AtlasWerks User's Guide

J. Samuel Preston

November 23, 2011

Contents

Acknowledgements

This work was made possible in part by NIH/NIBIB Grant 5R01EB007688 and NIH/NCRR Grant 1P41RR023953

Chapter 1

Introduction

1.1 Overview

AtlasWerks is an open-source (BSD license) software package for medical image atlas generation. It provides efficient CPU and GPU implementations of non-linear deformation algorithms for single machines and clusters, command line frontends for atlas and deformation generation, and a number of utility programs for handling 3D medical image data. This document gives a high-level overview of the concepts involved in atlas generation. Chapters 1-4 cover some background on atlas generation and the algorithms used in AtlasWerks. Chapter 5 covers the nuts and bolts of running the programs included in AtlasWerks, and the impatient reader can skip directly there.

1.2 Introduction

Many applications in medicine require an anatomical 'reference space' in which measurements from different specimines can be taken and compared against each other for the purposes of segmentation, disease classification, etc. [need citation]. Original methods involved using a specific subject as the reference and taking measurments with respect to that specific anatomy [talairach space]. Aligning all images to a specific example, however, biases results towards the template anatomy chosen. It is preferable to have the template represent the 'average' anatomy, for example so that differences of an individual from the template represents a difference from the average. When an atlas image is mentioned in this document, it refers to such an average image.

1.3 Average Images

In order to describe the process of generating an atlas, the idea of an 'average image' must first be breifly discussed. Images are just a grid of intensity values, and given a set of images of the same size, the average image could simply be computed as the average intensity value at each pixel. However, this does not result in what we would expect from an average of a number of related images (see Figure 1.1). Instead, what we expect is an image which represents the average 'shape' of an object, not the average intensity values. Figure 1.2 shows a result closer to what we would expect. In order to create such an average, we will deform each image towards an average. For this reason, we must talk briefly about image deformation and registration.

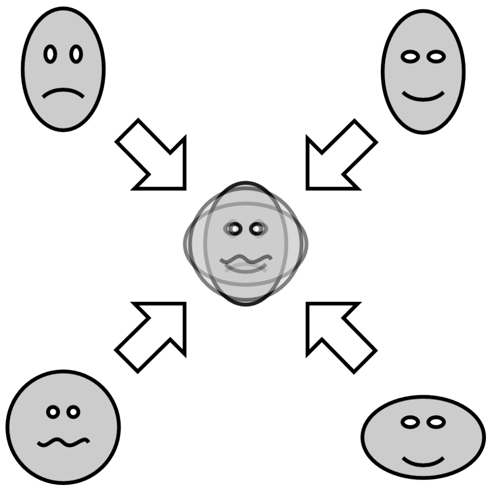

Figure 1.1: Atlas Generation via Intensity Averaging

1.4 Image Registration and Atlas Building

The basic problem of image registration is to find a mapping that will deform one image (the 'moving' image) to match, as closely as possible, another image (the 'static' or 'template' image). That is, the mapping is applied to the moving image to create a new image, the 'deformed' or 'registered' image, that matches the static image as closeley as possible. See Figure 1.3.
In our case, this mapping takes the form of a vector field, generally at the same resolution as the image, giving the motion of each point due to the deformation (Figure 1.4).
In order for the deformations generated by this software to be useful for medical image analysis, they must represent a 'reasonable' mapping. For instance, while we may want to allow some scaling of the objects being deformed, we probably do not want the deformation field to shrink any structure to zero volume. Also, we probably do not want vector fields to cross through themselves in non-physical ways. In order to guarantee these goals, we use a Large Deformation Diffeomorphic Metric Mapping (LDDMM) framework. This method generates

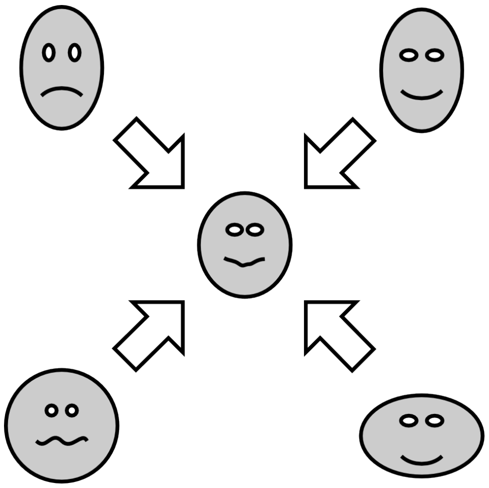
Figure 1.2: Atlas Generation via Average Shape

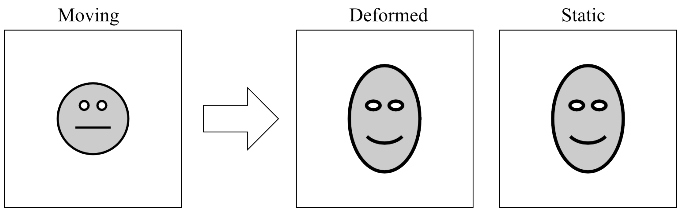

Figure 1.3: Image Registration

smooth, invertible deformations by solving PDEs representing the deformation as the flow of a viscous fluid (representing the deforming image space) influenced by a force attempting to match the images.

It is important, however, to note that the direction of the deformation is the opposite of the expected direction. The reason for this comes from the way the deformed image is created. Given the moving image and a deformation, we want to generate the deformed image. An image is simply a regular grid of intensity values. We know the grid values for the moving image, and we are trying to assign values at the gridpoints of the deformed image. If the deformation field gave the motion of each gridpoint from the moving image to the deformed image, these points would in general fall at non-gridpoint locations, and we would have the problem of how to assign values to the surrounding points (Figure 1.5). Even if we were able to assign these values correctly, for large deformations there may be points in the deformed image for which no deformation vectors come near, resulting in 'holes' in the deformed image. The process is greatly simplified by having the deformation point from the deformed image grid back to the initial location of that point in the moving image. Using this method, each point in the deformed image is assigned a value by finding its location in

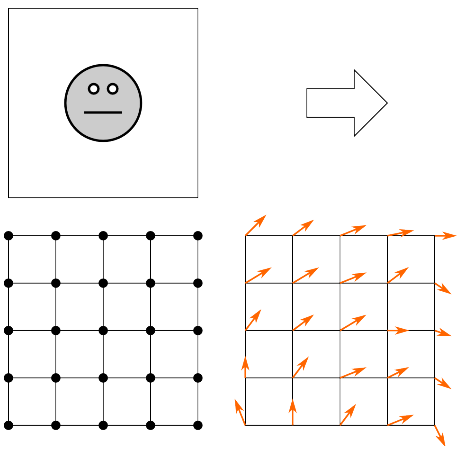

Figure 1.4: Vector Field Mapping

the moving image, and inerpolating the surrounding values to get the value to assign to that point (Figure 1.6).

Now that we see how the direction of the affect of applying a deformation to an image is opposite the direction of the deformation, Figure 1.7 introduces the notation we will use in this section. The arrow represents the deformation (as well as the direction – vectors of ϕ point from locations in the initial moving image to corresponding locations in the deformed image). In order to create I^{def}, grid locations in the space of I^{def} are deformed by ϕ^{-1}, and I^{M} is sampled at these points.

There are three methods for generating deformations. All are based on the LDDMM equations, but use different methods for solving them. The differences between the solvers are discussed in greater detail in Chapter 3, but for now it is sufficient to say that the 'Greedy' solver is faster and takes less memory, and is often sufficient if the desired result is simply the deformed image. If other properties of the deformation (also discussed later) are of interest, the 'Relaxation' (often just called LDMM) or 'Shooting' solver is probably needed.

In order to generate an atlas, an alternating algorithm is employed which first holds fixed a set of deformations (from each input image to the mean), deforming each input image towards the mean, and taking the intensity average of the deformed images as the current estimate of the mean. The mean image is then held fixed, and used to update the deformations from each input image to this mean. The deformations are usually initialized with an identity transform, so the initial estimate of the atlas image is a simple intensity average of the input images. As iterations progress, the estimage of the mean is refined from something like Figure 1.1 to something like Figure 1.2.

