulting in wasted space filled by black bars. We will not consider letterboxing to be a solution to the problem put forth in Section 1.1.

1.3.2 Changing Aspect Ratio

In order to change the aspect ratio of an image using scaling alone, a different scaling factor must be applied to each of the two directions. In the case of scaling a widescreen film to fit a television screen, the vertical scaling factor must be greater than the horizontal scaling factor. Intuitively, you could think of vertically stretching the letterboxed image described in the previous paragraph to fit the dimensions of the screen (or, equivalently,

squishing the image horizontally). In general, this is a bad idea, since everything in the stretched (or squished) image would look disproportionately thin and tall. Figure 1.3.1 shows us what the example from Figure 1.2.1 would look like if it were scaled to the aspect ratio of a television screen.

Some images, however, appear to be more resilient to stretching than others. For instance, photographs of the sky can be stretched or squished in either direction by a considerable amount without looking unnatural. Intuitively, it is easy to see why this is so; a photograph of a uniformly blue sky looks exactly the same no matter how much you stretch it in either direction. Throw in some clouds and the photograph will start to look different when you stretch it, but due to their amorphous nature, stretched-out clouds will still look like clouds.

In contrast, photographs containing highly-defined structures like buildings and vehicles are much less forgiving. The effect is particularly disturbing with photographs that prominently feature human beings, perhaps because we interact with properly-proportioned human beings so frequently in daily life that we can detect even the slightest amount of distortion in a photograph of a human being. These observations, subjective as they are, lead us towards the notion of *saliency*, which we will discuss in Section 2.1.

Figure 1.3.1. The example from Figure 1.2.1, scaled to the $1.33 : 1$ aspect ratio.

1.4 Alternatives to Cropping and Scaling

1.4.1 Open Matte

The discussion of pan-and-scan in Section 1.2 is complicated by the fact that, historically, many widescreen films were originally shot in a taller aspect ratio [10]. Markings on the camera's viewfinder served as guides for the directors and camera operators to see where the frames would be cropped. Usually, the cropping would take place in the movie theater, with the top and bottom portions of the projector obstructed by a *soft matte* (as opposed to a *hard matte*, which masks the top and bottom portions of the camera lens during the filming process itself).

When releasing soft-matted films to the home video market, studios have the option of including the full, unmatted frames (known as *open matte* frames), sparing important characters and objects on the left and right edges of each frame from being cropped away. Unfortunately, this occasionally results in microphones and other objects that are supposed to be hidden by the soft matte being inadvertently revealed. For example, Figure 1.4.1 shows a scene from *A Fish Called Wanda* in which John Cleese's character is naked when his family walks in on him. In the open matte version, we see that the actor is actually wearing pants, and the comedy of the scene is ruined.

Figure 1.4.1. Example of an open matte blooper. [10]
Still frame from *A Fish Called Wanda*, © MGM.

Needless to say, open matte, like letterboxing, is a non-solution for our purposes, because it relies on the availability of additional visual information above and beyond what was originally shown in theaters. In our quest to fit a widescreen film to a standard television screen, we will assume that the only content available to us is that which is present in the source material. If neither cropping nor scaling is a satisfactory solution to this problem, are we out of options?

1.4.2 Digital Recomposition

In a fantasy world where filmmakers have unlimited resources, the entire film could be shot from scratch again, with the director reframing each shot to conform to the new aspect ratio. As it turns out, such a fantasy world does exist, though it is a virtual one—beginning with *A Bug's Life*, Pixar Animation Studios has released each of their digitally animated films on DVD in both the widescreen version originally shown in theaters, and a newly reframed version for television screens. Figure 1.4.2 illustrates an example, with the original widescreen version on the left and a reframed version on the right. At first glance, the latter might simply look like an open matte version of the former, but a closer look reveals that the characters have been moved leftward in relation to their surroundings.

In 1999, inspired by Pixar's reframing work, a new digital recomposition system was announced called FlikFX [12], named after the hero in *A Bug's Life*. If the ludicrous results

Figure 1.4.2. Example of Pixar's digital recomposition. [11]
Still frame from *A Bug's Life*, © Pixar Animation Studios.

Figure 1.4.3. Example of the fictitious FlikFX Digital Recomposition System. [12]
Still frame from *Lawrence of Arabia*, © Columbia Pictures.

(see Figure 1.4.3 for an example) were not enough to tip a reader off that it was a hoax, one had simply to look at its purported "release date" to realize that it was nothing but a clever April Fools' joke. Nevertheless, whether or not its creators intended for anything serious to come out of it, this website set the stage (or the screen, as it were) for the retargeting algorithms that we will discuss in Chapter 2.

I shall end this chapter with a manually-recomposited version (Figure 1.4.4) of the widescreen example from Figure 1.2.1. Of course, this is possible only because the original example was a self-created one, and I had the foreground and background elements stored as separate layers in a Photoshop file. Wouldn't it be lovely if we could obtain similar results from a regular, "flat" photograph—and algorithmically, to boot?

Figure 1.4.4. The example from Figure 1.2.1, recomposited for the $1.33:1$ aspect ratio.

2
Previous Work

2.1 Measuring Saliency

The general problem of resizing an image, possibly changing its aspect ratio, while preserving the original image's content and intent as best possible, is often referred to in the literature as *retargeting*. In Section 1.3.2, we observed that some images are more resilient to distortion than others. More generally, some *regions* of an image may be more resilient to distortion than their neighbors. These regions tend to be those containing visual information that one might consider to be relatively "unimportant" like the sky, vast stretches of uniform-looking terrain, or blurry shapes in the background.

We formalize these intuitions by introducing the concept of a *saliency map*, also called an *energy function* or a *significance map* in various papers. That is, to every pixel of an image, we assign a *saliency* value that describes its relative importance. Obviously, the measure of saliency is an extremely subjective notion that is difficult to quantify. Nevertheless, many different attempts to quantify it can be found not only in the literature, but in actual use in a variety of image processing and computer vision algorithms, image retargeting being but a tip of the iceberg.

One of the most widely-used measures of image saliency is the magnitude of the *image gradient*. Intuitively, the gradient magnitude at each pixel is a measure of how dissimilar it is to its neighboring pixels. Large gradient magnitudes correspond to sharp changes in color and intensity, indicating the presence of edges or highly-detailed textures. Conversely, pixels with low gradient magnitudes are those that appear to "blend in" with their surroundings. The image gradient magnitude, at which we will take a closer look in Section 3.4.1, is a simple yet effective measure of saliency, which is why it is used in all three retargeting algorithms described in this chapter, sometimes in conjunction with more sophisticated techniques to measure saliency.

One such technique is *face detection*, an increasingly ubiquitous technology that has been integrated into many modern digital cameras. As we hypothesized in Section 1.3.2, our intimate familiarity with the human form may be reason enough to accord human features special importance in the quantification of saliency. The "model of saliency-based visual attention for rapid scene analysis" developed by Itti, Koch, and Niebur in 1998 [2] has also been incorporated by many other researchers into their saliency maps. This model, inspired by a biological model of primate vision, uses a neural-network process to analyze images at various scale levels and identify regions with different color, intensity, and orientation properties than their surroundings.

2.2 Seam Carving

In Section 1.2, we suggested that the width of an image could be reduced through cropping, i.e. the removal of vertical columns from the extreme left and right sides of the image. What if, instead of removing columns from the sides of an image, we removed columns from the middle? For an image with prominent features on the extreme left and right sides, and nothing but blue sky in the middle, removing a column of sky may well be the optimal retargeting strategy.

Of course, most images would not lend themselves so easily to the removal of whole columns, so why not allow the removal of curved paths instead, which bend and curve around prominent objects to avoid distorting them? We have to impose only two conditions on each of these paths—they must contain exactly one pixel in each row (or one pixel in each column if the goal is to reduce the image height) so that the resulting image is still rectangular in shape, and they should be connected (at least in a discrete sense, so that two pixels on the path that are vertically one pixel apart should also be no more than one pixel apart horizontally) in order to preserve spatial continuity.

This is precisely the strategy proposed by Shai Avidan and Ariel Shamir [1] in their groundbreaking 2007 paper. Referring to these connected, monotonic paths as *seams*, they named this technique *seam carving*. Not only can seams be removed to reduce an image's width or height—they can also be inserted to increase an image's width or height, as shown in Figure 2.2.1. Given an energy function (their term for saliency), Avidan and Shamir's algorithm uses dynamic programming to identify the optimal order of seam removal or insertion, whereby seams with the lowest energy (saliency) are removed first. Although this paper remains agnostic as to the specific choice of the energy function; most of its examples use the magnitude of the image gradient.

A year later, they teamed up with Michael Rubinstein [3] to improve the seam-carving algorithm. Recognizing that seam removal sometimes results in an increase in the total energy of an image (due to previously non-adjacent pixels coming into contact with each other), their improved algorithm removes seams by looking forward at the resulting image, so as to minimize the increase in total energy. In addition, they adapted their algorithm to work with video in addition to still images. It should be noted that video retargeting is not as simple as applying an image retargeting algorithm to each individual frame, because there is no guarantee that continuity will be maintained between frames.

Figure 2.2.1. Example of seam carving. [1]
Above: Original image, with lowest-energy horizontal and vertical seams shown in red.
Below: Retargeted image.

The most intuitive way to handle video retargeting is to treat a video clip as a cube, with two dimensions of space and a third dimension of time—one might think of it as stacking all the individual frames on top of each other. This way, any notion of spatial continuity employed in an image retargeting algorithm can naturally be extended to temporal continuity, simply by treating time as if it were a third spatial dimension. Rubinstein, Shamir, and Avidan extended the concept of seams to video by defining a three-dimensional seam as a connected two-dimensional manifold that spans both the width and the height of the video frame.

Seam carving is relatively easy to understand and implement, as evidenced by the sheer number of home-brewed implementations that sprouted all over the Internet shortly after Avidan and Shamir published their first paper. However, seam carving has three major disadvantages which make it, in my opinion, less promising than the warping-based approaches that we will examine in the next section. First, the direct removal of pixels

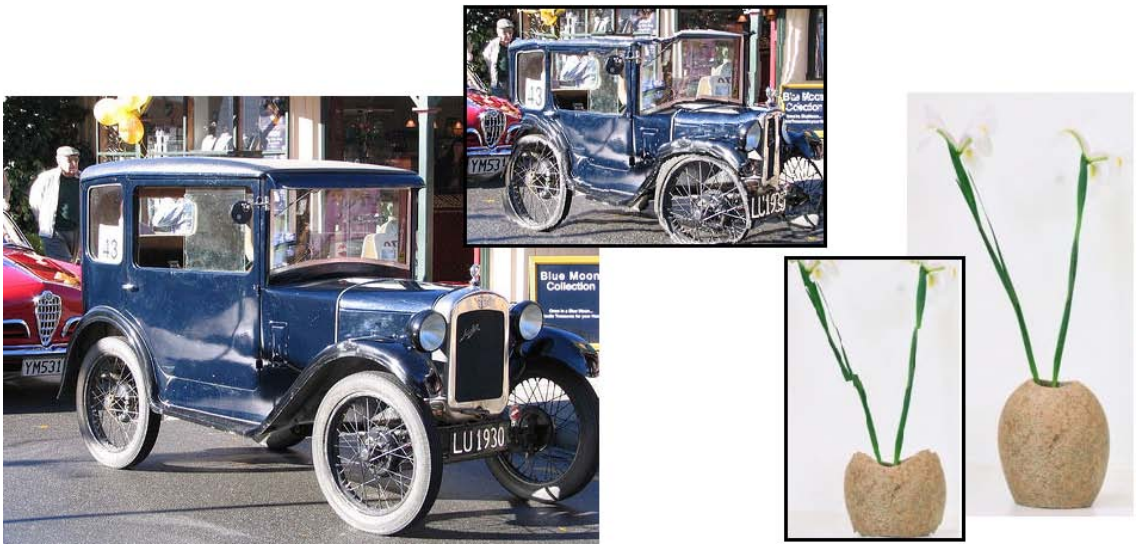


Figure 2.2.2. Examples where seam carving fails. [1]

inevitably causes visual discontinuities in the resulting image. The algorithm aims to minimize these discontinuities by removing low-energy pixels, but realistically, with most images, optimal seams cannot always be found, which brings us to the second problem. The requirement that seams be connected and span the image's width or height often makes it difficult to find seams that do not intersect prominent features in the image, as is the case with the two examples shown in Figure 2.2.2. The third disadvantage is that seam carving can only occur in one direction at a time, since seams are by definition either horizontal or vertical. In Section 2.3.2, we will see that retargeting an image along both axes at once can yield more consistently balanced results.

