2011-07-13

3D Image Reconstruction and Level Set Methods

Spencer R. Patty
Brigham Young University - Provo

3D Image Reconstruction and Level Set Methods

Spencer Patty

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mathematics

Brigham Young University

August 2011

Abstract

3D Image Reconstruction and Level Set Methods

Spencer Patty

Department of Mathematics

Master of Science

We give a concise explication of the theory of level set methods for modeling motion of an interface as well as the numerical implementation of these methods. We then introduce the geometry of a camera and the mathematical models for 3D reconstruction with a few examples both simulated and from a real camera. We finally describe the model for 3D surface reconstruction from n-camera views using level set methods.

Keywords: Level Set Method, PDE, Differential Geometry, Computer Vision, 3D Reconstruction

Acknowledgments

Thanks to my wife, I love you so much! Thanks to my parents and grandparents for teaching me the value of hard work and learning. Thanks to my advisor for encouraging me to seek out and pursue something I could find passion in.

Contents

LIST OF FIGURES

Chapter 1. The Level Set Method

We give an introduction and derivation of the level set methods that will be used. While there are many papers and books on this subject, we have found that clear and concise introductions to the matter are in short supply, and so we proceed in an attempt to give the reader a clear entrance to the language and theory used in this subject.

1.1 Introduction

The level set method is concerned with the tracking of interfaces as they move according to some physical or theoretical phenomena. These interfaces are typically assumed to be closed, meaning there is an inside and an outside. While there are applications that don't require this, we will focus on those that do. This could be anything from tracking the edge of a forest fire to image segmentation or even the sloshing of water. The key difference between this set of models and other marker based models is that the level sets allow for simple topological changes in the object being studied, whereas the marker models have a very difficult time when a splicing or merging happens. For instance in a fire, there are often times when,

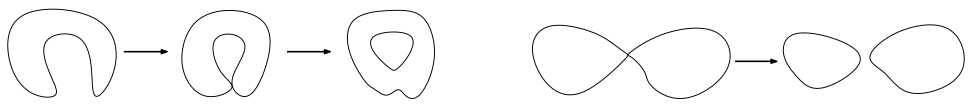

(a) Pinching Interface (b) Splitting Interface

Figure 1.1: These are a few of the topological changes that an interface may experience as it evolves. Figure 1.1(a) shows a pinching or merging of two sides with each other and Figure 1.1(b) shows the breaking apart or splitting of the interface. With marker methods these are nearly impossible to model, but the level set method handles any topological change naturally.

because of wind, the edge of the fire ends up enclosing a segment of the forest and we get an island of unburnt trees in the middle of the fire. With the level set methods this merger and enclosure can be described easily whereas with the standard markers it is unclear how one should separate and create two sets of connected markers. A marker method typically involves a parameterization along the edge of the interface and evenly spaced points along the parameterization, called markers, that are moved according to the phenomena being described. When one of these merging or splicing instances occur, it is rarely obvious how to join or split the two parameterizations into one. Hence it is difficult to deal with these topological changes with marker based methods. This is easily done, however, using level sets as seen in Figure 1.2.

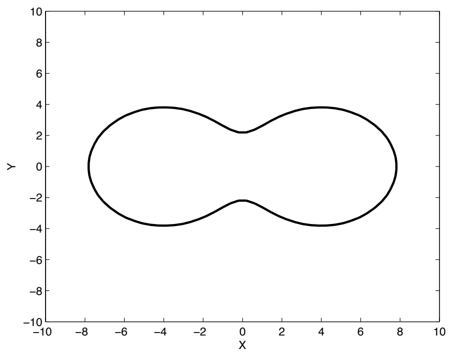
The level set methods also scale with size and dimension of the objects. The method works the same with two dimensional objects as objects in 4 or 5 dimensions. So once an algorithm is developed, it is straight forward to adapt it to higher dimensions of objects. Thus whether we are dealing with images as a 2D object or as projections of a 3D object we can consider them in similar manners and the algorithms will be easily adapted for each desired result. We now proceed to give a derivation of the level set equations and model. The numerical solution of these equations will be discussed in Chapter 2.

1.2 Derivation of Model

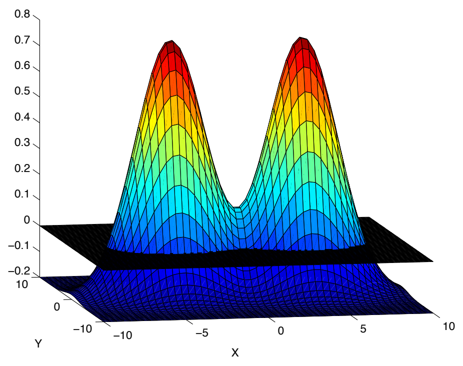
Given an initial closed $n-1$ dimensional hyper-surface Γ, we want to evolve it over time $\Gamma(t)$ as it propagates along its normal direction according to a speed function F. We thus embed Γ as the zero level set of a higher dimensional function ϕ. Let $\phi(\vec{x}, t=0)$, for $\vec{x} \in \mathbb{R}^n$, be defined by

$$\phi(\vec{x}, t=0) = \pm d$$

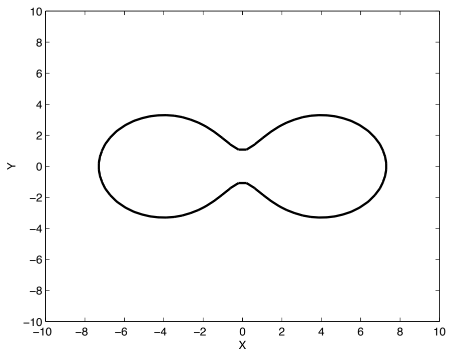
where d is the distance from $\vec{x}$ to $\Gamma(t=0)$, and the plus sign is chosen for $\vec{x}$ outside the initial hyper-surface and the minus sign for $\vec{x}$ inside the initial hyper-surface. We call this



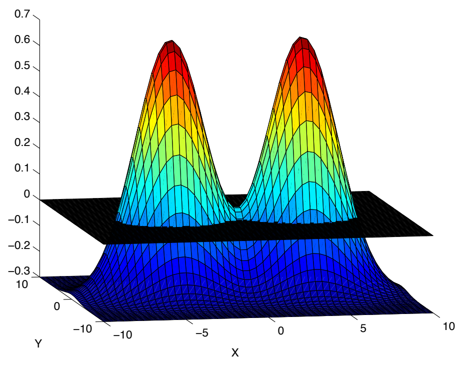
(a) initial interface



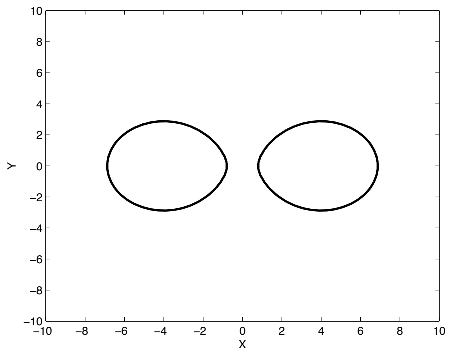
(b) initial surface



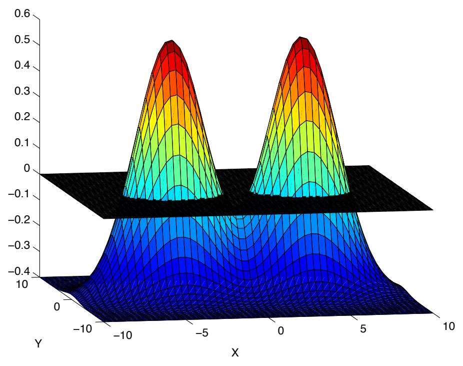
(c) splitting interface



(d) surface shifting down



(e) split interface



(f) final surface

Figure 1.2: The ease of using level sets to describe the splitting of a contour. We simulate the contour splitting by simply moving our surface up or down. For each step, we take the zero level set contour as our interface. Since our interface moving is only based on our surface moving, there is no difficulty with these types of topological changes in the level set method. As a side note, the surfaces represented here are not signed distance functions (the class of surfaces used in the level set method) but they do demonstrate the concept and ease of dealing with these types of topological changes with level sets.

a signed distance function and the benefits of choosing such a function will be discussed in Section 1.4. Thus we have a function $\phi(\vec{x}, t = 0) : \mathbb{R}^n \rightarrow \mathbb{R}$ such that

$$\Gamma(t = 0) = \{\vec{x} \mid \phi(\vec{x}, t = 0) \ = \ 0\}$$

and we want to produce an equation for evolving the function $\phi(\vec{x}, t)$ which will always contain $\Gamma(t)$ as the level set $\phi = 0$. In addition, we want the speed of motion in the normal direction to be F or in other words, for each $\vec{x}(t) \in \Gamma(t)$,

$$\vec{x}_t \cdot \vec{n} = F(\vec{x}(t))$$

where $\vec{n}$ is normal to the front at $\vec{x}(t)$ and $F(\vec{x}(t))$ is the speed function for that point.

We want the zero level set of ϕ to match our propagating hyper-surface, so we have the condition

$$\phi(\vec{x}(t), t) = 0, \quad \forall\, \vec{x}(t) \in \Gamma(t)$$

Thus

$$\frac{d}{dt}\phi(\vec{x}(t), t) = 0$$

so by chain rule,

$$\phi_t + \nabla\phi(\vec{x}(t), t) \cdot \vec{x}_t(t) = 0.$$

Now since $\vec{n} = \nabla\phi / |\nabla\phi|$, we have that $\nabla\phi = \vec{n}|\nabla\phi|$, so

$$\phi_t + \vec{n} \cdot \vec{x}_t(t)|\nabla\phi| = 0$$

or in other words

$$\phi_t + F|\nabla\phi| = 0,$$

which is our level set evolution equation as derived in [19] and [17].

1.3 Curvature of Curves and Surfaces

Before we continue, we must introduce the idea of the curvature of curves and surfaces as studied in differential geometry. There are many different metrics of curvature for different size spaces but since we are only concerned with two and three dimensional spaces, we will introduce the concepts of **curvature** κ of a curve in 2D, **Gaussian curvature** κ_G of a surface in 3D (often denoted K), and **mean curvature** κ_M of a surface in 3D (often denoted H). As a note, while we mention κ_G, it is not prevalent in our applications so we will focus most of our attention on the other two curvatures.

The idea of curvature is essentially to describe how much a curve or surface deviates from its tangent plane or how sensitive the tangent line is as we move along the curve or surface. Thus if the curve or surface is flat, we say the curvature is zero. Essentially, the higher the magnitude of the curvature at a point, the more quickly the curve or surface changes as you move away from that point.

1.3.1 Curvature in 2D. Curvature κ of a curve is most simply described as the inverse of the radius of curvature at that point, i.e. $\kappa = 1/R$, where the radius of curvature, R, is the radius of the best fitting circle (called the osculating circle) to the curve at that point. This is theoretically done as the limit of the radius of the circle that goes through the point of interest and one on either side as they slide towards the point where we are measuring the curvature. Hence it is the circle that best fits the curve at that point.

The curvature of a curve, $\phi(x, y) = c$, in 2D is calculated to be

$$\kappa = \frac{\phi_{xx}\phi_y^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}}$$

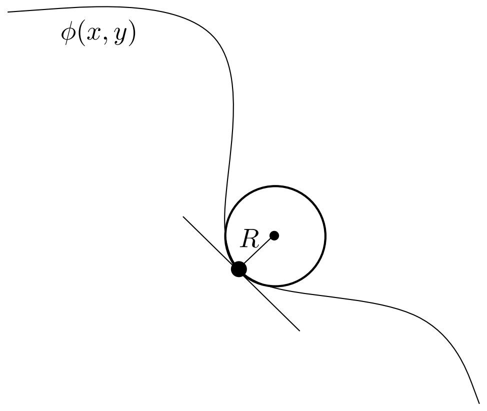


Figure 1.3: Curvature, $\kappa = 1/R$, is the inverse of the radius of the osculating circle at the point on the curve.

which has the simple representation of being the divergence of the unit normal vector,

$$\kappa = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right)$$

Example 1.1. The curvature of any straight line $\phi(x,y) = 0$ where $\phi(x,y) = y - mx - b$ is calculated by first finding the unit normal vector of $\phi(x,y)$.

$$\frac{\nabla \phi}{|\nabla \phi|} = \begin{bmatrix} -m/\sqrt{1+m^2} \\ 1/\sqrt{1+m^2} \end{bmatrix}.$$

Then the curvature is the divergence of this constant vector and so

$$\kappa = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} = 0.$$

This is exactly what we would expect since the line does not deviate from its tangent line at all.

Example 1.2. As a more interesting example, we find the curvature of the unit circle

$\phi(x, y) = 0$ where $\phi(x, y) = x^2 + y^2 - 1$. The unit normal is given by

$$\frac{\nabla\phi}{|\nabla\phi|} = \begin{bmatrix} 2x/\sqrt{4x^2 + 4y^2} \\ 2y/\sqrt{4x^2 + 4y^2} \end{bmatrix} = \begin{bmatrix} x/\sqrt{x^2 + y^2} \\ y/\sqrt{x^2 + y^2} \end{bmatrix}.$$

Thus we calculate the curvature as the divergence of this vector

$$\kappa = \nabla \cdot \begin{bmatrix} x/\sqrt{x^2 + y^2} \\ y/\sqrt{x^2 + y^2} \end{bmatrix} = \frac{x^2 + y^2}{(x^2 + y^2)^{3/2}} = \frac{1}{(x^2 + y^2)^{1/2}} = 1,$$

since $x^2 + y^2 = 1$. So the unit circle has constant curvature $\kappa = 1$. In fact for a circle of any radius, it is it's own osculating circle, hence the curvature is just the inverse of its radius. We see that smaller radius circles bend more quickly and so have higher curvature than larger radius circles.

1.3.2 Curvature in 3D. For surfaces embedded in 3D, there are two forms of curvature that arise. We briefly explain the idea of principal curvatures of a surface and then introduce the two forms of curvature. At any point on the surface, there are an infinite number of curves that pass through that point along the surface. The principal curvatures, κ_1 and κ_2, are respectively the maximal and minimal curvatures of the 2D curves passing through that point. See [2] for details.

Gaussian curvature κ_G gives us an idea of how locally convex (positive value) or locally saddle (negative value) the surface is at that point. It is calculated as the product of the principal curvatures, $\kappa_G = \kappa_1\kappa_2$. Gaussian curvature is more theoretically appealing to differential geometers since it is always the same value for a surface no matter what space the surface is embedded in, but is not as useful to us as the mean curvature is. Some applications using level set methods call for the use of Gaussian curvature, however most call for use of mean curvature. The Gaussian curvature of a surface, $\phi(x, y, z) = c$, embedded in 3D space is somewhat tedious to obtain and does not have a simple representation and so we give it as derived in [20]

$$\kappa_G = \frac{\begin{bmatrix} \phi_x^2(\phi_{yy}\phi_{zz} - \phi_{yz}^2) + \phi_y^2(\phi_{xx}\phi_{zz} - \phi_{xz}^2) + \\ \phi_z^2(\phi_{xx}\phi_{yy} - \phi_{xy}^2) + 2[\phi_x\phi_y(\phi_{xx}\phi_{yz} - \phi_{xy}\phi_{zz}) + \\ \phi_y\phi_z(\phi_{xy}\phi_{xz} - \phi_{yz}\phi_{xx}) + \phi_x\phi_z(\phi_{xy}\phi_{yz} - \phi_{xz}\phi_{yy})] \end{bmatrix}}{(\phi_x^2 + \phi_y^2 + \phi_z^2)^2}.$$

Mean curvature, κ_M, is defined as the average of the two principal curvatures, $\kappa_M = (\kappa_1 + \kappa_2)/2$. **Minimality** is defined on surfaces that have either a boundary at infinity or a closed boundary in 3D space (like a bent hanger). We say a surface is minimal if every other surface on that same boundary has more surface area than our surface. Mean curvature gives us an idea of how "close" to a minimal surface we are: meaning the smaller the magnitude of curvature, the more like a minimal surface our surface is.

Example 1.3. Some geometers play with soap film because the surfaces made by the soap film, for instance when beginning the process of making bubbles on a bent coat hanger, are minimal surfaces and have the property that mean curvature is zero. When you blow through the hanger on the film the curvature becomes nonzero but still constant. When the pressure becomes big enough, the film releases from the hanger and often a bubble is formed which has nonzero constant mean curvature.

We lastly note that mean curvature does depend on what space we are embedded in which is why it is not as theoretically appealing to geometers, but that very reason is why they will work so well in many applications for us seeing as we often want our interface to be a sort of minimal surface on our data. We caution that these next formulae are only valid when our surface is embedded in 3D space. Mean curvature of a surface $\phi(x, y, z) = c$ in 3D is calculated to be

$$\kappa_M = \frac{\begin{bmatrix} \phi_x^2\phi_{yy} - 2\phi_x\phi_y\phi_{xy} + \phi_y^2\phi_{xx} + \phi_x^2\phi_{xz} - \\ 2\phi_x\phi_z\phi_{xz} + \phi_z^2\phi_{xx} + \phi_y^2\phi_{zz} - 2\phi_y\phi_z\phi_{yz} + \phi_z^2\phi_{yy} \end{bmatrix}}{(\phi_x^2 + \phi_y^2 + \phi_z^2)^{3/2}},$$

which again is most easily represented as the divergence of our unit normal vector,

$$\kappa_m = \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|} \right).$$

Example 1.4. Given the conic surface $\sqrt{x^2 + y^2 - 1} = z$, we calculate the mean and Gaussian curvatures. We note that this surface is not differentiable at the origin and so curvature does not exist there, but everywhere else it does. As a note, this is not a problem for us as it is undefined on a zero measure set which will not affect our using the curvature calculated for any subsequent analysis. We simply give it an arbitrary value of zero and move on. We mention this because in Section 2.1 we will see that these areas of non differentiability, called the **skeleton** of the surface, occur frequently but do not affect any of our results if they occur on sets of measure zero. To proceed with calculating our mean curvature, we let $\phi(x, y, z) = \sqrt{x^2 + y^2} - 1 - z = 0$, from which we calculate the unit normal vector to be

$$\frac{\nabla\phi}{|\nabla\phi|} = \begin{bmatrix} x/\sqrt{2}\sqrt{x^2 + y^2} \\ y/\sqrt{2}\sqrt{x^2 + y^2} \\ -1/\sqrt{2} \end{bmatrix}.$$

Then our mean curvature is half the divergence of this vector and so

$$\begin{aligned}
\kappa_m &= \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|}\right) \\
&= \frac{1}{2^{1/2}}\left(\frac{1}{\sqrt{x^2+y^2}} - \frac{x}{2}(\sqrt{x^2+y^2})^{-3/2}(2x) + \frac{1}{\sqrt{x^2+y^2}} - \frac{y}{2}(\sqrt{x^2+y^2})^{-3/2}(2y)\right) \\
&= \frac{1}{2^{1/2}}\left(\frac{2}{\sqrt{x^2+y^2}} - \frac{x^2+y^2}{(x^2+y^2)^{3/2}}\right) \\
&= \frac{1}{2^{1/2}\sqrt{x^2+y^2}}
\end{aligned}$$

To calculate the Gaussian curvature, we use the same $\phi(x, y, z)$ but must calculate all the first and second derivatives. Since $\phi_{xz} = \phi_{yz} = \phi_{zz} = 0$ the formula reduces to

$$\kappa_G = \frac{(\phi_{xx}\phi_{yy} - \phi_{xy}^2)}{\phi_z^2|\nabla\phi|^4}$$

which evaluates to be equal to 0. So the cone is a surface with zero Gaussian curvature.

Example 1.5. As another example, we calculate the curvatures for the sphere. We would expect this to have constant Gaussian curvature and mean curvature. The surface is given by $\phi(x, y, z) = x^2 + y^2 + z^2 - a^2 = 0$. The unit normal is

$$\frac{\nabla\phi}{|\nabla\phi|} = \begin{bmatrix} x/\sqrt{x^2+y^2+z^2} \\ y/\sqrt{x^2+y^2+z^2} \\ z/\sqrt{x^2+y^2+z^2} \end{bmatrix}.$$

Hence our mean curvature is the divergence of the unit normal

$$\kappa_M = \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|}\right) = \frac{x^2+y^2+z^2}{(x^2+y^2+z^2)^{3/2}} = \frac{1}{\sqrt{(a^2)}} = \frac{1}{a}.$$

Our Gaussian curvature is calculated to be

$$\begin{aligned}
\kappa_G &= \frac{(2x)^2(4-0)+(2y)^2(4-0)+(2z)^2(4-0)+0}{\left((2x)^2+(2y)^2+(2z)^2\right)^2} \\
&= \frac{4^2(x^2+y^2+z^2)}{4^2(x^2+y^2+z^2)^2} \\
&= \frac{1}{x^2+y^2+z^2} \\
&= \frac{1}{a^2}
\end{aligned}$$

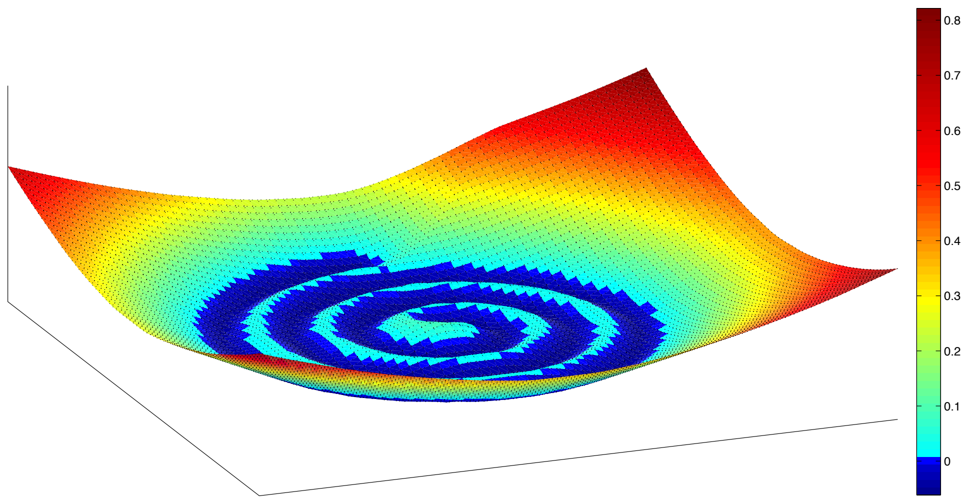
since $x^2+y^2+z^2=a^2$.

1.4 Signed Distance Functions

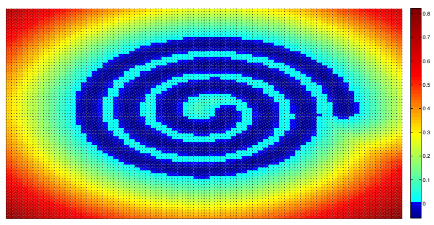
The reader will notice that in Section 1.2 we have embedded our desired interface into a one dimensional higher surface called a signed distance function as the zero level set. We will often give names to the inside, outside and boundary of the interface Γ. We typically call the boundary or interface $\partial\Omega$ the region inside or enclosed by our interface Ω^- and the region outside our interface Ω^+. Then our surface is described as

$$\phi(x) = \begin{cases} -d(x) & x \in \Gamma^- \\ 0 & x \in \partial\Gamma \\ d(x) & x \in \Gamma^+ \end{cases}$$

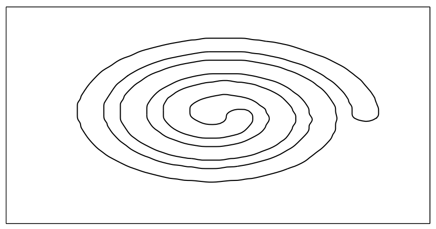
Doing so allows us to immediately know whether we are inside or outside of the interface by the sign of the function. If the value on our surface is negative, then we are inside the interface, otherwise we are outside of it. For example, we show the signed distance function of a coiled up snake contour in Figure 1.4. Note that the scale shows where it is inside (negative) and outside (positive) of the contour.



(a) side view of signed distance function



(b) top view of signed distance function



(c) snake contour interface

Figure 1.4: The signed distance function and zero level set for the snake contour interface. Note that negative values on the surface are inside the contour and positive values are outside. The height of the surface represents the minimal distance to an edge of the contour.

The height of the surface gives us the minimal distance from that point in space to the edge of our interface. Hence to find the point on the interface, $\mathbf{x}_I$, which is closest to our current position, x, we travel from our current location in the normal direction to the surface the distance of our height

$$x_I = x - \phi(x)\vec{n}.$$

Now, since we have the condition that $|\nabla\phi| = 1$,

$$\vec{n} = \frac{\nabla\phi}{|\nabla\phi|} = \frac{\nabla\phi}{1} = \nabla\phi.$$

Hence on our signed distance function, the closest point on the interface is given by

$$x_I = x - \phi(x)\nabla\phi.$$

Likewise, on our signed distance function, mean curvature is simplified greatly to be just the Laplacian of the surface

$$\kappa_M = \nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right) = \nabla\cdot\left(\frac{\nabla\phi}{1}\right) = \nabla\cdot\nabla\phi = \Delta\phi.$$

As a final note about signed distance functions. If we have two signed distance functions ϕ_1 and ϕ_2 for two distinct interfaces, it is simple to create the signed distance function for the union or intersection of those two interfaces.

$\phi(x) = \min(\phi_1(x), \phi_2(x))$ represents the distance function for the **union** $\Gamma_1^- \cup \Gamma_2^-$.

$\phi(x) = \max(\phi_1(x), \phi_2(x))$ represents the distance function for the **intersection** $\Gamma_1^- \cap \Gamma_2^-$.

$\phi(x) = \max(\phi_1(x), -\phi_2(x))$ represents the distance function for the **set subtraction** $\Gamma_1^- \backslash \Gamma_2^-$.

1.5 The Choice of Forcing Function

Now we return to finish the discussion of the use of level set models. As can be seen, the derivation and set of definitions is rather small and concise. The real innovation and challenge in using this model is coming up with a function F, which will cause the surface or interface to evolve in the desired manner. Often this is given by the physics of the system under observation, but many times especially with the applications in image processing, it is not clear what the forcing function should be. There have been many applications for smoothing that suggest the use of mean curvature based functions. For instance, if we wish the interface to move normal to itself relative to its curvature, we might use the function,

$$F(\mathbf{x}) = 1 - \varepsilon\kappa_M(\mathbf{x}).$$

where $\varepsilon > 0$ is a scaling constant for how fast to move by curvature. Common values are $\varepsilon = .1, .01,$ and $.001$. For the snake interface, Figure 1.4, evolution under the above function will cause the interface to unwind itself and form into a circle (a shape of constant mean curvature) which then will continue to decrease in size until it disappears entirely as seen in Figure 1.5.

