Direct occlusion handling for high level image processing algorithms

Carl Ryan Kelso
Direct Occlusion Handling for High Level Image Processing Algorithms

by

Carl Ryan Kelso

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Associate Professor Dr. Juan C. Cockburn
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
February 2009

Approved by:

Dr. Juan C. Cockburn, Associate Professor
Thesis Advisor, Department of Computer Engineering

Dr. Shanchieh Yang, Associate Professor
Committee Member, Department of Computer Engineering

Dr. Joseph Geigel, Associate Professor
Committee Member, Department of Computer Science
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

Direct Occlusion Handling for High Level Image Processing Algorithms

I, Carl Ryan Kelso, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Carl Ryan Kelso

Date
Dedication

To my parents and my advisors.
Their combined patience, support, and advice made this possible for me.
Acknowledgments

I am grateful for the help my sister Catherine provided in the process of writing this work. Her tireless assistance provided not only a connection to the vast body of work in the field of neural science, but also provided much of the polish my document needed. She, and her professors, have proven to be an extremely valuable resource in my research.
Abstract

Direct Occlusion Handling for High Level Image Processing Algorithms

Carl Ryan Kelso

Supervising Professor: Dr. Juan C. Cockburn

Many high-level computer vision algorithms suffer in the presence of occlusions caused by multiple objects overlapping in a view. Occlusions remove the direct correspondence between visible areas of objects and the objects themselves by introducing ambiguity in the interpretation of the shape of the occluded object. Ignoring this ambiguity allows the perceived geometry of overlapping objects to be deformed or even fractured. Supplementing the raw image data with a vectorized structural representation which predicts object completions could stabilize high-level algorithms which currently disregard occlusions. Studies in the neuroscience community indicate that the feature points located at the intersection of junctions may be used by the human visual system to produce these completions. Geiger, Pao, and Rubin have successfully used these features in a purely rasterized setting to complete objects in a fashion similar to what is demonstrated by human perception. This work proposes using these features in a vectorized approach to solving the mid-level computer vision problem of object stitching. A system has been implemented which is able extract L and T-junctions directly from the edges of an image using scale-space and robust statistical techniques. The system is sensitive enough to be able to isolate the corners on polygons with 24 sides or more, provided sufficient image resolution is available. Areas of promising development have been identified and several directions for further research are proposed.
Contents

Dedication ................................................................. iii

Acknowledgments ......................................................... iv

Abstract ................................................................. v

Glossary ................................................................. xiv

1 Introduction .............................................................. 1
   1.1 Motivation ....................................................... 5
   1.2 System Overview .............................................. 8

2 Background .............................................................. 12
   2.1 Biological Foundations ....................................... 12
   2.2 Edges .............................................................. 20
   2.3 Point Features ................................................ 26
   2.4 Vectorization ................................................... 31
   2.5 Object Completion ............................................ 36

3 System Design ........................................................ 42
   3.1 The Language .................................................. 42
   3.2 The Edge Detector Stage ..................................... 44
   3.3 The Vectorization Stage ..................................... 50
   3.4 The Completion Stage ....................................... 54
   3.5 The Interfaces ................................................ 57
4 Test Cases .......................................................... 60
  4.1 Edge Detector .................................................. 60
    4.1.1 The Gaussian Step Test ............................... 61
    4.1.2 Subsystem Effects ................................. 62
    4.1.3 Adjusted Variance, Fixed Blurs per Sub-Sample . 64
    4.1.4 Fixed Variance, Adjusted Blur Count Per Sub-Sample 65
    4.1.5 Matched Variance and Blurs per Sub-Sample .... 66
    4.1.6 Rotational Versus Separable Kernels ............ 66
    4.1.7 Differential Kernel Selection .................... 67
  4.2 Polynomial Edge Fitting ................................. 69
    4.2.1 Approximating Simple Shapes .................... 70
    4.2.2 Junction Extraction In Polygons .................. 71
    4.2.3 Fitting To Extracted Edges ...................... 73
    4.2.4 The Thresholds .................................. 74
    4.2.5 The M-Estimator ................................ 77
  4.3 Curve Completion .......................................... 79
    4.3.1 Polygonal Curve Closure ......................... 80
    4.3.2 Edge Closure Limits ............................. 80
    4.3.3 Fusors ........................................... 81
    4.3.4 Splitters ....................................... 82
  4.4 Complex Test Images .................................... 83
  4.5 Comparison to Prior Work ............................... 86