Shai Avidan now works for Adobe Systems Inc., and seam carving has been implemented in the latest version of Adobe Photoshop (i.e. CS4). Available in the *Edit* menu under the name *Content-Aware Scale*, the new seam-carving interface allows the user to "protect" a manual selection (which can be made using the standard Photoshop tools) from distortion. This feature is helpful in cases where the automatically-generated saliency map is inadequate, as can be seen in Figure 2.2.3.

Figure 2.2.3. The example from Figure 1.2.1, retargeted using Photoshop's seam-carving feature, with *(left)* and without *(right)* manual "protection" of the two characters.

2.3 Warping-Based Retargeting

If seam carving is seen as a sophisticated version of cropping, the analogous counterpart to scaling would be *warping*, or non-uniform scaling. Technically, a warp is simply the result of applying an arbitrary transformation map to an image, a process we will examine more closely in Section 3.5. Informally, to warp an image is to stretch and squish different parts of the image differently. Recalling our preliminary discussion of saliency in Sections 1.3.2 and 2.1, it is easy to see how warping lends itself well to image retargeting. Non-uniform scaling inevitably creates distortion, but by spreading the distortion over less salient regions of an image, thus avoiding distortion in high-saliency regions, we may be able to obtain more visually pleasing results than if the distortion were distributed evenly across the whole image.

2.3.1 An Algorithm Using Least Squares

In 2007, Lior Wolf, Moshe Guttmann, and Daniel Cohen-Or proposed an algorithm for retargeting video in real time [5], based on finding a least-squares solution to an overdetermined system of linear equations. Suppose that we wish to alter the width of a video clip, without vertically moving any of the pixels. This amounts to redefining the

x-coordinate of each pixel in each frame, upon which the algorithm imposes the following constraints:

Boundary substitutions: The leftmost pixels should stay fixed, while the rightmost pixels should conform to the target width.

Importance modeling: Salient pairs of horizontally adjacent pixels should remain approximately one pixel apart, while less salient pairs can be allowed to drift closer together or further apart.

Spatial and time smoothness: Each column of pixels should have approximately the same x-coordinate. In addition, each pixel should have approximately the same x-coordinate as in the previous frame.

In Section 4.2, we will see how to translate these constraints into an overdetermined system of linear equations, for which we can then find a least-squares solution. The saliency measure used by Wolf, Guttmann, and Cohen-Or is a combination of the image gradient magnitude, face detection, and motion detection. The inclusion of face detection ensures that human faces remain proportionally intact, although the same cannot necessarily be said of their bodies, as we see in Figure 2.3.1.

Figure 2.3.1. Example of the retargeting algorithm by Wolf, Guttmann, and Cohen-Or. [5]

2.3.2 An Algorithm Using Global Optimization

Both retargeting algorithms described thus far in this chapter work only in one direction at a time. Figure 2.3.2 demonstrates why such algorithms are inherently limited. The second column of images from the left contains the results of width reduction of the original images (shown in the leftmost column), using the retargeting algorithm by Wolf, Guttmann, and Cohen-Or described in Section 2.3.1. The *girl* image in the top row is retargeted more successfully than the *car* image in the bottom row, to which a significant amount of visible distortion has been introduced by the retargeting process. If an alternate strategy is employed whereby each image is first uniformly scaled down to its desired width, then retargeted up to its original height, the *car* image fares better than the *girl* image, as shown in the third column from the left. Unfortunately, as is evident from this example, there is no way to predict which of the two strategies would be more successful.

The rightmost column of images in Figure 2.3.2 presents the results of a retargeting algorithm published by Yu-Shuen Wang, Chiew-Lan Tai, Olga Sorkine, and Tong-Yee Lee

Figure 2.3.2. A comparison of the algorithms.

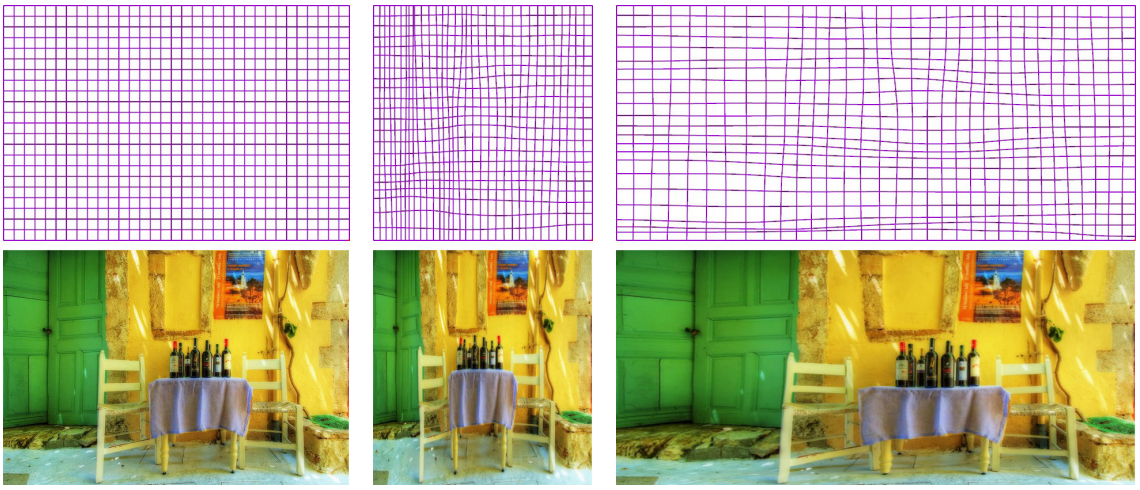


Figure 2.3.3. Example of the retargeting algorithm by Wang, Tai, Sorkine, and Lee. [4]

in 1998. They observed that one-dimensional retargeting algorithms propagate distortion only in the direction in which an image is being resized. This works well with the *girl* image in Figure 2.3.2 because the left half of the picture is a primarily low-saliency region. The *car* image, on the other hand, has little room for distortion along the x-axis; the top half of the image, however, consists mainly of low-saliency pixels, which is why the algorithm of Wolf, Guttmann, and Cohen-Or performs better when the indirect strategy of downscaling the image prior to retargeting along the y-axis is employed.

Based on this observation, Wang, Tai, Sorkine, and Lee formulated an algorithm that allowed distortion to be propagated along any direction in the two-dimensional plane. This algorithm partitions the source image into a mesh of quads, as illustrated in Figure 2.3.3. The vertices of the mesh are then retargeted using an iterative, global optimization process that minimizes quad deformation and grid line bending. As can be seen in Figures 2.3.2 and 2.3.4, this algorithm consistently performs better than the other two algorithms described in this chapter.

Figure 2.3.4. Comparison of several retargeting algorithms: *(from left to right)*the original image, the result of seam carving, the result of the retargeting algorithm described in Section 2.3.1, and the result of the retargeting algorithm described in Section 2.3.2

3
Pixels and Fundamentals

The papers referenced in the previous chapter were written, by and large, for an audience whose primary interest lies in computing rather than in mathematics. Consequently, the notation used in each paper is sometimes less precise and consistent than a mathematician might expect. In this chapter, I will attempt to construct, as formally as possible, a mathematical framework for image retargeting that we will use in the rest of this paper.

3.1 Notational Conventions

Throughout this paper, the set of *natural numbers*, written $\mathbb{N}$, will not include zero. We will use the square-bracket notation to denote closed intervals over the real numbers. In addition, we will introduce a discretization operator, written as a subscripted + symbol, to restrict a continuous set to integral values, as seen in the following definition:

Definition 3.1.1. Let $a, b \in \mathbb{R}$.

1. The **continuous interval** from a to b, written $[a, b]$, is the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$.

2. The **discrete interval** from a to b, written $[a, b]^+$, is the set $[a, b] \cap \mathbb{Z}$. $\triangle$

We will, however, use the notation (a, b) exclusively to denote an ordered pair, *not* an open interval. Furthermore, if f is a function that takes an ordered pair as its argument, we will sometimes write $f_{a,b}$ instead of $f(a, b)$, in order to reduce the number of parentheses and thus enhance readability.

3.2 Images as Functions

What is an image? To a computer program, an image is typically a matrix (or two-dimensional array), with each entry describing the visual appearance of each pixel. Monochromatic images, also known as grayscale images, have only one value attached to each pixel, describing the intensity of light—zero corresponds to black (or the absence of light), while the maximum value corresponds to white (or the maximal amount of light). Since data is stored discretely on a computer, this intensity value can only take on discrete values; the greater the number of bits used to represent each intensity value, the greater the number of distinct intensity values, and the "smoother" the appearance of the image.

Color images, on the other hand, are generally represented using more than just a single value. Among the many *color spaces* (i.e. ways to represent color information) available, the one most frequently used is the RGB color space, which stores three separate intensity values, one for each of the red, green, and blue *channels*, respectively. Alternate color spaces include RGBA (which adds an *alpha*, i.e. transparency, channel to the RGB color space), CYMK (which separates color information into four channels, namely cyan, yellow, magenta, and black), and HSV (which represents color information in terms of its hue, saturation, and brightness value).

For our purposes, it will be more convenient to think of images not as matrices, but as functions mapping points in a discrete and finite subset of a plane to points in a color space; this will allow us to then extend the notion of images so that they could be continuous,

which will be useful for reasons that will become clear in subsequent sections. With the above observations in mind, let us begin to build a mathematical model of an image.

Definition 3.2.1. Let R be a discrete and finite subset of $\{x \in \mathbb{R} \mid x \geq 0\}$ such that $0 \in R$, and let $n \in \mathbb{N}$. Then R^n is a **color space** with n channels. △

Definition 3.2.2. A **size** is an ordered pair (m, n), where $m, n \in \mathbb{N}$. If something is said to be of size (m, n), we will refer to m and n as its **width** and **length**, respectively. △

Definition 3.2.3. Let $m, n \in \mathbb{N}$.

1. The **discrete image space** of size (m, n) is $[0, m-1]^+ \times [0, n-1]^+$.
2. The **continuous image space** of size (m, n) is $[0, m-1] \times [0, n-1]$.
3. If $\mathbb{I}$ is a discrete image space of size (m, n), the **interpolation** of $\mathbb{I}$, written $\mathbb{I}^\star$, is the continuous image space of size (m, n).
4. If $\mathbb{I}$ is a continuous space of size (m, n), the **discretization** of $\mathbb{I}$, written $\mathbb{I}^+$, is the discrete image space of size (m, n). △

Definition 3.2.4. Let $\mathbb{I}$ be an image space, and let C be a color space.