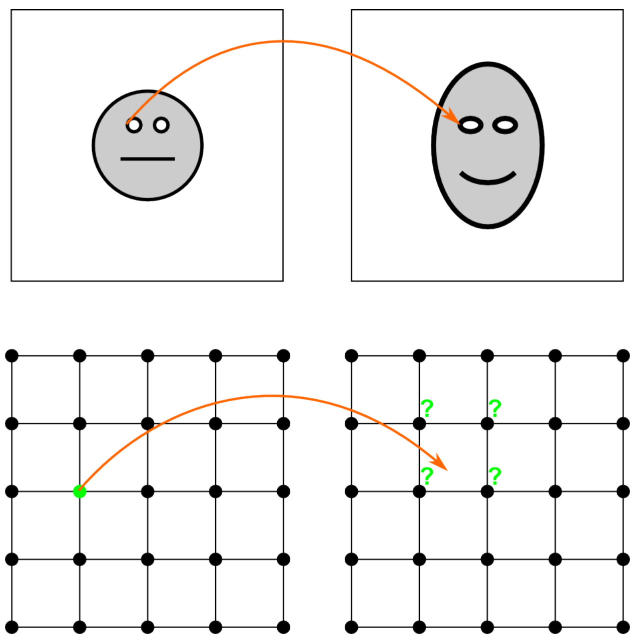

Figure 1.5: Image Registration Problem

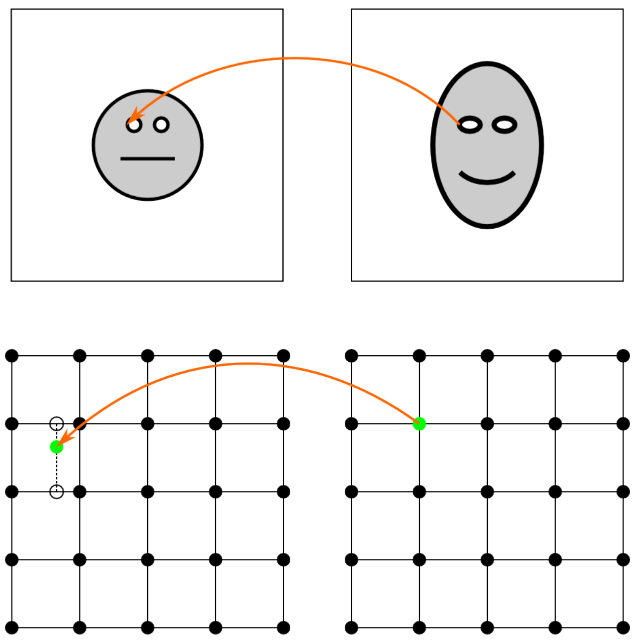

Figure 1.6: Image Interpolation

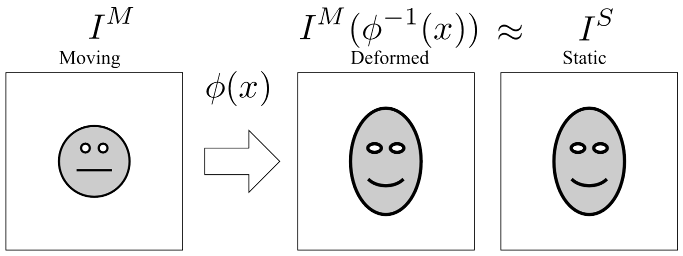

Figure 1.7: Image Registration

Chapter 2

Optimization Parameters

2.1 Fluid Parameters

As mentioned in the introduction, AtlasWerks generates deformations via a fluid analogy, where one can imagine the moving image as being embedded in a viscous fluid. The only force acting on this fluid is one attempting to match the embedded image with the static image. The fluid is allowed to flow until stability is reached, and the path of each particle in the fluid makes up the final deformation field that represents our registration.
From the fluid equations, we get constraints on the properties of the deformation itsself. Specifically, the energy given by the differential operator **L**:

$$\mathbf{L} = \alpha\nabla^2 + \beta\nabla(\nabla\cdot) + \gamma \tag{2.1}$$

should be minimized. Details regarding this operator can be found elsewhere [need citation], but it is sufficient for most users to note the three parameters controlling the eauation, α, β, and γ. While not exact, the α term can be thought of as controlling the smoothness of the deformation (larger α results in a smoother deformation), the β term can be thought of as controlling the 'compressability' of the fluid (larger β results in greater volume preservation), and γ is a regularization term that also penalizes the total size of the deformation. All values should be positive, and α and γ should be nonzero.

2.2 Inexact Matching Parameter

The force used to match the images comes from a simple sum-of-squares intensity difference. That is, an ideal matching would minimize the value $\sum_i I^{\mathrm{def}} - I^{\mathrm{S}}$, where I^{def} is the deformed image, I^{S} is the static image, and i is the number of pixels/voxels in the image. For this reason, it is important that images to be matched have similar intensity ranges. For CT scans this is generally not a problem, but MRIs often require intensity normalization to achieve best results. Also, while ideally the deformed image would match the static image

Table 2.1: Optimization Parameters

Parameter	Description
α	Controls smoothness of deformation (higher value = smoother)
β	Controls compressability of deformation (higher value = greater volume preservation)
γ	Regularization parameter / penalizes deformation size (higher value = smaller deformations)
λ	Controls tradeoff betweeen smooth deformations and exact matching (higher values = more exact matching)
σ	Same purpose as λ, but $\frac{1}{\sigma^2} = \lambda$ (lower values = more exact matching)
Step Size (Δ)	Affects speed and stability of optimization. Lower values result in slower but more stable convergence.

exactly, with real images this is never the case. Therefore, the final energy to be minimized is a combination of the energy from the **L** operator and the sum-of-squares image intensity difference:

$$\mathbf{E} = E_{vec} + \lambda E_{im} \tag{2.2}$$

The parameter λ controlls the tradeoff between exact image matching and smooth deformation fields – low values of λ result in smoother deformation fields, while higher values of λ result in more exact matching. The term λ is sometimes written as $\frac{1}{\sigma^2}$, in which case σ is the parameter, and has the opposite effect (lower values of sigma result in more exact matching). A summary of the parameters appear in Table 2.1

2.3 Step Size

The step size parameter sets the size of the update at each iteration. A higher value will cause the optimization to converge faster, but too high a value will cause the optimization to diverge ('blow up'). A smaller step size will cause more predictable convergence, but may require more iterations (and therefore more time) to converge. In general this parameter must be chosen experimentally by watching the energy convergence for a specific run. For greedy optimization (explained in Chapter 3) there is 'Max Perturbation' parameter, which is equivalent to step size, except it is constrained to fall in the range (0,1).

2.4 Multiscale

In order to speedy optimization and avoid local minima during optimization, a coarse-to-fine registration approach is used, where downsampled versions of the images are first registered to take care of coarse feature matching, followed by succesively higher resolution versions to match finer and finer detail. At the final scale level, the original images are directly used in the optimization. Each scale level can have its own set of parameters, although in most cases only the step size will be varied across scale levels.

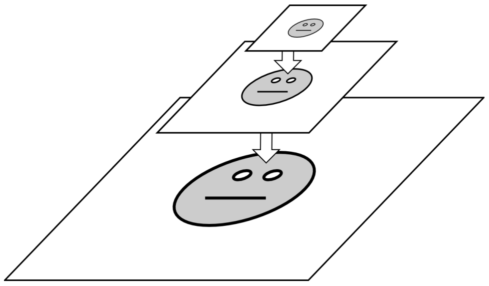

Figure 2.1: Multiscale optimization starts with the lowest scale level and decreases the downsample factor until optimization is being performed on the original image

Chapter 3

Optimization Algorithms

AtlasWerks supports three different algorithms for computing deformations based on the LDDMM equations – Greedy Optimization, Relaxation Optimization, and Shooting Optimization. In order to discuss the different methods, it is necessary to discuss a few more details of the registration method. Specifically, it must be noted that it is not just the final deformation that is solved for, but the path each point follows as it deforms from the initial moving image to the final deformed image. Thinking of the fluid analogy, a fluid particle (representing a specific location in the image) flows from its position in the initial image to a final location, giving the deformed image. The path that is taken is likely not a straight line – it depends on the imaging force and the flow of other particles at any given time (see Figure 3.1). For convenience, time is generally parameterized such that time $t = 0$ corresponds to the identity transform (resulting in the initial moving image), and time $t = 1$ corresponds to the final deformed image.

3.1 Greedy Optimization

In greedy optimization, there is no explicit representation of time. At each iteration the optimal update direction for the current total deformation is calculated, and a step in this direction is composed with the current deformation to give the updated deformation. Only the total composed deformation is retained from iteration to iteration, so information about the optimal path is not retained (or ever truly computed, for that matter). As the optimal path is not computed, the deformation energy is not computed at each step (only the image difference energy), so this optimization does not use the λ σ parameter. Figure 3.2 shows how greedy optimizaiton affects a single point in a single dimension over a series of iterations.

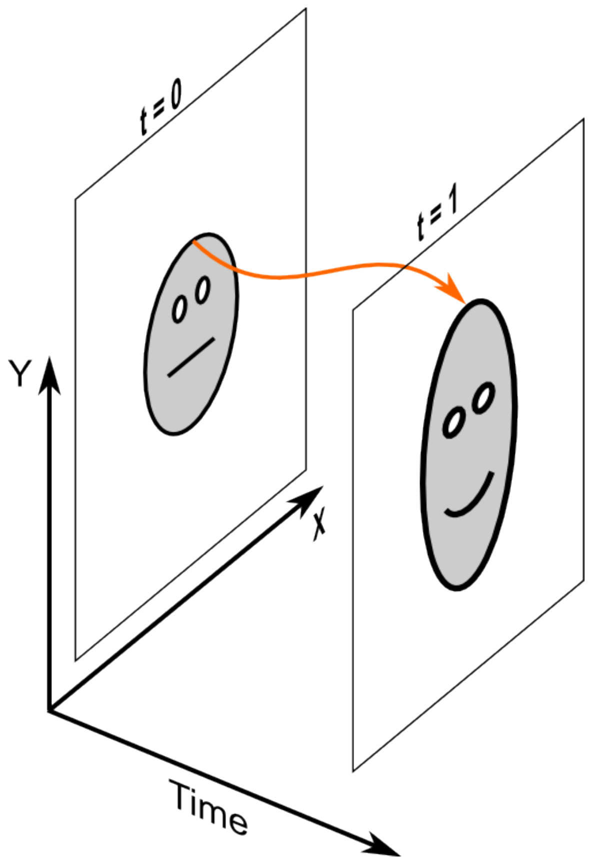

Figure 3.1: Time-parameterized deformation of a single point

3.2 Relaxation Optimization

In relaxation optimization, time is explicitly discretized, with a vector field at each time giving the motion of particles at that timestep. During an iteration, each vector field is updated to give a better approximation to the true deformation. Figure 3.3 shows how relaxation optimizaiton affects a single point in a single dimension over a series of iterations.