5 Conclusions ....................................................... 89

Bibliography ......................................................... 95

A The Chord to Arc-length Relation ......................... 101

B Gaussian Steps .................................................. 104
List of Tables

2.1 A variety of derivative kernels ........................................ 20
2.2 The fundamental derivative kernels used by Lindeberg . . . 25
3.1 A variety of M-Estimators ............................................. 51
4.1 Effect of varying the X-threshold on polygon edge segments. 75
4.2 Steps to convergence of several M-Estimators ............... 78
List of Figures

1.1 In the figure above, a green circle is visually in front of a red rectangle. The dotted edge bordering both is intrinsic to the circle, but extrinsic to the rectangle. .......................... 2

1.2 A piece of paper, a hand over it, and the resulting paper with the simplest shape prediction. Final image modified in Corel Photopaint 8. ........................................ 4

1.3 A red ball rolls behind a blue pillar. The visible region on the left shrinks until vanishing completely. In every frame, the ball is partially visible. Images created in Corel Bryce 5. 7

1.4 A flow chart depicting the system. ......................... 9

2.1 Demonstration of the presence of a blind spot in human vision. 17

2.2 Demonstration of edge completion in the presence of a blind spot. ............................................. 18

2.3 Demonstration of curve completion in the presence of a blind spot. ............................................. 18

2.4 Kanizsa’s triangle. This figure produces the appearance of a white triangle over three circles. Note the appearance of a white on white edge at the border of the triangle. There is no local gradient to hint at its presence. ............... 19

2.5 An image illustrating many of the different possible types of junctions. From left to right, these are the typical L, T, Y, followed by a concave Y, A reflective X, and a clear X. Image created in Corel Bryce 5.0. ....................... 28
2.6 A well known optical illusion demonstrating the three dimensional bistability of Y-junctions. Are the red regions squares on top of boxes, below the boxes, or are they diamonds? ....................................................... 29

2.7 Images of Kanizsa’s Square (a), the four crosses image (b), and the spontaneous splitting two fish image (c) described by Geiger et al.[9]. ................................................................. 29

2.8 The relationship between an edge and its detected points. . . 32

3.1 Demonstration of fit sensitivity across gaps without cumulative minimum thresholding ............................................ 53

3.2 A small sample BSP space. The green vector is a local root node. It splits the space into two hemi-planes. In the front hemi-plane is the blue vector, which has a fit length that passes the hemi-plane border. The red vector in the back hemi-plane, on the other hand can safely ignore things in the front hemi-plane since its fit is shorter than the distance to the border. ..................................................... 56

4.1 The Gaussian test images. Each image varies in standard deviation of blur from 1 px to 128 px as the image is traversed from top to bottom respectively. ......................... 61

4.2 Each image is processed using a blur filter with a variance of $\frac{1}{2}$ px. The process without sub-sampling used 16 blurs, while the process with sub-sampling used 8 per sub-sample. The lower images are vectorized versions of the outputs for clarity ................................................................. 63

4.3 Each image in this set was blurred 8 times per sub-sample with re-up-sampling enabled. ................................. 64
4.4 Each image in this set was blurred with a $3 \times 3$ Gaussian filter with variance of $\frac{1}{2}px^2$, and sub-sampled with re-up-sampling enabled.

4.5 This looks at the variance around the ideal blur per sub-sample described by Eaton et al.[7]. The base image is the quadratically increasing Gaussian step to highlight smaller scales.

4.6 This test bank blurs each image to a standard deviation of $2px$ before sub-sampling with a blur kernel of the specified variance.

4.7 A comparison of the edge detector output using rotation-ally symmetric versus separable blur kernels. Each image is blurred to a standard deviation of $2px$ before sub-sampling.

4.8 A comparison of the output produced by three common derivative kernels (and their convolution for second order derivatives).

4.9 Five simple convex shapes. From left to right: A square, pentagon, octagon, dodecagon and circle. From top to bottom: The generated edge with a single vertex (at top for all except the circle, which is at right), polynomial fits, and junction closures.