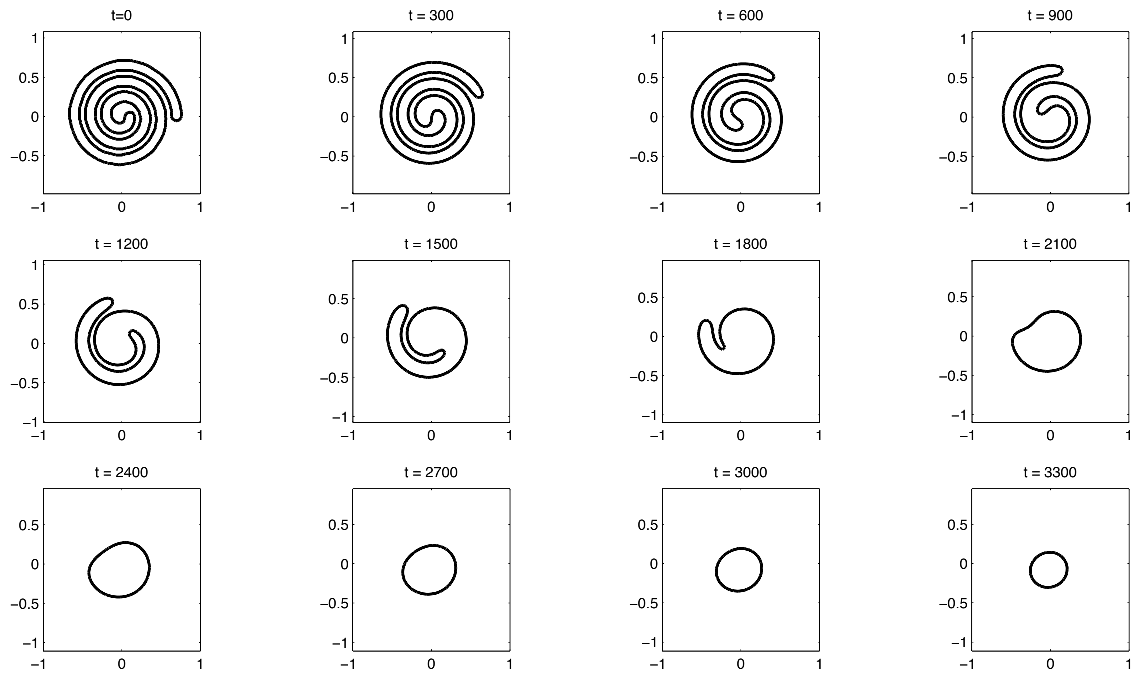


Figure 1.5: The evolution of the snake contour, Figure 1.4, under mean curvature, $F(\mathbf{x}) = 1 - \varepsilon\kappa_M(\mathbf{x})$, where $\varepsilon > 0$ is a small scaling constant.

Many other papers have used variational or energy minimization techniques. Sethian's book [20] discusses the techniques for the fast construction of extension velocities where the forcing function is only given or really defined on the interface. In Chapter 4, we will introduce a forcing function that will cause our surface to evolve for 3D surface reconstruction from multiple camera views of a scene.

Chapter 2. Numerical Schemes for Solving the Level Set Equation

When solving the level set equation, we often deal with a moving front that behaves like a wave. Hence, we use many techniques that were developed for hyperbolic equations. We make the choice of using upwind methods whenever there is a motion that behaves like propagating waves. We will first define the unit normal vector and then will develop a scheme for the gradient and mean curvature. Finally we will combine everything into a scheme that represents the general motions of advection, motion in normal direction and motion by curvature

$$\phi_t + U \cdot \nabla\phi + F_0|\nabla\phi| = \epsilon\kappa|\nabla\phi|,$$

where $\epsilon > 0$ is a proportionality constant, $U = (u, v, w)$ is the advection vector field and F_0 is the speed of motion in the outer normal direction.

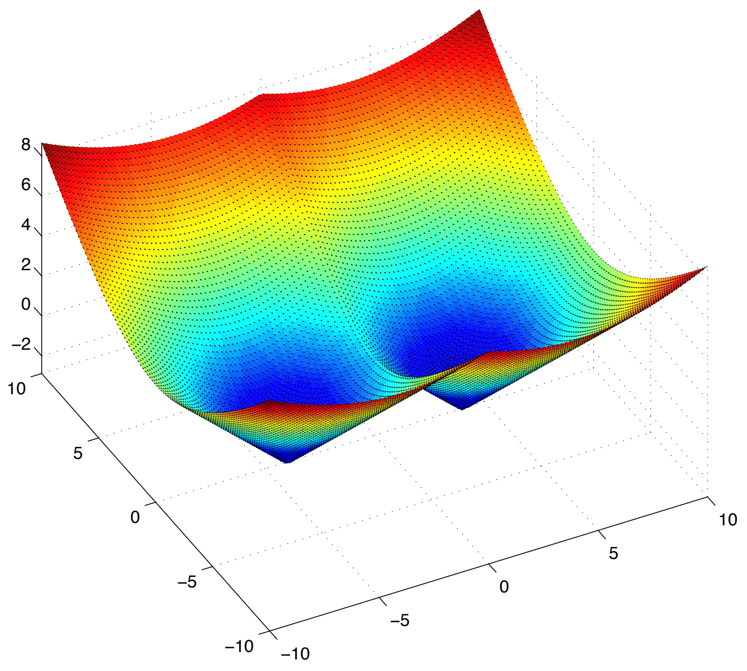
2.1 The Unit Normal

To begin with, we note that our ϕ at any time could have places where the gradient is not even technically defined. This can develop on the **skeleton** from using the choice of signed distance function where a point is equidistant from two or more places on the initial front.

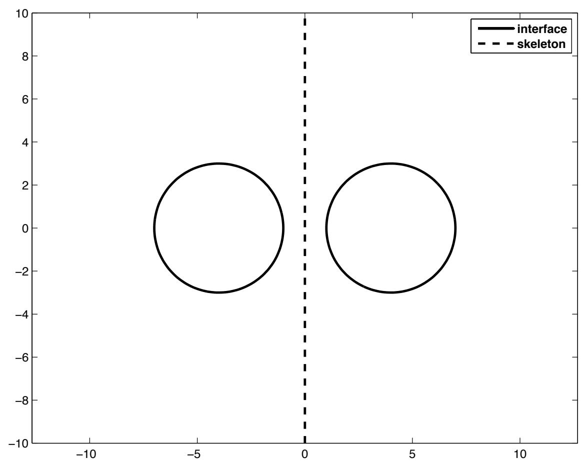
The unit normal is defined as

$$\vec{n} = \frac{\nabla\phi}{|\nabla\phi|}.$$

so we must come up with a standard logical way to deal with this possible nonexistence. Now in reality, these kinks smooth themselves out over time just by doing finite differencing, but we still need our unit normal to handle any possible problems. In two dimensions, we will



(a) the skeleton is where the surface is not differentiable



(b) skeleton with the interface.

Figure 2.1: A skeleton is formed when a point is equidistant from more than one location on the interface. The surface is continuous but not differentiable. This is a small zero measure set and does not affect our numerical stability at all.

take the unit normal to be in the direction of the average of the four one-sided derivatives in x and y

$$\vec{n}^* = \frac{(D_x^+, D_y^+)}{\left((D_x^+)^2 + (D_y^+)^2\right)^{0.5}} + \frac{(D_x^-, D_y^+)}{\left((D_x^-)^2 + (D_y^+)^2\right)^{0.5}} + \frac{(D_x^+, D_y^-)}{\left((D_x^+)^2 + (D_y^-)^2\right)^{0.5}} + \frac{(D_x^-, D_y^-)}{\left((D_x^-)^2 + (D_y^-)^2\right)^{0.5}},$$

which we then normalize,

$$\vec{n} = \frac{\vec{n}^*}{|\vec{n}^*|}.$$

Likewise in 3 dimensions, it is the average of the 8 one sided derivatives in x, y and z. This has the property that it always points where we would think to place it if we had to fill it in. This choice of unit normal deals well with the kinks and gives us the correct result where ϕ is actually differentiable.

2.2 Mean Curvature

We recall that mean curvature is the divergence of the unit normal vector

$$\kappa_M = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right),$$

and notice that mean curvature is a nonlinear diffusion process and so information propagates in all directions. This is not a hyperbolic motion but instead a diffusive motion. Hence, we can use a centered difference for all the first order partial derivatives instead of needing an upwind scheme when calculating curvature. Thus, motion by mean curvature which looks like

$$\phi_t = \varepsilon \kappa |\nabla \phi|$$

is approximated by

$$\frac{\phi_{ijk}^{n+1} - \phi_{ijk}^n}{\Delta t} = \epsilon K_{ijk} \left((D_{ijk}^{ox})^2 + (D_{ijk}^{oy})^2 + (D_{ijk}^{oz})^2 \right)^{1/2},$$

where K_{ijk} is the centered difference approximation to mean curvature at $\phi(x_i, y_j, z_k)$ and the derivative operators are as defined in subsection A.4.1. This leads to a simple scheme

$$\phi_{ijk}^{n+1} = \phi_{ijk}^{n} - \Delta t \left(\epsilon K_{ijk} \left((D_{ijk}^{ox})^2 + (D_{ijk}^{oy})^2 + (D_{ijk}^{oz})^2 \right)^{1/2} \right).$$

2.3 The Normed Gradient

We implement an upwind scheme when numerically solving for the norm of the gradient. When we are dealing with the motion in the normal direction, we want the gradient to reflect information flowing from upstream. We will use the max and min switches as discussed in subsection A.4.1. We define the normed gradients to be

$$\begin{aligned} \nabla^+ &= [\max(D_{ikj}^{-x}, 0)^2 + \min(D_{ijk}^{+x}, 0)^2 \\ &\quad \max(D_{ikj}^{-y}, 0)^2 + \min(D_{ijk}^{+y}, 0)^2 \\ &\quad \max(D_{ikj}^{-z}, 0)^2 + \min(D_{ijk}^{+z}, 0)^2]^{1/2}, \end{aligned}$$

$$\begin{aligned} \nabla^- &= [\max(D_{ikj}^{+x}, 0)^2 + \min(D_{ijk}^{-x}, 0)^2 \\ &\quad \max(D_{ikj}^{+y}, 0)^2 + \min(D_{ijk}^{-y}, 0)^2 \\ &\quad \max(D_{ikj}^{+z}, 0)^2 + \min(D_{ijk}^{-z}, 0)^2]^{1/2}, \end{aligned}$$

then to make a pass at simple motion in normal direction, $\phi_t = -F|\nabla \phi|$, we approximate with a forward difference in time and use the switch

$$\frac{\phi_{ijk}^{n+1} - \phi_{ijk}^{n}}{\Delta t} = - \left[\max(F_{ijk}, 0)\nabla^+ + \min(F_{ijk}, 0)\nabla^- \right]$$

which yields a simple first order approximation

$$\phi_{ijk}^{n+1} = \phi_{ijk}^{n} - \Delta t \left[\max(F_{ijk}, 0)\nabla^{+} + \min(F_{ijk}, 0)\nabla^{-}\right],$$

2.4 Combinations of Simple Motions

We have now discussed the different types of motions that could occur and we want to put them all together using a forcing function

$$F = F_{\text{prop}} + F_{\text{curv}} + F_{\text{adv}},$$

where

$F_{\text{prop}} = F_0$ is the propagation expansion speed in the outer normal direction

$F_{\text{curv}} = -\epsilon\kappa$ is the dependence of speed in normal direction on curvature

$F_{\text{adv}} = \vec{U}(\vec{x}, t) \cdot \vec{n}$ is the advection speed and $\vec{U}(\vec{x}, t)$ is the vector field.

Then with this forcing function F, we get

$$\phi_t + F|\nabla\phi| = 0$$

which simplifies to

$$\phi_t + F_0|\nabla\phi| + \vec{U} \cdot \nabla\phi = b\kappa|\nabla\phi|,$$

and we use (in 2D) the scheme

$$\phi_{ij}^{n+1} = \phi_{ij}^n + \Delta t \left[\begin{array}{l} -\left[\max((F_0)_{ij},0)\nabla^+ + \min((F_0)_{ij},0)\nabla^-\right] \\ - \left[\begin{array}{l} \max(u_{ij}^n,0)D_{ij}^{-x} + \min(u_{ij}^n,0)D_{ij}^{+x} \\ + \max(v_{ij}^n,0)D_{ij}^{-y} + \min(v_{ij}^n,0)D_{ij}^{+y} \end{array} \right] \\ + \epsilon K_{ij}^n \left((D_{ij}^{ox})^2 + (D_{ij}^{oy})^2\right)^{1/2} \end{array} \right],$$

where $\vec{U} = (u, v)$ and K_{ij}^n is the centered difference approximated curvature at ϕ_{ij}^n.

2.5 The Re-initialization Equation

We have talked about the evolution of the equation by level set method, but despite our best efforts the surface ϕ often drifts away from being a signed distance function. There are a number of fixes to this including reinitializing the surface or designing a forcing function, which keeps the level curves a constant distance from each other off the important zero level set. We will discuss here the technique for reinitializing the surface ϕ to be a signed distance function. This technique can be done every couple steps of standard evolution to reset the shape of the surface around the current zero level set, however in practice doing this can lead to a shifting of the actual front. Also if we reinitialize too often, we can bring in other errors and slow down our program too much. Hence a balance should be struck between actual evolution and reinitialization of the surface shape.

With this in mind, we observe that the signed distance function is the steady state solution ($\tau \to \infty$) of the initial value problem:

$$\text{IVP} \quad \begin{cases} \frac{\partial\psi}{\partial\tau} + \text{sign}(\phi)(|\nabla\psi| - 1) = 0 \\ \psi(\vec{x}, 0) = \phi(\vec{x}, t) \end{cases}, \tag{2.1}$$

where

$$\text{sign}(\phi) = \begin{cases} 1 & \text{in } \Omega^+ \\ 0 & \text{on } \partial\Omega \\ -1 & \text{in } \Omega^- \end{cases} \tag{2.2}$$

The equation

$$\phi_t + \text{sign}(\phi_o)(|\nabla\phi| - 1) = 0,$$

where $\phi = \phi_o$ to begin with, is called the **re-initialization equation**. The number of iterations that it is run dictates how far from the surface it is reinitialized. Each iteration corrects another $|\Delta x|$ distance from the boundary to be the distance function with the sign for inside and outside. This technique is described in full generality in [16].

A practical rule says that accuracy is required only near the initial font itself. A discrete value based on grid distances can suffice far away from the boundary. We caution that overuse of the re-initialization technique can cause the interface boundary itself to shift to nearby grid points. Likewise it takes a great deal of time to reshape the entire surface. It can be a valuable tool especially for creating our initial signed distance function but it can be overused.

2.6 Efficient Schemes for Level Set Methods

The above scheme is the most simple approximation of the general level set equation. For different applications, it might be necessary to increase the accuracy of each finite difference, or you could change from a forward time derivative to using a Runge-Kutta scheme. Likewise, many other methods of solving this equation have been explored and published which are much more efficient and accurate. However the understanding that comes from the above analysis and model is invaluable and the other schemes are in most cases just more complicated versions of the choices we made above. Sethian formulates a finite element triangulated version of the above in his book *Level Set Methods and Fast Marching Methods* [20]. **The Fast Marching method** is likewise implemented and then made efficient using an upwind scheme combined with a heap sort algorithm. The actual implementation of a fast efficient level set method is beyond the scope of this paper, but the ideas follow the same patterns as described above. With the scheme that was displayed above, we can construct a simple level set equation solver which handles easily basic cases, but the adoption of professional software is required for many applications if we want to be more accurate and more efficient. In fact, in many applications we only care about what is happening near our interface and so there have been some great advances in data structures that save time and space. The **narrow band level set method** utilizes the min-heap sort data structure and only calculates the signed distance function on a narrow band around the edge of the interface. Sethian describes this narrow band in full in Chapter 8 of [20] as an efficient implementation. Another data structure of note is the **octree** introduced by Strain in [21]. It is an adaptive method which allows for quick accurate calculations of interface movements. In C++, one of the best libraries goes by the name of DT-Grids and is introduced in [13].

Ian Mitchell at the University of British Colombia has developed a matlab toolbox for level set methods which is very usable and is set up in a general framework so as to be able to implement different applications easily. There are likewise many C++ software libraries which implement different data structures specifically designed and optimized for use in level set frameworks. These can be rather useful and when implementing a specific application, it is a good idea to start with one of these toolboxes or libraries and then add the necessary methods to the already optimized software.

Chapter 3. The Language and Geometry of Cameras.

3.1 Introduction

We will first introduce the mathematical models that simulate the actions of a camera. Once we understand how a camera can be represented, we will proceed to discuss the traditional techniques of combining multiple views of a stationary scene through various cameras and reproducing the 3D scene. This is often called 3D reconstruction from multiple images. To begin with, we must introduce the mathematical structures that will allow us to model cameras.

3.2 Homogeneous Coordinates or Projective Coordinates

The first idea to be introduced is how we will represent points in space. There are many coordinate systems that could be used. When we talk about a point in space, we typically think of its (x, y, z) coordinates. The set of such points is called **Euclidean space**. Geometry as taught in high school holds here as we can discuss distances between things, angles between intersecting lines and so forth. However, this is not the only way nor always the best way to represent the location of our points in space for our class of problems. In fact because of the type of operations we will be using to describe a camera, we will subsequently introduce what is called the **projective coordinate system** or **homogeneous coordinate system** as a better representation of our points in space.

Many of our operations are called **rigid body transformations** which essentially preserve all the relative geometries of the object or scene being acted upon. For instance if there is a wooden block on the table and we pick it up and toss it around and then set it down on a chair, we have done a rigid body transformation to the block. It has been moved and maybe rotated but the corners are still the same distance apart and the block has not changed shape at all. Hence the name rigid body. This type of transformation occurs when modeling cameras and in Euclidean space does not have a linear (matrix) representation. However in our projective coordinate system it does have a linear (matrix) representation. We choose the projective coordinate system because many of our operations which are non-linear in Euclidean space have linear representations in the projective space and so make life easier. We introduce a few of these operations in full detail.

An **affine transformation** is any operation that involves a scaling, rotation, and/or translation of the object being operated on. For a given vector $\mathbf{x} \in \mathbb{R}^n$, the rotational and scaling part is described by a matrix $R \in \mathcal{M}^{n\times n}(\mathbb{R})$ where the scaling in each dimension is given by the norm of each column of R and the translational part is a vector $T \in \mathbb{R}^n$. Then the transformation is typically written as $g = (R, T)$ where $g(\mathbf{x}) = R\mathbf{x} + T$.

The class of **rigid body transformations** is a subset of affine transformations characterized by $\det(R) = 1$ and R being orthonormal, $R^T R = I$. In other words, there is no scaling and the geometry of any given scene under the transformation is preserved. These transformations are sometimes called **special Euclidean transformations** or simply $SE(3)$
.

In order to represent these transformations as linear operators, we choose to represent a vector $\mathbf{x} \in \mathbb{R}^n$ as a $n+1$ dimensional vector in the space known as $\mathbb{P}^n$ or the **n-dimensional projective space**.

3.2.1 The Projection Coordinate System. Our projective space, $\mathbb{P}^n$, is characterized as being $\mathbb{R}^{n+1}\backslash\{\vec{0}\}$ with the equivalence relation for $\mathbf{X}, \mathbf{Y} \in \mathbb{P}^n$, $\mathbf{X} \simeq \mathbf{Y}$ if $\mathbf{X} = \mu\mathbf{Y}$ for some $\mu \in \mathbb{R}$. So we choose the class representative to be normalized in the $n+1$th element of $\mathbf{X}$. Thus a point $\mathbf{x} \in \mathbb{R}^n$ has projective coordinate $\mathbf{X} = [\mathbf{x}^T, 1]^T$. This class representative is also in many cases called the **homogeneous coordinate** of $\mathbf{x}$. Thus given a point $\mathbf{X} \in \mathbb{P}^n$, we can recover the equivalent point in $\mathbb{R}^n$ to be $[\frac{x_1}{x_{n+1}}, \ldots, \frac{x_n}{x_{n+1}}]^T$. There are many reasons why this space is where we choose to work. We have already mentioned that it allows us to represent affine transformations as linear since for $\mathbf{X} \in \mathbb{R}^n$,

$$g(\mathbf{x}) = R\mathbf{x} + T = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}.$$

So let

$$g = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)},$$

then for $\mathbf{X} \in \mathbb{P}^n$

$$g(\mathbf{X}) = g\mathbf{X} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \mathbf{X}.$$

Note that for a rotational matrix, $R^{-1} = R^T$, so our inverse transformation g^{-1} exists and is given by

$$g^{-1} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T T \\ 0 & 1 \end{bmatrix}.$$

Another property which becomes very useful is that in $\mathbb{P}^n$, points at infinity have a representation just like any other point. As an example, if you have ever stood on a set of railroad tracks that were straight as can be, you would notice that the parallel lines of the tracks actually converge together in our view. The point that they converge to would be the point at infinity in the direction of those parallel lines. The points $[\mathbf{x}, 0]$ do not have an equivalent representation in $\mathbb{R}^n$ but we can think of them as the limit of $[\mathbf{x}, t]$ with $t \to 0$. Thus our equivalence class representative $[\mathbf{x}, 0]$ is the the limit of $[\mathbf{x}/t, 1]$ as $t \to 0$ or in other words is the limit point at infinity in the direction of $\mathbf{x}$. Thus a point at infinity is just another point in $\mathbb{P}^n$. In projective space, lines are thus considered parallel if their intersection is at infinity.

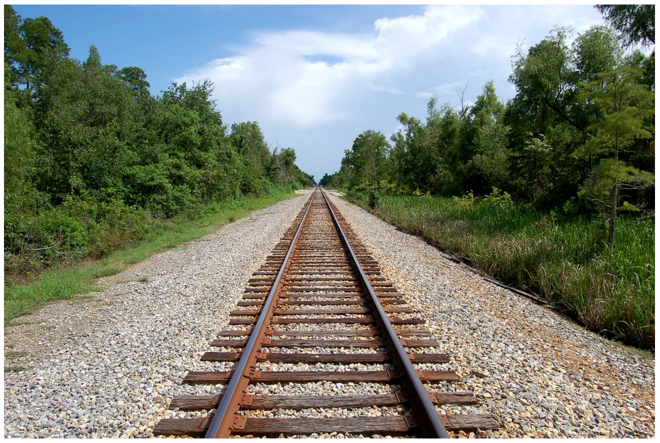


Figure 3.1: The point at infinity in the direction of the railroad tracks is where the parallel lines intersect in the picture. Photo taken by William Vann, used with permissions.

3.3 Camera Coordinate Transform

We want to simulate a rigid body transformation, $g_{cw} = (R, T)$, consisting of a rotation, R, and a translation, T, that can move us from world (w) to camera coordinates (c) and its inverse g_{wc} which moves back that will have the z-axis in the camera coordinates pointing toward the origin of the world coordinate system.

Having such a transform will prepare us for doing the projective transformations in the camera coordinates. To create this transform, we first pick a location, denote it $C \in \mathbb{R}^3$, for the camera's origin in the world coordinates. This is our translation component. Then we need a vector w of rotation and an angle, t to rotate about that vector that will give us a rotation matrix,

$$R = e^{\hat{w}t},$$

where the hat operator $\hat{(\cdot)}$, moves the vector to it's cross product form

$$\hat{w} = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}.$$

that is, the cross product is turned into matrix multiplication

$$u \times v = \hat{u}v.$$

Note that it is only a purely rotational matrix if w was normalized in the first place. So all formulas hereafter assume we have a unit vector w around which we are rotating.

The exponentiation above can be greatly simplified for skew symmetric matrices. **Rodrigues' Formula** gives, as proved in [11], that for any skew symmetric matrix like $\hat{w}$,

$$e^{\hat{w}t} = I + \sin(t)\hat{w} + (1 - \cos(t))\hat{w}^2. \tag{3.1}$$

To go backwards from R to (ω, t) is given by the following theorem.

Theorem 3.1. *Given a rotation matrix $R \in \mathbb{R}^{3\times3}$, there is some unit vector $w \in \mathbb{R}^3$ and $t \in \mathbb{R}$ such that*

$$R = e^{\hat{w}t}$$

called the **logarithm of** *R, given by the following: If $R = I$, then $t = 0$ and w can be any vector. If $R \neq I$,*

$$t = \cos^{-1}\left(\frac{trace(R) - 1}{2}\right)$$

and

$$w = \frac{1}{2\sin(t)}\begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}.$$

Proof. This is fairly easily seen by using the Rodrigues' formula, equation 3.1, and noticing that $\hat{w}^2 = ww^T - I$. □

Now, our rotation matrix must take the z-axis and rotate it down to the normalized $-C/\|C\|$. In particular the subspace perpendicular to the z-axis and $-C/\|C\|$ must be invariant under this transformation. In fact the vector

$$w = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times -\frac{C}{\|C\|} = \begin{bmatrix} C_2/\|C\| \\ -C_1/\|C\| \\ 0 \end{bmatrix}$$

is our very candidate after it is normalized. Now the angle of rotation t is the angle between the z-axis and $-C/\|C\|$ so,

$$\cos t = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot -\frac{C}{\|C\|} = -\frac{C_3}{\|C\|}$$

and hence,

$$t = \arccos(-\frac{C_3}{\|C\|})$$

is our angle of rotation. Thus our rotation matrix from camera to world coordinates is

$$R_{wc} = e^{\frac{\hat{w}}{\|w\|}t},$$

where t and w are defined above.