1. An **image** over $\mathbb{I}$ is a function $I \colon \mathbb{I} \to C$.
2. I is a **discrete image** if $\mathbb{I}$ is a discrete image space.
3. I is a **continuous image** if $\mathbb{I}$ is a continuous image space.
4. If I is a discrete image over $\mathbb{I}$, a **interpolation** of I, written $I^\star$, is a continuous image over $\mathbb{I}^\star$ such that $I^\star(x, y) = I(x, y)$ for each $(x, y) \in \mathbb{I}$.
5. If I is a continuous image over $\mathbb{I}$, the **discretization** of I, written I^+, is the unique discrete image over $\mathbb{I}^+$ such that $I^+(x, y) = I(x, y)$ for each $(x, y) \in \mathbb{I}^+$. △

3.3 Image Interpolation

In Section 1.3.1, we observed that all digital images are discrete, a concept we formalized in Definition 3.2.4 (2). Given a discrete image, suppose that we want to know what the color information "should be" at a point between pixels. We introduced the notion of image interpolation in Definition 3.2.4 (4) precisely to answer such a question, though we have yet to specify how an interpolation may be obtained.

Definition 3.3.1. Let D be the set of all discrete images, and let $D^\star$ be the set of all complete images. An **interpolator** is a function $B\colon D \to D^\star$. △

In other words, an interpolator takes an image I and returns an interpolation of I. It should be noted that each image does not have a unique interpolation; correspondingly, many different interpolators exist, and not all of them are equally useful. One could, for example, define an interpolation of an image that maps all non-integral points to a constant value; such an interpolation is unlikely to be of practical use. Examples of commonly-used interpolators include (in order of increasing complexity) the nearest neighbor, bilinear, and bicubic interpolators, which approximate the color information along each row and column as a stepwise function, a piecewise linear function, and a piecewise cubic function, respectively. Let us now proceed with a derivation of a bilinear interpolation.

3.3.1 Bilinear Interpolation

Let $\mathbb{I}$ be a discrete image space of size (m, n), let I be an image over $\mathbb{I}$, let B_L be the bilinear interpolator, and let $I^\star = B_L(I)$. Let $x \in [0, m-1]$ and $y \in [0, n-1]$; let $X = \lfloor x \rfloor$ and $Y = \lfloor x \rfloor$. Then $X \in [0, m-1]^+$ and $Y \in [0, n-1]^+$. Assuming that $X \neq m-1$ and $Y \neq n-1$, we know that $I_{X,Y}$, $I_{X,Y+1}$, $I_{X+1,Y}$, and $I_{X+1,Y+1}$ are well-defined. These coordinates, along with (x, y), are pictured in Figure 3.3.1.

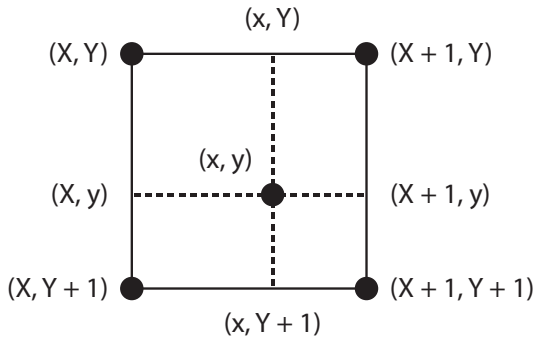


Figure 3.3.1. Bilinear interpolation.

If we assume that $I^\star$ is linear along the straight line joining (X, Y) and $(X+1, Y)$, we can write $I^\star_{x,Y}$ as a linear combination of $I_{X,Y}$ and $I_{X+1,Y}$, as follows:

$$I^\star_{x,Y} = \big((X+1) - x\big)I_{X,Y} + (x - X)I_{X+1,Y} \tag{3.3.1}$$

Similarly,

$$I^\star_{x,Y+1} = \big((X+1) - x\big)I_{X,Y+1} + (x - X)I_{X+1,Y+1} \tag{3.3.2}$$

Then, assuming that $I^\star$ is linear along the straight line joining (x, Y) and $(x, Y+1)$,

$$\begin{aligned} I^\star_{x,y} &= \big((Y+1) - y\big)I_{x,Y} + (y - Y)I_{x,Y+1} \\ &= \big((X+1) - x\big)\big((Y+1) - y\big)I_{X,Y} + (x - X)\big((Y+1) - y\big)I_{X+1,Y} \\ &\quad + \big((X+1) - x\big)(y - Y)I_{X,Y+1} + (x - X)(y - Y)I_{X+1,Y+1}. \end{aligned} \tag{3.3.3}$$

If $x = X$ and $y = Y$, we see that

$$\begin{aligned} I^\star_{x,y} &= \big((x+1) - x\big)\big((y+1) - y\big)I_{X,Y} + (x - x)\big((Y+1) - y\big)I_{X+1,Y} \\ &\quad + \big((x+1) - x\big)(y - y)I_{X,Y+1} + (x - x)(y - y)I_{X+1,Y+1} \\ &= I_{X,Y} = I_{x,y}, \end{aligned} \tag{3.3.4}$$

so the interpolation is valid according to Definition 3.2.4 (4).

The case where $X = m - 1$ or $Y = n - 1$ remains unaccounted for; Equation (3.3.3) is not well-defined in this case, since $(X+1, Y+1)$ and at least one of $(X, Y+1)$ and

$(X+1, Y)$ are not in the range of I. It turns out, however, that if $X = m-1$, then $x = X$ (since $x \leq m-1$), and if $Y = n-1$, then $y = Y$ (since $y \leq n-1$); thus, the coefficients of the undefined quantities in Equation (3.3.3) reduce to zero.

3.4 Image Saliency

In Section 2.1, we introduced the concept of a saliency map, which ascribes, to each pixel of an image, a saliency value describing its relative importance. Let us formalize this notion:

Definition 3.4.1. Let $\mathbb{I}$ be an image space, and let I be an image over $\mathbb{I}$. A **saliency map** of I is a function $S \colon \mathbb{I} \to \{x \in \mathbb{R} \mid x \geq 0\}$. △

Obviously, saliency maps are not unique; in Section 2.1, we discussed several different saliency measures that are used in practice. In the following section, we will take a closer look at the image gradient magnitude, one of the most widely-used measures of image saliency.

3.4.1 Image Gradient

Definition 3.4.2. Let $\mathbb{I}$ be a continuous image space, and let I be an image over $\mathbb{I}$. Suppose that I is differentiable everywhere except at the boundary of $\mathbb{I}$. Then, for each (x, y) in the interior of $\mathbb{I}$,

1. The **horizontal gradient** and **vertical gradient** of I at (x, y), written $\dfrac{\partial I}{\partial x}(x, y)$ and $\dfrac{\partial I}{\partial y}(x, y)$, are the partial derivatives of I with respect to its first and second parameters, respectively, evaluated at (x, y).

2. The **gradient magnitude** of I at (x, y), written $\|\nabla I\|(x, y)$, is given by
$$\|\nabla I\|(x, y) = \sqrt{\left(\frac{\partial I}{\partial x}(x, y)\right)^2 + \left(\frac{\partial I}{\partial y}(x, y)\right)^2}.$$ △

One realizes right away that this definition is of no practical use for actually computing the image gradient magnitude. The requirement that images be not only continuous but

differentiable is almost impossibly restrictive; since all digital images are discrete, we are faced with the task of finding an interpolation that is differentiable, which is no mean feat. Even if we were able to find such an interpolation, calculating its theoretical gradient magnitude at every point would still be computationally unfeasible. In practice, the image gradient magnitude is usually found through standard approximations like the Sobel operator [6], which we will describe in Section 3.4.3. Before we do that, we need to define the image convolution operation.

3.4.2 Image Convolution

Definition 3.4.3. Let $k, m, n \in \mathbb{N}$.

1. A **discrete image convolution kernel** of size k is a function $K\colon \left([-k,k]^+\right)^2 \to \mathbb{R}$.

2. Let K be a discrete image convolution kernel of size k. Then the **matrix representation** of K is the $(2k+1) \times (2k+1)$ matrix

$$\begin{bmatrix} K_{-k,-k} & K_{-k+1,-k} & \cdots & K_{0,-k} & \cdots & K_{k-1,-k} & K_{k,-k} \\ K_{-k,-k+1} & K_{-k+1,-k+1} & \cdots & K_{0,-k+1} & \cdots & K_{k-1,-k+1} & K_{k,-k+1} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ K_{-k,0} & K_{-k+1,0} & \cdots & K_{0,0} & \cdots & K_{k-1,0} & K_{k,0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ K_{-k,k-1} & K_{-k+1,k-1} & \cdots & K_{0,k-1} & \cdots & K_{k-1,k-1} & K_{k,k-1} \\ K_{-k,k} & K_{-k+1,k} & \cdots & K_{0,k} & \cdots & K_{k-1,k} & K_{k,k} \end{bmatrix}.$$

3. Let $\mathbb{I}$ be a discrete space of size (m,n), let I be an image over $\mathbb{I}$, and let K be a discrete convolution kernel of size k. Then the **convolution** of K with I, written $K \bowtie I$, is an image such that, for each $x \in [0, m-1]^+$ and $y \in [0, n-1]^+$,

$$(K \bowtie I)(x,y) = \sum_{v=-k}^{k} \sum_{w=-k}^{k} \left(K_{v,w} I_{x+v,y+w}\right). \qquad \triangle$$

This last definition is slightly problematic; for points (x,y) that are within k pixels of the boundary of I, some pairs $(x+v, y+w)$ are not in the range of I. There are several common strategies to deal with an undefined value $I_{x,y}$, including assigning it a constant

value (usually the element in the color space corresponding to black), and setting it to equal $I_{x',y'}$, where (x', y') is the point on the boundary of I that is closest to (x, y).

3.4.3 The Sobel Operator

Now, we are ready to introduce one of the standard approximations for the image gradient, namely the Sobel operator mentioned in Section 3.4.1. Let $\mathbb{I}$ be a discrete image space, let I be an image over $\mathbb{I}$, and let K_x and K_y be the discrete image convolution kernels represented by the matrices

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \tag{3.4.1}$$

respectively. Then $(K_x \bowtie I)(x, y) \approx \dfrac{\partial I}{\partial x}(x, y)$ and $(K_y \bowtie I)(x, y) \approx \dfrac{\partial I}{\partial y}(x, y)$ for each $(x, y) \in \mathbb{I}$ [6].

3.5 Image Transformation

In Section 2.3, we said that a warp is the result of applying a transformation map to an image. We hereby put forth the following definitions:

Definition 3.5.1. Let $\mathbb{I}$ and $\mathbb{J}$ be image spaces, let I be an image over $\mathbb{I}$, and let S be a saliency map of I.

1. An **image transformation map** is a function $T \colon \mathbb{I} \longrightarrow \mathbb{J}$. If we **apply** T to I, the resulting image is $(I \circ T)$.

2. Let T be an image transformation map. We say that T **respects** S if high-saliency regions of $\mathbb{I}$ experience less distortion under T than low-saliency regions. Note that, despite our attempts at formalism, this is one definition that is impossible to express formally, because the success or failure of a transformation map in retargeting an image is a highly subjective element that cannot be quantified.

3. Let $\bar{S}$ be the set of all saliency maps, and let $\bar{\mathbb{I}}$ be the set of all image spaces. An **image retargeting algorithm** is a function $A\colon \bar{S} \to \bar{\mathbb{I}}$ such that $A(S)$ is an image transformation map that respects S.

There are two general ways to specify an image transformation map. The first is a *forward map*, in which one specifies the destination points to which each pixel in the source image should be mapped; the second is an *inverse map*, in which one specifies, for each pixel in the target image, the corresponding point in the source image. Due to the way we shall set up our algorithm described in Chapter 4, we shall employ a forward map.