4.10 A demonstration of the limit of junction detectability in the estimator. The polygons from left to right are 24, 25, and 26 sided. All junctions are found on the 24 sided figure. The top right corner is lost on the 25 sided figure. Only seven junctions are detected in the 26 sided figure, one of which is caused by a vertex on the edge.
4.11 Four simple convex shapes. From left to right: A square, pentagon, octagon and dodecagon. From top to bottom: The original image, the extracted edge, the polynomial fits, and resulting junction closures. 73
4.12 A trefoil: a three dimensional knot. Rendered in Corel Bryce 5. 74
4.13 The result of varying the X-threshold on fits. 75
4.14 The result of varying the W-threshold on fits. 76
4.15 The result of varying the G-threshold on fits. 76
4.16 The test edge for comparing various M-Estimators. 77
4.17 The results of applying each M-Estimator to the test image.
The columns from left to right are the results from the Least Squares, Fair, Geman-McClure, Cauchy, and Tukey estimators respectively. From top to bottom are the progressive fits to the system, followed by the final fits, and the resulting completions. 78
4.18 The result of varying the W-threshold on fits. The distance between each pair is measured in the length of the side of a square. 81
4.19 The result of varying the W-threshold on fits. 82
4.20 A demonstration of the system processing splitting figures. 83
4.21 A demonstration of the system processing complex figures. 84
4.22 Additional complex figure examples. 85
A.1 A circle with tangent vectors $\vec{A}$ and $\vec{B}$, with angles $\phi$ and $\theta$ defined. 102
B.1 Three Gaussian step images used to test the edge detector’s responsiveness at different scales. The images are designed to produce a vertical edge down the center of the image. An ideal edge detector will return an uninterrupted edge at this location. Two additional edges are expected at the border where the gradient stops changing.
Glossary

A

amodal completions (amodal) When the human visual system completes objects, it may not necessarily create the sensation of a visible edge. If the completion occurs in the behind an occluder, it is referred to as an amodal completion. An example is the perception of completed circles in a Kaniza’s Square. While the edge is not visible, it can be traced., p. 11.

articulated An object composed of many components connected via joints. An example of an articulated object is a skeleton.

B

Binary Spatial Partitioning (BSP) A specific type of spatial indexing system. Binary Spatial Partitioning organizes spatially correlated information by recursively splitting the data into convex subsets based on an easily calculated metric. Through careful choice of the splitting metric, and balancing, it is possible to reduce searching in the space from $O(n^2)$ to $O(n \log n)$, p. 39.

blob A localized region of pixels or features derived from an image with similar properties. If these properties are easy to calculate then tracking blobs can be a fast and efficient means of following objects without high level processing., p. 3.

blue-noise Noise which is dominant in high spatial frequencies, but not present in low spatial frequencies., p. 20.

bottom-up A methodology of solving a complex problem which focuses on the nature of the underlying data or system before attempting
to tackle the issue. This may take longer than a top down strategy, but can yield better results., p. 1.

C

Canny’s edge detector  An edge detector developed in the 1980s which takes raw input data from a typical linear filter based derivative operator and performs several non-linear operations to refine edges down to a width of a single pixel., p. 10.

center-surround A receptive field pattern for a neuron. This pattern is typically recognizable by a cluster of inputs surrounded by a second cluster with opposite polarity. This structure is able to recognize the presence of details, which stimulate the inner field without affecting the outer field. The same structure in the eye may connect different colors to the center and the surround respectively, allowing most humans to distinguish between adjacent hues., p. 15.

Cesaro Equation An equation which relates a curve’s curvature to its arc-length.

Charge Coupled Devices (CCD) A capacitive array originally designed for use as a memory device. They have found application as the image sensors in digital cameras. See also Complementary Metal Oxide Semiconductors., p. 13.

chiasma The point in the brain where information from both eyes is sorted by hemisphere so information from a person’s left and right hemispheres of vision can be processed for stereo correlations., p. 15.

clothoid Alternatively known as a Euler spline or Cornu spline, the clothoid is a spiral whose curvature changes linearly with respect to arc-length. It minimizes the change in curvature between endpoints connected by a segment of it., p. 33.
Complementary Metal Oxide Semiconductor (CMOS)  A class of digital circuits which uses a capacitive barrier to control the active state of a gate. CMOS circuits tend to be very low power, and have found application as the light sensitive components in certain image sensors. See also Charge Coupled Devices., p. 13.