3.3 Shooting Optimization

Shooting optimization is much like relaxation optimization, but instead of storing a series of velocity fields, only a single ‘momentum’ image is stored. From this, a series of velocity fields can be computed via the evolution equations discussed later [need internal / external ref]. These momentum images are updated at each iteration to get succesively better approximations to the ideal deforma-

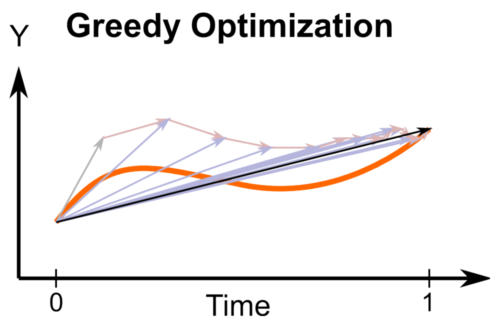

Figure 3.2: This graph shows the motion of a single point in a single dimension during greedy optimization. The ‘ideal’ path for this point is shown in orange. Since greedy optimization has no explicit representation of time, each iteration updates the total deformation to better match the final image. The blue-gray arrows represent the final deformation at different iterations, while the red-gray arrows represent the updates at each iteration.

tions. Figure 3.4 shows how shooting optimizaiton affects a single point in a single dimension over a series of iterations.

3.4 Atlas Building

For atlas building, the direction of the individual deformations changes for greedy vs. LDMM relaxation/shooting. Figures 3.5 and 3.6 show the differences in direction. The reasons for this are explained in detail in Appendix E, but it comes from the fact that the relaxation deformations can be reliably inverted, while the greedy deformations cannot.

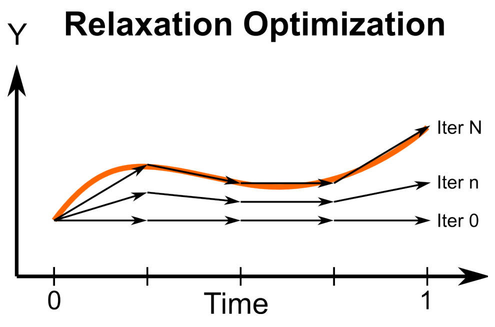

Figure 3.3: This graph shows the motion of a single point in a single dimension during relaxation optimization. The 'ideal' path for this point is shown in orange. Initially (iter=0) all vector fields are set to zero, so there is no deformation. As iterations progress, the vector fields are updated to provide closer and closer approximations to the ideal deformation.

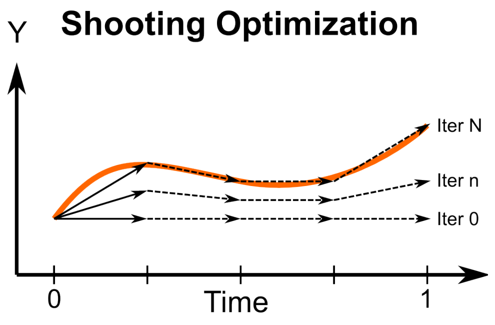

Figure 3.4: This graph shows the motion of a single point in a single dimension during shooting optimization. Time is discretized much like in relaxation optimization, but only an initial momentum is stored, and the rest of the deformation is computed via the evolution equations.

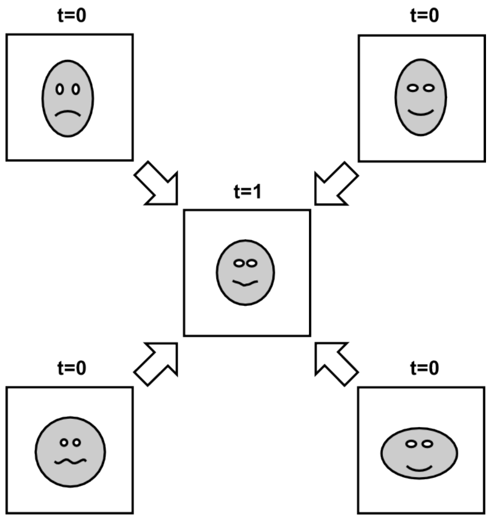

Figure 3.5: Atlas computation via the greedy algorithm. The mean is at timepoint 1, the individual images at timepoint 0 for each deformation.

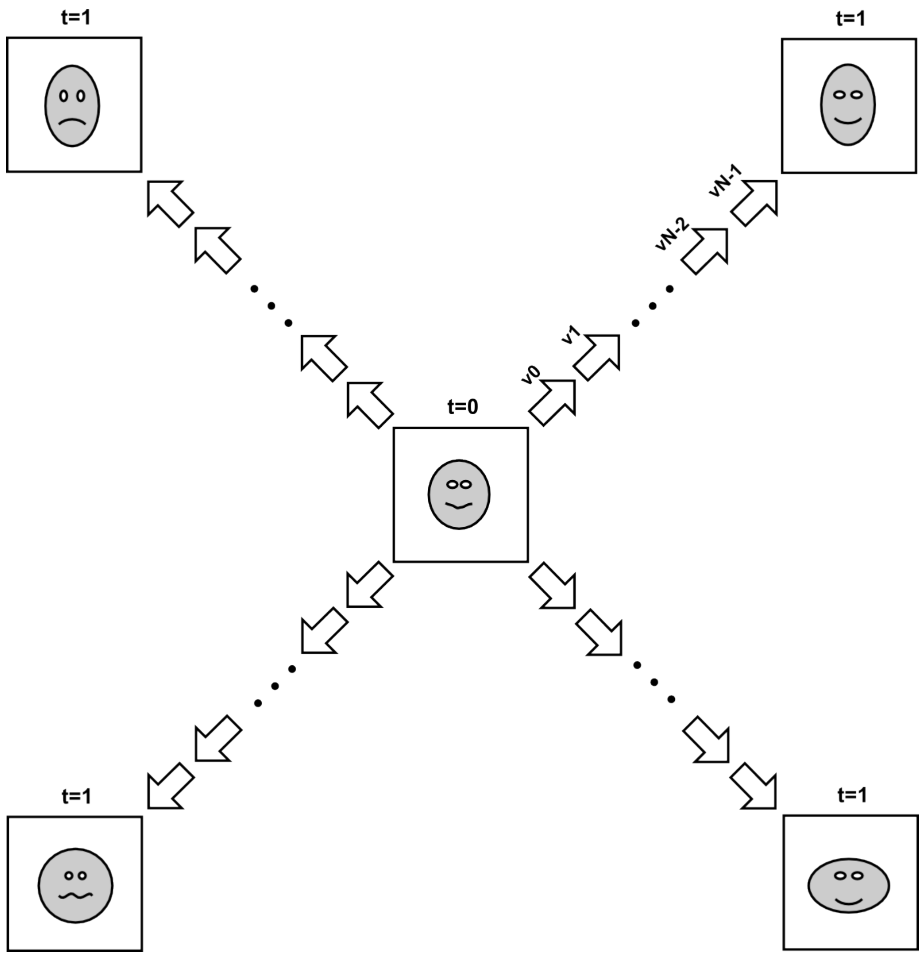

Figure 3.6: Atlas computation via LDDMM relaxation or shooting. The mean is at timepoint 0, the individual images at timepoint 1 for each deformation.

Chapter 4

Manifolds

4.1 Shape Manifold

In order to discuss some of the ideas behind the algorithms in AtlasWerks more fully, we must make a digression into the idea of a manifold. A manifold is simply a surface such that at any point on the surface, for a small enough area, the geometry looks like euclidean (flat) space of the dimension of the manifold. The canonical example of a manifold is the surface of a sphere. The surface of a sphere is a 2D surface in 3D space ($\mathbb{R}^3$). It is curved everywhere, but for a small enough patch of surface, it is very close to flat (see Figure 4.1).