The complete rigid body transformation from homogeneous camera coordinates, X_c, to homogeneous world coordinates, X_w, is given by

$$X_w = g_{wc}(X_c) = \begin{bmatrix} R_{wc} & C \\ 0 & 1 \end{bmatrix} X_c.$$

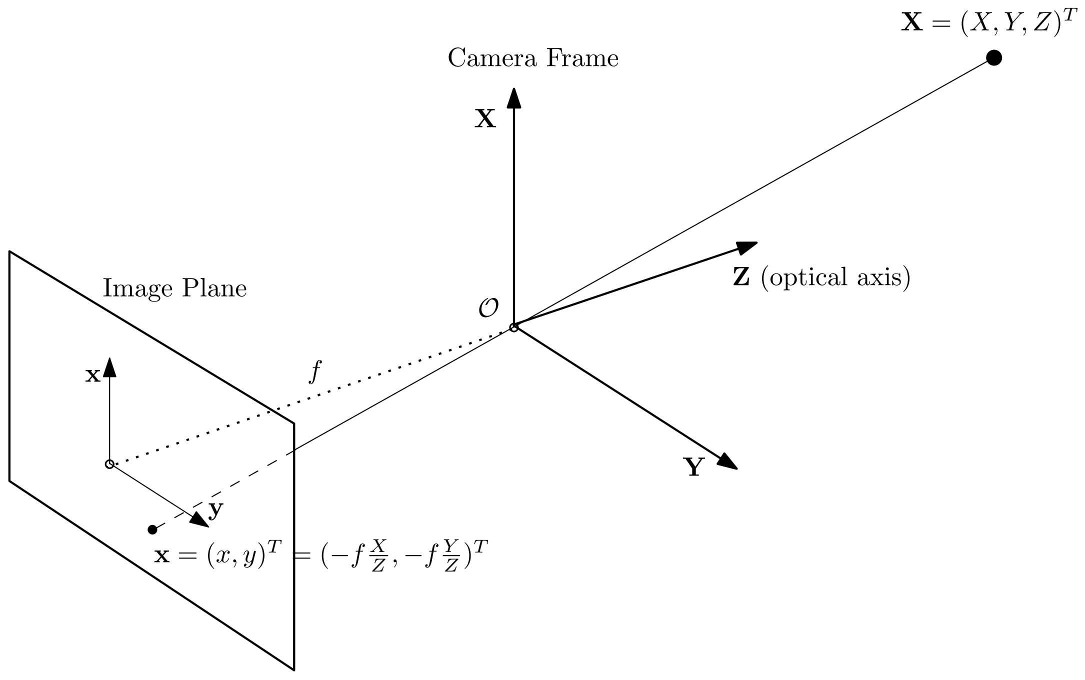
Likewise, the transform from world to camera in homogeneous coordinates can be represented as

$$X_c = g_{cw}(X_w) = \begin{bmatrix} R_{wc}^T & -R_{wc}^T\vec{C} \\ 0 & 1 \end{bmatrix} X_w.$$

3.4 The Geometry of a Camera

In this section we will introduce the mathematical models used to describe a single camera. We start with 3D coordinates which are already transformed into the camera coordinate frame. The camera can be thought of as a projection operator onto an image plane. While this projection is inherently nonlinear, by using the homogeneous coordinate system this becomes essentially a linear operation with a free scaling parameter. We start with the basic geometry and then introduce the notation we will be using to represent the camera. There are many excellent references which go into great detail about this form of representing the camera and give some bells and whistles for a very many different types of cameras and projections. A few of note are [11] and [4].

3.4.1 An Ideal Camera. In modeling cameras, we typically will make some assumptions about the lens of the camera. Standard assumptions are that the lens is a **thin lens** meaning it is nearly flat and therefore has little to no distortion of the image. While we know this is a false assumption, there are many algorithms which can undo the distortion using varying forms of interpolation and curve fitting, so it is a fairly simple preconditioning process to get our data into a form that matches this assumption. The most basic model called the **ideal camera** is the limit of the thin lens model. Sometimes it is also called a **pinhole camera**. We assume simply that the lens is condensed down to a pin hole through which light passes without distortion. We typically place the center of the lens in the origin of the camera coordinate frame. We then place the image frame which represents the film or digital receptors a distance of f units away from the origin along the camera Z axis. Another



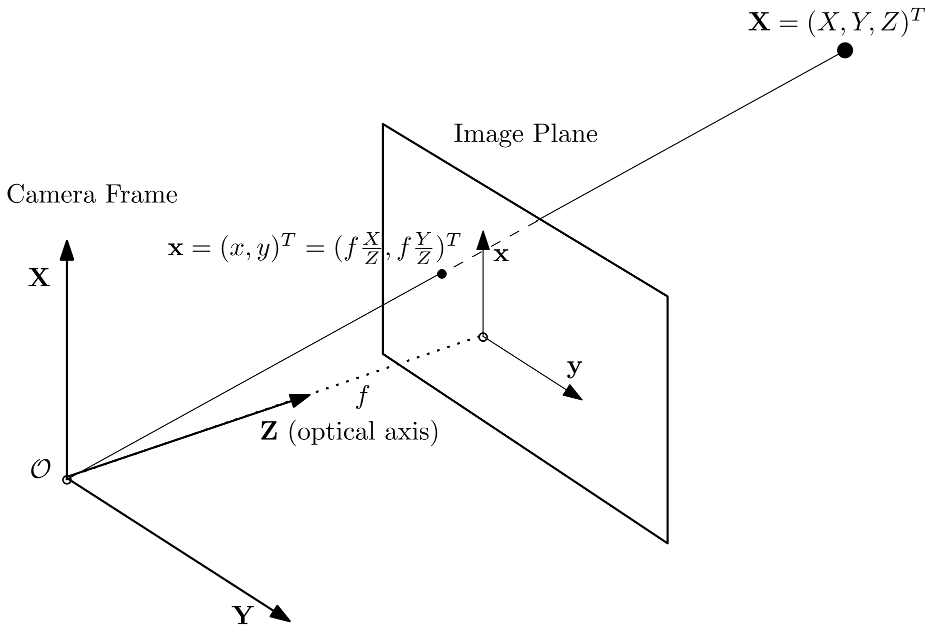
$\mathbf{X} = (X, Y, Z)^T$
Camera Frame
$\mathbf{X}$
$\mathbf{Z}$ (optical axis)
Image Plane
$\mathcal{O}$
f
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{Y}$
$\mathbf{x} = (x, y)^T = (-f\frac{X}{Z}, -f\frac{Y}{Z})^T$

Figure 3.2: The ideal camera model. The point $\mathbf{X} = (X, Y, Z)^T$ is given in coordinates of the camera frame. Note that the image plane is located behind the camera lens. This is the true form of model but it inverts the image and the coordinates on the image plane involve a negative sign: $\mathbf{x} = (-f\frac{X}{Z}, -f\frac{Y}{Z})^T$.

simplifying assumption is that the Z axis of the camera frame is the same as the optical axis of the camera. In other words, distances of objects directly in front of the camera are given by the Z coordinate in the camera frame. Then using geometry of similar triangles, it is easy to deduce that the coordinates of the point $\mathbf{X} = (X, Y, Z)^T$ on the image plane is $\mathbf{x} = (x, y)^T = (-f\frac{X}{Z}, -f\frac{Y}{Z})^T$.

We will often suppose that the image plane is instead located in front of the lens. By doing this, the view on the image plane is oriented the way we would expect and the negative signs are removed in the formula. The coordinates are given by $\mathbf{x} = (x, y)^T = (f\frac{X}{Z}, f\frac{Y}{Z})^T$.

We say the camera projects the point in space onto the image plane through the pinhole at the origin of the camera frame.



$\mathbf{X} = (X, Y, Z)^T$
Image Plane
Camera Frame
$\mathbf{x} = (x, y)^T = (f\frac{X}{Z}, f\frac{Y}{Z})^T$
$\mathbf{x}$
$\mathbf{X}$
$\mathbf{y}$
f
$\mathbf{Z}$ (optical axis)
$\mathcal{O}$
$\mathbf{Y}$

Figure 3.3: The ideal camera model. The point $\mathbf{X} = (X, Y, Z)^T$ is given in coordinates of the camera frame. By changing the location of the image plane to be in front of the lens, the coordinates on the image plane become $\mathbf{x} = (f\frac{X}{Z}, f\frac{Y}{Z})^T$.

3.4.2 Camera Projection. In projective coordinates, the camera becomes a linear operator because of our equivalency class from scalar multiplication. Since

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \simeq \lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} Zx \\ Zy \\ Z \end{bmatrix} = \mathcal{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

where $\mathcal{P}$ is a general 3×4 projection matrix. As a side note, we will often replace the Z factor by a λ to remind us that it is really a free parameter in the projective space. We will discuss the interpretation of λ and when it is possible to recover the original Z in our reconstruction methods in section 3.6.

We will break this projection matrix $\mathcal{P}$ down into a product of matrices that have meaning for our specific camera. As can be seen, the most simple projection matrix would be

$$\mathcal{P}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

the identity projector onto $\mathbb{R}^3$. Our projector matrix will also include information about how the camera turns the coordinates into pixels. In other words, the final output of a camera is not just in coordinates on the image plane but is in pixels on a finite **focal plane** with the origin being typically in the upper left corner if we were looking through the camera out.

Before finishing the projector matrix, we need to introduce some more terminology. The **principal point** of the image plane is the origin of the image plane coordinate frame. The rest of the parameters are called **intrinsic parameters** for the camera.

Intrinsic Parameters of the Camera	
f	focal length, the distance from $\mathcal{O}$ to image plane along the optical axis
(u, v)	axes of focal plane in pixels
θ	skew of pixel axes u and v. Typically $\theta = \frac{\pi}{2}$
k_u	scale of units in x direction of pixel, $x = k_u u$
k_v	scale of units in y direction of pixel, $k_v v = x\cos(\theta) - y\sin(\theta)$
(u_0, v_0)	coordinate of the principal point in pixels

Then the matrix

$$K = \begin{bmatrix} fk_u & -fk_u\cot(\theta) & u_0 \\ 0 & \frac{fk_v}{\sin(\theta)} & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

represents the transformation from the image plane to the pixel focal plane. Notice that f is always multiplied by k_u or k_v. This is the source of our scale ambiguity and so we typically assume unless otherwise known that $f = 1$. We also note that while we will do 3D reconstructions, our solution will not necessarily be the correct scale but will be accurate in its relative scale. Hence we can simplify many future problems by using this scale ambiguity to normalize various mathematical objects in those problems.

We already have introduced $g_{cw} = (R, T)$ which will transform our 3D point from the world coordinate frame to the camera coordinate frame. So, our entire transform from world coordinates to the pixel focal plane is given by

$$\begin{aligned}
\vec{x} &= \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\mathcal{P}_0\mathbf{X} = K\mathcal{P}_0 g_{cw}\mathbf{X_0} \\
&= \begin{bmatrix} fk_u & -fk_u\cot(\theta) & u_0 \\ 0 & \frac{fk_v}{\sin(\theta)} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix}.
\end{aligned}$$

3.4.3 Other Aspects of the Single Camera and some Mathematical Notation.

We say a camera is **calibrated** , if we know the intrinsic parameters, ie. the K matrix, corresponding to that camera. For calibrated cameras, we often think of K as being the identity since we can always premultiply everything by K^{-1}. Hence in our representations for a calibrated camera, we often leave the scaling depth parameter λ in front (to remind us that it is a free parameter for moving back to the Euclidean space) and can write the model as

$$\lambda\vec{x} = \mathcal{P}_0\mathbf{X}$$

A **homography** is any linear invertible transformation of $\mathbb{P}^n$. It can be described by an

$(n+1) \times (n+1)$ nonsingular matrix H. Hence we write our homography as,

$$\mathbf{x}' = H\mathbf{x}$$

3.5 Epipolar Geometry or The Geometry of Two Cameras

Epipolar geometry is the geometry corresponding to two camera views of the same object. With multiple views, we begin to be able to talk about the 3D structure of our scene. Under a single camera view, depth is not a concept that can be discussed, however once a second view is added, we can use the fairly simple geometry and a little linear algebra to do reconstructions of the scene up to a scale factor and a linear transformation (sometimes called a homography or a projective transformation in the literature) on the projective or homogeneous coordinates. In fact when we are completely calibrated in both cameras and have a notion of scale in the image, we can determine the linear transformation and remove it, giving us the correct idea of proportions and angles as well as the right scale. This typically takes some sort of calibration board which has been designed for obtaining scale measurements such as in Figure 3.5. However even without an idea of scale and distances in our image, the relative distances and measurements are correct.

Since our camera projection model is nonlinear in Euclidean space, we would expect figuring out the projector and undoing the projection to be a nonlinear problem as well. This is true and many models have been built which attempt to do this reconstruction using nonlinear techniques. However there are linear models as well which are much simpler and accomplish almost the same result. We will introduce one of those models called the Eight-Point Algorithm and then discuss some nonlinear modifications that can be given to enhance the result. When all is said and done, all these techniques reduce to a triangulation minimization problem under some metric that is linear or nonlinear.

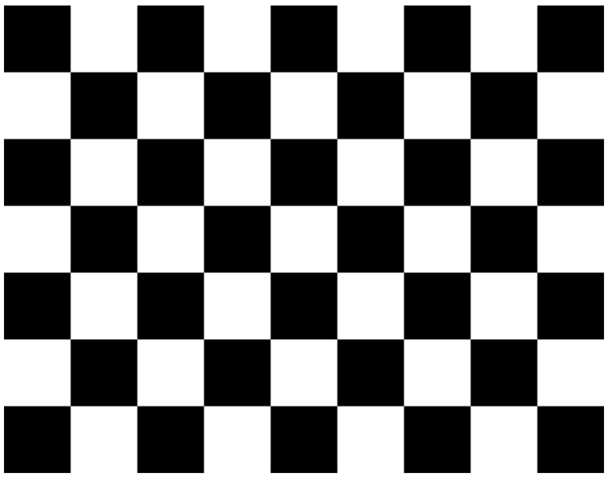


Figure 3.4: A calibration board to be used for discovering the internal parameters of a camera often looks like a chessboard because of the simple detection of contrasting squares and the size of each square is easily measured.

3.5.1 The Epipolar Constraint. Before we can get to the techniques for 3D reconstruction, we must introduce the language and geometry associated with two views. We will let the origin of camera 1 be o_1 and likewise o_2 for camera 2. Since each camera is represented by a projection onto an image plane, there is a rotation and translation $[R, T]$ that moves the coordinates from image 1 to image 2. It really does not matter which camera is labeled 1 or 2 (or later N) but once we have designated them, we will stick to them and all of the following notation will be associated with that specific designation of camera 1 and camera 2.

Now, given a point $\mathbf{X}_0$ in our 3D space, if $\mathcal{P}_1$ is the projection corresponding to camera 1 and $\mathcal{P}_2$ to camera 2, then $\lambda_1\mathbf{x}_1 = \mathcal{P}_1\mathbf{X}_0$ and $\lambda_2\mathbf{x}_2 = \mathcal{P}_2\mathbf{X}_0$. And since we have a relationship between our two cameras given by $[R, T]$ we get that

$$\lambda_2\mathbf{x}_2 = R\lambda_1\mathbf{x}_1 + T.$$

Now we wish to remove our parameters λ_1 and λ_2 and obtain a relationship between $\mathbf{x}_1$

and $\mathbf{x}_2$ directly. To do so, we will premultiply our equation above by $\hat{T}$ to get

$$\lambda_2 \hat{T}\mathbf{x}_2 = \lambda_1 \hat{T} R \mathbf{x}_1 + \hat{T}T,$$

but since $\hat{T}T = T \times T = 0$, we have

$$\lambda_2 \hat{T}\mathbf{x}_2 = \lambda_1 \hat{T} R \mathbf{x}_1.$$

Finally, we premultiply both sides by x_2^T and since $\hat{T}x_2 = T \times x_2$ is perpendicular to $\mathbf{x}_2$, we have $\mathbf{x}_2^T \hat{T}\mathbf{x}_2 = 0$. So

$$0 = \lambda_1 \mathbf{x}_2^T \hat{T} R \mathbf{x}_1,$$

or in other words we have what is called the **epipolar constraint**

$$\mathbf{x}_2^T \hat{T} R \mathbf{x}_1 = 0.$$

We define $E = \hat{T}R$ which is called the **essential matrix**. Then the epipolar constraint is given by

$$\mathbf{x}_2^T E \mathbf{x}_1 = 0.$$

More details will be given about the set of essential matrices characterized by being the product of a skew symmetric matrix and a rotation matrix in Section 3.5.3.

The geometric interpretation of this epipolar constraint as pointed out in Chapter 5 of [11] is as follows. We consider the triangle formed from the points $\mathbf{X}$, and the two camera origins o_1 and o_2 as seen in Figure 3.5. The three vectors of the triangle are on a single plane and so their triple product (the area of parallelepiped formed by them) is zero. The epipolar constraint is just this same triple product written in terms of the second camera frame since $R\mathbf{x}_1$ is the direction of vector from o_1 to $\mathbf{X}$ and T is the direction from o_1 to o_2.

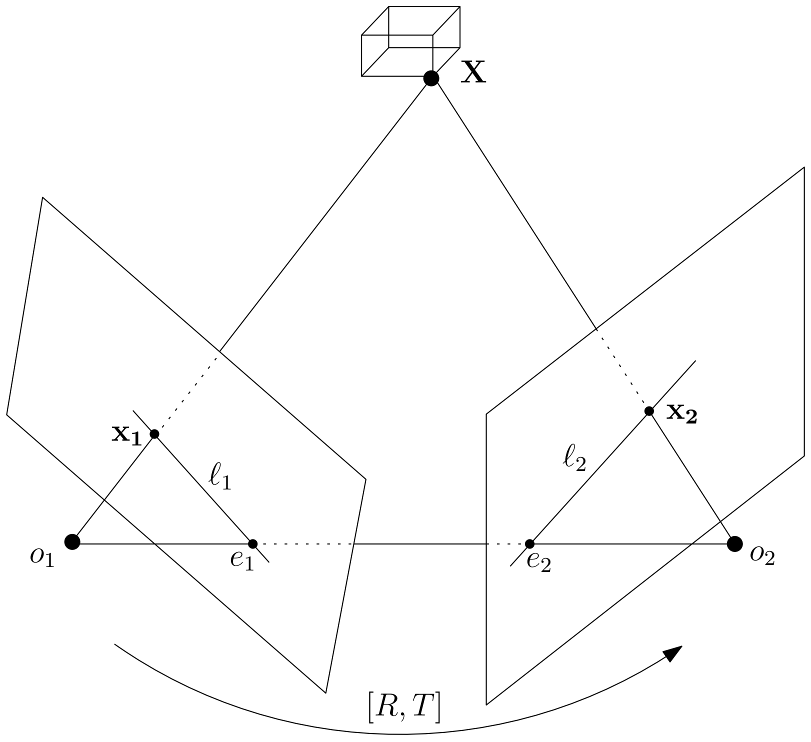


Figure 3.5: Epipolar geometry shows the projection of point $\mathbf{X}$ onto each camera plane together with the epipoles and epipolar lines corresponding to each image plane.

So $\mathbf{x}_2^T \hat{T} R \mathbf{x}_1 = x_2 \cdot (T \times R\mathbf{x}_1) = 0$.

From this epipolar constraint we obtain a few other geometric objects which have use in our discussion. The projection of the origin of camera 2 onto the image plane of camera 1 is called an **epipole**, denoted e_1, likewise on image plane 2, the projection of camera 1's origin is e_2. The projection of the line passing through $\mathbf{X}$ and o_2 on camera 1's image plane is called the **epipolar line of X**, denoted ℓ_1, likewise the projection of the line passing through $\mathbf{X}$ and o_1 onto image plane 2 is given by ℓ_2. We define the **epipolar plane** to be the plane passing through $\mathbf{X}$, o_1 and o_2. The epipolar lines can also be thought of as the intersection of the epipolar plane with each image plane respectively. Now since the vector $R\mathbf{x}_1$ is the same direction as the vector o_1 to $\mathbf{X}$ in image plane 2, then $T \times R\mathbf{x}_1$ define the coefficients up to scale of the line ℓ_2. Thus we obtain

$$\ell_2 \simeq E\mathbf{x}_1, \qquad \ell_1 \simeq E^T\mathbf{x}_2,$$

where we remember that $\simeq$ is equality up to multiplication by a scalar. Combining these with the epipolar constraint, we get that

$$\ell_i^T x_i = 0 \qquad \text{for } i = 1, 2.$$

Likewise we recognize that $e_2 \simeq T$ and $e_1 \simeq R^T T$ so

$$e_2^T E = 0, \qquad Ee_1 = 0,$$

and since each epipole lies on the epipolar line corresponding to it's image plane,

$$\ell_i^T e_i = 0 \qquad \text{for } i = 1, 2.$$

3.5.2 Planar Homography. Now, if all our points are on a plane, then we will use a planar homography instead of our epipolar constraint in reconstruction. We construct a **planar homography** using the normal vector, N, of the plane on which every point is located. We let d be the distance from camera 1 to the plane given by

$$d = N^T\mathbf{X}_1.$$

Then since we have

$$\mathbf{X}_2 = R\mathbf{X}_1 + T,$$

we get

$$\begin{aligned}\mathbf{X}_2 &= R\mathbf{X}_1 + T \\ &= R\mathbf{X}_1 + T\frac{N^T\mathbf{X}_1}{d} \\ &= \left(R + \frac{TN^T}{d}\right)\mathbf{X}_1 \\ &= K\mathbf{X}_1.\end{aligned}$$

Then our planar homography is given by

$$K = R + \frac{TN^T}{d} \in \mathbb{R}^{3\times 3}$$

and so for any $\mathbf{X}_1$ on our plane, the corresponding $\mathbf{X}_2$ in camera 2 coordinates is given by

$$\mathbf{X}_2 = K\mathbf{X}_1.$$

We can now project each coordinate onto its respective image plane to get

$$\mathbf{x}_2 \simeq K\mathbf{x}_1.$$

Thus our planar homography takes the place of the epipolar constraint when all points are on a single plane. Reconstruction techniques for the planar homography can be found in Section 5.3.2 of [11].

3.5.3 The Essential Matrix. We give a few useful theorems and characterizations about the essential matrices, in the essential space, $\varepsilon = \{E = \hat{T}R \in \mathbb{R}^{3\times 3} \mid R \in SO(3) \text{ and } T \in \mathbb{R}^3\}$.

Proposition 3.2. *Each essential matrix, $E \in \varepsilon \subset \mathbb{R}^{3\times 3}$ has rank 2.*

Proof. We let $E = \hat{T}R$. Then since R is a rotation matrix, it is invertible and so $\operatorname{rank} R = 3$. Now it can be easily proved that $\operatorname{rank} \hat{T} = 2$ since it is a 3×3 skew symmetric matrix. Hence $\operatorname{rank} E = 2$. □

Theorem 3.3. *Given a real $n \times n$ matrix E where $rank(E) = n-1$, we can obtain the singular value decomposition, $E = U\Sigma V^T$, where U and V are in $SO(n)$, meaning they are orthonormal and have determinant $= 1$. (Note that the traditional SVD decomposition only guarantees orthonormality and determinant $= \pm 1$.) and $\Sigma = diag([\sigma_1, \sigma_2, \dots, \sigma_{n-1}, 0])$.*

Proof. Take our standard SVD decomposition of $E = U\Sigma V^T$. Since U and V are already guaranteed to be orthogonal, we must show that we can write $E = \tilde{U}\Sigma\tilde{V}^T$ such that $\det \tilde{U} = 1$ and $\det \tilde{V} = 1$.

Now since $rank(E) = n-1$, we know that $\sigma_n = 0$ hence we write E as

$$E = \sigma_1 * u_1 v_1^T + \sigma_2 * u_2 v_2^T + \cdots + \sigma_{n-1} * u_{n-1} v_{n-1}^T + 0 * u_n v_n^T,$$

where the u_i (respectively v_i) are the column vectors of U (respectively V). We now consider some cases where modification might be required:

If $\det(U)$ or $\det(V) = 1$ then we let $\tilde{U} = U$ or $\tilde{V} = V$.

If $\det(U) = -1$, then let $\tilde{U} = [u_1, u_2, \dots, u_{n-1}, -u_n] = U * \text{diag}([1, 1, \dots, 1, -1])$.

If $\det(V) = -1$, then let $\tilde{V} = [v_1, v_2, \dots, v_{n-1}, -v_n] = V * \text{diag}([1, 1, \dots, 1, -1])$.

Thus $\det(\tilde{U}) = \det(\tilde{V}) = 1$ and,

$$\begin{aligned} \tilde{U}\Sigma\tilde{V}^T &= \sigma_1 * \tilde{u}_1 \tilde{v}_1^T + \cdots + \sigma_{n-1} * \tilde{u}_{n-1} \tilde{v}_{n-1}^T + 0 * \tilde{u}_n \tilde{v}_n^T \\ &= \sigma_1 * u_1 v_1^T + \cdots + \sigma_{n-1} * u_{n-1} v_{n-1}^T + 0 * u_n v_n^T \\ &= U\Sigma V^T \\ &= E. \end{aligned}$$

$\square$

The next theorem is a useful characterization of essential matrices proved by Huang and Faugeras in [7].

Theorem 3.4. *A nonzero matrix, $E \in \mathbb{R}^{3\times 3}$, is an essential matrix if and only if it has singular value decomposition*

$$E = U\,\mathrm{diag}\{\sigma, \sigma, 0\}V^T,$$

where $\sigma > 0$ and $U, V \in SO(3)$.

Proof. ($\Rightarrow$) By definition, $E = \hat{T}R$ where $R \in SO(3)$ and $T \in \mathbb{R}^3$. Then there is some orthogonal matrix $R_0 \in \mathbb{R}^{3\times 3}$ such that

$$\hat{T} = R_0^T \begin{bmatrix} 0 & \|T\| & 0 \\ -\|T\| & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} R_0.$$

Then

$$E^T E = R^T \hat{T}^T \hat{T} R = (R_0 R)^T \begin{bmatrix} \|T\|^2 & 0 & 0 \\ 0 & \|T\|^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} (R_0 R),$$

where we note that $R_0 R$ is also orthogonal and so the singular values of E are $\|T\|$, $\|T\|$, and 0. Then for the singular value decomposition $E = U\Sigma V^T$, by theorem 3.3, our matrices U and V, for are in $SO(3)$.

($\Leftarrow$) Assume we have the singular value decomposition of $E = U\Sigma V^T$ where $\Sigma = \mathrm{diag}\{\sigma, \sigma, 0\}$ and U and V are in $SO(3)$. Now let $R_z(\theta)$ be the matrix that represents a rotation of

θ radians around the z-axis. Then for $e_3 = [0,0,1]^T$,

$$R_z\left(+\frac{\pi}{2}\right) = e^{\hat{e}_3 \frac{\pi}{2}} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and $R_z\left(-\frac{\pi}{2}\right) = R_z\left(+\frac{\pi}{2}\right)^T$. Then let

$$(\hat{T}_1, R_1) = (UR_z\left(+\frac{\pi}{2}\right)\Sigma U^T, UR_z^T\left(+\frac{\pi}{2}\right)V^T) \tag{3.2}$$

$$(\hat{T}_2, R_2) = (UR_z\left(-\frac{\pi}{2}\right)\Sigma U^T, UR_z^T\left(-\frac{\pi}{2}\right)V^T). \tag{3.3}$$

It is easily verified that $\hat{T}_1$ and $\hat{T}_2$ are skew symmetric and $R_1, R_2 \in SO(3)$ since each are the product of matrices in $SO(3)$. Likewise $E = \hat{T}_1 R_1 = \hat{T}_2 R_2$.