3.5.1 Forward Mapping

Definition 3.5.2. Let $\mathbb{I}$ and $\mathbb{J}$ be image spaces, and let $T\colon \mathbb{I} \to \mathbb{J}$ be an image transformation map. Then T is a **forward map** if $\mathbb{I}$ is discrete and $\mathbb{J}$ is continuous. △

Forward mapping has its share of problems. The two chief concerns are that pixels in the source image may be mapped to non-integral points in the target image, and that there may be regions in the target image to which no pixel in the source image is mapped. I have no doubt that forward mapping is a well-studied subject, and that superior solutions than the ones I offer here can be found in the literature, but I chose to propose (and implement) my own workarounds to these two issues, namely *bilinear extrapolation* and *Manhattan-distance interpolation*, as described in the next two sections.

3.5.2 Bilinear Extrapolation

First, let us address the former concern, which is that pixels in the source image may be mapped to points between pixels in the target image. In order to distribute the color information of the source pixel to the surrounding pixels, we use an extrapolation process that mirrors bilinear interpolation, as described in Section 3.3.1, except in reverse. That is, we assign color values to the four nearest pixels by assuming that T is linear in the region

bounded by these pixels, as follows:

$$(I \circ T)\big(\lfloor X_{x,y} \rfloor, \lfloor Y_{x,y} \rfloor\big) \approx \big(\lceil X_{x,y} \rceil - X_{x,y}\big)\big(\lceil Y_{x,y} \rceil - Y_{x,y}\big) I(x,y)$$

$$(I \circ T)\big(\lfloor X_{x,y} \rfloor, \lceil Y_{x,y} \rceil\big) \approx \big(\lceil X_{x,y} \rceil - X_{x,y}\big)\big(Y_{x,y} - \lfloor Y_{x,y} \rfloor\big) I(x,y)$$

$$(I \circ T)\big(\lceil X_{x,y} \rceil, \lfloor Y_{x,y} \rfloor\big) \approx \big(X_{x,y} - \lfloor X_{x,y} \rfloor\big)\big(\lceil Y_{x,y} \rceil - Y_{x,y}\big) I(x,y)$$

$$(I \circ T)\big(\lceil X_{x,y} \rceil, \lceil Y_{x,y} \rceil\big) \approx \big(X_{x,y} - \lfloor X_{x,y} \rfloor\big)\big(Y_{x,y} - \lfloor Y_{x,y} \rfloor\big) I(x,y)$$

3.5.3 Manhattan-Distance Interpolation

After bilinear extrapolation has been completed, as described above, they may be pixels in the resulting image that have received no color information. We will assign color values to these pixels by a process based on the Manhattan distance metric.

Definition 3.5.3. Let $\mathbb{I}$ be a discrete image space, and let $(x,y),(x',y') \in \mathbb{I}$. Then the **Manhattan distance** between (x,y) and (x',y') is $|x-x'| + |y-y'|$. △

Let (x,y) be an "empty" pixel in the target image, i.e. one that has received no color information. Observe in Figure 3.5.1 that the pixels forming a diamond centered at (x,y) all have the same Manhattan distance from (x,y); a diamond is to Manhattan distance as a circle is to Euclidean distance. Hence, it seems reasonable to expect each point in such a diamond to contribute color information equally to (x,y). Our algorithm thus proceeds as follows: Identifying the diamond with the smallest "radius" centered at (x,y) such that at least one pixel forming the diamond has received some color information, we assign to (x,y) the mean of these pixels' color values.

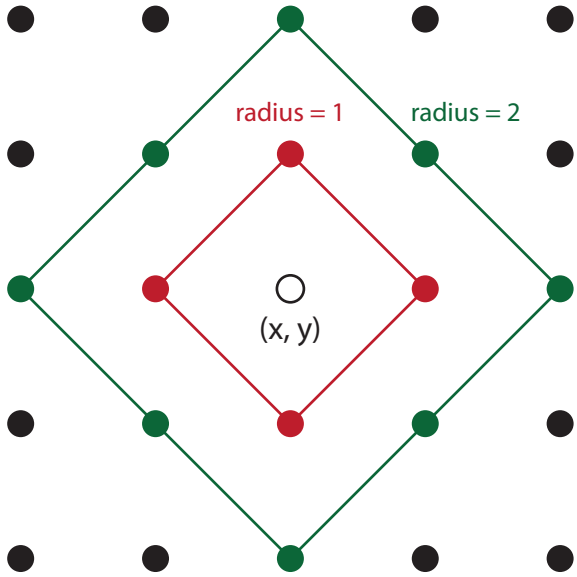


Figure 3.5.1. A diamond is to Manhattan distance as a circle is to Euclidean distance.

4
A First Attempt

4.1 Towards a New Retargeting Algorithm

In this chapter, I will develop a new retargeting algorithm that is based primarily on that of Wolf, Guttmann, and Cohen-Or described in Section 2.3.1 (henceforth referred to as Wolf's algorithm), also drawing on ideas from the algorithm of Wang, Tai, Sorkine, and Lee described in Section 2.3.2 (henceforth referred to as Wang's algorithm). I chose to model my algorithm after Wolf's algorithm primarily because I was attracted to the idea of using the least-squares method of finding an approximate solution to a system of overdetermined equations; the widespread use of linear algebra in computational applied mathematics affords me a wealth of highly-optimized numerical linear algebra libraries to draw upon.

As we saw in Section 2.3.2, Wang's algorithm consistently produces better results across a range of images because it retargets images globally, propagating distortion along both dimensions at once. Our goal, therefore, will be to formulate a new set of constraints that support global retargeting. We will start by examining each of Wolf's constraints, which we will express mathematically using the notation we developed in Chapter 3. Then, by discarding the constraints that are not in alignment with our goal, and adding new ones,

we will arrive at a new set of constraints that will hopefully give us the results we desire. For now, we will concern ourselves with the retargeting of still images; we thus ignore Wolf's time-smoothness constraints.

4.2 Constraints

Let $\mathbb{I}$ be a discrete image space of size (m, n), and let $\mathbb{J}$ be a continuous image space of size (m', n'), for some $m, n, m', n' \in \mathbb{N}$. Let I be an image over $\mathbb{I}$, and let S be a saliency map of I. Finally, let $T\colon \mathbb{I} \longrightarrow \mathbb{J}$ be a transformation map that respects S, and let $X\colon \mathbb{I} \longrightarrow [0, m' - 1]$ and $Y\colon \mathbb{I} \longrightarrow [0, n' - 1]$ be such that $T_{x,y} = (X_{x,y}, Y_{x,y})$.

4.2.1 Boundary substitutions

Wolf's boundary-substitution constraints specify that the leftmost pixels should stay fixed, while the rightmost pixels should conform to the target width. That is, for each $y \in [0, n-1]^+$,

$$X_{0,y} = 0 \tag{4.2.1}$$

and

$$X_{m-1,y} = m' - 1. \tag{4.2.2}$$

Since we want our retargeted image to fit the target dimensions, we will keep these boundary subsitutions. Furthermore, since we want to retarget both the x- and y-coordinates at once, we will add the corresponding constraint that, for each $x \in [0, m-1]^+$,

$$Y_{x,0} = 0 \tag{4.2.3}$$

and

$$Y_{x,n-1} = n' - 1. \tag{4.2.4}$$

4.2.2 Importance modeling

Wolf's so-called importance-modeling constraints specify that salient pairs of horizontally adjacent pixels should remain approximately one pixel apart, while less salient pairs can be allowed to drift closer together or further apart. Put differently, given any pair of horizontally adjacent pixels, the ideal horizontal distance between them is one pixel, which we can express mathematically as follows:

$$X_{x+1,y} - X_{x,y} = 1, \text{ where } x \in [0, m-2]^+ \text{ and } y \in [0, n-1]^+. \tag{4.2.5}$$

Recall that our goal is to construct an overdetermined system of linear equations, to which we will find a least-squares solution, which is a solution that minimizes the sum of squares of the residuals (i.e. the difference between the left- and right-hand sides of each equation). Suppose we were to multiply both sides of Equation (4.2.5) by some non-negative real number W. Its residual would then be $W(X_{x+1,y} - X_{x,y} - 1)$. Clearly, the larger W is, the larger is the residual; consequently, the larger is the residual's contribution to the sum to be minimized, and the larger is the equation's effect on the whole system. Thus, W acts as a weighting factor for the equation.

The obvious strategy, therefore, is to weight each instance of Equation (4.2.5) by a value that represents the saliency of both pixels referenced by the equation. Reasonable choices for such a value include the sum of the two saliency values, and the product of the two saliency values. Wolf's solution, on the other hand, is simply to include each equation twice, so that for each $x \in [0, m-2]^+$ and $y \in [0, n-1]^+$, the following constraints apply:

$$S_{x,y}(X_{x+1,y} - X_{x,y}) = S_{x,y} \tag{4.2.6}$$

and

$$S_{x+1,y}(X_{x+1,y} - X_{x,y}) = S_{x+1,y}. \tag{4.2.7}$$

Since we want to allow parts of the image to be scaled upwards or downwards, we are unable to specify, ahead of time, the ideal distance between any pair of points. We will, therefore, discard these importance-modeling constraints in favor of new ones that we will formulate in Section 4.2.4.

4.2.3 Spatial smoothness

Wolf's constraints specify that each column of pixels should have approximately the same x-coordinate. In other words, each pair of vertically adjacent pixels should stay vertically adjacent, as illustrated in Figure 4.2.1. That is, for each $x \in [0, m-1]^+$ and $y \in [0, n-2]^+$,

$$X_{x,y} - X_{x,y+1} = 0. \tag{4.2.8}$$

These constraints are unweighted in Wolf's algorithm, which considers column bending to be equally undesirable in all columns. In order to propagate distortion along both spatial dimensions, however, it seems reasonable for us to allow more column bending in less salient regions of an image, by weighting Equation 4.2.8 in the manner described in Section 4.2.2:

$$(S_{x,y} \diamond S_{x,y+1})(X_{x,y} - X_{x,y+1}) = 0, \tag{4.2.9}$$

where $\diamond$ is some binary operation that combines two saliency values in some fashion. As mentioned in Section 4.2.2, the sum and product operations both seem like reasonable options for $\diamond$. Similarly, we will constraint each pair of horizontally adjacent pixels to stay horizontally adjacent, as illustrated in Figure . That is, for each $x \in [0, m-2]^+$ and $y \in [0, n-1]^+$,

$$(S_{x,y} \diamond S_{x+1,y})(Y_{x,y} - Y_{x+1,y}) = 0. \tag{4.2.10}$$

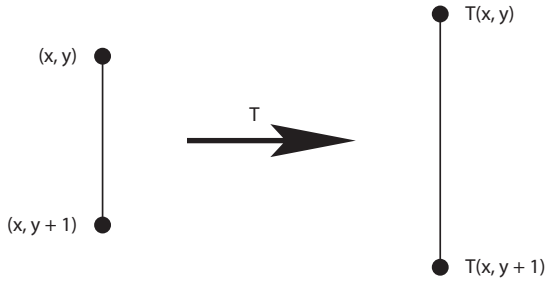


Figure 4.2.1. Visual motivation for Equations 4.2.8 and 4.2.9.

Figure 4.2.2. Visual motivation for Equation 4.2.10.