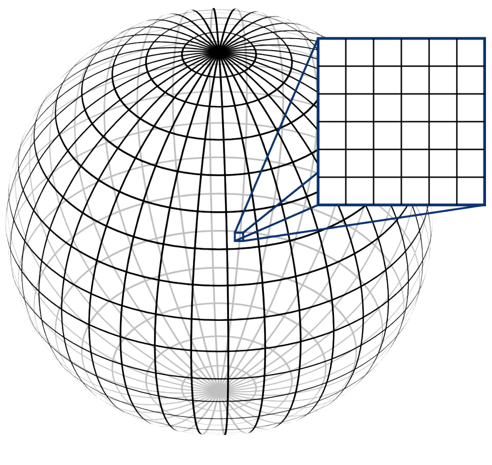

Figure 4.1: The surface of a sphere as a manifold – at any point, it can locally be approximated as flat

The manifold we are interested in, however, is in very high-dimensional space. We are interested in a surface embedded in the space of all images. For simplicity, let us consider only greylevel images of size 256x256. We know that these images are simply an array of intensity values. For a computer to represent these, the intensity values must be a type such as int or float, which has a countable set of possible values, but for our purposes let us also assume that intensity values are just a Real number. Each image can then just be thought of as a vector of length $\mathbf{N} = 65536$, which represents a point in $\mathbb{R}^{\mathbf{N}}$. To reiterate, *any* 256x256 greylevel image corresponds to a point in this high-dimensional space, and any point in the space could be viewed as a 256x256 greylevel image. Most of the images in this space would look like random noise, but a point might also represent a greyscale picture of your dog or an image of a slice of a brain. If we take all of the points that represent possible MRI images of human brains, we would expect them to cluster tightly into a lower-dimensional surface in this high-dimensional space. Given some constraints on the smoothness of this surface, it constitues a manifold (see Figure 4.2).

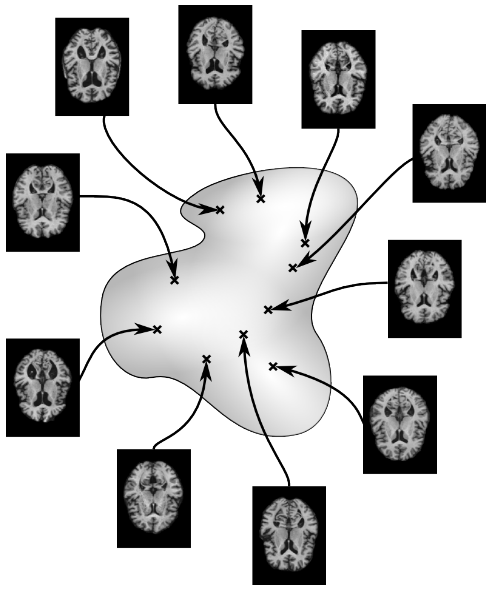

Figure 4.2: Our representation of a high-dimensional manifold in a very high dimensional space.

4.2 Geodesics

Given two of these brain images (two points on the manifold), we can compute a deformaiton between the two images as described earlier. Every intermediate point along the deformation gives an image of an intermediate brain, and therefore lies on the manifold. The energy minimized in Equation 2.2 then gives us a 'distance' along the manifold between these two points – in fact, the minimized distance gives the *shortest* path between the points. We call this path a geodesic, which is just a generalization of a straight line to curved space. On the sphere in Figure 4.1, a geodesic between two points is a segment of the great circle containing the two points. On our 'brain manifold' we can compute not only the geodesic between two points, but also the continuation of the geodesic past our chosen points (see Figure 4.3).

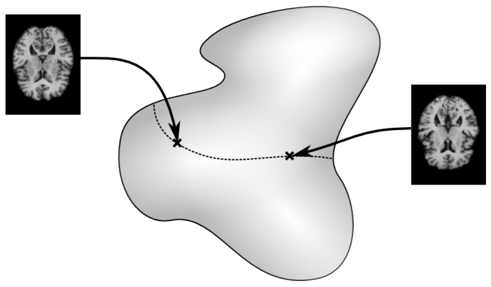

Figure 4.3: A geodesic on a manifold

4.3 Atlas building and optimization methods

Given this notion of distance, our atlas is the point which minimizes the sum of the squared distances from each input brain to the atlas – a generalization of the standard idea of a mean. Figure 4.4 shows this idea.

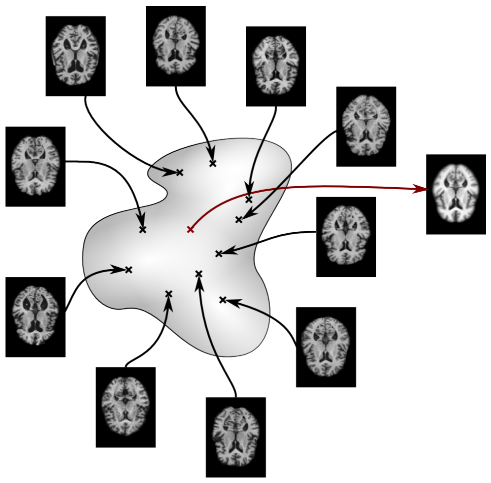

Figure 4.4: Atlas generation on our manifold

We can also now see the differences in the deformations produced by greedy and relaxation optimization. As shown in Figure 4.5, while the greedy deformation produces a single deformation able to deform the moving image to the static image, relaxation optimization produces a series of steps along the geodesic which represent the actual path of the deformation.

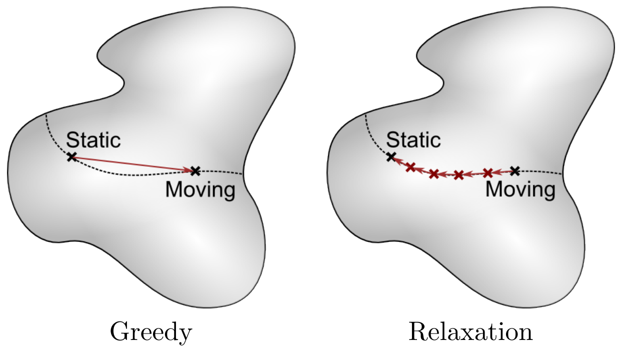

Figure 4.5: Differences in deformation results between Greedy and Relaxation Optimization methods

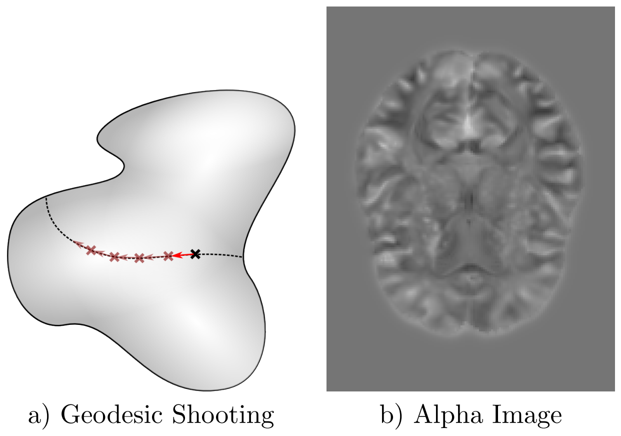

Figure 4.6: (a) Results of geodesic shooting on a manifold. (b) A ‘momentum’ image, defining the path for which an image will evolve.

Chapter 5

Applications

This chapter covers the basics of running the applications contained in AtlasWerks. Table 5.1 gives a list of the most-used applications and their purpose.

5.1 Parameter Files

The deformation and atlas building applications included in AtlasWerks take a large number of parameters, far more than can be easily entered on the command line. Therefore the parameters are contained in an XML parameter file, and read in when the application is run. As with all applications in AtlasWerks, a brief overview of available commands can be generated by passing the `-h` command. For commands taking a parameter file, the `-f` command is the most important, as it specifies the parameter file to read. These applications also accept the `-g` option, which generates a 'default' parameter file, showing all of the options available to the program. A partial listing of such a default parameter file is shown in Listing 5.1. Note the comments that are generated, which give an idea

Table 5.1: Application Descriptions

Application	Description
GreedyWarp / GreedyAtlas	Greedy deformation and atlas building
LDMMWarp / LDMMAtlas	LDMM relaxation and shooting deformation and atlas building
txWerks / txApply	Utilities for composing and applying deformations
LDMMVelocityUtils	Utilities for working with time-varying velocity fields
GeodesicShooting	Calculation of image evolution along a geodesic
ImMap	GUI application for comparing images and surfaces

Listing 5.1: Section of a default parameter file generated with via the -g flag

```
<!--top-level node-->
<LDMMWarpParameterFile>
  <!--General settings for multiscale manager-->
  <MultiscaleSettings>
    <!--Use sinc upsampling for images? (trilinear interpolation is
    the default)-->
    <UseSincImageUpsample val="false" />
  </MultiscaleSettings>
  <!--Multiple of the following can occur...-->
  <!--Settings for single-scale LDMM registration-->
  <LDMMScaleLevel>
    <!--setting for a single scale level-->
    <ScaleLevel>
      <!--factor by which to downsample images-->
      <DownsampleFactor val="1" />
    </ScaleLevel>
    <!--Settings for LDMM iteration-->
    <Iterator>
      <!--Differential operator parameters-->
      <DiffOper>
        <!--Controls the 'viscosity' of the fluid-->
        <Alpha val="0.01" />
        <!--Controls the 'viscosity' of the fluid (penalizes expansion/
        compression)-->
        .
        .
        .
```

of what the parameter controls. Generally only a small subset of the parameters need to be changed from the default. Although each application takes different parameters, many of the parameters are the same. The following parameter file snippets come from a standard configuration for LDMMWarp, but most of the parameters apply to other applications as well. The descriptive comments and many unused parameters have been removed to improve readability. If a parameter is not included in a parameter file, its default value (as generated with the `-g` option) is used. Some parameters (such as input files) must be specified.
Listing 5.2 shows the specification of the input files to the deformtion. ImagePreprocessor defines some default operations that can be performed on the input images before they are passed to optimization. Many of the unused options have been removed from this listing. The given parameters direct the application to rescale the intensities in the input images from $[-1024, 425]$ to the range $[0, 1]$, clamping any values outside the input window range. The final two options direct the preprocessing to be performed on both the Moving and Static images (it is sometimes useful to only preprocess one of the images). Listings 5.3 and 5.4 show input parameters for LDMMAtlas, where a list of input files must be specified. Both specify the same set of files – one via a printf-style format string, and one via a list of input files.
Listing 5.5 shows the parameters that govern the multiscale optimization. Once again, many parameters that are left at their default values are removed from the parameter file – if they are omitted, the default value will be used. The listing shows two scale levels (`<LDMMScaleLevel>...</LDMMScaleLevel>`). The first is the full-resolution (`<DownsampleFactor val="1"/>`) scale level, which will be run for 100 iterations. The α, β, γ, and σ parameters are set, and adaptive step sizes with a max. perturbation of 0.1 is used. Note that in the lower resolution scale level (`<DownsampleFactor val="2"/>`), the parameters are omitted. The