□

Theorem 3.5. *Now given an essential matrix, E, there are exactly two unique decompositions $E = \hat{T}_1 R_1 = \hat{T}_2 R_2$ given by Equations 3.2 and 3.3.*

Proof. The proof can be found on in **Theorem 5.7** on page 116 of [11]. It is not difficult but is rather tedious and offers no additional insight. Hence we refer the reader to their proof. □

3.6 Traditional Reconstruction Techniques

We first introduce, in Section 3.6.1, the most simple yet effective linear technique for finding the essential matrix, $E \in \{\hat{T}R \mid R \in SO(3), T \in \mathbb{R}^3\}$, from two sets of correlated points $\mathbf{x}_1$ and $\mathbf{x}_2$ and decompose E it into the product of a rotation, R, and a translation, T. This allows us to calculate the depth of the points and thus effect a 3D reconstruction of our points. This algorithm is called the **eight point algorithm** and is excellent for data points which have no error in them. It was introduced in 1981 by Longuet-Higgins in [10] and has been used by practitioners since then. It is a linear algorithm and fairly simple to implement but rather effective in getting close to the true solution R and T and is often used to get an initial guess for other nonlinear methods. We then give a nonlinear refinement, in Section 3.6.2, which works well to enhance the accuracy of the solution under measurement error in the data. This triangulation algorithm will give back the optimal R and T as well as the data points with the error removed. It performs fairly well under small to moderate error but very poorly under high levels of noise due to the nonlinearity. It is recommended on the basis of many empirical tests in [12] that under high levels of noise, the linear method be used as it consistently obtained a close answer and the others performed much more poorly.

3.6.1 The 8-Point Algorithm. We will solve for the essential matrix, E, in the essential space $\varepsilon = \{E \in \mathbb{R}^{3\times 3} \mid E = \hat{T}R$ where $T \in \mathbb{R}^3$ and $R \in SO(3)\}$ that satisfies the epipolar constraint $\mathbf{x}_2^{jT} E \mathbf{x}_1^j = 0$ for all correlated data points $\mathbf{x}_1 = \{\mathbf{x}_1^j\}_{j=1}^n$ and $\mathbf{x}_2 = \{\mathbf{x}_2^j\}_{j=1}^n$. We will then obtain a decomposition of E into $R \in SO(3)$ and $T \in \mathbb{R}^3$ such that $E = \hat{T}R$.

To begin, we write our epipolar constraint $\mathbf{x}_2^{jT} E x_1^j = 0$ as a linear system. We vectorize our essential matrix

$$E = \begin{bmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{bmatrix}$$

to be

$$E_v = \begin{bmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 \end{bmatrix}^T \in \mathbb{R}^9.$$

Then let our data points be formed into a matrix $A = [\mathbf{x}_1^1 \otimes \mathbf{x}_2^1 \ \dots \ \mathbf{x}_1^n \otimes \mathbf{x}_2^n] \in \mathbb{R}^{9\times n}$ where

$\otimes$ is the Kronecker product. In other words if $\mathbf{x}_1 = [x_1^1, x_1^2, x_1^3]^T$, then

$$\mathbf{x}_1 \otimes \mathbf{x}_2 = \begin{bmatrix} x_1^1 \mathbf{x}_2 \\ x_1^2 \mathbf{x}_2 \\ x_1^3 \mathbf{x}_2 \end{bmatrix} \in \mathbb{R}^9.$$

Then our epipolar constraint for all n data points can be written as the linear equation

$$A^T E_v = 0. \tag{3.4}$$

We can now solve the least squares problem for a unique nontrivial vector E_v of this equation granted that A has the property

$$\operatorname{rank} A \geq 8.$$

This will always happen provided that there are 8 correlated sets of points, that is we have correlated points $\mathbf{x}_1^i$ and $\mathbf{x}_2^i$ for $i = 1, \ldots, n$ and $n \geq 8$. A requirement on those points is that they are in what is called a **general position** which essentially means that no four points are collinear or that they do not lie on the same plane. When all the points lie on a plane in space, the epipolar constraint is not sufficient to solve for their positions and an additional assumption must be made since the epipolar matrix actually becomes a planar homography and multiple nontrivial solutions to equation 3.4 exist. This case is treated thoroughly in Section 5.3.2 of the book [11] by Ma et al.

Now it is fairly simple to prove that the eigenvector, v, corresponding to the smallest nonzero eigenvalue of $A^T A$ gives the minimum nontrivial solution to the least squares problem

$$\min_{v \in \mathbb{R}^9} \|A^T v\|_2^2.$$

Now, if the points are in general position then this is the last column vector, v_9, of the matrix $V = [v_1, \dots, v_9]$ in the singular value decomposition of A

$$A = U\Sigma V^T$$

and if the points have no measurement error, it will correspond to a vector which exactly solves the problem

$$A^T E = 0.$$

Otherwise it will be the least squares minimum of $\|A^T E\|_2^2$. We note that it may not attain a value of zero because the measured data points may not exactly satisfy the epipolar constraint due to the measurement error in them.

Now, we let $E_v = v_9$ and then reshape it back to it's 3×3 matrix form, E. This matrix satisfies our epipolar constraint, $\mathbf{x}_2^T E \mathbf{x}_1 = 0$, but E may not necessarily be in our essential space ε. We will thus project it onto the essential space using the Frobenius norm as our metric.

Theorem 3.6. *The essential matrix $E_0 \in \varepsilon$ that minimizes the Frobenius norm $\|E - E_0\|_F$ for a matrix $E \in \mathbb{R}^{3\times 3}$ is given by*

$$E_0 = U \operatorname{diag}\{\lambda, \lambda, 0\} V^T,$$

where E has the singular value decomposition $E = U \operatorname{diag}\{\lambda_1, \lambda_2, \lambda_3\} V^T$ and $\lambda = (\lambda_1 + \lambda_2)/2$.

Remark. Now typically we will use the essential matrix $E_0 \in \varepsilon$ that has norm 1 which is equivalent to letting $E_0 = U \operatorname{diag}\{1, 1, 0\} V^T$. This works since there is a scale ambiguity already inherent in our problem so we can assume a normalized translation vector T in our reconstructions. Then since $\|R\| = 1$, and $\|T\| = 1$ we get that $\|E\| = \|\hat{T}R\| = 1$ as well.

We note as well that while E_0 is the solution with the minimal Frobenius norm, it may not be the optimal solution to our original problem of finding an essential matrix, $E \in \varepsilon$, that gives

$$x_2^{jT} E x_1^j = 0,$$

for $j = 1, \ldots, n$.

Now, given our matrix E_0, we decompose it into its two possible T and R such that $E_0 = \hat{T}_1 R_1 = \hat{T}_2 R_2$ following the decomposition given in equations 3.2 and 3.3. We also note that the sign of E was also arbitrary and so we follow the same procedure to project $-E$ onto the essential space and decompose it into $\hat{T}_3 R_3$ and $\hat{T}_4 R_4$. Hence, we have four possible solutions for our rotation R and translation T. It has been shown in [11] that one of these four solutions will always give rise to a 3D reconstruction that has positive depth. The method recommended for determining which of these to use is by checking the sign of the depth of the 3D points. The one with the most positive depths is the solution to be used. A simple way to check the sign of the depth is as follows. The 3D point has positive depth if $(\hat{T} x_1^j)^T (\hat{x}_1^j R^T x_2^j) > 0$.

Summary of the 8-Point Algorithm	
Step 1.	Create matrix of data points $A \in \mathbb{R}^{9\times n}$ with columns $\mathbf{x}_1^j \otimes \mathbf{x}_2^j$ for $j = 1, \ldots, n$. We solve the linear system $AE_s = 0$ corresponding to the epipolar constraint, by letting $E_s = v_9$ be the last column vector of the matrix V from the singular value decomposition of $A = U\Sigma V^T$. Unstack the vector $E_s \in \mathbb{R}^{9\times 1}$ into the matrix $E \in \mathbb{R}^{3\times 3}$. We note that E is not necessarily in our essential space but is the least squares minimum of the system $\mathbf{x}_2^{jT} E \mathbf{x}_1^j$ for $j = 1, \ldots, n$.
Step 2.	Project the solution E onto the essential space under the Frobenius norm. Let $E = U\,\mathrm{diag}\{\sigma_1, \sigma_2, \sigma_3\} V^T$ be the singular value decomposition. Then the matrix, E_0, given by $E_0 = U\,\mathrm{diag}\{1, 1, 0\} V^T$ is the normalized essential matrix which minimizes the Frobenius norm $\|E - E_0\|_F$.
Step 3.	Decompose $E_0 = U\Sigma V^T$ into the the 2 pairs of translation and rotation parts as defined in Theorem 3.4 $$\begin{aligned}(\hat{T}_1, R_1) &= (UR_z\left(+\frac{\pi}{2}\right)\Sigma U^T, UR_z^T\left(+\frac{\pi}{2}\right)V^T),\\ (\hat{T}_2, R_2) &= (UR_z\left(-\frac{\pi}{2}\right)\Sigma U^T, UR_z^T\left(-\frac{\pi}{2}\right)V^T),\end{aligned}$$ Likewise decompose $-E$ via Step 2. and 3. to get a total of 4 possible pairs of translation and rotations, (T_i, R_i) for $i = 1 \ldots 4$.
Step 4.	Choose the solution $E_0 = \hat{T}R$ from one of the four decompositions, (T_i, R_i) for $i = 1, \ldots, 4$, which yields positive depth solutions in the 3D reconstruction. The sign of the depth is positive if $$(\hat{T}x_1^j)^T(\hat{x}_1^j R^T x_2^j) > 0.$$ We note that this solution $E_0 = \hat{T}R$ is not necessarily the true solution but instead is our best least squares approximation to it. So we recommend to choose the decomposition which yields the most positive depths of the four.

3.6.2 Refinement of Solution via Triangulation. Since the 8-point algorithm makes the assumption that there is no error in the data, and all real images have a bit of error, we need a way to refine our solution, X, R and T, to be optimal where we use the term optimal in the sense of minimal under least squares. Hence the method of triangulation is optimal for our purposes. In [12], Ma et al demonstrate the equivalence of many existing techniques that were being used in industry to the general triangulation method under the epipolar constraint. The following discussion of refinements follows the ideas set forth by them to improve the estimates of R and T and to get estimates of the true data $\mathbf{x}$ from measured data points with error in them, $\tilde{x}$. We proceed as follows.

Given n measured and correlated data points $\tilde{\mathbf{x}}_1 = \{\tilde{\mathbf{x}}_1^j\}_{j=1}^n$ and $\tilde{\mathbf{x}}_2 = \{\tilde{\mathbf{x}}_2^j\}_{j=1}^n$ for each camera, we want to find the set of points $\mathbf{x}_1$ and $\mathbf{x}_2$ that minimize

$$\phi(\mathbf{x}_1, \mathbf{x}_2, R, T) = \sum_{j=1}^{n} \|\tilde{\mathbf{x}}_1^j - \mathbf{x}_1^j\|^2 + \|\tilde{\mathbf{x}}_2^j - \mathbf{x}_2^j\|^2,$$

subject to $\forall j \in \{1, \ldots, n\}$,

$$\begin{aligned}
&\mathbf{x}_2^{jT} \hat{T} R x_1^j = 0\\
&\mathbf{x}_1^{jT} e_3 = 1\\
&\mathbf{x}_2^{jT} e_3 = 1,
\end{aligned}$$

where $e_3 = [0, 0, 1]^T$ is the 3rd unit vector in $\mathbb{R}^3$. Using the method of Lagrange multipliers, we reduce this constrained optimization problem to an unconstrained one, as shown in Appendix B.1. In actuality we give two equivalent formulations of the reduced problem which were arrived at separately in Appendix B.1.

$$\min_{\{\mathbf{x}_1,\mathbf{x}_2,R,T\}} \phi(\mathbf{x}_1,\mathbf{x}_2,R,T) = \sum_{j=1}^{n} \frac{\left(\mathbf{x}_2^{jT}\hat{T}R\tilde{\mathbf{x}}_1^j + \tilde{\mathbf{x}}_2^{jT}\hat{T}R\mathbf{x}_1^j\right)^2}{\left\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\right\|^2 + \left\|\hat{e}_3\hat{T}R\mathbf{x}_1^j\right\|^2} \tag{3.5}$$

$$= \sum_{j=1}^{n} \frac{\left(\mathbf{x}_2^{jT}\hat{T}R\tilde{\mathbf{x}}_1^j\right)^2}{\left\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\right\|^2} + \frac{\left(\tilde{\mathbf{x}}_2^{jT}\hat{T}R\mathbf{x}_1^j\right)^2}{\left\|\hat{e}_3\hat{T}R\mathbf{x}_1^j\right\|^2}. \tag{3.6}$$

Now, again following the discussion in [11],[12] and [18], we proceed to simplify our problem by reducing considerably the parameter space. This is done by exploiting the geometry and using it to simplify our search for optimal $\mathbf{x}_1$ and $\mathbf{x}_2$ to be a search over a single one dimensional variable, θ. We proceed as follows:

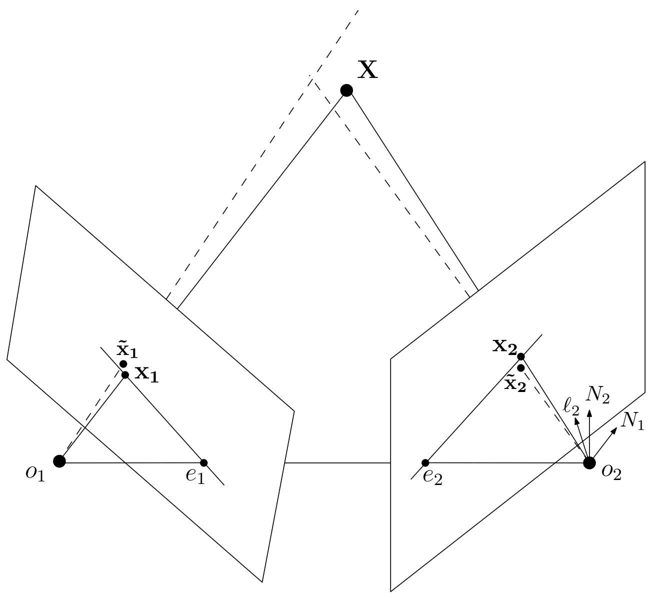
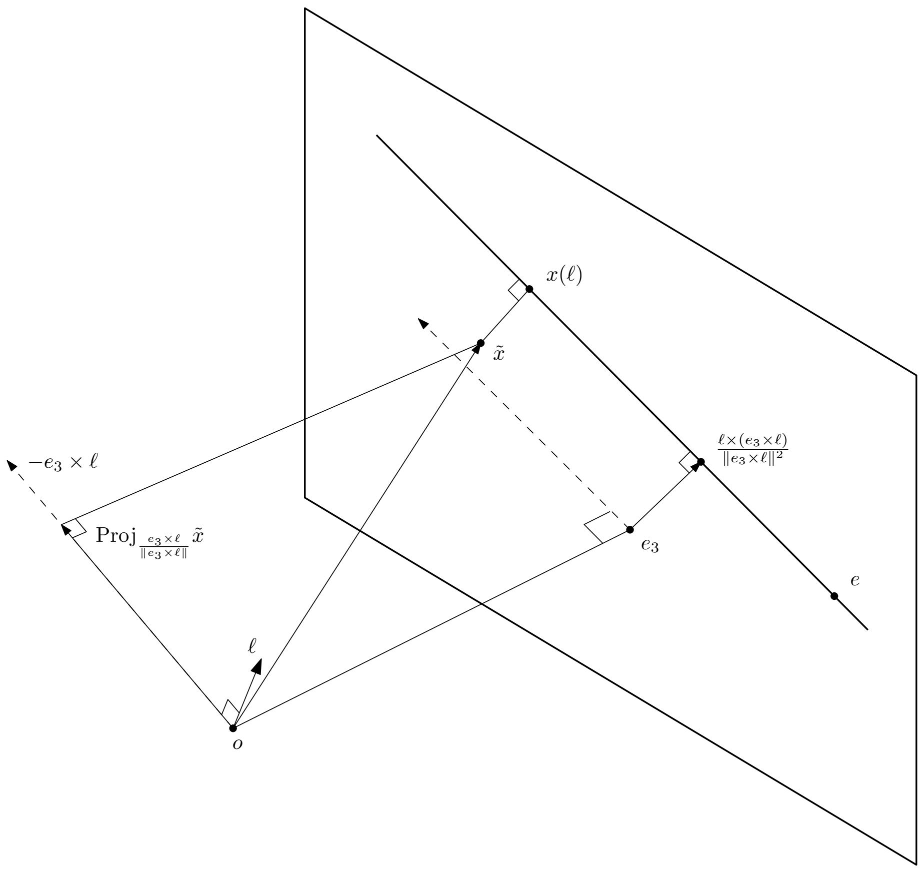


Figure 3.6: Given the epipolar plane with ℓ_2 as it's normal vector, we can calculate the nearest $\mathbf{x}_i$ on plane to $\tilde{\mathbf{x}}_i$ which satisfies the epipolar constraint.

We show in Appendix B.2 that for any measured data point, $\tilde{\mathbf{x}}$, on the image plane and epipolar line, we can explicitly compute the closest point to $\tilde{\mathbf{x}}$ on that given epipolar line in the following manner. Let ℓ be the unit normal vector to the epipolar plane corresponding to our desired epipolar line as seen in Figure 3.6, (ie. $\ell = \frac{e \times x}{\|e \times x\|}$ where e is the epipole and x is any other point on the epipolar line). Then $\mathbf{x}(\ell)$, the closest point to $\tilde{\mathbf{x}}$ on the epipolar line is characterized by

$$\mathbf{x}(\ell) = \frac{\hat{e}_3 \ell \ell^T \hat{e}_3^T \tilde{\mathbf{x}} + \hat{\ell}^T \hat{\ell} e_3}{e_3^T \hat{\ell}^T \hat{\ell} e_3}, \tag{3.7}$$

where $e_3 = [0, 0, 1]^T$. Hence $\ell \simeq \mathbf{x} \times e$ and so if we are in the second image coordinates, $\ell_2 = \mathbf{x}_2 \times e_2$ and hence we can easily calculate the same vector in the first image plane to be $\ell_1 = R^T \ell_2$.



$x(\ell)$
$\tilde{x}$
$-e_3 \times \ell$
$\frac{\ell \times (e_3 \times \ell)}{\|e_3 \times \ell\|^2}$
$\mathrm{Proj}_{\frac{e_3 \times \ell}{\|e_3 \times \ell\|}} \tilde{x}$
e_3
e
ℓ
o

Figure 3.7: The closest point to $\tilde{\mathbf{x}}$ on the given epipolar line is given by Equation 3.7 where ℓ is the unit vector normal to the epipolar plane through the points o, e, and $\mathbf{x}(\ell)$).

Thus we have

$$\mathbf{x}_1(\ell_1) = \frac{\hat{e}_3 \ell_1 \ell_1^T \hat{e}_3^T \tilde{\mathbf{x}}_1 + \hat{\ell}_1^T \hat{\ell}_1 e_3}{e_3^T \hat{\ell}_1^T \hat{\ell}_1 e_3} \tag{3.8}$$

and

$$\mathbf{x}_2(\ell_2) = \frac{\hat{e}_3 \ell_2 \ell_1^T \hat{e}_3^T \tilde{\mathbf{x}}_2 + \hat{\ell}_2^T \hat{\ell}_2 e_3}{e_3^T \hat{\ell}_2^T \hat{\ell}_2 e_3}. \tag{3.9}$$

Now using these explicit forms, we can write our minimization distance $\|\tilde{\mathbf{x}} - \mathbf{x}(\ell)\|^2$ as

$$\begin{aligned} \|\tilde{\mathbf{x}} - \mathbf{x}(\ell)\|^2 &= \|\tilde{\mathbf{x}}\|^2 + \frac{\ell^T \left(I + 2\hat{\tilde{\mathbf{x}}}\hat{e}_3 - \hat{e}_3^T \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T \hat{e}_3 \right) \ell}{\ell^T \left(\hat{e}_3^T \hat{e}_3 \right) \ell} \\ &= \|\tilde{\mathbf{x}}\|^2 + \frac{\ell^T A(\tilde{\mathbf{x}}) \ell}{\ell^T B \ell}, \end{aligned}$$

where $A(\tilde{\mathbf{x}}) = I + 2\hat{\tilde{\mathbf{x}}}\hat{e}_3 - \hat{e}_3^T \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T \hat{e}_3$ and $B = \hat{e}_3^T \hat{e}_3$. Thus we have our function to be optimized as a function of ℓ_2^j

$$\begin{aligned} \phi(\ell_2, R, T) &= \|\tilde{\mathbf{x}}_2^j - \mathbf{x}_2^j(\ell_2^j)\|^2 + \|\tilde{\mathbf{x}}_1^j - \mathbf{x}_1^j(R^T \ell_2^j)\|^2 && (3.10) \\ &= \sum_{j=1}^{n} \|\tilde{\mathbf{x}}_2^j\|^2 + \frac{\ell_2^{jT} A(\tilde{\mathbf{x}}_2^j) \ell_2^j}{\ell_2^{jT} B \ell_2^j} + \|\tilde{\mathbf{x}}_1^j\|^2 + \frac{\ell_2^{jT} R A(\tilde{\mathbf{x}}_1^j) R^T \ell_2^j}{\ell_2^{jT} R B R^T \ell_2^j}. && (3.11) \end{aligned}$$

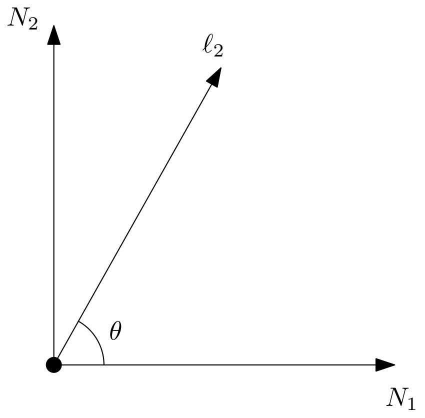


Figure 3.8: We parameterize $\ell_2 = \cos(\theta) N_1 + \sin(\theta) N_2$ where N_1 and N_2 are orthogonal basis elements for $\mathbb{R}^3$ along with the epipole, e_2 ($\simeq T$), in the second camera's reference frame.

Now, this can be further simplified by remembering that ℓ_2 is the unit normal vector to the epipolar plane given by o_2, $\mathbf{x}_2$ and e_2. Hence if we create a set of orthonormal basis elements with e_2 as one of the directions and N_1 and N_2 the other two basis elements, then

ℓ_2 is actually found in the plane perpendicular to e_2. The gemoetry is demonstrated in Figure 3.6. Hence ℓ_2 is a unit vector in the plane given by the vectors (N_1, N_2) and so we can parameterize ℓ_2 by its angle formed with N_1. Hence $\ell_2 = \cos(\theta)N_1 + \sin(\theta)N_2$ and we can use θ as our variable of optimization instead of ℓ_2. As a simple note, we can also restrict T to be in $\mathbb{S}^2$ since scale is still arbitrary so we can think of T as being a direction on the boundary of the unit sphere. In [12], Ma, et al, point out that this all becomes an optimization on the manifold $(R, T, \theta) \in SO(3) \times \mathbb{S}^2 \times \mathbb{R}^n$ since there are n different data points in each camera and so n θ values to be optimized. In addition, they point out that we are searching for the optimal θ in a bounded region since geometry dictates it will be between the θ_1 and θ_2 where $\ell_1(\theta_1)$ and $\ell_2(\theta_2)$ are the normal unit vectors of the epipolar planes through $\tilde{(\mathbf{x})}_1$ and $\tilde{\mathbf{x}}_2$ respectively.

They recommend an alternating minimization scheme to find the minimum.

Alternating Scheme for Nonlinear Refinement	
Step 1.	Initialize R, T to be the solutions of the eight-point algorithm with the measured data, $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$. Minimize equation 3.5 or 3.6 in R and T. Use the explicit characterizations of each, $(w, t) \in \mathbb{R}^3 \times \mathbb{R}$ where $R = \exp(\hat{w}t)$ as described in Section 3.3 and $T \in \mathbb{S}^2$, to do avoid doing the optimization on a manifold.
Step 2.	Given the new R and T, minimize equation 3.11 with respect to ℓ_2.
Step 3.	Go back to Step 1. if $\phi(\mathbf{x}_1, \mathbf{x}_2, R, T)$ is not below some predetermined error limit.

Remark. A final point of caution before we proceed. When error becomes very large, bifurcations can happen in our minimization function which will cause our nonlinear optimization to find a solution much worse even than the linear eight point algorithm. This is described in great detail in the paper [12] by Ma et al. They make a recommendation to help alleviate this effect by adjusting the linear eight point algorithm slightly to use the eigenvectors corresponding to the smallest and second smallest eigenvalues of $X^T X$ to construct E. They recommend to construct E for each eigenvector and then choose the solution which gives the most positive depths in the reconstruction. This can be done without explicitly computing the depth by checking the sign of the depth. Thus it is positive if

$$(\hat{T} x_1^j)^T (\hat{x}_1^j R^T x_2^j) > 0.$$

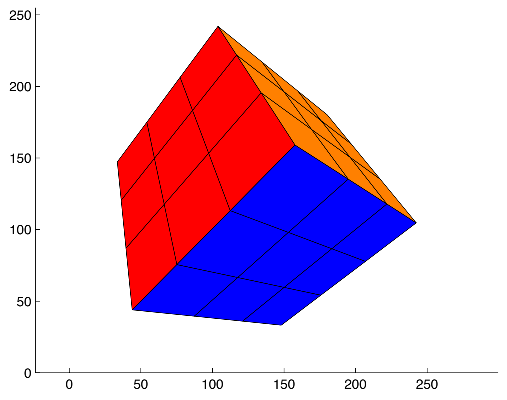
It is recommended to do so because as error gets high, the last two eigenvalues get closer and closer and the corresponding eigenvectors can switch roles. Now while this is a good practice, it does not always prevent the nonlinear effect from falling into a worse approximation. In fact after doing some empirical tests they recommend it is best to simply abandon the nonlinear adjustment under extremely high amounts of noise and solely use the eight point linear approximation.

3.7 Some Examples of Reconstruction from Simulated and Actual Images

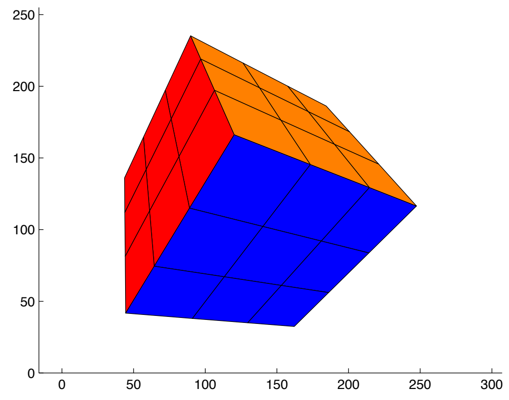
We give a few examples of the results from using the eight point algorithm and the nonlinear adjustment for reconstruction.

3.7.1 Rubik's Cube Simulation. We started by creating a virtual Rubik's Cube in MATLAB and simulating two camera projections. We then add noise to the data and run it through our reconstruction algorithm. The two camera images are shown in Figure 3.7.1 and the 3D reconstruction is shown in Figure 3.7.1.