Compute Unified Device Architecture (CUDA)  A C-like programming language developed by NVIDIA to extend access to the functionality of their highly parallelized floating point GPU hardware., p. 9.

D

Difference of Gaussian (DoG)  The difference of two gaussian kernels with different variances. The DoG operator can be used as an approximation of the LoG operator. It is the basis for the SIFT scale-space., p. 15.

Difference of Offset Gaussian (DoOG)  A generalization of the DoG function, the DOOG function allows for an offset between the peaks of the gaussian functions. This offset allows the resulting functions to have similar structural appearance to the Gabor wavelet., p. 15.

Discrete Fourier Transform (DFT)  A transformation designed to represent a sampled periodic window of data as a sum of sinusoids of various frequencies and orientations. The inverse of this transform is the IDFT.

E

dedge  The interface between distinct regions. In an image, an edge is usually treated as bands of locally maximal gradients. Edges are one-dimensional., p. 1.

extrinsic  An edge is extrinsic to an object if the visible edge is defined
by the border of another object’s physical geometry. If an object has extrinsic edges, portions of it are hidden from view., p. 2.

F

feature Two definitions for image features exist in modern literature. They may either consist of all recognizable aspects of an image, including, but not limited to edges, blobs, corners, and interest points. The alternate definition limits the term exclusively to zero-dimensional locations of significance. In this paper, the second definition is used., p. 1.

G

gradient A transition in color, brightness, or another image metric. In mathematics, the gradient is a vector which points in the local direction of greatest change in a function., p. 1.

H

high-level Algorithms which benefit from a structural understanding of a scene. High-level algorithms include tracking algorithms, motion capture, 3D extraction and image registration techniques. These algorithms are only as stable as their underlying structural interpretation of the scene is, and suffer directly from assumptions made in the mid-level., p. 1.

Hough transform A transformation used to convert an imagespace into a space representing the likelihood of the presence of a geometric shape at a known location. The simplest Hough transform is one which converts pixels to lines. Variants able to extract curves and circles of varying sizes also exist, however, the more complex the search space, the more difficult it is to find local peaks in it.
intrinsic An edge is intrinsic to an object if the visible edge is defined by the border of the object’s physical geometry., p. 2.

Inverse Discrete Fourier Transform (IDFT) A transformation designed to revert a frequency domain image back to the spatial domain. This transform is the inverse operation of the Discrete Fourier Transform.

Iteratively Re-weighted Least Squares (IRLS) A process by which an approximation for an overdetermined system is fit to a model by evaluating the quality of each point in the fit and weighting the influence of that point in the final output respectively., p. 34.

Joint Photographic Experts Group (JPG) The common name for a lossy image file format developed by the Joint Photographic Experts Group. The actual format is the JPEG File Interchange Format (JFIF), which was designed to compress images by transforming them in a fashion which makes it simple to preserve image data preferred by the human brain when interpreting an image., p. 58.

junction The intersection of edges in a view.

Kanizsa’s Square Similar to a Kanizsa’s Triangle, except four pacmen are arranged at the corners of an illusory square.

Kanizsa’s Triangle A well known figure across many visual fields. This figure is composed of three pacmen evenly spaced with the openings pointing to the center of the figure. The human visual system typically interprets this as three black circles on top of a white background with a white triangle on top of them. The closure
of the pacmen to circles is an example of amodal completion, while the completion of edges to the triangle demonstrates modal completion. The figure may further be accompanied by the corners of a hollow triangle, similar to the configuration of a Koch’s snowflake.

**L**

**L-junction**  The intersection of two edge segments at their endpoints. This may be formed by an actual physical corner, or by similarly colored objects occluding one another. Since the presence of this type of junction may indicate the presence of an occlusion, it can be used to launch an attempted completion process. A successful completion hints that the inside of the L is in the background, while a failed completion hints that it is in the foreground. Counter examples include the lack of an illusory square in the Four Plusses image, and the perceived object ordering when looking at the crook of an arm., p. 10.