Listing 5.2: Sample input image parameters

```xml
<MovingImage val="/home/me/data/MyImage01.mha" />
<StaticImage val="/home/me/data/MyImage02.mha" />

<ImagePreprocessor>
  <IntensityWindow>
    <RescaleIntensities val="true" />
    <UseInputIntensityWindow val="false" />
    <InputWindowMin val="-1024" />
    <InputWindowMax val="425" />
    <OutputWindowMin val="0" />
    <OutputWindowMax val="1" />
  </IntensityWindow>
</ImagePreprocessor>
<PreprocessMovingImage val="true" />
<PreprocessStaticImage val="true" />
```

Listing 5.3: Sample input images for atlas building

```xml
  <WeightedImageSet>
      <InputImageFormatString>
          <FormatString val="/home/me/data/MyInputFile%03d.nrrd" />
          <Base val="0" />
          <NumFiles val="4" />
          <Weight val="1" />
      </InputImageFormatString>
      <ScaleImageWeights val="true" />
  </WeightedImageSet>
```

Listing 5.4: Sample input images for atlas building

```xml
  <WeightedImageSet>
      <WeightedImage>
          <Filename val="/home/me/data/MyInputFile000.nrrd" />
      </WeightedImage>
      <WeightedImage>
          <Filename val="/home/me/data/MyInputFile001.nrrd" />
      </WeightedImage>
      <WeightedImage>
          <Filename val="/home/me/data/MyInputFile002.nrrd" />
      </WeightedImage>
      <WeightedImage>
          <Filename val="/home/me/data/MyInputFile003.nrrd" />
      </WeightedImage>
      <ScaleImageWeights val="true" />
  </WeightedImageSet>
```

Listing 5.5: Sample scale level parameters

```
<LDMMScaleLevel>

  <ScaleLevel>
    <DownsampleFactor val="1" />
  </ScaleLevel>

  <Iterator>
    <DiffOper>
      <Alpha val="0.01" />
      <Beta val="0.01" />
      <Gamma val="0.001" />
    </DiffOper>
    <Sigma val="2" />
    <UseAdaptiveStepSize val="true" />
    <MaxPert val="0.1" />
  </Iterator>

  <NIterations val="100" />

</LDMMScaleLevel>

<LDMMScaleLevel>
  <ScaleLevel>
    <DownsampleFactor val="2" />
  </ScaleLevel>

  <Iterator>
    <MaxPert val="0.1" />
  </Iterator>

  <NIterations val="200" />

</LDMMScaleLevel>

<NTimeSteps val="5" />
```

values are inherited from the previously-defined scale level, so as long as we want the same parameters at each scale level, they can be omitted. Note that they will inherit from the previous scale level in the order they appear in the parameter file, not the previous scale level based on downsample level. In fact, in this case the scale level with DownsampleFactor=2 is defined after the full-resolution scale level, even though they will actually be run in the opposite order. They could have been definied in the opposite order in the parameter file, but the first scale level defined should give the parameters that will be set across all scale levels. Note also the NTimeSteps parameter – this is set outside of a scale level, as it must be held constant across all scale levels. It is the number of discrete timesteps our deformation is broken into. It must have a fairly small value, as the memory requirements increase nearly linearly with the number of timesteps. See Chapter 2 for more details on the meanings and effects of the parameters.

Listing 5.5 shows some of the output options for LDMMWarp. The name of output files is generally formed from a concatenation of the OutputPrefix value, the input file name, and a suffix indicating what the output represents (deformed image, inverse deformed image, etc.) Note that 'velocity fields' here represent the individual timestep offsets, while 'deformation fields' represent the entire composed deformation.

Listing 5.7 shows some miscellaneous parameters that are never the less quite important. UseGPU determines whether the results will be calculated on the CPU or the GPU, GPUId chooses the device ID of the GPU to use (if there is more than one GPU available), and ShootingOptimization chooses whether

Listing 5.6: Sample output parameters

```
<OutputPrefix val="" />
<OutputSuffix val="mha" />
<WriteDefImage val="true" />
<WriteInvDefImage val="true" />
<WriteDefField val="true" />
<WriteInvDefField val="true" />
<WriteVelocityFields val="false" />
<WriteIntermediateImages val="false" />
<WriteAlphas val="false" />
<WriteAlpha0 val="false" />
<WriteInitialScaleLevelDefImage val="false" />
<WriteFinalScaleLevelDefImage val="false" />
```

Listing 5.7: Sample optimization options

```
<ShootingOptimization val="false" />

<UseGPU val="false" />
<GPUId val="0" />
```

standard relaxation or shooting optimization is used to compute the deformation.

Given this introduction, most of the parameters will be decipherable from the auto-generated default parameter files for each application. If a parameter in undecipherable, it is probably safe to leave it at it's default value (or just delete it, accomplishing the same thing). For most real-world data from MRI or CT, three or four scale levels with downsample factors doubling at each level yeild the best results.

Chapter 6

ToDo

6.1 Describe naming conventions for direction of deformation

6.2 Describe necessary preprocessing

Appendix A

Time Varying Vector Fields

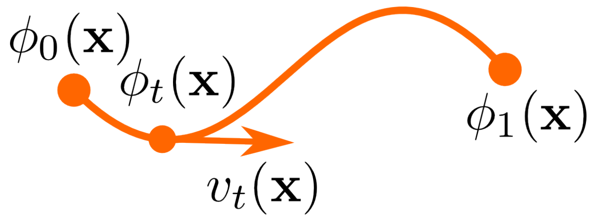

Figure A.1: Time-parameterized deformation of a single point

As described briefly in Chapters 3 and 4, the deformations from LDDMM are generated as a time-varying velocity field $v(x,t)$. Time is parameterized on the arbitrary interval $[0,1]$, where $t=0$ corresponds to the initial image and $t=1$ corresponds to the final deformed image. The total deformation for point x at time t is given by $\phi(x,t)$:

$$\phi(x,t) = x + \int_0^t v(\phi(x,t),t)dt \tag{A.1}$$

Or, written with time as a subscript:

$$\phi_t(x) = x + \int_0^t v_t \circ \phi_t(x)dt \tag{A.2}$$

Alternatively, the relation can be written as:

$$\frac{d}{dt}\phi_t(x) = v_t \circ \phi_t(x) \tag{A.3}$$

Note that v_t is given in an Eulerian reference frame, where $v_t(x)$ gives the velocity of the particle that is at point x at time t. If we want the velocity at some time t of the particle that was at point x at $t=0$, we must trace it's path forward in time and then find the velocity at this new location, calculated as

$v_t \circ \phi_t(x)$. We will call a vector field such as ϕ, which maps a point x at one timepoint to its new location $x' = \phi(x)$ at another timepoint, a deformation field or h-field. Vector fields such as v that map a point x to its displacement at a given timepoint are called velocity fields, or v-fields.
Another very useful notation, introduced in [2], is:

$$\phi_{s,t} = \phi_t \circ \phi_s^{-1} \tag{A.4}$$

This means that $\phi_{s,t}(x)$ gives the position of a particle at time t that was at position x at time s. In this notation, the full deformation ϕ would be $\phi_{0,1}$, and ϕ^{-1} is $\phi_{1,0}$.