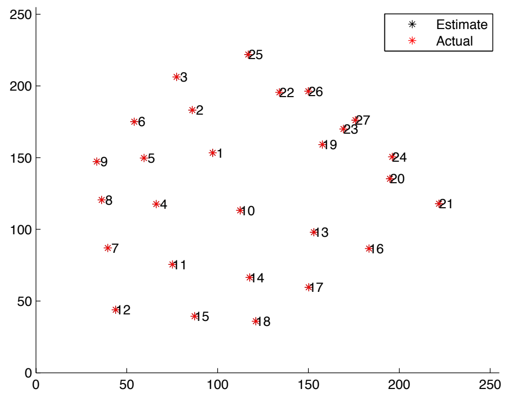
3.7.2 An Example with an Actual Camera. We next take two pictures of the bookshelf in my office using the camera on my computer. We use a feature finding and tracking algorithm, as described in Chapter 4 and 11 of [11], to identify 49 likely corresponding sets of points. We then run those points through our linear 8 point algorithm and and nonlinear adjustment algorithms to obtain the 3D reconstruction seen in Figure 3.7.2.



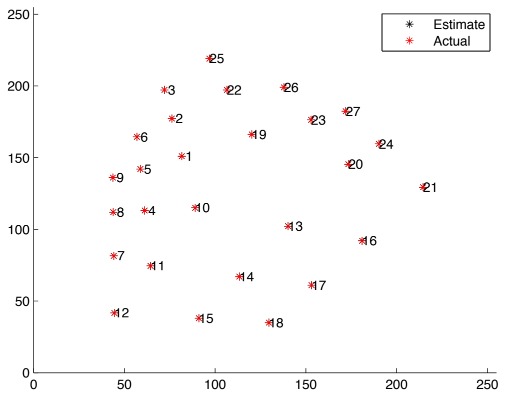
(a) Rubiks cube camera view 1



(b) Rubiks cube camera view 2

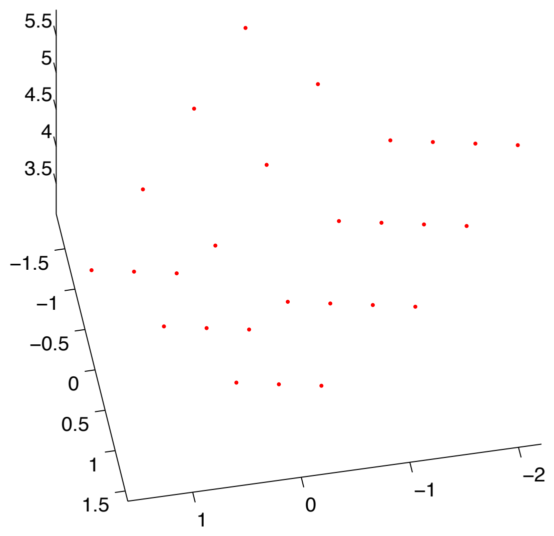


(c) Rubiks cube reconstruction view 1

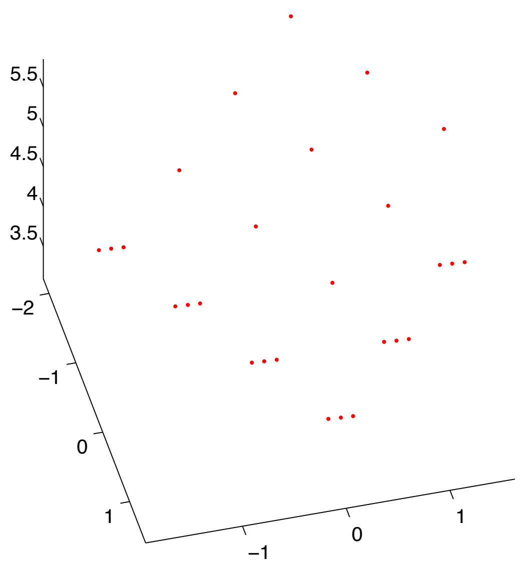


(d) Rubiks cube reconstruction view 2

Figure 3.9: The simulated reconstruction of three faces of a Rubik's cube from two camera views. The points are reconstructed in 3D then reprojected onto the image plane and overlaid on the originals. It is possible to find deviations in the re-projection from the original but they are very close.

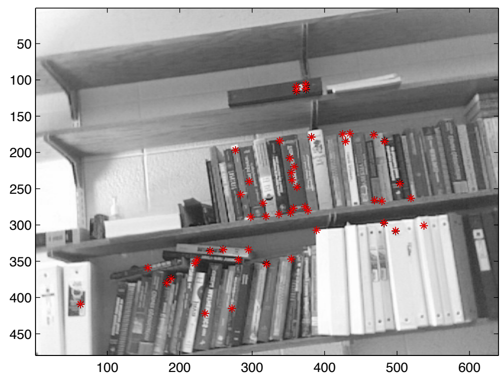


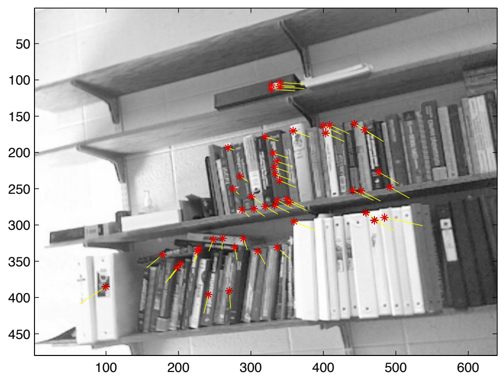
(a) Rubiks cube 3D reconstruction

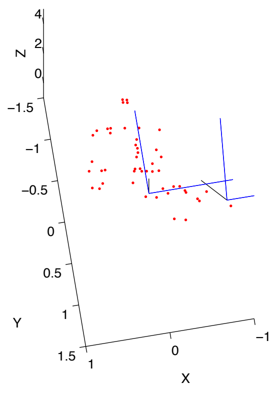


(b) Rubiks cube 3D reconstruction from different view

Figure 3.10: The 3D structure of the points on the Rubik's cube is apparent. We see that the three sides have points that are parallel and the shape is what we would expect. These views are different from the locations that the cameras took. Once the 3D structure is known, we can simulate the camera view from a unique position.


(a) Camera 1 view of bookshelf


(b) Camera 2 view of bookshelf


(c) 3D reconstruction view 1

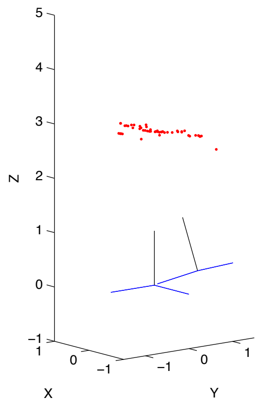

(d) 3D reconstruction view 2

Figure 3.11: We use a camera to obtain two views of a bookshelf as seen in Figures 3.11(a) and 3.11(b). The points are identified, tracked and correlated then are passed through our reconstruction algorithm to obtain the 3D structure as seen in Figures 3.11(c) and 3.11(d). The axes represent the orientation of each camera.

We give a short but complete description of a dense stereo reconstruction model that leads to a partial differential equation with motion in the normal direction, which can be solved using level set methods. This model was introduced by Olivier Faugeras and Renaud Keriven in [3] and later enhanced in Chapter 18 of [15] which was written by Faugeras, Gomes, and Keriven.

We suppose that objects in our scene are given by a smooth C^2 surface, $\hat{S}$ in R^3. The goal is to derive a partial differential equation that will start with an initial surface S_0 and evolve the surface in time toward $\hat{S}$ according to

$$\mathbf{S}_t = \beta \mathbf{N}$$

where $\mathbf{S}$ is a point on the surface and $\mathbf{N}$ the inner unit normal. Essentially we must come up with a β which will cause our initial surface to evolve in the normal direction which has the surface $\hat{S}$ as the steady state.

We let $I_i(m)$ represent the image intensity of pixel m in camera i. Intensity refers to the color or greyscale value of the image pixel. So I_i represents the view of camera i of the scene.

We define an functional which will serve as an error measure for our approximation to the surface. We wish to find a surface which will minimize it and so the calculus of variations tells us that it's Euler Lagrange equation must be satisfied. As we will see shortly the Euler-Lagrange equation gives us a partial differential equation which is characterized by motion in the normal direction of the surface. Hence, it can be solved using the level set method. This Euler-Lagrange equation will thus give us the forcing function, β, for our level set equation.

We let our error measure functional be

$$C(\mathbf{S},\mathbf{N}) = \int_S \varphi(\mathbf{S},\mathbf{N}) d\sigma$$

where φ measures the error made in the 3D reconstruction of assuming there is an object at point **S** with normal **N**. The integration is taken over the entire surface. We use the normalized cross correlation as our error measure which will be explained hereafter. This compares small windows around each point to see how similar they are and returns a value between -1 and 1. A value of 1 is strong correlation and will correspond to a good surface reconstruction while the small correlation or negative values will signify big changes needed in the surface location. We allow for multiple cameras and the fact that not every camera can see every point, called **occlusion** in the literature. Thus the correlation between camera i and camera j at a point § which has coordinate m_i in camera i image plane is given by

$$\rho_{ij} = \frac{\langle I_i, I_j, \rangle}{|I_i||I_j|}(\mathbf{S},\mathbf{N},m_i)$$

where cross correlation is essentially integration over windows of size $(2p+1)\times(2q+1)$ pixels around a point m_i and its corresponding pixel m_j in the other camera. We make some modifications to the second window which correspond to it being the reflection of the first window off the tangent plane at the point on the surface into the second image plane. We will formally describe it in Chapter 5.

$$\langle I_i, I_j\rangle(\mathbf{S},\mathbf{N}) \quad = \quad \frac{1}{4pq}\int_{-p}^{p}\int_{-q}^{q}\left[I_i(m_i+m) - \bar{I}_i(m_i)\right]\left[I_j(m_j+Am) - \bar{I}_j^*(m_j)\right] dm$$

and

$$|I_i| = \sqrt{\langle I_i, I_i\rangle}$$

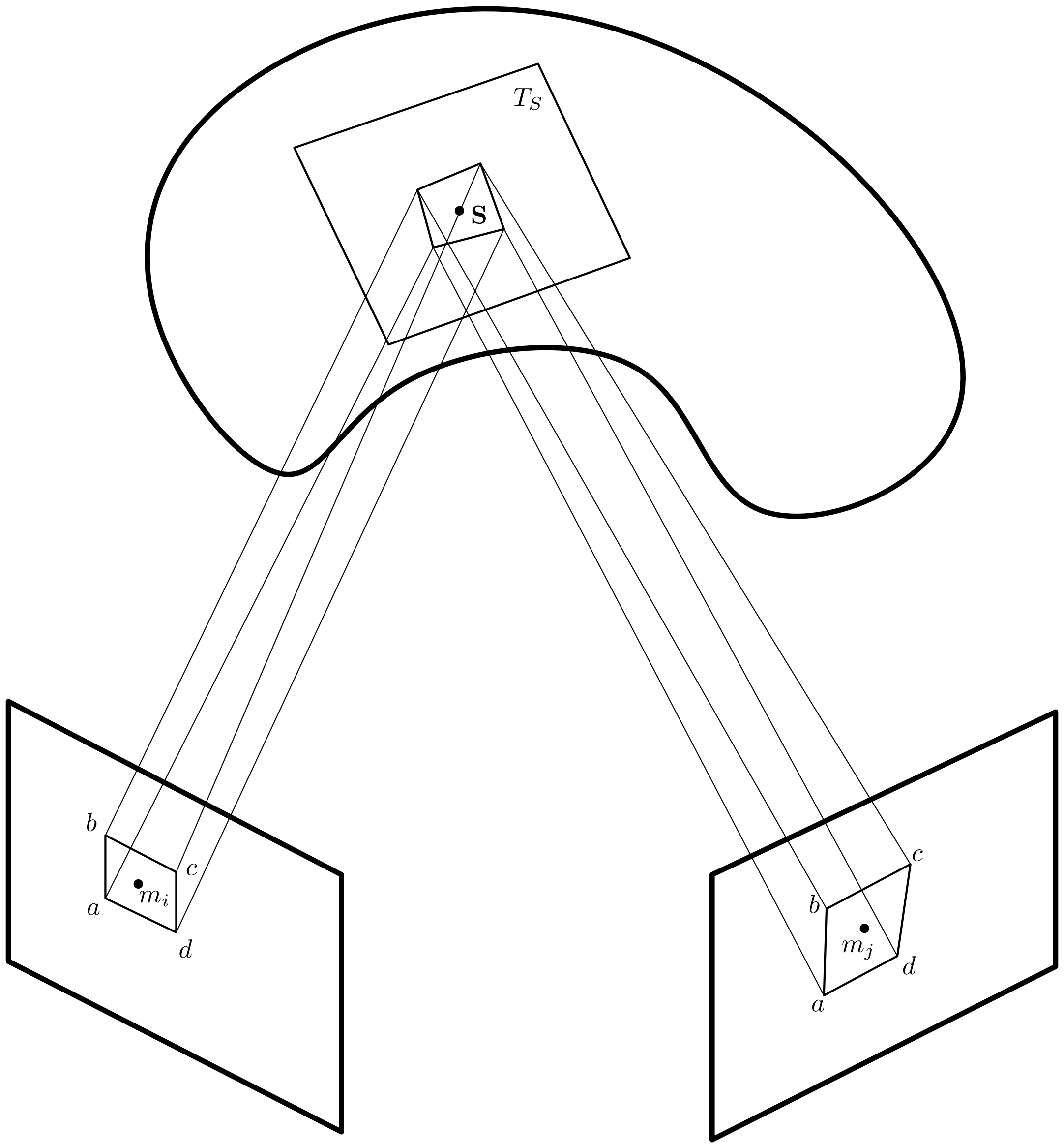


Figure 4.1: The window of size $(2p+1) \times (2q+1)$ pixels around the point m_i projected from camera i under the planar homography, K, is no longer necessarily rectangular in camera j around m_j.

Now to model occlusion, we let Γ be the set of cameras which can see the point $\mathbf{S}$ on the surface, then we let φ take into account the average of all the correlations of cameras which can see $\mathbf{S}$. Finally in order to have φ stay positive which causes us to have a stable evolution, we let g be any decreasing function that maps our correlation to positive values, ie. $g : [-1, 1] \to \mathbb{R}^+$. Then

$$\varphi(\mathbf{S}, \mathbf{N}) = g(\frac{1}{|\Gamma|(|\Gamma| - 1)} \sum_{i,j \in \Gamma, i \neq j} \rho_{ij})$$

For practical purposes, this is a symmetric error criterion which takes into account occlusion and visibility that works for multiple cameras. We will typically let $g(x) = 1 - x$ but any decreasing function g that satisfies the above will work.

Now, the Euler Lagrange equations for this model lead to a partial differential equation with movement solely in the normal direction and a magnitude of

$$\beta = 2H\varphi - (\varphi_{\mathbf{S}} + 2H\varphi_{\mathbf{N}}) \cdot \mathbf{N} - \text{Trace}\left((\varphi_{\mathbf{SN}})_{T_S} + \partial\mathbf{N} \circ (\varphi_{\mathbf{NN}})_{T_s}\right)$$

where everything is evaluated at the point $\mathbf{S}$ with normal $\mathbf{N}$ of the surface. T_S denotes the tangent plane to the surface at $\mathbf{S}$ and $d\mathbf{N}$ denotes the differential of the Gauss map of the surface. Finally H is the mean curvature of the surface and $\varphi_{\mathbf{SN}}$, $\varphi_{\mathbf{NN}}$ are the second order derivatives of φ with $(\varphi_{\mathbf{SN}})_{T_S}$ and $(\varphi_{\mathbf{NN}})_{T_S}$ being their restrictions to the tangent space T_S. The proof of this result can be found in the Appendix B of [3].

Now for our level set implementation, we embed the surface $\mathbf{S}$ as the zero level set of a function $u(\mathbf{X}, t)$ where $\mathbf{X} \in \mathbb{R}^3$. Then our surface at time t is given by

$$S(t) = \{\mathbf{X} \in \mathbb{R}^3 \mid u(\mathbf{X}, t) = 0\}$$

and evolves according to the level set equation

$$u_t(\mathbf{X}, t) = \beta(\mathbf{X}, t)|\nabla u(\mathbf{X}, t)|.$$

With this embedding, we have that the inward normal is given by

$$\mathbf{N} = -\frac{\nabla u}{|\nabla u|}$$

and

$$2H = \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right).$$

So our forcing function β is given by

$$\beta = \varphi \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) + \left(\varphi_{\mathbf{S}} + \varphi_{\mathbf{N}} \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right)\right) \cdot \frac{\nabla u}{|\nabla u|} - \operatorname{Trace}\left((\varphi_{\mathbf{SN}})_{T_S} + \partial\mathbf{N} \circ (\varphi_{\mathbf{NN}})_{T_s}\right)$$

Hence we only need to calculate $\varphi_{\mathbf{S}}$, $\varphi_{\mathbf{N}}$, $\varphi_{\mathbf{SN}}$, and $\varphi_{\mathbf{NN}}$ which will be given in Section 5.2. We also note that β is only truly defined on the surface and so we need to extend it to all other points in $\mathbb{R}^3$ which will be explained in Section 5.4.

Chapter 5. Elements of Level Set Method Stereo Problem

We give the specifics of our level set model for stereo surface reconstruction from n camera views taken simultaneously of a scene. The following will work for a general number of camera shots of the scene and as such we will give the general formula and then do the specifics in terms of two cameras. We assume our object will be constructed in the coordinate system of camera 1. Thus

$$\mathcal{P}_1 = [I_3, 0] \in \mathbb{R}^{3\times 4}$$

and we assume we know the Euclidean transformation from camera 1 to camera 2 given by

$$\mathcal{P}_2 = [R, T] \in \mathbb{R}^{s\times 4}.$$

We make the simplifying assumption that the tangent plane at S well approximates the surface nearby and so there is a planar homography, $K \in \mathbb{R}^{3\times 3}$, that maps from coordinate m_i on the focal plane of camera i to m_j on camera j. Where we typically will normalize by the third component of m_i or m_j to get the 2D coordinate on the plane

$$m_i = [x_i, y_i, 1]^T \simeq [x_i, y_i]^T.$$

Then for a general 3-D point, M_1, on a surface in camera 1 coordinates, the corresponding M_2 in camera 2 coordinates is given by

$$\begin{aligned} M_2 &= RM_1 + T \\ &= RM_1 + T\left(\frac{N^T M_1}{d}\right) \\ &= \left(R + \frac{TN^T}{d}\right) M_1 \\ &= KM_1 \end{aligned}$$

where N is the unit normal vector to the surface at M_1 in the first camera's coordinate system. Therefore we let $d = N^T M_1$ be the distance from the origin to the tangent plane to the surface at M_1. We say K is the planar homography corresponding to M_1 and M_2. As a side note, the same could be done for the normalized points m_i and m_j and the result is that in their respective focal planes,

$$m_2 \simeq Km_1.$$

Finally, our error functional, $C(S) = \int_S \varphi(\mathbf{S}, \mathbf{N}) d\sigma$, gives us Euler-Lagrange equations characterized by motion in the normal direction. We can represent the system in the level set equation

$$u_t = \beta |\nabla u|$$

with $u(X, 0) = S_0$ given by some initial surface like a cylinder and magnitude of the forcing function given by

$$\beta = \begin{cases} 2H\varphi & \text{regularization term} \\ -(\varphi_{\mathbf{S}} + 2H\varphi_{\mathbf{N}}) \cdot \mathbf{N} & \text{first order data term} \\ -\operatorname{Trace}\left((\varphi_{\mathbf{SN}})_{T_S} + \partial\mathbf{N} \circ (\varphi_{\mathbf{NN}})_{T_s}\right) & \text{higher order terms} \end{cases}$$

where each is evaluated at point **S** with normal **N** of the surface. T_S is the tangent plane to the surface at S. H is the mean curvature, and dN is the derivative of the Gauss Map. As a side note, the authors of this model recommend that the initial surface completely enclose the desired final result to give optimal convergence under this pde. Also, the authors point out in [15] that simulations with the higher order terms in β were not significantly different from simulations without them. We may, in cases where speed is an issue, neglect to compute the higher order terms and our result will not be noticeably different that otherwise.

5.1 The measure of error, $\varphi(\mathbf{S},\mathbf{N})$

Let $\varphi(S,N)$ be the functional defined by

$$\varphi(S,N) = g\left(\frac{1}{|\Gamma||\Gamma-1|}\sum_{ij\in\Gamma,i\neq j}\frac{\langle I_i,I_j\rangle}{|I_i||I_j|}(S,N)\right)$$

where $g(x)=1-x$, Γ is the set of cameras which can see the point S and,

$$\langle I_i,I_j\rangle(S,N) \;=\; \frac{1}{4pq}\int_{-p}^{p}\int_{-q}^{q}\left[I_i(m_i+m)-\bar{I}_i(m_i)\right]\left[I_j(m_j+Am)-\bar{I}_j^*(m_j)\right]dm$$

We note that $m_i=\mathcal{P}_iS$ and where we denote the integral over the box around m_i or m_j, by

$$\int^* = \frac{1}{4pq}\int_{-p}^{p}\int_{-q}^{q}$$

and the bar denotes the averages

$$\begin{aligned}\bar{I}_i(m_i) &= \int^* I_i(m_i+m)dm\\ \bar{I}_j(m_j) &= \int^* I_j(m_j+m)dm\\ \bar{I}_j^*(m_j) &= \int^* I_j(K(m_i+m))dm\end{aligned}$$

We simplify our later calculations by approximating K by an affine transformation $K(m_i + m) = m_j + Am+$ higher order terms, where if the row vectors of K are

$$K = \begin{bmatrix} -k_1- \\ -k_2- \\ -k_3- \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

then since $m = [x, y, 0]^T$ we get the first order approximation to K

$$A = \frac{1}{k_3 m_i} \begin{bmatrix} k_1 - x_j k_3 \\ k_2 - y_j k_3 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

Finally we remove the last column and write $m = [x, y]^T$ to have $A \in \mathbb{R}^{2\times 2}$. We will also need the derivatives of K and A in the tangential and normal directions so since $K = dR + T\mathbf{N}^T$, the derivative with respect to N_i, each element of $\mathbf{N}$ is

$$K_{N_i} = d_{N_i} R - T N_{N_i}^T = S_i R + T \cdot [0, 1, 0]$$

where the 1 is in the i-th spot of the last vector. Thus we can calculate $(Am)_{N_i}$ which will be given by its product with the vector $m = [x, y]^T$ and the rows of $K_{N_i} = [(k_1)_{N_i}; (k_2)_{N_i}; (k_3)_{N_i}]$.

$$(Am)_{N_i} = -\frac{(k_3)_{N_i} m_i}{k_s m_i} Am + \frac{1}{k_3 m_i} \begin{bmatrix} (k_1)_{N_i} - x_j (k_3)_{N_i} \\ (k_2)_{N_i} - y_j (k_3)_{N_i} \end{bmatrix} m \in \mathbb{R}^{2\times 1}$$

Then

$$(Am)_N = [(Am)_{N_1}, (Am)_{N_2}, (Am)_{N_3}] \in \mathbb{R}^{2\times 3}.$$

Similarly, we get the derivative with respect to S_i in $\mathbf{S}$.

$$K_{S_i} = d_{S_i} R = N_i R$$

and

$$(Am)_{S_i} = -\frac{(k_3)_{S_i} m_i}{k_s m_i} Am + \frac{1}{k_3 m_i} \begin{bmatrix} (k_1)_{S_i} - x_j (k_3)_{S_i} \\ (k_2)_{S_i} - y_j (k_3)_{S_i} \end{bmatrix} m \in \mathbb{R}^{2\times 1}$$

So

$$(Am)_S = [(Am)_{S_1}, (Am)_{S_2}, (Am)_{S_3}] \in \mathbb{R}^{2\times 3}$$

Now we will solve this system using the level set methods so the surface is given by $\{\mathbf{X} \in \mathbb{R}^3 \mid u(\mathbf{X}, t) = 0\}$. So our level set equation is given by

$$u_t(\mathbf{X}, t) = \beta(\mathbf{X}, t) |\nabla u(\mathbf{X}, t)|$$

Now, the normal to the surface at any point is given by

$$N = -\frac{\nabla u}{|\nabla u|}$$

On the surface of the object, the parameterization of the point can be given by $S(v, w) = (v, w, \frac{-vu_x - wu_y}{u_z} + c)$, where c is a constant, so we have

$$\begin{aligned} S_v &= (1, 0, -\frac{u_x}{u_z})^T \\ S_w &= (0, 1, -\frac{u_y}{u_z})^T \end{aligned}$$

We can likewise calculate the unnormalized vectors

$$\begin{aligned} N_v &= -\frac{\partial}{\partial x} \frac{\nabla u}{|\nabla u|} \\ N_w &= -\frac{\partial}{\partial y} \frac{\nabla u}{|\nabla u|} \end{aligned}$$

Thus we can calculate the differential of the Gaussian Map, ∂N

$$\partial N = \begin{bmatrix} \frac{e}{E} & \frac{f}{h} \\ \frac{f}{h} & \frac{g}{G} \end{bmatrix}$$

where

$$\begin{aligned} e &= -N_v \cdot S_v, & E &= S_v \cdot S_v \\ f &= -N_w \cdot S_v, & F &= S_v \cdot S_w \\ g &= -N_w \cdot S_w, & G &= S_w \cdot S_w \\ h &= S_v \times S_w \end{aligned}$$

5.2 Formal Definitions of the derivatives of φ

Finally, we will need φ_S, φ_N, φ_{SN} and φ_{NN} to calculate everything that goes into our forcing function, β.

$$\begin{aligned} \varphi_S &= \frac{\partial g}{\partial x} \sum_{i,j\in\Gamma,i\neq j} \frac{\langle I_i, I_j\rangle_S}{|I_i||I_j|} - \frac{\langle I_i, I_j\rangle}{|I_i|^2|I_j|^2} (|I_i||I_j|)_S \\ \varphi_N &= \frac{\partial g}{\partial x} \sum_{i,j\in\Gamma,i\neq j} \frac{\langle I_i, I_j\rangle_N}{|I_i||I_j|} \\ \varphi_{SN} &= \frac{\partial g}{\partial x} \sum_{i,j\in\Gamma,i\neq j} \frac{\langle I_i, I_j\rangle_{SN}}{|I_i||I_j|} - \frac{\langle I_i, I_j\rangle_N^T}{|I_i|^2|I_j|^2} (|I_i||I_j|)_S \\ \varphi_{NN} &= \frac{\partial g}{\partial x} \sum_{i,j\in\Gamma,i\neq j} \frac{\langle I_i, I_j\rangle_{NN}}{|I_i||I_j|} \end{aligned}$$

Now in order to calculate each of these quantities, we will ned the derivatives of our correlation measure. For a point on the surface $S = (S_1, S_2, S_3)$, we give the notation

$$\langle \nabla I_i^T, I_j^* \rangle \frac{\partial m_i}{\partial S} = \int^* \nabla I_i(m_i + m)^T \frac{\partial m_i}{\partial S} \left[I_j(m_j + Am) - \bar{I}_j^*(m_j) \right] dm$$

So likewise,

$$\langle I_i, \nabla I_j^{*T} \rangle \frac{\partial m_j}{\partial S} = \int^* \left[I_i(m_j + m) - \bar{I}_i(m_i) \right] \nabla I_j(m_j + Am)^T \frac{\partial m_j}{\partial S} dm.$$

We let R_k denote the k-th row of our rotation matrix R and R_k^j the j-th element of the k-th row. Likewise we let $T = [T_1, T_2, T_3]^T$. Then for $m_i = S/S(3)$ and $m_j = \frac{RS+T}{R_3 S + T_3}$, we get

$$\frac{\partial m_i}{\partial S} = \begin{bmatrix} 1/S(3) & 0 & -\frac{S(1)}{S(3)^2} \\ 0 & 1/S(3) & -\frac{S(2)}{S(3)^2} \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

$$\frac{\partial m_j}{\partial S} = \frac{1}{(R_3 S + T_3)^2} \begin{bmatrix} R_1(R_3 S + T_3) - R_3(R_1 S + T_1) \\ R_2(R_3 S + T_3) - R_3(R_2 S + T_2) \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

and $\nabla I_i \in \mathbb{R}^{3\times 1}$ $(\mathbb{R}^{2\times 1})$.