**Laplacian of Gaussian (LoG)**  The Laplacian of Gaussian in cartesian coordinates is the sum of all second order derivatives of the gaussian function., p. 15.

**Lateral Geniculate Nucleus (LGN)**  A region of the brain between the chiasma and the visual cortex dedicated to the processing of information from one hemifield of both eyes. The later regions of the LGN perform the first stereo integration of this data., p. 15.

**low-level**  Algorithms designed to operate directly with pixels to shape the data they present in order to improve their ease of use. Low level algorithms include edge detection, mapping between color spaces, and feature point detectors. They also include image filtering algorithms including linear filters, transforms, and median passes. These algorithms must be able to work in the presence of an unknown quantity of noise, preparing the data for higher level processes., p. 1.
**M**

**mid-level**  Algorithms designed to convert a raster image to a structural image. This provides high-level algorithms with a means of operating on objects, rather than raw image data. An example of a mid-level algorithm would be one which finds lines represented as peaks in a Hough transform. The Hough transform simplifies the task of the mid-level algorithm by presenting raw pixel data in an alternative form. The output of the clustering algorithm is a line or group of lines present in the image., p. 21.

**modal completions (modal)** When the human visual system completes objects, it may do so assuming that the completed portion lies either in front of or behind a secondary surface. If the completion occurs in the front, it is referred to as a modal completion. An example is the appearance of a white on white edge on the border of Kanizsa’s Triangle or Square., p. 11.

**motion capture** The process of extracting the motion of an entity. This may be performed mechanically, optically, or through the use of inertia or acceleration sensitive sensors., p. 1.

**O**

**occlusion** An occlusion is when one object partially or fully hides a second object from view., p. 2.

**optical axis** The axis of rotational symmetry in an optical system., p. 13.

**P**

**Portable Network Graphics (PNG)** A lossless graphics file format designed to allow storage and compression of image data in a variety of layouts, including paletted colors, full 24-bit, or 32-bit including transparency., p. 58.
R

**recognition**  The act of determining what an object is based on its visible structure., p. 1.

**Rochester Institute of Technology (RIT)**  Rochester Institute of Technology - The red-brick technical institute in Rochester NY notable for it’s fast paced quarter system and co-op program.

S

**Scale-Invariant Feature Transform (SIFT)**  Scale Invariant Feature Transform - An algorithm designed to locate stable feature points in images. SIFT points are invariant against rotations, translations, and a limited amount of affine transformation. They are notable for two features. First, they require the image be broken down into a DoG scale space, which is systematically searched for features. Second, the points have a complex descriptor incorporating information from the region local to the point in its given scale., p. 7.

**scale-space**  A volume of information derived from an image where pixels at successively coarser scales encompass information from wide regions of an image. Scale-space levels are generated through gaussian blurring, and provide a means for algorithms to behave in a scale invariant fashion without requiring convolutions with progressively larger kernels., p. v.

**self-occlusion**  When an object is capable of hiding parts of itself from view, it is referred to as self-occlusion. For convex objects, the surface facing away from the viewer is occluded by the bulk of the object. For concave and articulated object, parts of the surface facing toward the viewer can be completely hidden by an intermediate surface, also facing the viewer., p. 5.
T

T-junction  The intersection of two edges, one at an endpoint. While not always easy to distinguish from a Y-Junction, T junctions provide the strongest occlusion ordering and edge completion hint. They are often formed when three objects overlap in a small region. The head of the T, in the general case, is intrinsic to the object closest to the viewer. The stem hints at a completion under the object at the head of the T. A counter example is a barber’s pole. The red and white stripes painted around the pole form T junctions between each other and the background., p. 11.

top-down  A methodology of solving a complex problem which attempts to break down the original problem into gradual sub problems. This methodology may yield faster results than bottom up solutions, but typically yield less information about the underlying data-set, and suffer from problems where assumptions about the underlying structures which differ from reality., p. 1.

tracking  Following an entity through an image sequence., p. 1.

trancendental function  Any function which is not expressible as a ratio of polynomials. Examples of transcendental functions include the exponential function, sinusoids, hyperbolic sinusoids, and logarithms., p. 37.

W

white-noise  Noise spread evenly across all spatial frequencies.