A.1 Composition of velocity fields

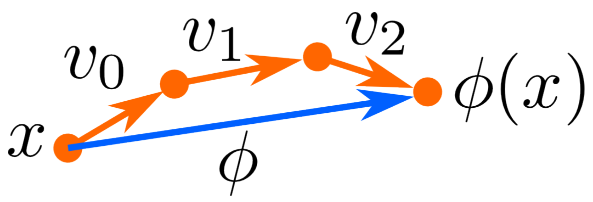

Figure A.2: Simple discretization of the deformation of a single point into three timesteps

Given a discrete set of velocity fields representing the deformation $\phi(x)$, the practical problem arises of creating the total deformation or a section of the deformation. Figure A.2 shows a simple deformation with three timesteps. If we label the time subscript of ϕ with the discrete timestep, instead of the time, we have ϕ_n, $n = 0..3$, where $\phi_0 = x$ and $\phi_3 = \phi$. The discrete version of equation A.2 is:

$$\phi_{n+1}(x) = x + \sum_{0}^{N-1} v_n \circ \phi_n(x) dt \tag{A.5}$$

For our three timestep example, this yeilds:

$$\begin{aligned}
&\phi_0(x) = x \\
&\phi_1(x) = x + v_0(x) \\
&\phi_2(x) = x + v_0(x) + v_2(x + v_0(x)) \\
&\phi_3(x) = x + v_0(x) + v_2\big(x + v_0(x)\big) + v_3\Big(x + v_0(x) + v_2\big(x + v_0(x)\big)\Big)
\end{aligned}$$

This can easily be calculated as:

Given: $v_n, n = 0 \ldots N-1$

```
$\phi_0 = x$
for $i = 0 \ldots N-1$ do
    $\phi_{i+1}(x) = \phi_i(x) + v_i(\phi_i(x))$
end for
```

In fact, given a series of velocity fields, there are four common deformations we wish to build: $\phi_{0,t}$, $\phi_{1,t}$, $\phi_{t,0}$, and $\phi_{t,1}$. The action on an image is to deform the image at time t to the initial or final timepoint in the first two cases, and to deform the image at the initial or final timepoint to time t in the latter two cases. The algorithms for iteratively constructing the deformaitons are given below:

Creating $\phi_{0,t}$:

```
Given: $v_n, n = 0 \ldots t$
$\phi_{0,0} = x$
for $i = 1 \ldots t$ do
    $\phi_{0,i}(x) = (x + v_{i-1}(x)) \circ \phi_{0,i-1}(x) = \phi_{0,i-1}(x) + v_{i-1}(\phi_{0,i-1}(x))$
end for
```

Creating $\phi_{t,0}$:

```
Given: $v_n, n = 0 \ldots t$
$\phi_{0,0} = x$
for $i = 1 \ldots t$ do
    $\phi_{i,0}(x) = \phi_{i-1,0}(x - v_{i-1}(x))$
end for
```

Creating $\phi_{t,T}$

```
Given: $v_n, n = t \ldots T$
$\phi_{T,T} = x$
for $i = T-1 \ldots t$ do
    $\phi_{i,T}(x) = \phi_{i+1,T}(x + v_i(x))$
end for
```

Creating $\phi_{T,t}$:

```
Given: $v_n, n = t \ldots T$
$\phi_{T,T} = x$
for $i = T-1 \ldots t$ do
    $\phi_{T,i}(x) = (x - v_i(x)) \circ \phi_{T,i+1}(x) = \phi_{T,i+1} - v_i(\phi_{T,i+1})$
end for
```

A.2 Jacobian Determinant Calculations

A note on the calculation of $|\mathrm{D}\phi_{i,0}|$:
While we represent $\phi_{t,0}$ as a vector field mapping points from timepoint t to timepoint 0, the path of a given point traced out from timepoint t to timepoint 0 is actually nonlinear and given by
...
As noted above $\phi_{t,0} =$, we can compute a better approximation of the determinant of the jacobian of $\phi_{t,0}$ as the product of the determinant of the jacobian

of the individual velocity steps:

$$\begin{aligned}
|\mathrm{D}\phi_{t,0}| &= |\mathrm{D}(\phi_{t-1,0} \circ (x - v_{t-1}))| \\
&= |(\mathrm{D}(\phi_{t-1,0}) \circ (x - v_{t-1}))\mathrm{D}(x - v_{t-1})| \\
&= |(\mathrm{D}\phi_{t-1,0}) \circ (x - v_{t-1})||\mathrm{D}(x - v_{t-1})| \\
&= |\mathrm{D}\phi_{t-1,0}| \circ (x - v_{t-1})|\mathrm{D}(x - v_{t-1})|
\end{aligned} \tag{A.6}$$

In other words, at each timestep the determinant of the jacobian of the deformation is the previous step's value deformed by the current deformation step, scaled by the determinant of the jacobian of the deformation step.
Similarly, it is easy to come up with an iterative approximation for $|\mathrm{D}\phi_{0,t}|$:

$$\begin{aligned}
|\mathrm{D}\phi_{0,t}| &= |\mathrm{D}(x + v_{t-1}) \circ \phi_{0,t-1}| \\
&= |\mathrm{D}\phi_{0,t-1}||\mathrm{D}(x + v_{t-1})| \circ \phi_{0,t-1}
\end{aligned} \tag{A.7}$$

A.3 Inverse Calculations

Given only a deformation field $\phi(x)$, finding the inverse $\phi^{-1}(x)$ can be an expensive operation. If only the inverse at a specific location is required (finding $\hat{x} = \phi^{-1}(x)$), one method is to first find the nearest grid location $\tilde{x}$ such that $\phi(\tilde{x}) \approx x$. Using the first-order taylor expansion of $\phi(\tilde{x} + \delta)$:

$$\phi(\tilde{x} + \delta) \approx \phi(\tilde{x}) + \delta D\phi(\tilde{x}) \tag{A.8}$$

we can improve the approximation for solving for the δ that gives us the best approximation of $\hat{x}$:

$$\delta = \big(D\phi(\tilde{x})\big)^{-1}\big(x - \phi(\tilde{x})\big) \tag{A.9}$$

And our new approximation is:

$$\phi^{-1}(x) = \tilde{x} + \delta \tag{A.10}$$

Finding $\phi^{-1}(x)$ given a series of velocity fields is easier. First, let us look at a small deformation $h(x) = x + v(x)$, and look for its inverse:

$$\begin{aligned}
h(x) &= x + v(x) \\
h^{-1}(x + v(x)) &= x \\
h^{-1}(x + v(x)) &\approx h^{-1}(x) + v(x) \cdot Dh^{-1}(x) \\
h^{-1} &\approx x - v(x) \cdot Dh^{-1}(x)
\end{aligned}$$

where the third step approximates the function by its first-order Taylor expansion. Note that if $Dh^{-1}(x) \approx I$, this is simply $h^{-1} \approx x - v(x)$, which is a reasonable assumption for a small deformation. Therefore, we can compose the inverted deformations to create the total deformation:

```
Given: $v_n, n = 0 \dots N-1$
$\phi_0^{-1} = x$
**for** $i = 0 \dots N-1$ **do**
   $\phi_{i+1}^{-1}(x) = \phi_i^{-1}(x - v_i(x))$
**end for**
```

Appendix B

LDDMM

For a detailed description of LDDMM see [2]. LDDMM seeks to minimize the objective function:

$$\hat{v} = \underset{v: \frac{d}{dt}\phi_t = v_t \circ \phi_t}{\operatorname{argmin}} \left(\int_0^1 \|v_t\|_V^2 dt + \lambda \|I_0 \circ \phi_1^{-1} - I_1\|_{L^2}^2 \right) \tag{B.1}$$

B.0.1 Greedy

By holding the current deformation fixed in Equation B.1 and solving for the negative gradient direction, we come up with the current best direction in which to change the vector field to minimize the energy functional. By taking a small step in this direction and recalculating, we achieve a 'greedy' iterative solution.

$$v_t = \mathbf{K}\Big(\lambda (I_0 \circ \phi_{t,0} - I_1) \Big) \tag{B.2}$$

$$\phi_{t+1} = \phi_t(x + \Delta v_t) \tag{B.3}$$

B.0.2 Relaxation

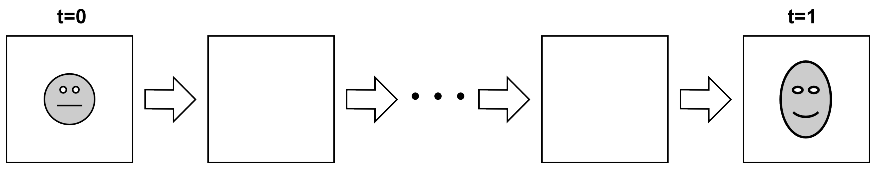

Figure B.1: LDDMM Relaxation

From Equation B.1 we can compute the variation of the LDDMM energy w.r.t. the velocity field at time t:

$$\nabla_v E_t = 2v_t - \mathbf{K}\Big(\frac{2}{\sigma^2}|\mathrm{D}\phi_{t,1}|\nabla \mathrm{J}_t^0[\mathrm{J}_t^0 - \mathrm{J}_t^1]\Big) \tag{B.4}$$

However, this requires computation of $|\mathrm{D}\phi_{t,1}|$, the jacobian determinant of $\phi_{t,1}$. This can become unstable as a numerical operation computed via finite differences. Instead, let us rewrite the image difference term:

$$\|I_0 \circ \phi_1^{-1} - I_1\|_{L^2}^2 \tag{B.5}$$

in terms of the dot product norm and our interpolation operator $\mathcal{T}$:

$$< \mathcal{T}_{1,0}I_0 - I_1,\ \mathcal{T}_{1,0}I_0 - I_1 > \tag{B.6}$$
$$=< \mathcal{T}_{1,0}I_0,\ \mathcal{T}_{1,0}I_0 > -2 < \mathcal{T}_{1,0}I_0,\ I_1 > + < I_1,\ I_1 > \tag{B.7}$$