5.3 Derivatives of the correlation integral

Then the derivative $\langle I_i, I_j \rangle_S \in R^{1\times 3}$ is given by

$$\langle I_i, I_j \rangle_S = \langle \nabla I_i^T, I_j^* \rangle \frac{\partial m_i}{\partial S} + \langle I_i, \nabla I_j^{*T} \rangle \left(\frac{\partial m_j}{\partial S} + (Am)_S \right)$$

The derivative $\langle I_i, I_j \rangle_N \in R^{1\times 3}$ is given by

$$\begin{aligned} \langle I_i, I_j \rangle_N &= \langle I_i^T, \nabla I_j^{*T} \rangle (Am)_N \\ &= \int^* \left[I_i(m_i + m) - \bar{I}_i(m_i) \right] \nabla I_j(m_2 + Am)^T (Am)_N dm \end{aligned}$$

Now we define the norm to be

$$|I_i|^2 = \langle I_i, I_i \rangle = \int^* \left[I_i(m_i + m) - \bar{I}_i(m_i) \right] \left[I_i(m_i + m) - \bar{I}_i(m_i) \right] dm$$

and

$$|I_j|^2 = \langle I_j^*, I_j^* \rangle = \int^* \left[I_j(m_j + Am) - \bar{I}_j^*(m_j)\right] \left[I_j(m_j + Am) - \bar{I}_j^*(m_j)\right] dm$$

thus the derivatives $|I_i|_S$ and $|I_j|_S \in \mathbb{R}^{1\times 3}$ are

$$\begin{aligned} |I_i|_S &= \frac{1}{|I_i|}\left[\langle \nabla I_i^T, I_i \rangle \frac{\partial m_i}{\partial S}\right] \\ |I_j|_S &= \frac{1}{|I_j|}\left[\langle \nabla I_j^{*T}, I_j^* \rangle \frac{\partial m_j}{\partial S} + \langle I_j^*, \nabla I_j^{*T} \rangle (Am)_S\right] \end{aligned}$$

For the higher order terms, we need $\langle I_i, I_j \rangle_{SN} \in \mathbb{R}^{3\times 3}$ which is given by

$$\begin{aligned} \langle I_i, I_j \rangle_{SN} &= \int^* \frac{\partial m_i}{\partial S}^T \nabla I_i(m_i + m) \nabla I_2(m_j + Am)^T (Am)_N dm \\ & -\frac{\partial \bar{I}_i^T}{\partial S}(m_i) \int^* \nabla I_j(m_j + Am)^T (Am)_N dm \\ & + \int^* \left[I_i(m_i + m) - \bar{I}_i(m_i)\right] \left[(Am)_S^T H_{I_j} (Am)_N + \nabla I_j^T (Am)_{SN}\right] dm \end{aligned}$$

where

$$\frac{\partial \bar{I}_i}{\partial S} = \int^* \nabla I_i^T (m_i + m) dm \frac{\partial m_i}{\partial S}$$

and H_{I_j} is the Hessian of $I_j(S, N)$.

Finally, we need the 3×3 matrix,

$$\langle I_i, I_j \rangle_{NN} = \int^* (I_i(mi + m) - \bar{I}_i(m_i)) \left[(Am)_N^T H_{I_j} (Am)_N + \nabla I_j^T A_{NN} m\right] dm$$

With all of these terms, we can calculate the desired quantities of φ and it's derivatives.

5.4 Extending β from the surface, S, to $\mathbb{R}^3$

To preserve the distance functions and speed up our final result as well as extend our β to all of $\mathbb{R}^3$ we follow the suggestion of Gomes and Faugeras in [5] to rewrite our level set equation so that the distance function is preserved over time. Their solution is to have the forcing function be defined on the zero level set and to let the extension be such that along characteristic lines, the magnitude is the same. With these assumptions, our level set equation becomes

$$u_t = \beta(X - u\nabla u)$$

where $X - u\nabla u = y$ such that $u(y) = 0$. This works well because for our specific problem, β is really only defined on the zero level set in the first place. Hence this extension fits well into our application. It is implemented using the narrow banded level set method which speeds up our calculations considerably.

Appendix A. An Introduction to Finite Differences for Numerical Analysis

A.1 What is a Finite Difference?

Consider the differential equation $y' = f(y)$ with initial condition $y(a) = y_0$ defined on some interval $I = [a, b]$. We desire to find a solution $y(x)$. Many such problems have no closed form solution and must be approximated numerically. Thus we let $\{x_0 = a, x_1, \ldots, x_{n+1} = b\}$ be a partition of I and we define $\Delta x_i = x_i - x_{i-1}$. Note that

$$\frac{y_{i+1} - y_i}{\Delta x_{i+1}} \approx f(y_i),$$

so that $y_{i+1} \approx y_i + f(y_i)\Delta x_{i+1}$. In general, we approximate $f^{(n)}(x_i)$ using a linear combination of $\{f(x_j)\}_{j=0}^{n+1}$. A finite difference approximation for $f^{(n)}(x_i)$ is a linear combination $\sum_{j=0}^{n+1} a_j f(x_j)$. A finite difference method creates a particular discretized system using finite differences which approximate the differential equation we wish to solve. We hope that this discretized system will accurately approximate the solution of the differential equation. For the purposes of our discussion here, we will deal with functions $f(x, t)$ of time and one dimension of space. We note that finite difference methods in multiple spatial dimensions are treated similarly.

Given a point x_i in a grid of points where $x_{i+1} - x_i = h$ for all i, (sometimes we use h to represent the grid size, and other times we will explicitly say Δx or Δt, especially if the sizes are different) we will use the Taylor series of $f(x)$ for points in neighborhood of x_i around x_i. As an example, say we wish to approximate $f'(x_i)$ using a forward difference, meaning we will use the points x_i and x_{i+1}, then the Taylor series of $f(x_{i+1})$ around x_i is

$$f(x_{i+1}) = f(x_i + h) \quad = f(x_i) + f'(x_i)(h) + f''(x_i)\tfrac{(h)^2}{2} + \mathcal{O}(h^3),$$

so if we subtract over $f(x_i)$ and solve for $f'(x_i)$ to get

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + f''(x_i)\frac{(h)}{2} + \mathcal{O}(h^2),$$

so we get our forward difference approximation of the first derivative at x_i

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + \mathcal{O}(h).$$

We note that this is a first order approximation to the first derivative. We say a finite difference approximating an n-th derivative is of order m if we can write it as a function of its neighbors, $f^{(n)}(x_i) = \sum_{j=0}^{n+1} a_j f(x_j) + \mathcal{O}(h^m)$.

Likewise, we could do a backward difference (using x_i and x_{i-1})

$$f(x_{i-1}) = f(x_i - h) = f(x_i) + f'(x_i)(-h) + f''(x_i)\frac{(-h)^2}{2} + \mathcal{O}(h^3)$$

so isolating $f'(x_i)$, we get our first order backward difference scheme for f',

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + \mathcal{O}(h).$$

We can do this in general for any set of grid points in a neighborhood of x_i, for example we have a second order centered difference (using an balanced amount of points on either side of x_i like for instance using x_{i+1}, x_i, and x_{i-1}) approximation to the first derivative We use the taylor series

$$\begin{aligned} f(x_{i+1}) &= f(x_i) + f'(x_i)(h) + f''(x_i)\frac{(h)^2}{2} + f'''(x_i)\frac{(h)^3}{6} + \mathcal{O}(h^4) \\ f(x_{i-1}) &= f(x_i) + f'(x_i)(-h) + f''(x_i)\frac{(-h)^2}{2} + f'''(x_i)\frac{(-h)^3}{6} + \mathcal{O}(h^4), \end{aligned}$$

so subtract the second from the first to get

$$f(x_{i+1}) - f(x_{i-1}) = 2f'(x_i)(h) + \mathcal{O}(h^3).$$

So we get

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + \mathcal{O}(h^2).$$

Notice that the h^2 terms in the Taylor series cancel each other out giving us, after dividing by h, a second order centered difference approximation to the first derivative. Likewise, we can solve for a centered difference approximation to the second derivative. Add the two above taylor series and subtract $2f(x_i)$ to get

$$f(x_{i+1}) - 2f(x_i) + f(x_{i-1}) = f''(x_i)(h)^2 + \mathcal{O}(h^4)$$

so we have

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} + \mathcal{O}(h^2).$$

Sometimes when we are dealing with the edges of our grid, we may need to create a higher order finite difference approximation which is one sided. For instance, to create a second order approximation at a point x_i using x_{i-1} and x_{i-2}, we first write out our Taylor series

$$f(x_{i-1}) = f(x_i) - f'(x_i)h + f''(x_i)\frac{h^2}{2} - f'''(x_i)\frac{h^3}{6} + f^{(4)}(x_i)\frac{h^4}{12} + \mathcal{O}(h^5),$$
$$f(x_{i-2}) = f(x_i) - 2f'(x_i)h + 2f''(x_i)h^2 - \frac{4}{3}f'''(x_i)h^3 + \frac{4}{3}f^{(4)}(x_i)h^4 + \mathcal{O}(h^5).$$

We want to remove the h^2 terms when we add them so we add -4 copies of the first equation to the second to get

$$-4f_{i-1} + f_i = -4f_i + f_i + 4hf_i' - 2hf_i' + \frac{2}{3}h^3 f_i''' - \frac{4}{3}h_i''' + \mathcal{O}(h^4)$$

and solving for f_i' we get the scheme

$$f_i' = \frac{3f_i - 4f_{i-1} + f_{i-2}}{2h} + \mathcal{O}(h^2).$$

A.1.1 General Method to Find Finite Difference Scheme. In order to find the combination of neighbor points that will yield an approximation to any derivative desired, we first give a general form of the Taylor series about a point x_i. For any number j, the Taylor series of $f(x_{i+j})$ around x_i is

$$f(x_{i+j}) = f(x_i + jh) = f(x_i) + f'(x_i)(jh) + f''(x_i)\frac{(jh)^2}{2} + f'''(x_i)\frac{(jh)^3}{3!} + \dots$$

We denote the set of points which will be used in our scheme around x_i by $J = [j_1 \ j_2 \ \dots \ j_n]$ where for example $j_l = 2$ refers to x_{i+2} or $j_l = -1$ refers to x_{i-1}. Then using the general Taylor series form above, we can write our set of Taylor series for each

$j \in J$ into a system of linear equations as

$$\begin{bmatrix} 1 & j_1h & \frac{(j_1h)^2}{2!} & \dots & \frac{(j_1h)^{n-1}}{(n-1)!} \\ 1 & j_2h & \frac{(j_2h)^2}{2!} & \dots & \frac{(j_2h)^{n-1}}{(n-1)!} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & j_nh & \frac{(j_nh)^2}{2!} & \dots & \frac{(j_nh)^{n-1}}{(n-1)!} \end{bmatrix} \begin{bmatrix} f(x_i) \\ f'(x_i) \\ \vdots \\ f^{(n-1)}(x_i) \end{bmatrix} = \begin{bmatrix} f(x_i + j_1h) \\ f(x_i + j_2h) \\ \vdots \\ f(x_i + j_nh) \end{bmatrix} + \mathcal{O}(h^n).$$

If we denote this matrix as A, the vectors of derivatives of $f(x_i)$ as F', and the vector of $f(x_{i+j})$ for $j \in J$, as F_J, then we have

$$AF' = F_J + \mathcal{O}(h^n),$$

We note that this matrix A is a vandermonde matrix and is known to be ill-conditioned in numerical linear algebra, but luckily we will not be creating matrices of larger size than probably even 6×6 or 7×7 and usually smaller so this shouldn't cause us much problem.

If we define $C = A^{-1}$, then the ith row of C contains the coefficients for the $(i-1)$th derivative of $f(x_i)$ for a scheme of order $\mathcal{O}(h^{n-i+1})$. In other words we then have

$$F' = CF_J + \begin{bmatrix} \mathcal{O}(h^n) \\ \mathcal{O}(h^{n-1}) \\ \vdots \\ \mathcal{O}(h^1) \end{bmatrix},$$

or if we denote C_i to be the ith row of C, then

$$f^{(i-1)}(x_i) = C_i \begin{bmatrix} f(x_i + j_1h) \\ f(x_i + j_2h) \\ \vdots \\ f(x_i + j_nh) \end{bmatrix} + \mathcal{O}(h^{n-i+1}) = C_iF_J + \mathcal{O}(h^{n-i+1}).$$

A.2 Finite Difference Schemes

Now, when we have a time dependent ODE or PDE, we can combine our finite difference approximations to form two kind of schemes which approximate our differential equation which are explicit schemes and implicit schemes. (Later on we will study combinations of explicit and implicit sometimes called hybrid schemes. In particular we will study the **Crank-Nicolson** method which is a powerful hybrid which will be used extensively for nonlinear wave equations).

An **explicit scheme** is one which calculates the state of the system at a future time from the state at the current time. In other words, we let a superscript, f^n denote the state of f at time n in our grid, then for a linear system we have some matrix operator A which gives us the change from time n to $n+1$ as

$$Af^n + b = f^{n+1}$$

for some b. And if our system is nonlinear, then we have some operator G such that

$$G(f^n) = f^{n+1}.$$

An **implicit scheme** finds a solution for the future time step by solving a system of equations involving future and current time steps. In other words, for a linear system we have some matrix operator B that gives us

$$Bf^{n+1} + d = f^n$$

for some vector d, and for a nonlinear system, we have some operator H such that

$$H(f^{n+1}) = f^n.$$

A **hybrid scheme** uses a combination of explicit and implicit schemes in an effort to obtain higher order convergence and maintain the stability of the implicit schemes. The most famous of these hybrid schemes is the Crank-Nicolson scheme which is accomplished by considering ourselves standing at the $t + 1/2$ time step and then using an average of implicit and explicit schemes in time. We will discuss the details of this below along with two examples: the linear heat equation and the nonlinear burger's wave equation.

A.2.1 Finite Difference Schemes for ODE.

We will give examples of explicit and implicit schemes as we attempt to solve the equation

$$\frac{dy}{dt} = sy, \qquad s > 0,$$

with initial condition $y(0) = y_0$ which has analytic solution $y(t) = y_0 e^{st}$.

For our explicit scheme, we will use the forwards Euler's method. We call it forward because we use a forward difference in time. Then we have

$$\frac{y^{n+1} - y^n}{\Delta t} = sy^n + \mathcal{O}(\Delta t),$$

so we get

$$y^{n+1} - y^n = s\Delta t y^n,$$

or

$$y^{n+1} = (1 + s\Delta t)y^n,$$

as our update. Thus given the initial data $y^0 = y_0$, we make each time step forward by calculating the new value using the above scheme. A solution was constructed in Matlab to solve the equation $y' = y$ from time 0 to time 5 using 100 time steps and initial data $y_0 = 1$. We display the results of our solution computed using the forward Euler's method,

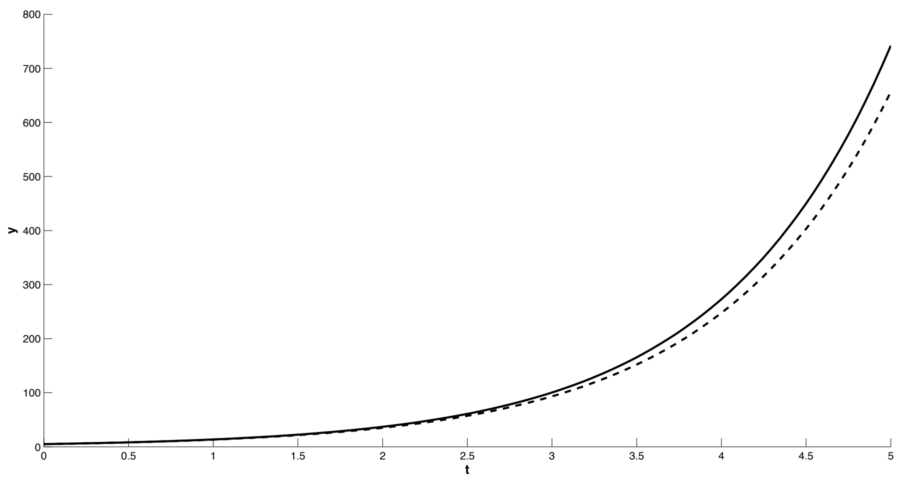


Figure A.1: The ODE, $y' = sy$, solved using forwards Euler scheme in a MATLAB plot. The solid line is the true solution and the dashed line is the approximation.

our explicit scheme.

We now solve this same system using an implicit scheme and then compare the benefits and disadvantages of both methods. For our implicit method, we suppose we are sitting at time $n + 1$ and then use the backward difference for our time derivative

$$\frac{y^{n+1} - y^n}{\Delta t} = sy^{n+1} + \mathcal{O}(\Delta t),$$

which when simplified gives us the update

$$(1 - s\Delta t)y^{n+1} = y^n$$

or

$$y^{n+1} = \frac{1}{1 - s\Delta t} y^n.$$

So for the same example above, we also display the solution computed using the backward Euler's method in Figure A.2.

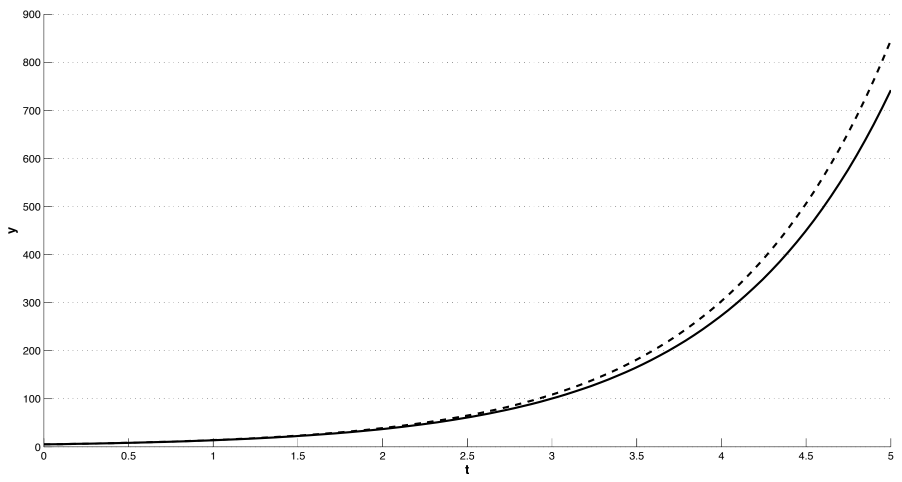


Figure A.2: The ODE, $y' = sy$, solved using backwards Euler scheme in a MATLAB plot. The solid line is the true solution and the dashed line is the approximation.

If we let the grid size $\Delta t \to 0$, we find that both of these methods approach the true solution. In this case, there was not too much difference in solving implicitly or explicitly, but for more complicated systems, the implicit method requires solving a system of equations at each time step, where as the explicit method is merely the action of an operator at each time step. For a class of differential equations which are called **stiff**, meaning there are certain parts of the equation which lead to rapid variation in the solution, the explicit methods are unstable unless we use extremely small time increments. But the implicit methods are stable for solving the stiff equations. The explicit methods generally require a much smaller time step than implicit methods, so the simplicity of each update being multiplication by a matrix requires a much greater number of evaluations to get the same quality of solution as the implicit methods.

It is up to the user to decide which schemes they decide to use for their differential equation. Each implicit or explicit method has advantages and disadvantages which must be weighed out before deciding which method is best for the specific differential equation.

A.2.2 Finite Difference Schemes for PDE. We will demonstrate an explicit and implicit scheme to solve the heat equation with diffusivity constant k, and initial heat distribution $f(x)$ for $x \in [0, l]$

$$\begin{cases} u_t = k u_{xx}, & \kappa > 0 \\ u(x,0) = f(x) & \text{initial condition} \\ u(0,t) = g(t),\ u(l,t) = h(t) & \text{boundary conditions.} \end{cases}$$

Explicit Scheme for the Heat Equation.. We will use the explicit forward-time centered-space (FTCS) method where subscripts denote location in space and superscripts denote location in time ($u(x_i, t_n) = u_i^n$). Then the approximation for the equation at $u(x_i, t_n)$ is

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = k\left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}\right),$$

which is $\mathcal{O}(\Delta t, (\Delta x)^2)$. Then if we let $r = \frac{k\Delta t}{(\Delta x)^2}$, we get the simplified update

$$\begin{aligned} u_i^{n+1} &= u_i^n\left(1 - \frac{2k\Delta t}{(\Delta x)^2}\right) + \frac{k\Delta t}{(\Delta x)^2}[u_{i+1}^n + u_{i-1}^n] \\ &= r u_{i-1}^n + (1-2r)u_i^n + r u_{i+1}^n. \end{aligned}$$

We notice that the next time step at x_i is updated by the values at the previous time step of x_{i-1}, x_i, and x_{i+1} which we display using a **stencil**. This type of stencil is characteristic of explicit schemes.

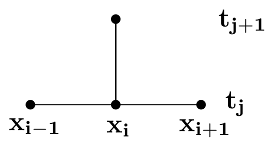


Figure A.3: Stencil for explicit (FTCS) scheme on heat equation.

Since our update equation is linear, we can create a matrix M which represents the update for all x_i to the next time step. We suppose that there are $m+2$ points in our grid in space numbered 0 to $m+1$ for each time step, and that the boundary values u_0 and u_{m+1} are given to us. (Remember that in order to have a unique solution, we must have boundary conditions defined and known for all all time.) Our system is then

$$M\vec{u}^n + \vec{b}^n = \vec{u}^{n+1},$$

where M is an $m \times m$ matrix

$$M = \begin{bmatrix} 1-2r & r & 0 & 0 & \dots & 0 \\ r & 1-2r & r & 0 & \dots & 0 \\ 0 & r & 1-2r & r & & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & & & r & 1-2r & r \\ 0 & 0 & \dots & 0 & r & 1-2r \end{bmatrix},$$

and b is an $n \times 1$ vector,

$$\vec{b}^n = \begin{bmatrix} ru_0^n \\ 0 \\ \vdots \\ 0 \\ ru_{m+1}^n \end{bmatrix}.$$

Written out completely, we get the system

$$\begin{bmatrix} 1-2r & r & 0 & 0 & \dots & 0 \\ r & 1-2r & r & 0 & \dots & 0 \\ 0 & r & 1-2r & r & & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & & & r & 1-2r & r \\ 0 & 0 & \dots & 0 & r & 1-2r \end{bmatrix} \begin{bmatrix} u_1^n \\ u_2^n \\ u_3^n \\ \vdots \\ u_{m-1}^n \\ u_m^n \end{bmatrix} + \begin{bmatrix} ru_0^n \\ 0 \\ 0 \\ \vdots \\ 0 \\ ru_{m+1}^n \end{bmatrix} = \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ u_{m-1}^{n+1} \\ u_m^{n+1} \end{bmatrix}$$

with initial conditions

$$\begin{bmatrix} u_1^0 \\ u_2^0 \\ u_3^0 \\ \vdots \\ u_{m-1}^0 \\ u_m^0 \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) \end{bmatrix}.$$

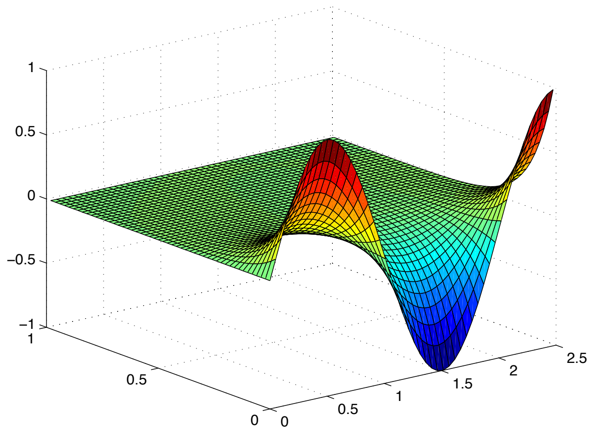
We implement this method in Matlab for the equation

$$u_t = ku_{xx}, \qquad k = 1, \tag{A.1}$$

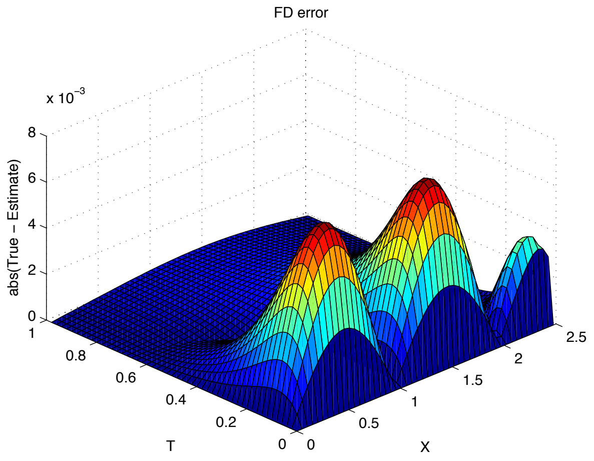
with $x \in [0, 5\pi/2]$ and $t \in [0, 1]$. We use the initial condition $f(x) = \sin(\pi x)$ and boundary conditions $f(0, t) = 0$ and $f(5\pi/2, t) = 1$. Since this is a stiff equation, the explicit method is highly unstable and we were required to make at least 3,200 time steps and only 100 space steps to deal with the stiffness.