X

X-junction  The intersection of two edges, without the clear termination of either. This type of intersection hints at the presence of an occluding object which is not opaque. Sharp variation in intensity or color across the junction along one of the edges may resolve
this ambiguity. Specialized processing for X-Junctions is beyond the scope of this paper.

Y

Y-junction  The intersection of three edges, all at an end point. When none of the edges provide evidence to support a smooth connection with another edge, they no longer can be treated as a T-Junction. Y-Junctions naturally can form from a perspective view of a 3D corner, such as the corner of a cube. The junctions may be interpreted as popping in, or popping out of the scene, depending on other visual cues. These junctions will not launch completion processes. Specialized processing for Y-Junctions is beyond the scope of this paper., p. 10.
Chapter 1

Introduction

Since its conception, computer vision as a field has evolved rapidly. Through its evolution it has enabled the development of systems which have found use in medicine, automation, navigation, and special-effects. Some of these systems can extract the 3D geometry of viewed objects [25]; others focus on the extraction of local motion [2]. With so many potential applications, the field has seen developments from both a top-down and a bottom-up approach simultaneously.

The top-down approach has yielded high-level algorithms which deal with complex tasks including motion capture, tracking, and recognition. The bottom-up approach has resulted in the development of low-level algorithms which provide means for detecting gradients [36], edges ([17], [19], [4]), and features ([14], [21], [26]). While these developments have come a long way, the high-level algorithms suffer from assumptions made by earlier processing stages.

Some of these issues can be dealt with by selecting alternative low-level algorithms. For instance, adjusting color-spaces can improve the detection of edges in an image [36] or reduce the search space in tracking algorithms [6]. Signal transformations, like the Fourier, Z, or Wavelet transforms, can shift
the entropy in an image so that relevant information is easier to find [16]. The use of scale-space can make an algorithm robust against changes in scale ([37], [38],[19],[20], [7]). Each of these techniques contributes to the field by introducing new assumptions about the nature of image structures; the benefits arise in areas where current predictions typically fail.

The structural issue which seems to cause the most systemic problems in the industry is a stable means of handling occlusions. An occlusion occurs when an object in an image is partially or fully hidden by a second entity. The edges of the occluding object modify the shape of the visible portions of the occluded object. Edges in an image which are formed by an object are said to be intrinsic to that object. If those edges are adjacent to regions belonging to another object, they are extrinsic to that other object (Figure 1.1).

The lack of a strong occlusion handling algorithm has severe implications. Since the appearance of the visible portions of an object can be changed by the presence of an occluder, tracking and recognition algorithms which do not attempt to predict the original shape can prematurely lose track of or falsely identify the objects they are attempting to interpret.

Figure 1.1: In the figure above, a green circle is visually in front of a red rectangle. The dotted edge bordering both is intrinsic to the circle, but extrinsic to the rectangle.

Typical motion capture algorithms like Wren’s PFinder [39] and the tracking
algorithm presented by Yamamoto et al. [41] make poor predictions about occlusions caused by secondary objects in a scene. The PFinder algorithm finds image regions which are likely to belong to a person. Once these regions are found, their configuration is used to recover the location and stance of the person. This technique allows the algorithm to ignore objects which occlude the subjects in a predictable fashion.

Shirts and pants on a person leave regions (blobs) of visible skin which are likely to correspond to certain body parts. However, if multiple people are in a scene, the blobs are not sorted by individual. Attempting to fit the regions extracted from multiple people to the model of one person will cause the algorithm to fail. This problem might be solved if the blobs could be grouped and stitched together into individuals before attempting to recover the person’s stance.

Problems caused by scenes with occlusions arise from naive assumptions about the objects in the scene being analyzed. One common approach to interpreting a scene is to use the figure-ground paradigm [27] (Figure-ground is synonymous with the object-background paradigm used by Gonzales et al. [11]). The paradigm treats all edges in a scene as intrinsic to the figures next to them. The background layer is treated as the exception to this, and is allowed no intrinsic edges.

This simple assumption has resulted in the development of high-level algorithms which work with considerable success on simple scenes, in stable, controlled environments. With time saved not solving for the occlusions, the algorithms are quick, supporting visually distinct objects over stable backgrounds. However, when using this method, if one object overlaps a second, algorithms will predict a hole in the second in the shape of the first (Figure
1.2). The fact that the figure-ground paradigm is a special case of a broader perceptual condition is highlighted by Rubin’s work [27].