Splitting the deformation at time t and using the adjoint we have:

$$=< \mathcal{T}_{1,t}\mathcal{T}_{t,0}I_0,\ \mathcal{T}_{1,t}\mathcal{T}_{t,0}I_0 > -2 < \mathcal{T}_{1,t}\mathcal{T}_{t,0}I_0,\ I_1 > + < I_1,\ I_1 > \tag{B.8}$$
$$=< \mathcal{T}_{t,0}I_0,\ \mathcal{T}_{1,t}^T\mathcal{T}_{1,t}\mathcal{T}_{t,0}I_0 > -2 < \mathcal{T}_{t,0}I_0,\ \mathcal{T}_{1,t}^T I_1 > + < I_1,\ I_1 > \tag{B.9}$$

Now, taking the variation w.r.t. v_t, we have:

$$2\nabla(\mathcal{T}_{t,0}I_0) \cdot \mathcal{T}_{1,t}^T\mathcal{T}_{1,t}\mathcal{T}_{t,0}I_0 - 2\nabla(\mathcal{T}_{t,0}I_0)\mathcal{T}_{1,t}^T I_1 \tag{B.10}$$
$$= 2\nabla(\mathcal{T}_{t,0}I_0)\mathcal{T}_{1,t}^T(\mathcal{T}_{1,0}I_0 - I_1) \tag{B.11}$$

Note on taking the variation:
To take the variation of $< \mathrm{A}x,\ \mathrm{A}x >$ w.r.t. x, the general product rule is $(\mathrm{A}x)'\mathrm{A}x{+}\mathrm{A}x(\mathrm{A}x)' = 2\mathrm{A}x(\mathrm{A}x)'$. However, since $< \mathrm{A}x,\ \mathrm{A}x >=< x,\ \mathrm{A}^T\mathrm{A}x >=< \mathrm{A}^T\mathrm{A}x,\ x >$, we can take $< \mathrm{A}x,\ \mathrm{A}x >' = x' \cdot \mathrm{A}^T\mathrm{A}x + \mathrm{A}^T\mathrm{A}x \cdot x' = 2x'\mathrm{A}^T\mathrm{A}x$.
We can see that equation B.11 is quite intuitive – we are simply taking the intensity difference at the final timepoint, and splatting it back to time t where it is multiplied by the gradient of the deformed image. From this, the full gradient becomes:

$$\nabla_v E_t = 2v_t - \mathbf{K}\Big(\frac{2}{\sigma^2}\nabla \mathrm{J}_t^0 \cdot \mathcal{T}_{1,t}^T(\mathrm{J}_1^0 - I_1)\Big) \tag{B.12}$$

Algorithm 1 LDDMM Relaxation Equation

```
Given: $I^0$, $I^T$,$\Delta$,$\lambda$
Allocated: $\mathrm{J}_t^0 t = 1 \ldots T$, $I^{\mathrm{diff}}$, $h$
{deformation}
$v_t$ = zeros, $t = 0 \ldots T - 1$
{scratch vars}
for # of iterations do
  {compute forward-deformed images}
  $h$ = Identity
  for $t = 1$ to $T - 1$ do
    $h = \phi_{t-1,0} \circ (x - v_{t-1})$
    $\mathrm{J}_{t+1}^0 = I^0 \circ h$
  end for
  {compute difference at final timepoint}
  $I^{\mathrm{diff}} = \mathrm{J}_T^0 - I^T$
  {compute update}
  $h$ = Identity
  for $t = T - 1$ to $0$ do
    $h = h - v_t \circ h$
    $\acute{v}_t = \lambda \mathbf{K} \big( \nabla \mathrm{J}_t^0 \cdot \mathcal{T}_h^T I^{\mathrm{diff}} \big)$
    $v_t = v_t - 2\Delta \big( v_t - \acute{v}_t \big)$
  end for
end for
```

Appendix C

LDDMM Geodesic Shooting

From [4] we have the LDDMM geodesic evolution equations for images:

$$v_t = -\mathbf{K}[\alpha_t \nabla I_t] \tag{C.1}$$

$$\frac{dI_t}{dt} + \nabla I_t^T v_t = 0 \tag{C.2}$$

$$\frac{d\alpha_t}{dt} + \nabla \cdot (\alpha_t v_t) = 0 \tag{C.3}$$

A simple explanation of geodesic shooting can be found in [1].
We note that at convergence, equation B.4 gives that $\nabla_v E_t = 0$, and therefore:

$$2v_t = \mathbf{K}\Big(\frac{2}{\sigma^2}|\mathrm{D}\phi_{t,1}|\nabla \mathrm{J}_t^0[\mathrm{J}_t^0 - \mathrm{J}_t^1]\Big) \tag{C.4}$$

$$\mathbf{L}v_t = \frac{1}{\sigma^2}|\mathrm{D}\phi_{t,1}|\nabla \mathrm{J}_t^0[\mathrm{J}_t^0 - \mathrm{J}_t^1] \tag{C.5}$$

A thorough discussion of geodesic shooting is given in [3]. From this we know that the quantity $\mathbf{L}v_t$, which we will refer to as the 'momentum', or a_t, is conserved throughout a 'geodesic' deformation.
Looking at equation C.5, we see that the momentum can be written:

$$\begin{aligned} a_t &= \mathbf{L}v_t \\ &= \frac{1}{\sigma^2}|\mathrm{D}\phi_{t,1}|\nabla \mathrm{J}_t^0[\mathrm{J}_t^0 - \mathrm{J}_t^1] \\ &= \alpha_t \nabla \mathrm{J}_t^0 \end{aligned} \tag{C.6}$$

where $\alpha_t = \frac{1}{\sigma^2}|\mathrm{D}\phi_{t,1}|[\mathrm{J}_t^0 - \mathrm{J}_t^1]$. This means that the momentum at any timepoint is the gradient of the deformed image at that timepoint, scaled by the scalar field α_t.

It is shown in [3], α_t can be computed at any timepoint t as

$$\alpha_t = |\mathrm{D}\phi_{t,0}|\alpha_0 \circ \phi_{t,0} \tag{C.7}$$

Given an initial α_0 from LDDMM (using T discrete timesteps) and the initial image I^0, This gives us a simple algorithm for propogating the deformation along the geodesic, given in Algorithm 2

Algorithm 2 LDDMM Geodesic Shooting Algorighnm

```
Given: $\alpha_0$, $I^0$
$\phi_{0,0}$ = Identity
$v_0 = \mathbf{K}(\alpha_0 \nabla I^0)$
for $t = 1$ to $T - 1$ do
    $\phi_{t,0} = \phi_{t-1,0} \circ (x - v_{t-1})$
    $\alpha_t = |\mathrm{D}\phi_{t,0}| \cdot (\alpha_0 \circ \phi_{t,0})$
    $\nabla \mathrm{J}^0_t = \nabla(I^0 \circ \phi_{t,0})$
    $v_t = \mathbf{K}(\alpha_t \nabla \mathrm{J}^0_t)$
end for
```

See Section A.2
We also note that α_{i+1} can be written in terms of α_i. From Equation A.6, and defining $h_i = x - v_i$:

$$\begin{aligned}
\alpha_{i+1} &= |\mathrm{D}h_i||\mathrm{D}\phi_{i,0}| \circ h_i\alpha_0 \circ \phi_i \\
&= |\mathrm{D}h_i||\mathrm{D}\phi_{i,0}| \circ h_i\alpha_0 \circ \phi_i \circ h_i \\
&= |\mathrm{D}h_i|\Big[|\mathrm{D}\phi_{i,0}|\alpha_0 \circ \phi_i\Big] \circ h_i \\
&= |\mathrm{D}h_i|\alpha_i \circ h_i
\end{aligned} \tag{C.8}$$

Similarly, we can compute $\nabla \mathrm{J}^0_i$ from $\nabla \mathrm{J}^0_{i-1}$:

$$\begin{aligned}
\nabla \mathrm{J}^0_{i+1} &= \nabla(I^0 \circ \phi_{i+1,0}) \\
&= \nabla(I^0 \circ \phi_{i,0} \circ h_i) \\
&= \nabla(\mathrm{J}^0_i \circ h_i) \\
&= (\mathrm{D}h_i)^T \nabla \mathrm{J}^0_i \circ h_i
\end{aligned} \tag{C.9}$$

Since from Equation C.6 we have:

$$v_i = \mathbf{K}(\alpha_i \nabla \mathrm{J}^0_i)$$

To summarize, Equations C.9 and C.8 give us update equations for α_i and $\nabla \mathrm{J}^0_i$ that rely only on α_{i-1} and $\nabla \mathrm{J}^0_{i-1}$:

$$\alpha_{i+1} = |\mathrm{D}h_i|\alpha_i \circ h_i \tag{C.10}$$

$$\nabla \mathrm{J}^0_{i+1} = (\mathrm{D}h_i)^T \nabla \mathrm{J}^0_i \circ h_i \tag{C.11}$$

Where $h_i = x + \mathbf{K}(\alpha_i \nabla \mathrm{J}^0_i)$.