Stability of Explicit FTCS Method for Heat Equation. As a general rule, for this FTCS method (Euler's method for time) to be stable, we must have $s = \frac{\kappa\Delta t}{(\Delta x)^2} \leq \frac{1}{2}$. This is a strict requirement and is obtained in studying the stability of the Euler's method typically using the Fourier Transform. We call this restriction a Courant-Friedrich-Levy

(CFL) Condition on grid sizes. For smaller numbers of time steps (larger time steps), our solution quickly blew up because $s > 1/2$. An explicit scheme typically requires many more time steps to achieve the convergence desired than an implicit scheme but it is typically a simple update. Our final result was fairly close to the actual solution which is known to be $F(x,t) = e^{-\pi^2 t} sin(\pi x)$. We will see quickly that the implicit method is much preferred than explicit methods for this equation.



(a) Explicit FTCS solution



(b) Absolute error in explicit FTCS solution

Figure A.4: Explicit FTCS method solution and absolute error for heat equation (A.1) in a MATLAB plot. This required a very fine grid to accomplish with this accuracy.

Implicit Scheme for the Heat Equation. Our implicit scheme is a backward-time centered-space (BTCS) where we work from the $n+1$ time step. Thus we get as our approximation of our heat equation at $u(x_i, t_n)$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = k\left(\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2}\right),$$

which is also $\mathcal{O}(\Delta t, (\Delta x)^2)$. We make the same simplification, $r = \frac{k\Delta t}{(\Delta x)^2}$, to get

$$u_i^{n+1} - u_i^n = r(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1})$$

or

$$-ru_{i+1}^{n+1} + (1+2r)u_i^{n+1} - ru_{i-1}^{n+1}) = u_i^n.$$

We note that the the next time step of x_i is dependent upon x_i in the previous step and x_{i+1}, x_{i-1} in the same time step which we also display in a stencil which is characteristic of implicit methods.

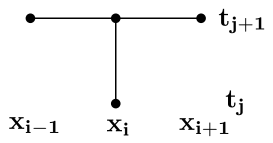


Figure A.5: Stencil for implicit (BTCS) scheme on heat equation

We can again write our system in matrix notation as

$$\begin{bmatrix} 1+2r & -r & 0 & 0 & \dots & 0 \\ -r & 1+2r & -r & 0 & \dots & 0 \\ 0 & -r & 1+2r & -r & & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & & & -r & 1+2r & -r \\ 0 & 0 & \dots & 0 & -r & 1+2r \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ u_{m-1}^{n+1} \\ u_m^{n+1} \end{bmatrix} + \begin{bmatrix} -ru_0^{n+1} \\ 0 \\ 0 \\ \vdots \\ 0 \\ -ru_{m+1}^{n+1} \end{bmatrix} = \begin{bmatrix} u_1^n \\ u_2^n \\ u_3^n \\ \vdots \\ u_{m-1}^n \\ u_m^n \end{bmatrix}$$

or

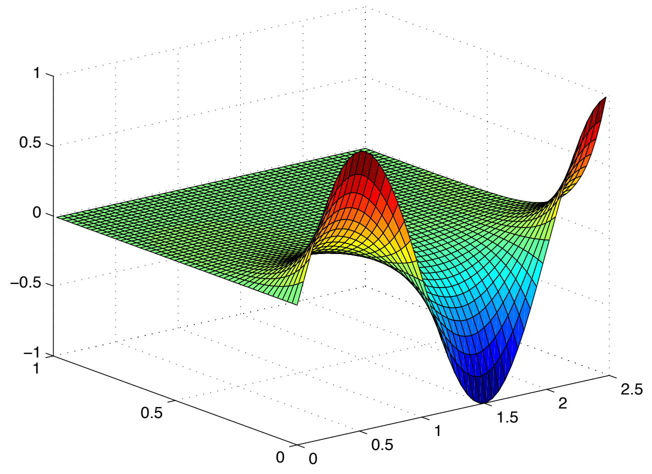
$$M\vec{u}^{n+1} + \vec{b}^{n+1} = \vec{u}^n,$$

or in other words, our update in time to is

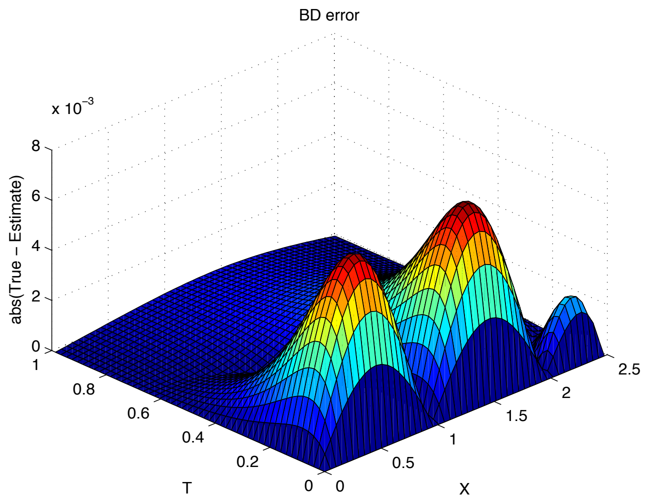
$$\vec{u}^{n+1} = M^{-1}\left(\vec{u}^n - \vec{b}^{n+1}\right).$$

We solve this last equation using a good numerical system solver instead of outright computing the inverse of M as calculating inverses can be numerically much more difficult and even unstable.

A.2.3 Stability of Implicit BTCS Scheme for Heat Equation. With the same system, equation (A.1), we only require 150 time steps and 100 space steps with the implicit scheme and are easily able to solve the inverse problem to get the same result as the 3,250 time steps and 100 space steps using the explicit system. The **implicit scheme is stable** and so there is no CFL condition on the time steps like in the explicit scheme. The inverse system is stable for any size time steps or space steps so the accuracy desired will dictate how small of grid size steps you need to use. We show the plots for the implicit BTCS solution in Figure A.6.



(a) Implicit BTCS solution



(b) Absolute error in implicit BTCS solution

Figure A.6: Implicit BTCS method solution and absolute error for heat equation (A.1) in a MATLAB plot. This was done on a fairly course grid.

A.2.4 Crank-Nicolson Scheme for Heat Equation. In the implicit scheme for the heat equation, we have a stable system for any Δt and Δt, whereas with the explicit scheme we had a CFL condition that

$$\frac{k\Delta t}{(\Delta x)^2} < \frac{1}{2}.$$

However, both of these schemes had order of convergence $\mathcal{O}\left(\Delta t, \Delta x^2\right)$. We will now formulate a scheme that has the same stability properties as the implicit scheme which is $\mathcal{O}\left(\Delta t^2, \Delta x^2\right)$. We will do this by considering ourselves at time step $t + 1/2$ and then using a centered difference in time ($\mathcal{O}\left(\Delta t^2\right)$ of step $\Delta t/2$. This can be accomplished through an average of the forward and backwards differences. So the heat equation becomes

$$\frac{u_i^{n+1} - u_i^n}{2(\Delta t/2)} + \mathcal{O}(\Delta t^2) = \frac{k}{2}\left(\frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2} + \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2}\right) + \mathcal{O}(\Delta x^2).$$

This turns out to have no CFL condition, meaning that the Crank-Nicolson scheme is unconditionally stable. The stencil for this Crank-Nicolson scheme is

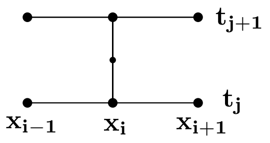
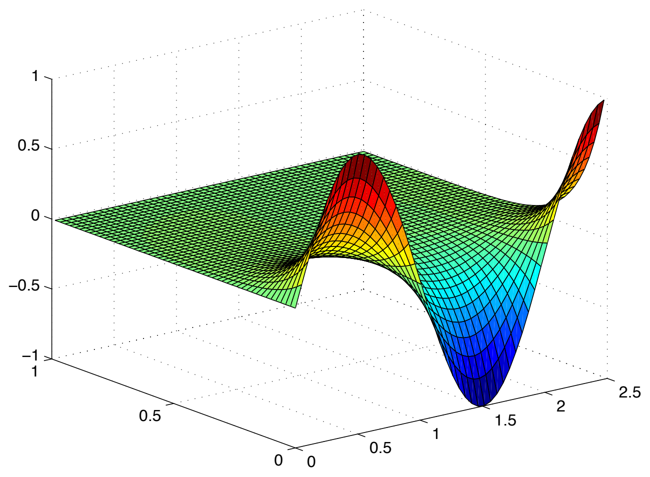
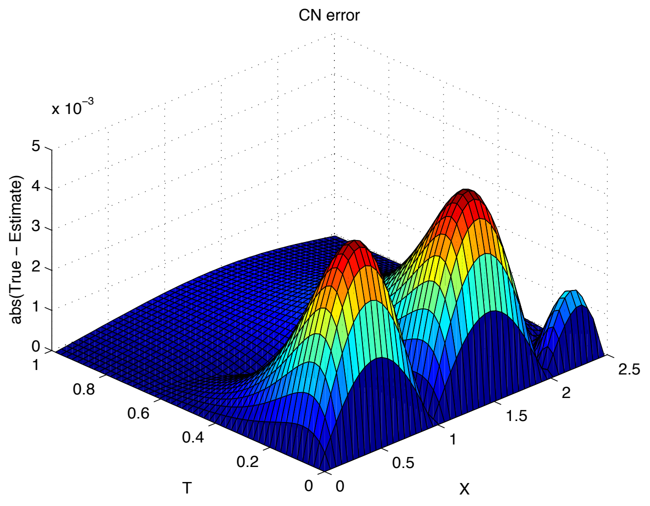


Figure A.7: Stencil for Crank-Nicolson scheme on heat equation

While we just state this result here, it follows nicely from the standard Fourier argument that is in every textbook of that treats this material. Likewise, instead of solving this system as has been done previously, we will leave the greater details in the discussion of the Crank-Nicolson scheme for a more complicated nonlinear wave equation and merely give the results for the heat equation using Crank-Nicolson in Figure A.8. Before we proceed to the wave equation, we give a side by side comparison of the absolute errors in the three models. Figure A.9 gives an error comparison against the true solution for time $t = 0.0268$. The forward difference scheme is given on very fine grid and the other two schemes are on the same course grid. We can easily see the varying accuracies of the models and especially note that the forward difference scheme required 3250 time steps to be equivalent to the 100 time steps of the other models.



(a) CN solution



(b) Absolute error in CN solution

Figure A.8: Crank-Nicolson method solution and absolute error for heat equation (A.1) in a MATLAB plot. This was done on a fairly course grid.

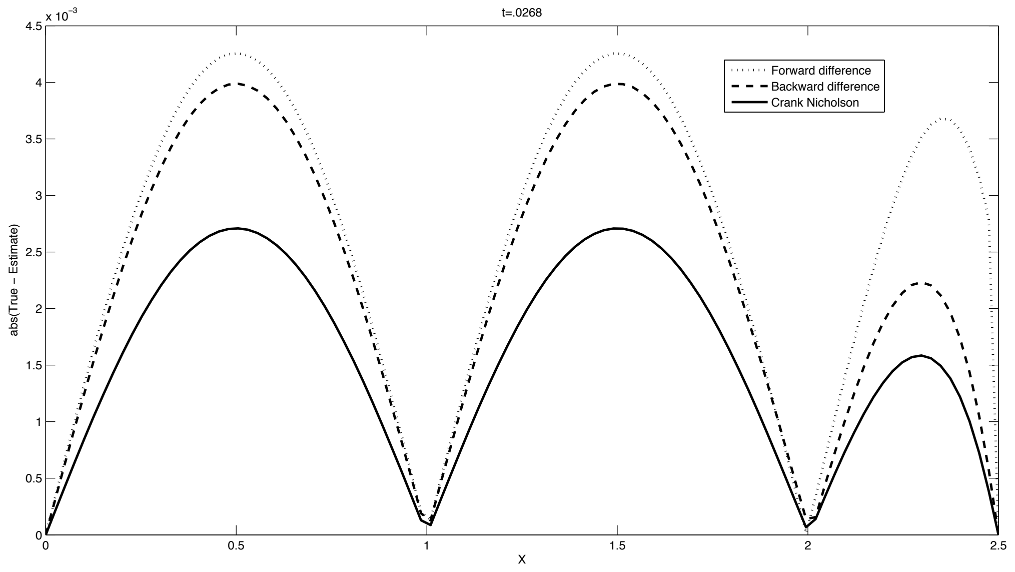


Figure A.9: The heat equation solved using forward difference, backward difference, and Crank-Nicolson schemes. This is a cut across space at the time $t = 0.0268$ in the evolution of the heat equation. The backward difference (dashed line) and Crank-Nicolson (solid line) schemes use the same course mesh while the forward difference (dotted line) scheme required a mesh roughly 32 times as dense to get equivalent results.

A.3 Burger's Wave Equation solved with Crank-Nicolson scheme.

We look for solutions to what is called the **general Cauchy problem** for any ordinary or partial differential equation.

$$u_t = f(u, u_x, u_{xx}, \dots)$$

$$u(0, x) = \tilde{u}(x) = \hat{u}(x) + v(x)$$

The right hand side can take on any form and use any number of derivatives of u. The Cauchy problem essentially asks the following question: How does a perturbation of a solution evolve under the it's differential equation? In other words are solutions stable or do they evolve into something else? We can consider $\hat{u}$ to be a solution to the equation and $v(x)$ a perturbation. We work to see what happens to this perturbed solution.

A specific equation which is studied is the Burger's equation. It is given by

$$f(u, u_x, u_{xx}) = -\frac{d}{dx}\left(\frac{u^2}{2}\right) + \nu u_{xx}$$

where ν is a small parameter which effects our diffusion. Thus we desire to find solutions to the equation

$$u_t + uu_x = \nu u_{xx}$$

given some perturbed initial solution.

In the case of Burger's equation, we transform our coordinates into the moving frame $(x, t) \to (x - st, t)$ to get the equivalent equation

$$u_t - su_x + uu_x - \nu u_{xx} = 0,$$

and initial condition

$$u(0, x) = \hat{u}(x) + v(x),$$

where $v(x)$ is a small perturbation.

We will create a finite difference scheme for this equation using the **Crank-Nicolson method** by taking an average of forward and backward difference schemes. To solve for the $n+1$th time step, where we know already the nth time step, we consider our scheme as if we were standing at the $n + 1/2$ time step and then take the average of the forward half time step and backward half time step. So our equation is

$$\begin{aligned}\frac{u_i^{n+1} - u_i^n}{\Delta t} + \left(\frac{u_i^{n+1} + u_i^n}{2} - s\right)\left(\frac{1}{2}\right)\left(\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} + \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}\right) \\ -\frac{\nu}{2}\left(\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} + \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}\right) = 0.\end{aligned} \tag{A.2}$$

To continue and simplify, we let $r = \frac{\Delta t}{(\Delta x)^2}$. Then we get the following nonlinear equation in the time $n+1$.

$$\begin{aligned} u_i^{n+1}\left[1 + r\nu + \frac{r\Delta x}{8}\left(u_{i+1}^{n+1} - u_{i-1}^{n+1} + u_{i+1}^{n} - u_{i-1}^{n}\right)\right] + u_{i-1}^{n+1}\left[\frac{-r\Delta x}{8}u_i^n - \frac{r\nu}{2} + \frac{rs\Delta x}{4}\right] \\ +u_{i+1}^{n+1}\left[\frac{r\Delta x}{8}u_i^n - \frac{r\nu}{2} - \frac{rs\Delta x}{4}\right] + u_i^n\left[r\nu - 1 + \frac{r\Delta x}{8}\left(u_{i+1}^n - u_{i-1}^n\right)\right] \\ +u_{i-1}^n\left[\frac{rs\Delta x}{4} - \frac{r\nu}{2}\right] + u_{i+1}^n\left[-\frac{r\nu}{2} - \frac{rs\Delta x}{4}\right] = 0. \end{aligned} \tag{A.3}$$

To see how we will solve this, we use the notation $w_i = u_i^{n+1}$ and $u_i = u_i^n$. Then our vector of variables is w since u is already known. We write the above equation as $G(w,u) = 0$ which we will find the roots of in w using Newton's method.

$$G(w,u) = \left[\begin{array}{c}
\frac{-1.5}{\Delta x} w_0 + \frac{2}{\Delta x} w_1 - \frac{1}{2\Delta x} w_2 \\[2em]
w_1 \left[1 + r\nu + \frac{r\Delta x}{8}\left(w_2 - w_0 + u_2 - u_0\right)\right] + w_0 \left[\frac{-r\Delta x}{8} u_1 - \frac{r\nu}{2} + \frac{rs\Delta x}{4}\right] \\
+ w_2 \left[\frac{r\Delta x}{8} u_1 - \frac{r\nu}{2} - \frac{rs\Delta x}{4}\right] + u_1 \left[r\nu - 1 + \frac{r\Delta x}{8}\left(u_2 - u_0\right)\right] \\
+ u_0 \left[\frac{rs\Delta x}{4} - \frac{r\nu}{2}\right] + u_2 \left[\frac{-r\nu}{2} - \frac{rs\Delta x}{4}\right] \\
\vdots \\
\vdots \\
w_i \left[1 + r\nu + \frac{r\Delta x}{8}\left(w_{i+1} - w_{i-1} + u_{i+1} - u_{i-1}\right)\right] + w_{i-1} \left[\frac{-r\Delta x}{8} u_i - \frac{r\nu}{2} + \frac{rs\Delta x}{4}\right] \\
+ w_{i+1} \left[\frac{r\Delta x}{8} u_i - \frac{r\nu}{2} - \frac{rs\Delta x}{4}\right] + u_i \left[r\nu - 1 + \frac{r\Delta x}{8}\left(u_{i+1} - u_{i-1}\right)\right] \\
+ u_{i-1} \left[\frac{rs\Delta x}{4} - \frac{r\nu}{2}\right] + u_{i+1} \left[\frac{-r\nu}{2} - \frac{rs\Delta x}{4}\right] \\
\vdots \\
\vdots \\
w_{m-1} \left[1 + r\nu + \frac{r\Delta x}{8}\left(w_m - w_{m-2} + u_m - u_{m-2}\right)\right] + w_{m-2} \left[\frac{-r\Delta x}{8} u_{m-1} - \frac{r\nu}{2} + \frac{rs\Delta x}{4}\right] \\
+ w_m \left[\frac{r\Delta x}{8} u_{m-1} - \frac{r\nu}{2} - \frac{rs\Delta x}{4}\right] + u_{m-1} \left[r\nu - 1 + \frac{r\Delta x}{8}\left(u_m - u_{m-2}\right)\right] \\
+ u_{m-2} \left[\frac{rs\Delta x}{4} - \frac{r\nu}{2}\right] + u_m \left[\frac{-r\nu}{2} - \frac{rs\Delta x}{4}\right] \\[2em]
\frac{1}{2\Delta x} w_{m-2} - \frac{2}{\Delta x} w_{m-1} + \frac{1.5}{\Delta x} w_m
\end{array}\right]$$

In order to do the newton step, we need to also calculate the Jacobian $DG(w,u)$. In order to do so, we define the following terms

$$\begin{aligned} a_i &= \frac{r\Delta x}{8}(-w_i - u_i) - \frac{r\nu}{2} + \frac{rs\Delta x}{4} \\ b_i &= 1 + r\nu + \frac{r\Delta x}{8}\left(u_{i+1} - u_{i-1} + w_{i+1} - w_{i-1}\right) \\ c_i &= \frac{r\Delta x}{8}(w_i + u_i) - \frac{r\nu}{2} - \frac{rs\Delta x}{4}. \end{aligned}$$

Then our Jacobian is

$$DG(w,u) = \begin{bmatrix} \frac{-1.5}{\Delta x} & \frac{2}{\Delta x} & \frac{-0.5}{\Delta x} & & & & \\ a_1 & b_1 & c_1 & & & & \\ & a_2 & b_2 & c_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & a_{m-1} & b_{m-1} & c_{m-1} \\ & & & & \frac{0.5}{\Delta x} & \frac{-2}{\Delta x} & \frac{1.5}{\Delta x} \end{bmatrix}.$$

And we can solve for the roots using Newtons method

$$x_{n+1} = x_n + (DG)^{-1}G(x_n, u).$$

A.3.1 Stability of Crank-Nicolson Method.

The Crank-Nicolson scheme is stable just like the implicit methods. There is no CFL condition on time steps. It is of higher order accuracy $\mathcal{O}(\Delta t^2, \Delta x^2)$ and involves a little more work in solving each update but is much more accurate than the implicit schemes. Sometimes the Crank-Nicolson scheme can be pretty complicated to implement but it is one of the preferred techniques for solving problems like this nonlinear burger's wave equation.

A.4 Hyperbolic Equations using Techniques from Computational Fluid Dynamics

A.4.1 Upwind Scheme. These previous schemes deal well when information is flowing in all directions. For instance, when we have a second order derivative, then the value at any point is being influenced by the points to its left and right. We get a smoothing type of effect. This is why we often call the heat equation (which has a second order derivative) a diffusive equation. When we are dealing with wave equations, we typically are dealing with information being propagated along a certain direction called **the characteristic line**. For instance, if we are dealing with the equation

$$u_t + au_x = 0,$$

where $a > 0$, then a wave will propagate to its right over time with speed a. Hence, the

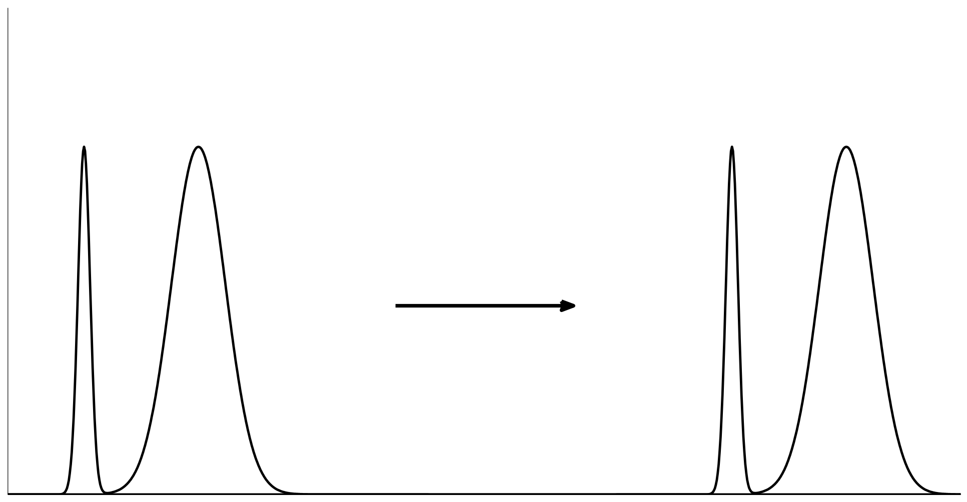


Figure A.10: Plot of $u_t + au_x = 0$ with $a > 0$. The wave travels from left to right with positive constant a.

information flows from left to right and our scheme should gather information to be propagated from the left. Now, the backwards difference operator for the spatial derivative will gather information from the left to propagate but the forwards difference operator gathers from the right, so for our specific example, our knowledge of which direction the wave should propagate tells us we need to use the backwards difference. We say that the scheme is an **upwind scheme** when it correctly passes information in the direction of the characteristic lines. In addition, we need our time step for the explicit scheme to be such that we are passing information slower than the speed of our wave is moving. This keeps the domain of dependence for each point on the correct side of the point and ensures that we do not lose information. Typically this translates into a Courant-Friedrich-Levy (CFL) condition on our time step of

$$\frac{a\Delta t}{\Delta x} < 1.$$

When we know which direction information is flowing, we can hard code the scheme used to reflect that knowledge. However, in many instances the direction of flow might change or we may not know which direction it is flowing in. We can still use a generic scheme which deals with this case.

Define forwards, backwards and centered difference operators on a function u to be

$$D_i^{-x} = \frac{u_i^n - u_{i-1}^n}{\Delta x},$$

$$D_i^{+x} = \frac{u_{i+1}^n - u_i^n}{\Delta x},$$

$$D_i^{ox} = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x},$$

Then we can write our first order explicit upwind scheme for the system, $u_t + au_x = 0$, as

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -\max(a,0)D_i^{-x} - \min(a,0)D_i^{+x}.$$

In other words, we have the **explicit upwind scheme** with first order accuracy,

$$u_i^{n+1} = u_i^n - \Delta t \left(\max(a,0) D_i^{-x} + \min(a,0) D_i^{+x} \right),$$

which is stable for $\left|\frac{a\Delta t}{\Delta x}\right| < 1$.

Notice that this correctly chooses the forward or backward scheme depending on the direction of characteristic flow. We can think of a^- and a^+ as switches to turn on or off the forward and backward difference flows. This generalizes to multiple spatial dimensions as well as higher order schemes. Example: for a second order scheme in space, we would use

$$u_x^- = \frac{3u_i^n - 4u_{i-1}^n + u_{i-2}^n}{2\Delta x}$$

and

$$u_x^+ = \frac{-u_{i+2}^n + 4u_{i+1}^n - 3u_i^n}{2\Delta x}.$$

A.4.2 Lax-Wendroff Scheme. There are other types of schemes which are not exactly upwind schemes, but still do a good job with preserving information flow. The Lax-Wendroff scheme uses the Taylor series expansion of $u(x_j, t_n + \Delta t)$ around $u(x_j, t_n)$ to get a higher order accurate system

$$u_j^{n+1} = u_j^n + (\Delta t) \frac{\partial u}{\partial t}_j^n + \frac{1}{2}(\Delta t)^2 \frac{\partial^2 u}{\partial t^2}_j^n + \mathcal{O}(\Delta t^3),$$

so we attempt to replace the time derivatives by space derivatives on the right hand side:

$$u_t = -a u_x$$

and

$$\begin{aligned}
u_{tt} = \frac{\partial}{\partial t}(u_t) &= \frac{\partial}{\partial t}(-au_x) \\
&= -a_t u_x - a u_{tx} \\
&= -a_t u_x - a\frac{\partial}{\partial x}(u_t) \\
&= -a_t u_x - a\frac{\partial}{\partial x}(-au_x) \\
&= -a_t u_x + a\frac{\partial}{\partial x}(au_x).
\end{aligned}$$

So if a is a constant, then this simplifies to

$$u_t = -au_x,$$

and

$$u_{tt} = a^2 u_{xx}.$$

Hence when we evaluate these at (x_j, t_n) and plug into our taylor expansion (for a, a constant), we get with centered differences in space,

$$u_j^{n+1} = u_j^n + \Delta t(-au_x)|_{(j,n)} + \frac{1}{2}(\Delta t)^2(a^2 u_{xx})\Big|_{(j,n)} + \mathcal{O}(\Delta t^3),$$

We finally have the **Lax-Wendroff scheme**, a second order scheme in space

$$u_j^{n+1} = u_j^n - \frac{a\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n) + \frac{a^2\Delta t^2}{2\Delta x^2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n),$$

which is stable for $\left|\frac{a\Delta t}{\Delta x}\right| < 2$.