4.2.4 A New Set of Constraints

Before we add any further constraints, let us take stock of the constraints that we have so far. We have the following boundary substitutions:

$$X_{0,y} = 0 \qquad \text{and} \qquad X_{m-1,y} = m' - 1 \qquad \text{for each } y \in [0, n-1]^+, \tag{4.2.11}$$

$$Y_{x,0} = 0 \qquad \text{and} \qquad Y_{x,n-1} = n' - 1 \qquad \text{for each } x \in [0, m-1]^+, \tag{4.2.12}$$

and the following constraints for pairs of vertically and horizontally adjacent pixels, respectively:

$$(S_{x,y} \diamond S_{x,y+1})(X_{x,y} - X_{x,y+1}) = 0 \quad \text{for each } x \in [0, m-1]^+, \quad y \in [0, n-2]^+, \tag{4.2.13}$$

and

$$(S_{x,y} \diamond S_{x+1,y})(Y_{x,y} - Y_{x+1,y}) = 0 \quad \text{for each } x \in [0, m-2]^+, \quad y \in [0, n-1]^+. \tag{4.2.14}$$

These constraints are obviously insufficient for our purposes, as they can be satisfied by any transformation map that maps each row of pixels to a row of pixels, and each column of pixels to a column of pixels, regardless of the distance between each row or column. As

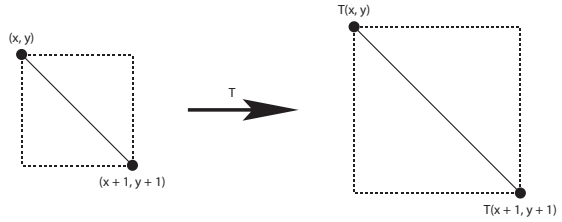


Figure 4.2.3. Visual motivation for Equations 4.2.15 and 4.2.16.

we observed in Section 4.2.2, we are unable to specify, ahead of time, the ideal distance between any pair of points.

Consider a pair of diagonally adjacent points (x, y) and $(x + 1, y + 1)$, pictured in Figure 4.2.3. Ideally, after retargeting, the horizontal distance between these pixels should equal the vertical distance:

$$X_{x+1,y+1} - X_{x,y} = Y_{x+1,y+1} - Y_{x,y}. \tag{4.2.15}$$

As before, we weight this equation by the saliency values of the two pixels, to obtain the following constraint:

$$(S_{x,y} \diamond S_{x+1,y+1})(X_{x,y} - Y_{x,y} - X_{x+1,y+1} + Y_{x+1,y+1}) = 0$$
$$\text{for each } x \in [0, m-2]^+ \text{ and } y \in [0, n-2]^+. \tag{4.2.16}$$

Using the same argument on a pair of diagonally adjacent points $(x, y + 1)$ and $(x + 1, y)$, pictured in Figure 4.2.4, we derive the following constraint:

$$(S_{x,y+1} \diamond S_{x+1,y})(X_{x,y+1} + Y_{x,y+1} - X_{x+1,y} - Y_{x+1,y}) = 0$$
$$\text{for each } x \in [0, m-2]^+ \text{ and } y \in [0, n-2]^+. \tag{4.2.17}$$

4.3 A Numerical Example

Let $m = n' = 3$ and $n = m' = 2$. In other words, the source image is 3 pixels wide and 2 pixels tall, and the target dimensions are 2×3. Applying the boundary substitutions in

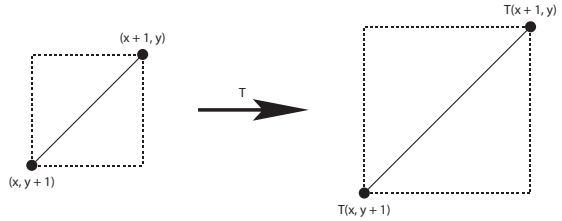


Figure 4.2.4. Visual motivation for Equation 4.2.17.

Equations 4.2.11 and 4.2.12, we have:

$$X_{0,0} = X_{0,1} = 0, \tag{4.3.1}$$

$$X_{2,0} = X_{2,1} = 1, \tag{4.3.2}$$

$$Y_{0,0} = Y_{1,0} = Y_{2,0} = 0, \tag{4.3.3}$$

and

$$Y_{0,1} = Y_{1,1} = Y_{2,1} = 2. \tag{4.3.4}$$

Thus, we are left with only two unknowns, which are $X_{1,0}$ and $X_{1,1}$. Then, applying the constraints described by Equation 4.2.13, we have

$$(S_{0,0} \diamond S_{0,1})(X_{0,0} - X_{0,1}) = (S_{0,0} \diamond S_{0,1})(0 - 0) = 0, \tag{4.3.5}$$

$$(S_{1,0} \diamond S_{1,1})(X_{1,0} - X_{1,1}) = 0, \tag{4.3.6}$$

and

$$(S_{2,0} \diamond S_{2,1})(X_{2,0} - X_{2,1}) = (S_{2,0} \diamond S_{2,1})(1 - 1) = 0. \tag{4.3.7}$$

Similarly, applying the constraints described by Equation 4.2.14, we have

$$(S_{0,0} \diamond S_{1,0})(Y_{0,0} - Y_{1,0}) = (S_{0,0} \diamond S_{1,0})(0 - 0) = 0, \tag{4.3.8}$$

$$(S_{1,0} \diamond S_{2,0})(Y_{1,0} - Y_{2,0}) = (S_{1,0} \diamond S_{2,0})(0 - 0) = 0, \tag{4.3.9}$$

$$(S_{0,1} \diamond S_{1,1})(Y_{0,1} - Y_{1,1}) = (S_{0,1} \diamond S_{1,1})(2 - 2) = 0, \tag{4.3.10}$$

and

$$(S_{1,1} \diamond S_{2,1})(Y_{1,1} - Y_{2,1}) = (S_{1,1} \diamond S_{2,1})(2 - 2) = 0. \tag{4.3.11}$$

Applying the constraints described by Equation 4.2.16, we have

$$(S_{0,0} \diamond S_{1,1})(X_{0,0} - Y_{0,0} - X_{1,1} + Y_{1,1}) = (S_{0,0} \diamond S_{1,1})(0 - 0 - X_{1,1} + 2) = 0 \tag{4.3.12}$$

and

$$(S_{1,0} \diamond S_{2,1})(X_{1,0} - Y_{1,0} - X_{2,1} + Y_{2,1}) = (S_{1,0} \diamond S_{2,1})(X_{1,0} - 0 - 1 + 2) = 0. \tag{4.3.13}$$

Finally, applying the constraints described by Equation 4.2.17 we have

$$(S_{0,1} \diamond S_{1,0})(X_{0,1} + Y_{0,1} - X_{1,0} - Y_{1,0}) = (S_{0,1} \diamond S_{1,0})(0 + 2 - X_{1,0} - 0) = 0 \tag{4.3.14}$$

and

$$(S_{1,1} \diamond S_{2,0})(X_{1,1} + Y_{1,1} - X_{2,0} - Y_{2,0}) = (S_{1,1} \diamond S_{2,0})(X_{1,1} + 2 - 1 - 0) = 0. \tag{4.3.15}$$

Discarding Equation (4.3.5) and Equations (4.3.7) through (4.3.11), all of which reduce to $0 = 0$, we are left with the following system of linear equations:

$$(S_{1,0} \diamond S_{1,1})X_{1,0} - (S_{1,0} \diamond S_{1,1})X_{1,1} = 0, \tag{4.3.16}$$

$$(S_{0,0} \diamond S_{1,1})X_{1,1} = 2(S_{0,0} \diamond S_{1,1}), \tag{4.3.17}$$

$$(S_{1,0} \diamond S_{2,1})X_{1,0} = -(S_{1,0} \diamond S_{2,1}), \tag{4.3.18}$$

$$(S_{0,1} \diamond S_{1,0})X_{1,0} = 2(S_{0,1} \diamond S_{1,0}), \tag{4.3.19}$$

and

$$(S_{1,1} \diamond S_{2,0})X_{1,1} = -(S_{1,1} \diamond S_{2,0}). \tag{4.3.20}$$

Let

$$\mathbf{A} = \begin{bmatrix} S_{1,0} \diamond S_{1,1} & -(S_{1,0} \diamond S_{1,1}) \\ 0 & S_{0,0} \diamond S_{1,1} \\ S_{1,0} \diamond S_{2,1} & 0 \\ S_{0,1} \diamond S_{1,0} & 0 \\ 0 & S_{1,1} \diamond S_{2,0} \end{bmatrix}, \ \mathbf{x} = \begin{bmatrix} X_{1,0} \\ X_{1,1} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} 0 \\ 2(S_{0,0} \diamond S_{1,1}) \\ -(S_{1,0} \diamond S_{2,1}) \\ 2(S_{0,1} \diamond S_{1,0}) \\ -(S_{1,1} \diamond S_{2,0}) \end{bmatrix}. \quad (4.3.21)$$

Then Equations (4.3.16) through (4.3.20) can be more concisely expressed as

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \quad (4.3.22)$$

Let $\mathbf{A}^{\mathrm{T}}$ be the transpose of $\mathbf{A}$. We can pre-multiply both sides of Equation (4.3.22) by $\mathbf{A}^{\mathrm{T}}$ to obtain

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{A}^{\mathrm{T}}\mathbf{b}. \quad (4.3.23)$$

Now, $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is a square matrix. If $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is invertible, then

$$\mathbf{x} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{\mathrm{T}}\mathbf{b}\right) \quad (4.3.24)$$

is the least-squares solution to Equation (4.3.22); a proof of this fact can be found in any standard linear algebra textbook.

4.3.1 Uniform Saliency Map

Suppose that $S_{x,y} = 1$ for each $(x, y) \in \mathbb{I}$, and let $\diamond$ be the product operation. Then

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 2 \\ -1 \\ 2 \\ -1 \end{bmatrix}. \quad (4.3.25)$$

By computing

$$\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1} = \left(\begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}^{-1} = \frac{1}{8} \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \quad (4.3.26)$$

and

$$\mathbf{A}^{\mathrm{T}}\mathbf{b} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ -1 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{4.3.27}$$

we have

$$\mathbf{x} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{\mathrm{T}}\mathbf{b}\right) = \frac{1}{8}\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}. \tag{4.3.28}$$

Hence $X_{1,0} = X_{1,1} = 0.5$; as we might expect, when the saliency map is uniform, the source image is scaled uniformly in each dimension. We obtain the same result if we let $\diamond$ be the sum instead of the product.

4.3.2 Non-Uniform Saliency Map with Sum of Saliency Constraints

Now, let $\diamond$ be the sum operation, and suppose that

$$\mathbf{S} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{4.3.29}$$

is the matrix representation of S, which is to say that $S_{x,y}$ is the value located at the xth column and yth row of $\mathbf{S}$. Then

$$\mathbf{A} = \begin{bmatrix} 1+1 & -(1+1) \\ 0 & 2+1 \\ 1+2 & 0 \\ 1+1 & 0 \\ 0 & 1+1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 0 & 3 \\ 3 & 0 \\ 2 & 0 \\ 0 & 2 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 2(2+1) \\ -(1+2) \\ 2(1+1) \\ -(1+1) \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ -3 \\ 4 \\ -2 \end{bmatrix}. \tag{4.3.30}$$

By computing

$$\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1} = \left(\begin{bmatrix} 2 & 0 & 3 & 2 & 0 \\ -2 & 3 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 2 & -2 \\ 0 & 3 \\ 3 & 0 \\ 2 & 0 \\ 0 & 2 \end{bmatrix} \right)^{-1} = \begin{bmatrix} 17 & -4 \\ -4 & 17 \end{bmatrix}^{-1} = \frac{1}{273}\begin{bmatrix} 17 & 4 \\ 4 & 17 \end{bmatrix} \tag{4.3.31}$$

and

$$\mathbf{A}^{\mathrm{T}}\mathbf{b} = \begin{bmatrix} 2 & 0 & 3 & 2 & 0 \\ -2 & 3 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 6 \\ -3 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} -1 \\ 14 \end{bmatrix}, \tag{4.3.32}$$

we have

$$\mathbf{x} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{\mathrm{T}}\mathbf{b}\right) = \frac{1}{273}\begin{bmatrix}17 & 4\\ 4 & 17\end{bmatrix}\begin{bmatrix}-1\\ 14\end{bmatrix} = \begin{bmatrix}1/7\\ 6/7\end{bmatrix}. \tag{4.3.33}$$

Hence $X_{1,0} = \frac{1}{7}$ and $X_{1,1} = \frac{6}{7}$.