Appendix D

Optimization of α_0

In optimizing α_0 instead of the entire velocity field series, a major goal is to reduce memory usage by not keeping all velocity fields in memory, but recalculating them via the shooting equations as necessary. We will be starting from the update equation B.4. Applying the $\mathbf{L}$ operator to this equation, we have:

$$\mathbf{L}\nabla_v E_t = 2\mathbf{L}v_t - \frac{2}{\sigma^2}|\mathrm{D}\phi_{t,1}|[\mathrm{J}_t^0 - \mathrm{J}_t^1]\nabla \mathrm{J}_t^0 \tag{D.1}$$

We are now only intersted in updating at $t = 0$, Which gives us our update step:

$$\alpha_0^{k+1} = \alpha_0^k - \Delta[\alpha_0^k - \lambda|\mathrm{D}\phi_{0,1}|(I^0 - I^T \circ \phi_{0,1})] \tag{D.2}$$

In order to apply this update step, we must calculate $\phi_{0,1}$ and $|\mathrm{D}\phi_{0,1}|$. We do this by calculating the v_t fields via the shooting algorithm 2. As we successively calculate v_t, we update our necessary quantities as:

$$\begin{aligned}
\phi_{0,t+1} &= \phi_{0,t} + v_t \circ \phi_{0,t} && \text{(D.3)}\\
|\mathrm{D}\phi_{0,t+1}| &= |\mathrm{D}\phi_{0,t} + (\mathrm{D}v_t \circ \phi_{0,t}) \cdot \mathrm{D}\phi_{0,t}| && \text{(D.4)}\\
&= |I + \mathrm{D}v_t \circ \phi_{0,t}||\mathrm{D}\phi_{0,t}| && \text{(D.5)}
\end{aligned}$$

Note that since $I \circ \phi = I$, we can write the determinant equation as:

$$\begin{aligned}
|\mathrm{D}\phi_{0,t+1}| &= |I \circ \phi_{0,t} + \mathrm{D}v_t \circ \phi_{0,t}||\mathrm{D}\phi_{0,t}| && \text{(D.6)}\\
&= |(I + \mathrm{D}v_t) \circ \phi_{0,t}||\mathrm{D}\phi_{0,t}| && \text{(D.7)}\\
&= |(I + \mathrm{D}v_t)| \circ \phi_{0,t} \cdot |\mathrm{D}\phi_{0,t}| && \text{(D.8)}
\end{aligned}$$

The total algorithm for optimizing α_0 is given in Algorithm 3.

Algorithm 3 Optimization of α_0

```
Given: $I^0$, $I^T$,$\Delta$
$\alpha_0$ = zeros
for # of iterations do
    $\alpha_{t=0} = \alpha_0$
    $\phi_{0,t=0}$ = Identity
    $\phi_{t=0,0}$ = Identity
    $v_{t=0} = \mathbf{K}(\alpha_{t=0}\nabla I^0)$
    for $t = 1$ to $T - 1$ do
        {update component calculations}
        $\phi_{0,t} = \phi_{0,t-1} + v_{t-1} \circ \phi_{0,t-1}$
        $\alpha_t = \mathcal{T}^T_{\phi_{0,t}}\alpha_0$
        $\phi_{t,0} = \phi_{t-1,0} \circ (x - v_{t-1})$
        $\nabla \mathrm{J}^0_t = \nabla(I^0 \circ \phi_{t,0})$
        {shoot forward to get next $v_t$}
        $v_t = \mathbf{K}(\alpha_t \nabla \mathrm{J}^0_t)$
    end for
    {final update component calculations}
    $\phi_{t=T,0} = \phi_{t=T-1,0} \circ (x - v_{t=T-1})$
    {update $\alpha_0$}
    $\alpha_0 = \alpha_0 - \Delta\Big[\alpha_0 - \lambda\big(\mathcal{T}^T_{\phi_{t=T,0}}(I^0 \circ \phi_{t=T,0} - I^T)\big)\Big]$
end for
```

Appendix E

Atlas Generation

As discussed in Chapter 4, atlas generation attempts to find an image $\hat{I}$ which minimizes the sum of squared distances from the input images. The optimization, then, attempts to jointly estimate the average image $\hat{I}$, as well as the individual deformations $\hat{\phi}_i$

$$\hat{\phi}_i, \hat{I} = \underset{I,\phi_i}{\operatorname{argmin}} \sum_{i=1}^{N} \left(\int_0^1 \|v_t\|_V^2 dt + \lambda \|I \circ \phi_{1,0}^i - I^i\|_{L^2}^2 \right) \tag{E.1}$$

This is solved via an alternating algorithm which first holds the current estimation of the mean image fixed and updates the individual deformations, which are then updated independently via the relaxation equations (B.1) or momentum optimization, followed by holding these deformations fixed and estimating the mean image. Since the individual deformations are fixed during this step, the optimization becomes:

$$\hat{I} = \underset{I}{\operatorname{argmin}} \sum_{i=1}^{N} \left(\|I \circ \phi_{1,0}^i(x) - I^i(x)\|_{L^2}^2 \right) \tag{E.2}$$

Performing a change of variable $x = \phi_{0,1}(y)$, we have:

$$\hat{I} = \underset{I}{\operatorname{argmin}} \sum_{i=1}^{N} \left(\|I(y) - I^i \circ \phi_{0,1}^i(y)\|_{L^2}^2 \right) \tag{E.3}$$

Taking the gradient w.r.t. I in the original space and setting this equal to zero, we can solve directly for $\hat{I}$:

$$\hat{I} = \frac{1}{N} \frac{\sum_{i=1}^{N} I^i \circ \phi_{0,1}^i |\mathrm{D}\phi_{0,1}^i|}{\sum_{i=1}^{N} |\mathrm{D}\phi_{0,1}^i|} \tag{E.4}$$

E.1 Adjoint Shooting Optimization of α_0

Algorithm 4 Optimization of α_0

```
Given: $I^0$, $I^1$,$\epsilon$,$T$
$\alpha_0$ = zeros
for # of iterations do
  {Shoot forward to $I^T$}
  $v_0 = -\mathbf{K}(\alpha_0 \nabla I^0)$
  $\phi_{0,t=0}$ = Identity
  $\phi_{t=0,0}$ = Identity
  for $t = 1$ to $T$ do
    $|\mathrm{D}\phi_{t,0}| = |\mathrm{D}\phi_{t-1,0}| \circ (x - \delta_t \cdot v_{t-1})|\mathrm{D}(x - \delta_t \cdot v_{t-1})|$
    $|\mathrm{D}\phi_{0,t}| = |\mathrm{D}(x + \delta_t \cdot v_{t-1})| \circ \phi_{0,t-1}|\mathrm{D}\phi_{0,t-1}|$
    $\phi_{0,t} = \phi_{0,t-1} + \delta_t \cdot v_{t-1} \circ \phi_{0,t-1}$
    $\phi_{t,0} = \phi_{t-1,0}(x - \delta_t \cdot v_{t-1})$
    $I_t = I^0 \circ \phi_{t,0}$
    $\alpha_t = |\mathrm{D}\phi_{t,0}|\alpha_0 \circ \phi_{t,0}$
    if $t < T$ then
      $v_t = -\mathbf{K}(\alpha_t \nabla I_t)$
    end if
  end for
  {Integrate adjoint equations back}
  $\hat{I}_T = I^1 - I_T$
  $\hat{v}_T = \mathbf{K}[-\hat{I}_T \nabla I_T]$
  $\tilde{I}_T = |\mathrm{D}\phi_{0,T}|\hat{I}_T \circ \phi_{0,T}$
  $\tilde{\alpha}_T = 0$
  for $t = T - 1$ to $0$ do
    $\tilde{\alpha}_t = \tilde{\alpha}_{t+1} - (\nabla I_{t+1} \cdot \hat{v}_{t+1}) \circ \phi_{0,t+1}$
    if $t > 0$ then
      $\tilde{I}_t = \tilde{I}_{t+1} + |\mathrm{D}\phi_{0,t+1}| \, (\nabla \cdot (\alpha_{t+1}\hat{v}_{t+1})) \circ \phi_{0,t+1}$
      $\hat{\alpha}_t = \tilde{\alpha}_t \circ \phi_{t,0}$
      $\hat{I}_t = |\mathrm{D}\phi_{t,0}|\tilde{I}_t \circ \phi_{t,0}$
      $I_t = I^0 \circ \phi_{t,0}$
      $\alpha_t = |\mathrm{D}\phi_{t,0}|\alpha_0 \circ \phi_{t,0}$
      $\hat{v}_t = \mathbf{K}[\alpha_t \nabla \hat{\alpha}_t - \hat{I}_t \nabla I_t]$
    end if
  end for
  {update $\alpha_0$}
  $\alpha_0 = \alpha_0 - \epsilon \cdot (\nabla I_0 \cdot K(\alpha_0 \nabla I_0) - \tilde{\alpha}_0)$
end for
```

Bibliography

[1] Beg, M., and Khan, A. Computing an average anatomical atlas using lddmm and geodesic shooting. In *Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on* (April 2006), pp. 1116–1119.

[2] Beg, M. F., Miller, M. I., Trouv, A., and Younes, L. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *International Journal of Computer Vision 61*, 2 (2005), 139–157.

[3] Miller, M. I., Trouvé, A., and Younes, L. Geodesic shooting for computational anatomy. *J. Math. Imaging Vis. 24*, 2 (2006), 209–228.

[4] Younes, L., Arrate, F., and Miller, M. I. Evolutions equations in computational anatomy. *NeuroImage 45*, 1, Supplement 1 (2009), S40 – S50. Mathematics in Brain Imaging.