A.4.3 Beam-Warming Scheme. The Beam-Warming Scheme is one of the most popular methods in computational fluid dynamics with Lax-Wendroff being next in preference. It takes the Lax-Wendroff scheme one more step and instead of using second order centered differences in space, we use the second order one sided difference from the direction of upwind. The **Beam-Warming scheme** for ($a > 0$) yields a second order accurate model

$$u_j^{n+1} = u_j^n - \frac{a\Delta t}{2\Delta x}(3u_j^n - 4u_{j-1}^n + u_{j-2}^n) + \frac{a^2\Delta t^2}{2\Delta x^2}(u_j^n - 2u_{j-1}^n + u_{j-2}^n),$$

which is stable for $\left|\frac{a\Delta t}{\Delta x}\right| < 2$. Hence the Beam-Warming method could be considered as an upwind version of Lax-Wendroff. Likewise for ($a < 0$), we have

$$u_j^{n+1} = u_j^n - \frac{a\Delta t}{2\Delta x}(-3u_j^n + 4u_{j+1}^n - u_{j+2}^n) + \frac{a^2\Delta t^2}{2\Delta x^2}(u_j^n - 2u_{j+1}^n + u_{j+2}^n).$$

A.4.4 Implicit Hyperbolic Schemes.

It is possible to write many of these schemes in implicit form, but the benefit of doing so compared to the extra computational and coding time makes it often not worth the effort.

Appendix B. Triangulation Formulation and Geometry

B.1 The Triangulation Problem

We derive and formulate the triangulation problem for refining our solution R, T given by the 8-point algorithm to accommodate n measured data points $\tilde{\mathbf{x}}_1 = \{\tilde{\mathbf{x}}_1^j\}_{j=1}^n$ and $\tilde{\mathbf{x}}_2 = \{\tilde{\mathbf{x}}_2^j\}_{j=1}^n$ under what we assume to be unstructured noise.

We wish to find $\mathbf{x}_1$ and $\mathbf{x}_2$ on the image planes that minimize in the least squares sense the difference from our measured points $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ which also satisfy our epipolar constraint

$$\mathbf{x}_2^{jT} \hat{T} R \mathbf{x}_1^j = 0.$$

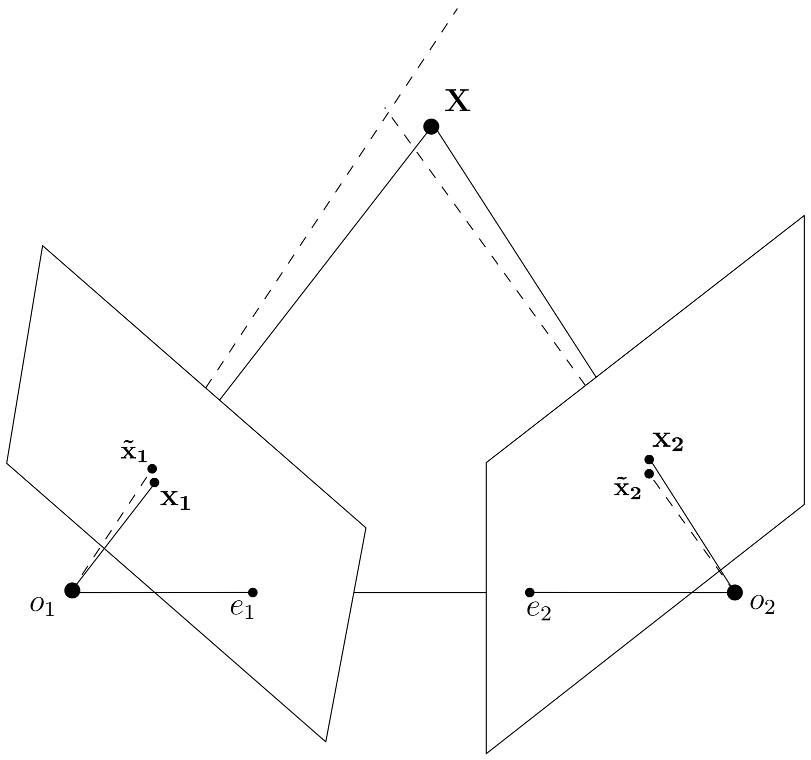
Hence we wish to find the points $\mathbf{x}_1$ and $\mathbf{x}_2$ that minimize the function

$$\phi(\mathbf{x}_1, \mathbf{x}_2, R, T) = \sum_{j=1}^{n} \|\tilde{\mathbf{x}}_1^j - \mathbf{x}_1^j\|^2 + \|\tilde{\mathbf{x}}_2^j - \mathbf{x}_2^j\|^2,$$

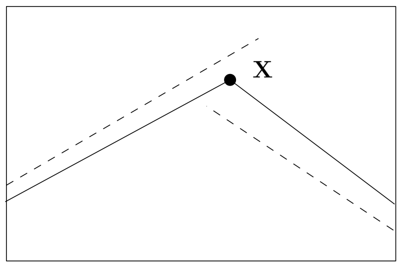
subject to the constraints

$$\begin{aligned} &\mathbf{x}_2^{jT} \hat{T} R \mathbf{x}_1^j = 0, \\ &\mathbf{x}_1^{jT} e_3 = 1, \\ &\mathbf{x}_2^{jT} e_3 = 1, \end{aligned}$$

where $e_3 = [0, 0, 1]^T$, $R \in SO(3), T$. The last two constraints are added to ensure our solution space for $\mathbf{x}_i$ is on the image planes for each camera. Now, since constrained optimization is difficult to work under, we obtain the unconstrained optimization problem through the



(a) Epipolar geometry with error



(b) the data points with error might not satisfy the epipolar constraint

Figure B.1: Under the assumption of noisy measured data, $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$, the epipolar constraint may not hold, ie. the three lines do not form a triangle. The triangulation problem posed to refine our solution with noise is to find $\mathbf{x}_1$ and $\mathbf{x}_2$ that satisfy the epipolar constraint and minimize the Euclidean distance to $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$. Figure B.1(a) shows the geometry with error and figure B.1(b) shows a close up and different angle of how the epipolar constraint may not be satisfied.

method of Lagrange multipliers. Hence, the unconstrained minimization function is

$$\phi(\mathbf{x}_1, \mathbf{x}_2, R, T, \lambda, \eta, \mu) = \sum_{j=1}^{n} \|\tilde{\mathbf{x}}_1^j - \mathbf{x}_1^j\|^2 + \|\tilde{\mathbf{x}}_2^j - \mathbf{x}_2^j\|^2 + \lambda^j \mathbf{x}_2^{jT} \hat{T} R \mathbf{x}_1^j + \eta^j (\mathbf{x}_1^{jT} e_3 - 1) + \mu^j (\mathbf{x}_2^{jT} e_3 - 1). \tag{B.1}$$

Now, the necessary conditions for a minimum is that the gradient $\nabla\phi$ must be zero, so we proceed by setting the derivative with respect to $\mathbf{x}_1^j$, $\mathbf{x}_2^j$, λ^j, η^j, and μ^j equal to zero. Thus for each $j \in \{1, \dots, n\}$, the following must be satisfied

$$\begin{aligned}
-2(\tilde{\mathbf{x}}_1^j - \mathbf{x}_1^j) + \lambda^j R^T \hat{T}^T \mathbf{x}_2^j + \eta^j e_3 &= 0 \\
-2(\tilde{\mathbf{x}}_2^j - \mathbf{x}_2^j) + \lambda^j \hat{T} R \mathbf{x}_1^j + \mu^j e_3 &= 0 \\
\mathbf{x}_2^{jT} \hat{T} R \mathbf{x}_1^j &= 0 \\
\mathbf{x}_1^{jT} e_3 &= 1 \\
\mathbf{x}_2^{jT} e_3 &= 1.
\end{aligned}$$

Now, in order to solve for our optimal λ^j, we use the first two equations to obtain

$$\mathbf{x}_1^j = \tilde{\mathbf{x}}_1^j - \frac{1}{2}\left(\lambda^j R^T \hat{T}^T \mathbf{x}_2^j + \eta^j e_3\right) \tag{B.2}$$

$$\mathbf{x}_2^j = \tilde{\mathbf{x}}_2^j - \frac{1}{2}\left(\lambda^j \hat{T} R \mathbf{x}_1^j + \mu^j e_3\right). \tag{B.3}$$

To simplify more, we multiply by $\hat{e}_3^T \hat{e}_3 = \text{diag}([1,1,0])$ and add e_3 to both sides to get

$$\begin{aligned}
\mathbf{x}_1^j &= \tilde{\mathbf{x}}_1^j - \frac{1}{2}\lambda^j \hat{e}_3^T \hat{e}_3 R^T \hat{T}^T \mathbf{x}_2^j \\
\mathbf{x}_2^j &= \tilde{\mathbf{x}}_2^j - \frac{1}{2}\lambda^j \hat{e}_3^T \hat{e}_3 \hat{T} R \mathbf{x}_1^j.
\end{aligned}$$

Now if we use equation B.2 to obtain our optimal λ^j, we must premultiply both sides of it by $\mathbf{x}_2^{jT} \hat{T} R$ and use our epipolar constraint to get

$$0 = \mathbf{x}_2^{jT} \hat{T} R \mathbf{x}_1^j = \mathbf{x}_2^{jT} \hat{T} R \tilde{\mathbf{x}}_1^j - \frac{1}{2}\lambda^j \mathbf{x}_2^{jT} \hat{T} R \hat{e}_3^T \hat{e}_3 R^T \hat{T}^T \mathbf{x}_2^j,$$

so

$$2\mathbf{x}_2^{jT} \hat{T} R \tilde{\mathbf{x}}_1^j = \lambda^j \left\| \hat{e}_3 R^T \hat{T}^T \mathbf{x}_2^j \right\|^2,$$

and our optimal λ^j satisfies

$$\lambda^j = \frac{2\mathbf{x}_2^{jT}\hat{T}R\tilde{\mathbf{x}}_1^j}{\left\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\right\|^2}. \tag{B.4}$$

Likewise, if we use equation B.3, we premultiply by $\mathbf{x}_1^{jT}R^T\hat{T}^T$ to get

$$\lambda^j = \frac{2\tilde{\mathbf{x}}_2^{jT}\hat{T}R\mathbf{x}_1^j}{\left\|\hat{e}_3\hat{T}R\mathbf{x}_1^j\right\|^2}. \tag{B.5}$$

Finally, if we add both solutions together, we obtain

$$\lambda^j = \frac{2\left(\mathbf{x}_2^{jT}\hat{T}R\tilde{\mathbf{x}}_1^j + \tilde{\mathbf{x}}_2^{jT}\hat{T}R\mathbf{x}_1^j\right)}{\left\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\right\|^2 + \left\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\right\|^2}. \tag{B.6}$$

Now, we return to our unconstrained minimization equation B.1 with our optimal values to obtain our simplified function which we can optimize not only over $\mathbf{x}_1$ and $\mathbf{x}_2$, but also over $R \in SO(3)$ and $T \in \mathbb{R}^3$. We substitute equation B.6 for λ^j to get,

$$\begin{aligned}
\phi(\mathbf{x}_1, \mathbf{x}_2, R, T) &= \sum_{j=1}^{n} \|\tilde{\mathbf{x}}_1^j - \mathbf{x}_1^j\|^2 + \|\tilde{\mathbf{x}}_2^j - \mathbf{x}_2^j\|^2 + \lambda^j \mathbf{x}_2^{jT}\hat{T}R\mathbf{x}_1^j + \eta^j(\mathbf{x}_1^{jT}e_3 - 1) + \mu^j(\mathbf{x}_2^{jT}e_3 - 1) \\
&= \sum_{j=1}^{n} \|\tilde{\mathbf{x}}_1^j - \mathbf{x}_1^j\|^2 + \|\tilde{\mathbf{x}}_2^j - \mathbf{x}_2^j\|^2 \\
&= \sum_{j=1}^{n} \|\frac{1}{2}\lambda^j \hat{e}_3^T\hat{e}_3 R^T\hat{T}^T\mathbf{x}_2^j\|^2 + \|\frac{1}{2}\lambda^j\hat{e}_3^T\hat{e}_3\hat{T}R\mathbf{x}_1^j\|^2 \\
&= \sum_{j=1}^{n} \frac{1}{4}\left(\lambda^j\right)^2 \left(\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\|^2 + \|\hat{e}_3\hat{T}R\mathbf{x}_1^j\|^2\right) \\
&= \sum_{j=1}^{n} \frac{1}{4}\left(\frac{2\left(\mathbf{x}_2^{jT}\hat{T}R\tilde{\mathbf{x}}_1^j + \tilde{\mathbf{x}}_2^{jT}\hat{T}R\mathbf{x}_1^j\right)}{\left\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\right\|^2 + \left\|\hat{e}_3\hat{T}R\mathbf{x}_1^j\right\|^2}\right)^2 \left(\|\mathbf{x}_2^{jT}\hat{T}R\hat{e}_3^T\|^2 + \|\hat{e}_3\hat{T}R\mathbf{x}_1^j\|^2\right),
\end{aligned}$$

and hence, we have

$$\phi(\mathbf{x}_1, \mathbf{x}_2, R, T) = \sum_{j=1}^{n} \frac{\left(\mathbf{x}_2^{jT} \hat{T} R \tilde{\mathbf{x}}_1^j + \tilde{\mathbf{x}}_2^{jT} \hat{T} R \mathbf{x}_1^j\right)^2}{\left\|\mathbf{x}_2^{jT} \hat{T} R \hat{e}_3^T\right\|^2 + \left\|\hat{e}_3 \hat{T} R \mathbf{x}_1^j\right\|^2}. \tag{B.7}$$

or substituting equations B.4 and B.5, we obtain

$$\phi(\mathbf{x}_1, \mathbf{x}_2, R, T) = \sum_{j=1}^{n} \frac{\left(\mathbf{x}_2^{jT} \hat{T} R \tilde{\mathbf{x}}_1^j\right)^2}{\left\|\mathbf{x}_2^{jT} \hat{T} R \hat{e}_3^T\right\|^2} + \frac{\left(\tilde{\mathbf{x}}_2^{jT} \hat{T} R \mathbf{x}_1^j\right)^2}{\left\|\hat{e}_3 \hat{T} R \mathbf{x}_1^j\right\|^2}. \tag{B.8}$$

B.2 The Geometry of the Parameterizations of the Optimization Variables

We remind the reader that ℓ_2 is the normal vector to the epipolar plane of our measurement error free data points in the second camera coordinates as shown in Figure B.2. We will talk about a vector ℓ in general, but in actual practice we will use ℓ_2.

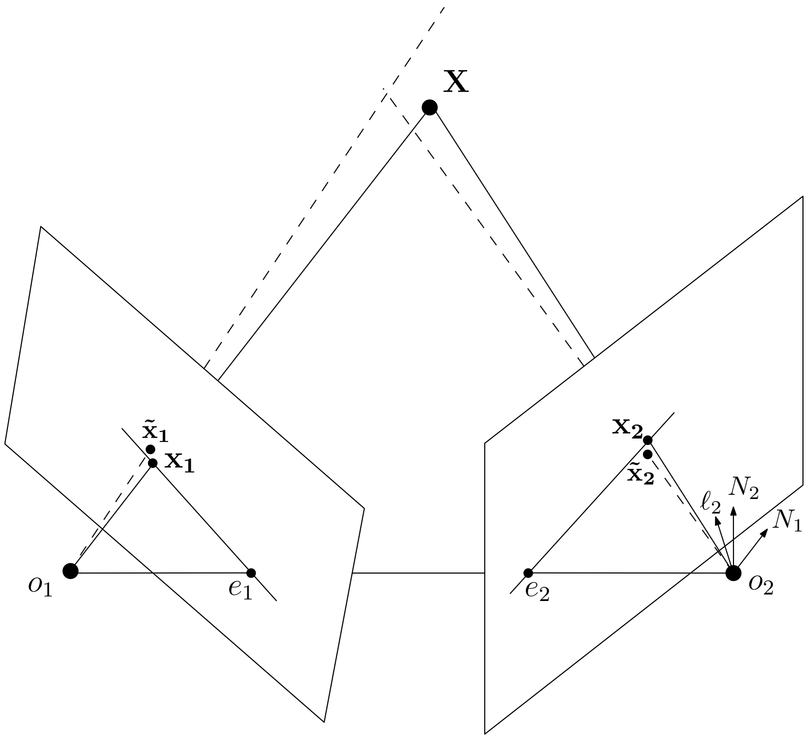


Figure B.2: We show the triangulation problem where ℓ is the normal vector to the epipolar plane. We will express our point $\mathbf{x}$ as a function of this vector ℓ given our measured data $\tilde{\mathbf{x}}$ which minimizes the distance from $\tilde{\mathbf{x}}$ to the epipolar plane given by ℓ.

We now prove the claim that the closest point, $\mathbf{x}$, to $\tilde{\mathbf{x}}$ on the epipolar line at the intersection of the epipolar plane with ℓ as it's unit normal vector and the image plane is given by

$$\mathbf{x}(\ell) = \frac{\hat{e}_3 \ell \ell^T \hat{e}_3^T \tilde{\mathbf{x}} + \hat{\ell}^T \hat{\ell} e_3}{e_3^T \hat{\ell}^T \hat{\ell} e_3}. \tag{B.9}$$

In Figures B.3, B.4(a) and B.4(b) the geometry of $\mathbf{x}(\ell)$ in the image plane is demonstrated from the three useful views. Now to prove this claim, we proceed to manipulate the above equation into recognizable geometric projections.

$$
\begin{aligned}
\mathbf{x}(\ell) &= \frac{\hat{e}_3\ell\ell^T\hat{e}_3^T\tilde{\mathbf{x}} + \hat{\ell}^T\hat{\ell}e_3}{e_3^T\hat{\ell}^T\hat{\ell}e_3} \\
&= \frac{(\hat{e}_3\ell)(\hat{e}_3\ell)^T\tilde{\mathbf{x}} + \hat{\ell}(\hat{e}_3\ell)}{(\hat{e}_3\ell)^T(\hat{e}_3\ell)} \\
&= \frac{(e_3\times\ell)(e_3\times\ell)^T\tilde{\mathbf{x}}}{\|e_3\times\ell\|^2} + \frac{\ell\times(e_3\times\ell)}{\|e_3\times\ell\|^2} \\
&= \mathrm{Proj}_{\frac{e_3\times\ell}{\|e_3\times\ell\|}}\tilde{\mathbf{x}} + \text{ the pt closest to } e_3 \text{ on epipolar line (see Fig. B.4(a) and B.4(b))}
\end{aligned}
$$

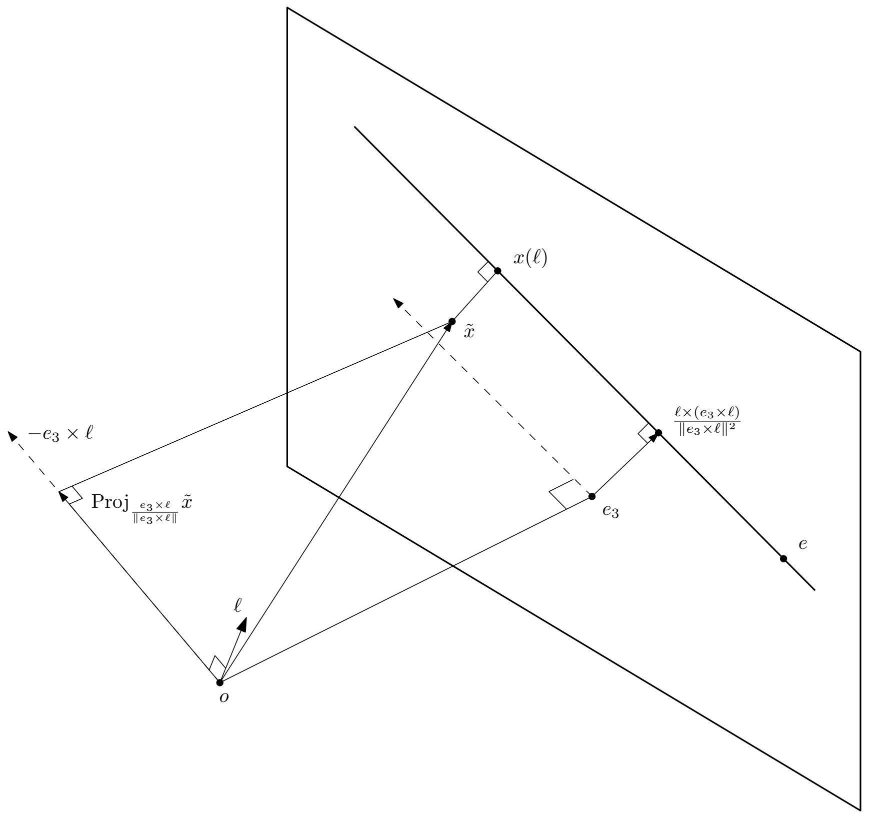
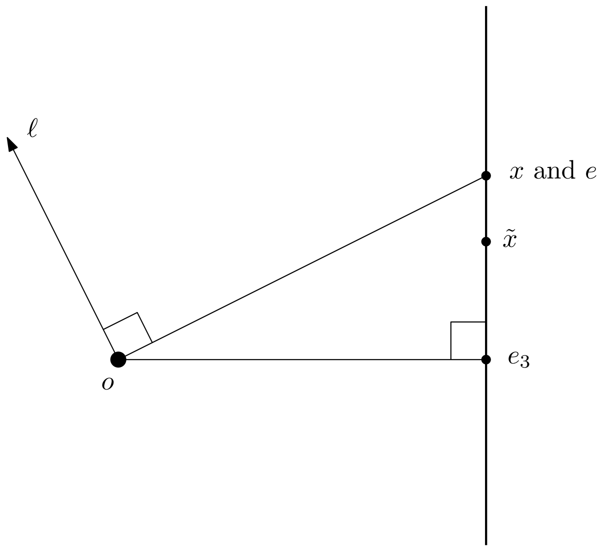
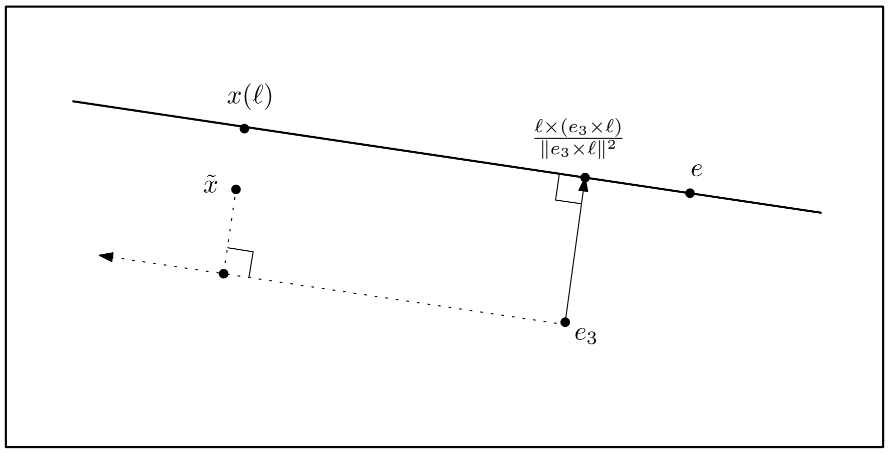

$x(\ell)$
$\tilde{x}$
$\frac{\ell\times(e_3\times\ell)}{\|e_3\times\ell\|^2}$
$-e_3\times\ell$
$\mathrm{Proj}_{\frac{e_3\times\ell}{\|e_3\times\ell\|}}\tilde{x}$
e_3
e
ℓ
o

Figure B.3: The point $\mathbf{x}(\ell)$ is given by $\mathbf{x}(\ell) = \mathrm{Proj}_{\frac{e_3\times\ell}{\|e_3\times\ell\|}}\tilde{\mathbf{x}} + \frac{\ell\times(e_3\times\ell)}{\|e_3\times\ell\|^2}$. It can be seen that it is the point on the epipolar line closest to $\tilde{\mathbf{x}}$. In Figure B.3, the dashed line on the image plane is the direction of the vector $-e_3\times\ell$ projected onto the image plane which also shows up in Figure B.4(b) as a dotted line.



(a) View of side of image plane along epipolar line



(b) Front view of image plane

Figure B.4: Other views of the image plane to see the geometry of the above discussion. The dotted line in Figure B.4(b) is the projection of the vector $-e_3 \times \ell$ onto the image plane.

Bibliography

[1] Richard L. Burden and J. Douglas Faires. *Numerical Analysis, Eighth Edition*. Thomson Brooks/Cole, 2005.

[2] Manfredo P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Inc., 1976.

[3] Olivier Faugeras and Renaud Keriven. Variational principles, surface evolution, pde's, level set methods and the stereo problem. Technical report, INRIA, November 1996.

[4] Olivier Faugeras, Quang-Tuan Luong, and with contributions from T. Papadopoulo. *The Geometry of Multiple Images: The Laws that Govern the Formation of Multiple Images of a Scene and some of their Applications*. The MIT Press, 2001.

[5] José Gomes and Olivier Faugeras. Reconciling distance functions and level sets. *Journal of Visual Communication and Image Representation*, 11:209–223.

[6] Richard I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:580–593.

[7] T.S. Huang and O.D. Faugeras. Some properties of the e matrix in two-view motion estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(12):1310 –1312, dec 1989.

[8] Allan Jepson and David J. Heeger. Linear subspace methods for recovering translational direction. In *Spatial Vision in Humans and Robots*, pages 39–62. Cambridge University Press, 1992.

[9] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time Dependent Problems*. Society for Industrial and Applied Mathematics, 2007.

[10] H. Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[11] Yi Ma, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer - Verlag New York, Inc., 2004.

[12] Yi Ma and Jana Ko seck. Optimization criteria and geometric algorithms for motion and structure estimation. *International Journal of Computer Vision*, 44:219–249, 2001.

[13] Michael B. Nielsen and Ken Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26(3):261–299, March 2006.

[14] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer Science + Business Media, LLC, 2003.

[15] Stanley Osher and Nikos Paragios. *Geometric Level Set Methods in Imaging, Vision and Graphics.* Springer-Verlag, 2003.

[16] Giovanni Russo and Peter Smereka. A remark on computing distance functions. *Journal of Computational Physics*, 163:51–67, 2000.

[17] Guillermo Sapiro. *Geometric Partial Differential Equations and Image Analysis.* Cambridge University Press, 1996.

[18] Shankar Sastry. Optimization criteria, sensitivity and robustness of motion and structure estimation. In *In Proceedings of ICCV workshop on Vision Theory and Algorithms*, pages 9–16, 1999.

[19] J. A. Sethian. Theory, algorithms, and applications of level set methods for propagating interfaces. *Acta Numerica*, 1995.

[20] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science.* Cambridge University Press, 1999.

[21] John A. Strain. Tree methods for moving interfaces. *Journal of Computational Physics*, 151:616–648, 1999.

Index