4.3.3 Non-Uniform Saliency Map with Product of Saliency Constraints

With the same saliency map S, let us see what happens when $\diamond$ is the product operation. In this case,

$$\mathbf{A} = \begin{bmatrix}1\cdot 1 & -(1\cdot 1)\\ 0 & 2\cdot 1\\ 1\cdot 2 & 0\\ 1\cdot 1 & 0\\ 0 & 1\cdot 1\end{bmatrix} = \begin{bmatrix}1 & -1\\ 0 & 2\\ 2 & 0\\ 1 & 0\\ 0 & 1\end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix}0\\ 2(2\cdot 1)\\ -(1\cdot 2)\\ 2(1\cdot 1)\\ -(1\cdot 1)\end{bmatrix} = \begin{bmatrix}0\\ 4\\ -2\\ 2\\ -1\end{bmatrix}. \tag{4.3.34}$$

By computing

$$\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1} = \left(\begin{bmatrix}1 & 0 & 2 & 1 & 0\\ -1 & 2 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & -1\\ 0 & 2\\ 2 & 0\\ 1 & 0\\ 0 & 1\end{bmatrix}\right)^{-1} = \begin{bmatrix}6 & -1\\ -1 & 6\end{bmatrix}^{-1} = \frac{1}{35}\begin{bmatrix}6 & 1\\ 1 & 6\end{bmatrix} \tag{4.3.35}$$

and

$$\mathbf{A}^{\mathrm{T}}\mathbf{b} = \begin{bmatrix}1 & 0 & 2 & 1 & 0\\ -1 & 2 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\ 4\\ -2\\ 2\\ -1\end{bmatrix} = \begin{bmatrix}-2\\ 7\end{bmatrix}, \tag{4.3.36}$$

we have

$$\mathbf{x} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{\mathrm{T}}\mathbf{b}\right) = \frac{1}{35}\begin{bmatrix}6 & 1\\ 1 & 6\end{bmatrix}\begin{bmatrix}-2\\ 7\end{bmatrix} = \begin{bmatrix}-1/7\\ 8/7\end{bmatrix}. \tag{4.3.37}$$

Hence $X_{1,0} = -\frac{1}{7}$ and $X_{1,1} = \frac{8}{7}$. We observe that $T_{1,0}$ and $T_{1,1}$ have strayed beyond the boundaries of the target dimensions. Unfortunately, this can happen sometimes using the least-squares method, and it results in cropping (since pixels in the source image that are retargeted outside the target dimensions would simply not be visible in the resulting image). Furthermore, this example illustrates another potential pitfall of an algorithm based on least-squares—the retargeting process did not preserve the order of pixels in the

source image. This is known as a foldover problem, and it can happen in the middle of an image (not just at the boundaries), manifesting itself in my results as cobweb-like artifacts.

4.4 Implementation

I implemented two versions of my algorithm in C++, one using the sum of saliency constraints and the other using the product of saliency constraints. I used Intel's open-source OpenCV library for computer vision and image processing [6], and Timothy Davis' SuiteSparseQR library to solve the least-squares systems [7]. For the saliency map, I used OpenCV's implementation of the *Scharr* operator, which is a convolution-type approximation to the image gradient, similar to but (according to OpenCV's documentation) more accurate than the Sobel operator described in Section 3.4.3. After performing the forward mapping, using the reverse bilinear extrapolation described in Section 3.5.1, the algorithm optionally performs the Manhattan-distance interpolation described in Section 3.5.3.

4.5 Results

Let us see how this algorithm fares with an image of the *Megazord* from the 1993 children's television series *Mighty Morphin' Power Rangers* (which used footage from the Japanese series *Kyōryū Sentai Zyuranger*). The image, which is 320 pixels wide and 240 pixels tall, is shown on the left of Figure 4.5.1; on its right is its saliency map generated by the algorithm. Figure 4.5.2 shows the result of our algorithm using both the sum of saliency and product of saliency constraints, in comparison with the result of uniform scaling in one direction using Adobe Photoshop's bilinear interpolation. We see that the product of saliency constraints produce better output than the sum of saliency constraints in this case. This is not always the case, however; the sum of saliency constraints do better with the example image from Figure 1.2.1, with the corresponding retargeting results shown

Figure 4.5.1. 320×240 *Megazord* image (© Saban/Toei), and its saliency map.

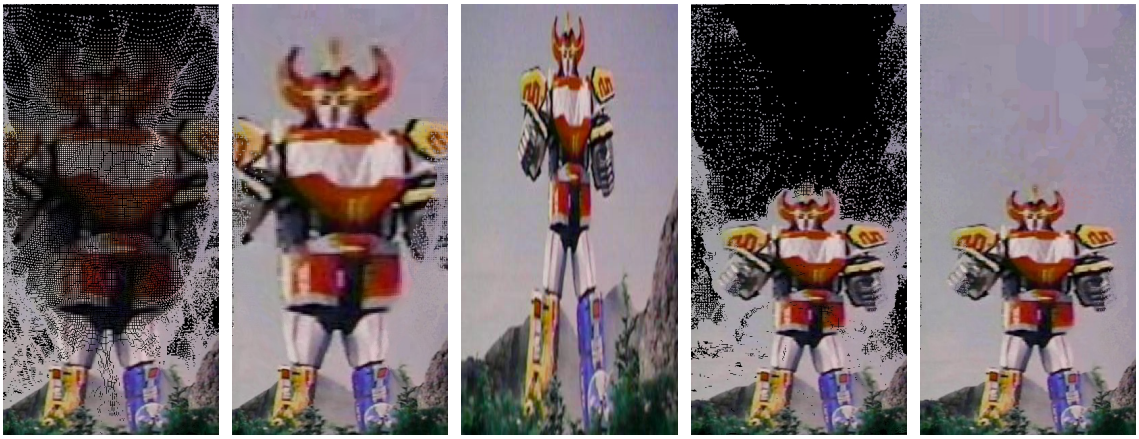


Figure 4.5.2. *Megazord* image retargeted to 320×640 using *(from left to right)*
1. sum of saliency constraints without interpolation,
2. sum of saliency constraints with interpolation,
3. bilinear interpolation,
4. product of saliency constraints without interpolation, and
5. product of saliency constraints with interpolation.

in Figure 4.5.3. Both results, however, show noticeable foldover artifacts, as described in Section 4.3.3. Finally, we see that both sets of constraints fail miserably at retargeting the *Megazord* image to 640×240, with a significant portion of the source image cropped away in each case. However, we observe that the highest-saliency region has been kept in the retargeted image, and in situations where there is no better alternative, it could be argued that cropping is the correct strategy to employ. I should remind us here that, due to the subjective nature of these results, there is no objective measure of how well the algorithm performed—one has simply to use his or her own judgment to decide on the quality of the results.

The most significant drawback of this algorithm (or at least my implementation of it) is its speed. On an HP Pavilion dv6t laptop running on an Intel Core i7-720QM quad-core processor (1.6 GHz) with 4 GB of RAM, the algorithm took approximately four minutes to retarget the *Megazord* image of size 320×240. That is clearly unacceptable performance for an algorithm that, one hopes, will eventually be adaptable for video retargeting. Ignoring the fact that the additional constraints introduced for video retargeting would add to the runtime, for a video clip that plays at approximately 24 frames per second (the standard for film), it would take approximately $4 \times 24 = 96$ minutes to retarget a single second of the video clip, and 96 hours to retarget a single minute of it.

Finding the least-squares solution is clearly the bottleneck here; we are dealing with an immense (though very sparse) coefficient matrix with approximately twice as many columns as the number of pixels in the source image (in this case $320 \times 240 = 76,800$, resulting in approximately $2 \times 76,800 = 153,600$ columns) and approximately four times as many rows as the number of pixels (i.e. approximately $4 \times 76,800 = 307,200$ rows). Hence the number of entries in the coefficient matrix is approximately $153,600 \times 307,200 = 47,185,920,000$—and this is a relatively small image! The resolution of a DVD is 720×480. One wonders

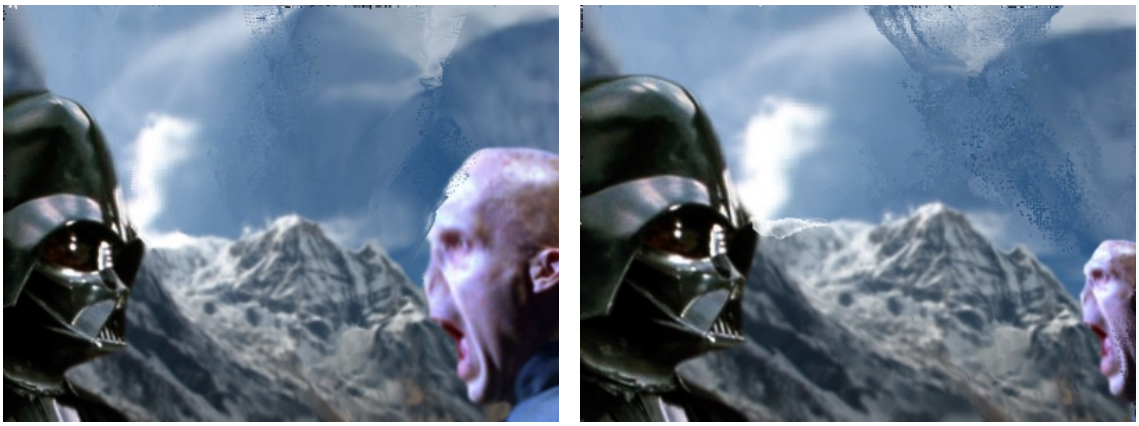


Figure 4.5.3. The image from Figure 1.2.1, retargeted to 640×240 using sum *(left)* and product *(right)* of saliency constraints, respectively.

Figure 4.5.4. *Megazord* image retargeted to 640×240 using sum of saliency constraints, without *(left)* and with *(right)* interpolation, respectively.

Figure 4.5.5. *Megazord* image retargeted to 640×240 using product of saliency constraints, without *(left)* and with *(right)* interpolation, respectively.

what kinds of optimizations Wolf and her colleagues were able to perform so as to achieve real-time performance with her algorithm.

4.6 Altering Boundary Constraints

Up to this point, we have only considered cases where the source and target images are both rectangular. There are, however, many potential applications for a more general retargeting algorithm that accepts and returns arbitrarily-shaped images. For example, it is not uncommon for people to display photographs in oval-shaped frames; using a retargeting algorithm to fit all salient content within the oval boundary is a more desirable solution than trimming the corners. The algorithm presented in this chapter can be easily adapted for this purpose simply by altering the boundary conditions so that each pixel on the boundary of the source image is mapped to a pixel on the boundary of the target image. Figures 4.6.1 and 4.6.2 demonstrate the result of retargeting the *Megazord* image (from Figure 4.5.1) to fit a diamond-shaped boundary using the sum of saliency and product of saliency constraints, respectively. The images on the left are the results shown without interpolation; the interpolated versions on the right have been manually overlaid with a black boundary (since my algorithm performed interpolation on whole image, including the portions outside the boundary of the diamond).

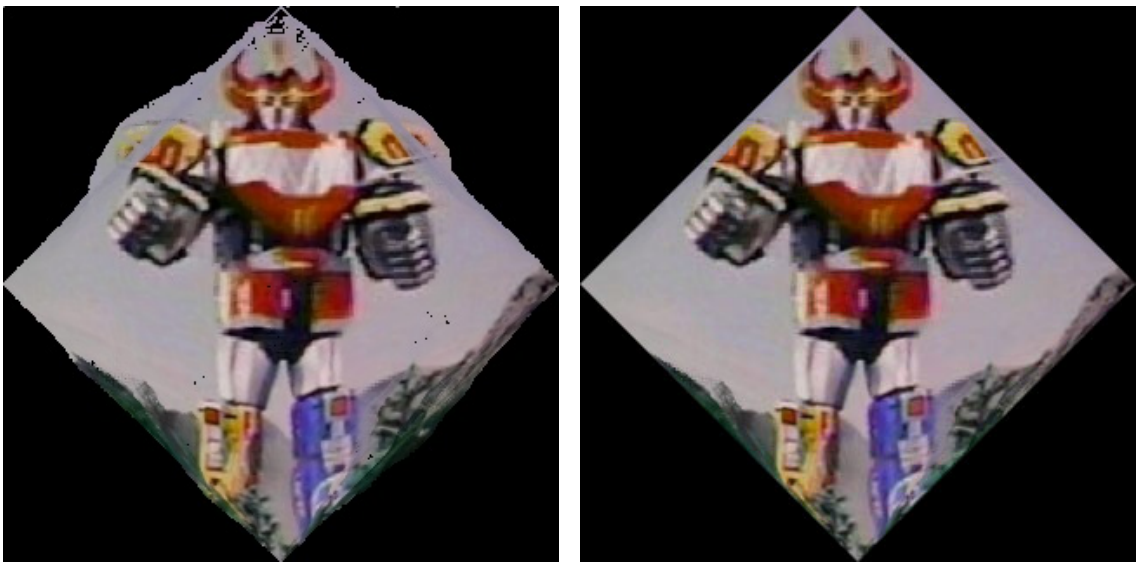


Figure 4.6.1. *Megazord* image, retargeted to a diamond inscribed within a 320×320 square, using sum of saliency constraints.

Figure 4.6.2. *Megazord* image, retargeted to a diamond inscribed within a 320×320 square, using product of saliency constraints.

5
A Second Attempt

5.1 The Need for Speed

Since speed was an issue with my previous approach, I attempted to come up with a faster retargeting algorithm. An obvious solution is to work at a coarser level than that of the pixel. One could follow in Wang's footsteps [4] and partition the image into a mesh of quads, applying the previous algorithm to the vertices of the mesh instead of individual pixels, and subsequently interpolating within each quad to complete the image retargeting map. Instead, I chose to investigate an alternative route, based on the divide-and-conquer paradigm, which has yielded efficient algorithms for many different problems. For example, the most efficient sorting algorithms—mergesort and quicksort—are based on the divide-and-conquer paradigm.

5.2 Derivation

Consider a rectangular region of the source image, bounded by the lines $x = u_1$, $x = u_3$, $y = v_1$, and $y = v_3$. Suppose that the corresponding region of the retargeted image is a rectangle bounded by the lines $x = x_1$, $x = x_3$, $y = y_1$, and $y = y_3$. Let us select an

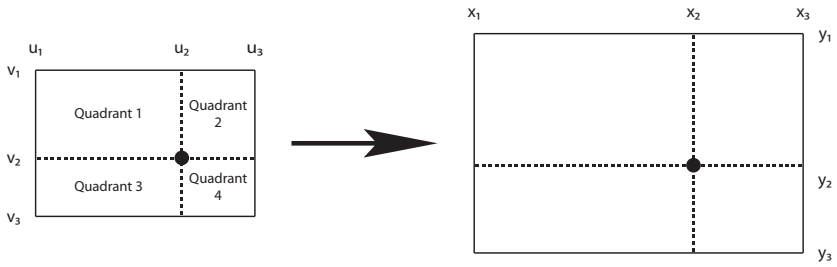


Figure 5.2.1. A rectangle partitioned into four quadrants.

arbitrary point (u_2, v_2) in the source image such that $u_1 < u_2 < u_3$ and $v_1 < v_2 < v_3$, and let (x_2, y_2) be the corresponding point in the retargeted image. This partitions the source and retargeted image into four quadrants, as shown in Figure 5.2.1.

Let R_1, R_2, R_3, and R_4 be the aspect ratios of the top-left, top-right, bottom-left, and bottom-right quadrants of the source image, respectively, and let S_1, S_2, S_3, and S_4 be the sum of saliency values of the pixels in each of those quadrants. Formally,

$$R_1 = \frac{u_2 - u_1}{v_2 - v_1}, \quad R_2 = \frac{u_3 - u_2}{v_2 - v_1}, \quad R_3 = \frac{u_2 - u_1}{v_3 - v_2}, \quad \text{and} \quad R_4 = \frac{u_3 - u_2}{v_3 - v_2}, \tag{5.2.1}$$

and

$$S_1 = \sum_{x=u_1}^{u_2} \sum_{y=v_1}^{v_2} s(x, y), \quad S_2 = \sum_{x=u_2}^{u_3} \sum_{y=v_1}^{v_2} s(x, y), \quad S_3 = \sum_{x=u_1}^{u_2} \sum_{y=v_2}^{v_3} s(x, y), \tag{5.2.2}$$

$$\text{and} \quad S_4 = \sum_{x=u_2}^{u_3} \sum_{y=v_2}^{v_3} s(x, y). \tag{5.2.3}$$

Ideally, we would like the aspect ratio of each quad to remain the same, and thus we formulate the following equations:

$$\frac{x_2 - x_1}{y_2 - y_1} = R_1, \quad \frac{x_3 - x_2}{y_2 - y_1} = R_2, \quad \frac{x_2 - x_1}{y_3 - y_2} = R_3, \quad \text{and} \quad \frac{x_3 - x_2}{y_3 - y_2} = R_4. \tag{5.2.4}$$

As we did with our previous algorithm, we weight each equation by the appropriate sum of saliency values:

$$S_1\left(\frac{x_2-x_1}{y_2-y_1}\right)=R_1S_1 \quad \text{or, equivalently,} \quad S_1x_2-R_1S_1y_2=S_1x_1-R_1S_1y_1$$
$$S_2\left(\frac{x_3-x_2}{y_2-y_1}\right)=R_2S_2 \quad \text{or, equivalently,} \quad S_2x_2+R_2S_2y_2=S_2x_3+R_2S_2y_1$$
$$S_3\left(\frac{x_2-x_1}{y_3-y_2}\right)=R_3S_3 \quad \text{or, equivalently,} \quad S_3x_2+R_3S_3y_2=S_3x_1+R_3S_3y_3$$

and

$$S_4\left(\frac{x_3-x_2}{y_3-y_2}\right)=R_4S_4 \quad \text{or, equivalently,} \quad S_4x_2-R_4S_4y_2=S_4x_1-R_4S_4y_1$$

Once again, we have an over-constrained system of linear equations, which we will attempt to solve in a least-squares manner. Let

$$\mathbf{A}=\begin{bmatrix}S_1 & -R_1S_1\\ S_2 & R_2S_2\\ S_3 & R_3S_3\\ S_4 & -R_4S_4\end{bmatrix}, \quad \mathbf{x}=\begin{bmatrix}x_2\\ y_2\end{bmatrix}, \quad \text{and} \quad \mathbf{b}=\begin{bmatrix}S_1(x_1-R_1y_1)\\ S_2(x_3+R_2y_1)\\ S_3(x_1+R_3y_3)\\ S_4(x_3-R_4y_3)\end{bmatrix} \tag{5.2.5}$$

Then our matrix equation is $\mathbf{Ax}=\mathbf{b}$. Recall from Section 4.3 that our least-squares solution is given by $\mathbf{x}=\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{\mathrm{T}}\mathbf{b}\right)$, where $\mathbf{A}^{\mathrm{T}}$ is the transpose of $\mathbf{A}$. We compute

$$\begin{aligned}\mathbf{A}^{\mathrm{T}}\mathbf{A}&=\begin{bmatrix}S_1 & S_2 & S_3 & S_4\\ -R_1S_1 & R_2S_2 & R_3S_3 & -R_4S_4\end{bmatrix}\begin{bmatrix}S_1 & -R_1S_1\\ S_2 & R_2S_2\\ S_3 & R_3S_3\\ S_4 & -R_4S_4\end{bmatrix}\\ &=\begin{bmatrix}{S_1}^2+{S_2}^2+{S_3}^2+{S_4}^2 & -R_1{S_1}^2+R_2{S_2}^2+R_3{S_3}^2-R_4{S_4}^2\\ -R_1{S_1}^2+R_2{S_2}^2+R_3{S_3}^2-R_4{S_4}^2 & {R_1}^2{S_1}^2+{R_2}^2{S_2}^2+{R_3}^2{S_3}^2+{R_4}^2{S_4}^2\end{bmatrix},\end{aligned}$$

$$\begin{aligned}&\mathbf{A}^{\mathrm{T}}\mathbf{b}\\ &=\begin{bmatrix}S_1 & S_2 & S_3 & S_4\\ -R_1S_1 & R_2S_2 & R_3S_3 & -R_4S_4\end{bmatrix}\begin{bmatrix}S_1(x_1-R_1y_1)\\ S_2(x_3+R_2y_1)\\ S_3(x_1+R_3y_3)\\ S_4(x_3-R_4y_3)\end{bmatrix}\\ &=\begin{bmatrix}{S_1}^2(x_1-R_1y_1)+{S_2}^2(x_3+R_2y_1)+{S_3}^2(x_1+R_2y_3)+{S_4}^2(x_3-R_4y_3)\\ R_1{S_1}^2(-x_1+R_1y_1)+R_2{S_2}^2(x_3+R_2y_1)+R_3{S_3}^2(x_1+R_3y_3)+R_4{S_4}^2(-x_3+R_4y_3)\end{bmatrix},\end{aligned}$$

and

$$\begin{aligned}&\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\\ &= \frac{1}{\det(\mathbf{A}^{\mathrm{T}}\mathbf{A})}\begin{bmatrix} {R_1}^2{S_1}^2 + {R_2}^2{S_2}^2 + {R_3}^2{S_3}^2 + {R_4}^2{S_4}^2 & R_1{S_1}^2 - R_2{S_2}^2 - R_3{S_3}^2 + R_4{S_4}^2 \\ R_1{S_1}^2 - R_2{S_2}^2 - R_3{S_3}^2 + R_4{S_4}^2 & {S_1}^2 + {S_2}^2 + {S_3}^2 + {S_4}^2 \end{bmatrix},\end{aligned}$$

where

$$\begin{aligned}\det\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right) = ({S_1}^2 + {S_2}^2 + {S_3}^2 + {S_4}^2)({R_1}^2{S_1}^2 + {R_2}^2{S_2}^2 + {R_3}^2{S_3}^2 + {R_4}^2{S_4}^2) - \\ (-R_1{S_1}^2 + R_2{S_2}^2 + R_3{S_3}^2 - R_4{S_4}^2)^2.\end{aligned}$$

Now, $\mathbf{x} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{\mathrm{T}}\mathbf{b}\right)$ gives us our desired solution, the full expression of which we will omit here.

5.3 Implementation and Results

A natural choice for our arbitrary point to be retargeted is the midpoint of the rectangle. Our algorithm, then, consists of partitioning the source image into four equal-sized quads, applying the above results to obtain the ideal position to which the midpoint should be retargeted; then, we repeat the exact same process with each of the four quads. Rinse, repeat, and recurse.

Figure 5.3.1 illustrates the result of this algorithm on the *Megazord* image from the previous chapter. As I expected, the results left something to be desired—I did not take any care to ensure that the boundary points of each quad were in alignment with each other. The speed, however, was incredible—the algorithm produced output within a fraction of a second. It remains to be seen whether better results can be obtained by tweaking the algorithm, or if the recursion idea is a lost cause. If one can figure out how to smooth out the kinks (quite literally), it may be a promising avenue for further research.

Figure 5.3.1. Result of recursive retargeting algorithm.

Appendix A
Selected Results

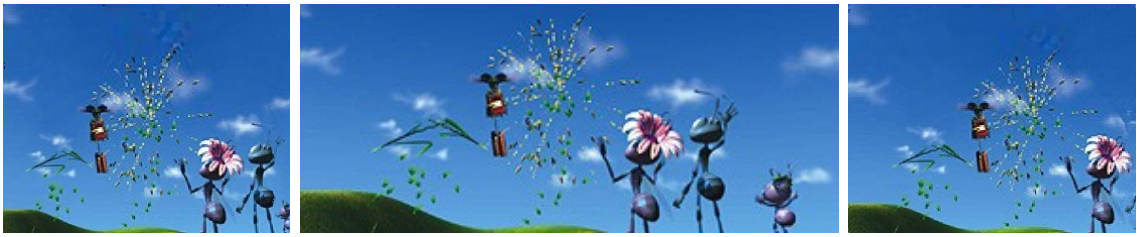
The following pages present, without commentary, a selection of results produced by my first retargeting algorithm, as described in Chapter 4. Each row consists of *(from left to right)* the retargeted result using the sum of saliency constraints, the original image, and the retargeted result using the product of saliency constraints.

Examples from Section 2.3

Example from Figure 2.3.2.

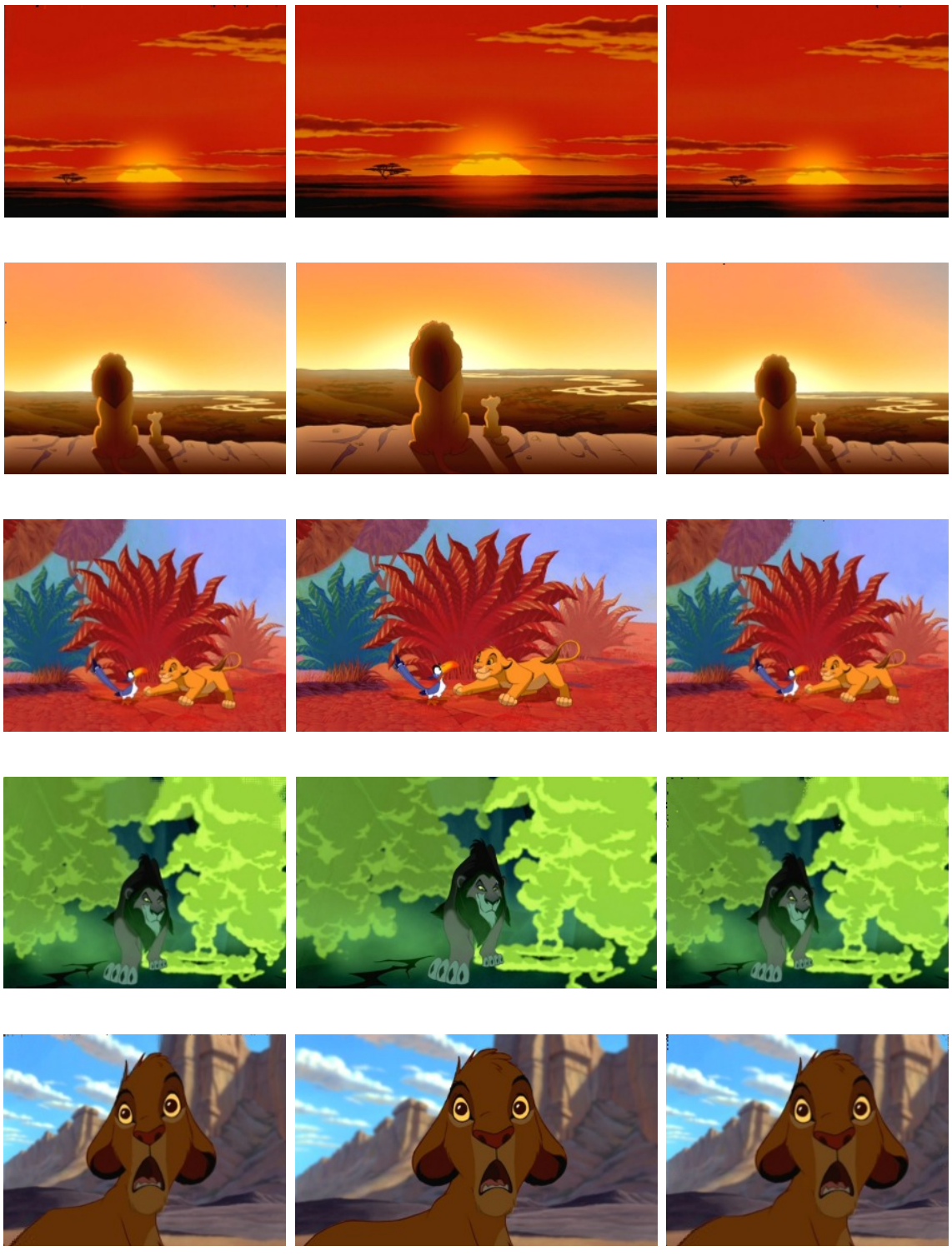
Example from Figure 2.3.2.

Example from Figure 2.3.4.



Example from *A Bug's Life* (© Pixar Animation Studios).

Examples from *The Lion King* (© Disney)

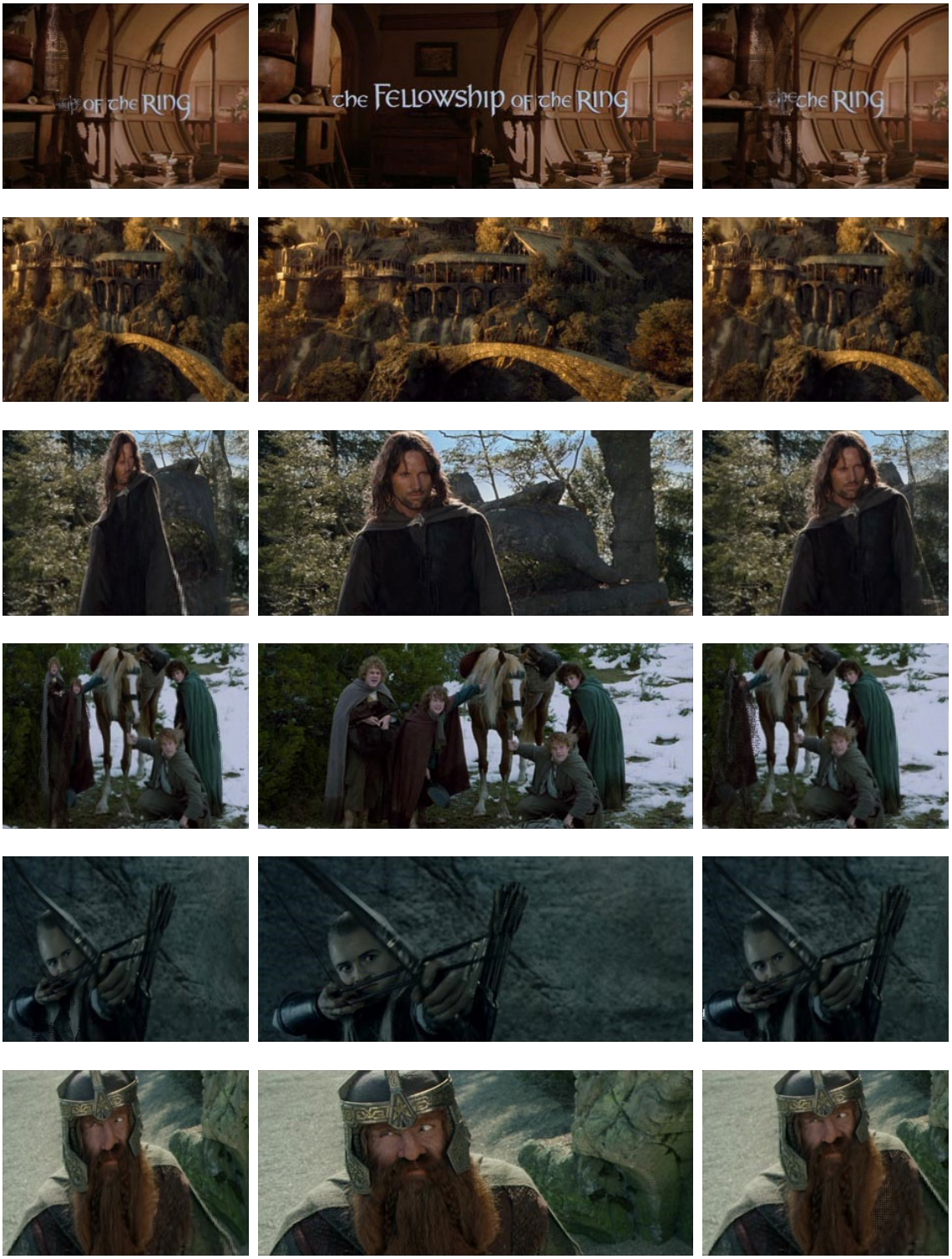


Examples from *Lawrence of Arabia* (© Columbia Pictures)

Examples from *Star Wars: Attach of the Clones* (© Lucasfilm)

Examples from *The Lord of the Rings: The Fellowship of the Ring*

(© New Line Cinema)



Examples from *The Lord of the Rings: The Two Towers*

(© New Line Cinema)

Bibliography

[1] Shai Avidan and Ariel Shamir, *Seam carving for content-aware image resizing*, ACM Trans. Graph. **26** (2007), no. 3, 10, DOI http://doi.acm.org/10.1145/1276377.1276390.

[2] Laurent Itti, Christof Koch, and Ernst Niebur, *A Model of Saliency-Based Visual Attention for Rapid Scene Analysis*, IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998), no. 11, 1254-1259.

[3] Michael Rubinstein, Ariel Shamir, and Shai Avidan, *Improved seam carving for video retargeting*, SIGGRAPH '08: ACM SIGGRAPH 2008 papers, 2008, pp. 1–9, DOI http://doi.acm.org/10.1145/1399504.1360615, (to appear in print).

[4] Yu-Shuen Wang, Chiew-Lan Tai, Olga Sorkine, and Tong-Yee Lee, *Optimized scale-and-stretch for image resizing*, SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers, 2008, pp. 1–8, DOI http://doi.acm.org/10.1145/1457515.1409071, (to appear in print).

[5] Lior Wolf, Moshe Guttmann, and Daniel Cohen-Or, *Non-homogeneous Content-driven Video-retargeting*, Proceedings of the Eleventh IEEE International Conference on Computer Vision (ICCV-07), 2007.

[6] Intel, *OpenCV 2.1 Documentation* (2009).

[7] Timothy Davis, *User's Guide for SuiteSparseQR, a multifrontal multithreaded sparse QR factorization package* (2009).

[8] *Widescreen-O-Rama!*: *All About Aspect Ratios (Why Widescreen Really* Is *Better)*, The Digital Bits, 2000 (accessed April 24, 2010). http://www.thedigitalbits.com/articles/anamorphic/aspectratios/widescreenorama.html.

[9] *What is Widescreen?*, Widescreen Advocate, 2002 (accessed April 24, 2010). http://www.widescreenadvocate.org/whatis.html.

[10] *Widescreen Information Page*, accessed April 24, 2010. http://www.twowiresthin.com/aspect/.

[11] Bill Hunt, *Pixar talks A Bug's Life*: *Interview with Leo Hourvitz and Bill Kinder*, The Digital Bits, 1999 (accessed April 24, 2010). `http://www.thedigitalbits.com/articles/pixarbugslife.html`.

[12] *FlikFX Digital Recomposition System*, The American Widescreen Museum, 1999 (accessed April 24, 2010). `http://www.widescreenmuseum.com/flikfx/loatest1.htm`.