![Figure 1.2: A piece of paper, a hand over it, and the resulting paper with the simplest shape prediction. Final image modified in Corel Photopaint 8.](image)

Several approaches to dealing with the occlusion problem have been taken. A common technique is to use blobs to analyze a scene. The generic blob approach groups regions in an image with similar local pixel properties [6]. These groups are rarely more complex than color histograms, brightness, or texture. Structures can be built up out of clouds of blobs which may be spatially or visually related. These clouds have been used successfully in laboratory motion capture techniques [40].

Blob based systems like this can be more stable than edge detectors in the presence of noise. Blob grouping suffers when unrelated objects have similar properties. For example, if a person is wearing a white shirt and writing on a white piece of paper, regions from both the shirt and paper may be joined into a single cloud. Alternatively, if the properties of an object change over time, due to lighting, shadows, or incandescence, the object may be split into pieces. This method is unable to predict the shape of partially obscured objects without knowledge of the occluded object itself.

The weakness of these algorithms rest in their dependency on raw low-level image data for the direct derivation of high-level properties from the images.
More robust motion capture techniques ([31], [5], [40], [39]) use a model of a person to supplement the information extracted from a scene. Data from broad regions in the image is accumulated to support a given model, reducing the effects of local noise in the raw data. With this additional a-priori information, the techniques are able to overcome self-occlusions.

While this is significant, the human visual system is able to correctly interpret the geometry of any convex object. It supports arbitrary curves as an input, allowing it to dynamically model objects in the scene. This allows it to generate models of objects which have never been seen before. A model based system provides stability, but is limited to interpreting only what it expects to see. The existence of optical illusions shows that the human visual system does use models to interpret the objects being viewed. The types of illusions that human vision is subject to shows that the models used are generic enough to apply to most items seen on a daily basis.

This work attempts to use research from both the computer vision and neuroscience communities to develop a system for extracting the structure of a scene. Included is a description of the steps involved in the creation of a system capable of partitioning a generic image. This is intended as a pre-processing step for higher-level algorithms. The work also attempts to identify directions for further research to improve upon the described technique.

### 1.1 Motivation

This research was originally motivated by the study of marker-less monocular human motion capture systems. Motion capture systems are a staple
of the video game and movie industries, allowing the movements of actors to be recorded and, later, mapped to articulated models of the characters. This technology has been used in games like *God of War*, and movies like *Beowulf*, *The Polar Express*, and for the character Gollum in *The Lord of the Rings* trilogy. Typical modern commercial systems require markers, small reflective spheres or lights on a skin tight suit, to recover the position and movements of actors. These appear in a video sequence as bright points which are simple to detect. Removing the constraints of a marker based system extends the functionality of motion capture to navigation and medicine, as well as many other fields.

Analysis of existing marker-less human motion capture systems revealed a number of weaknesses. A further review of the literature showed these same problems were present across a wide range of high-level algorithms. Only rarely do algorithms attempt to predict geometry in the presence of occlusions, and when they do, they usually are limited to solving self occlusions. The key concept which is overlooked in these systems is the distinction between intrinsic and extrinsic edges. Edges are intrinsic to an object if their appearance is determined by the object’s physical geometry. Extrinsic edges are determined by a second object. Extrinsic edges caused by shadows or occluding objects can cause objects to be split or deformed.

Some systems attempt to resolve the structural issue by grouping blobs, or similar image regions. These systems suffer from problems which arise when independent blob sources are in a scene. If a blob based capture or tracking system partitions an image based on skin tone, and multiple people are in the scene, the tracking system must be able to distinguish between the blobs originating from different people. This is achievable if the blobs
are guaranteed to never overlap, but outside of clean-room scenes, this constraint is difficult to guarantee. Tracked objects moving toward or away from the camera can be lost if the algorithm is not scale invariant [6]. Scale invariance can be achieved through the use of a scale-space volume. This technique uses successive Gaussian blurs and sub-samples to allow analysis of coarse and fine features with the same technique. An implementation of Collin’s Mean-Shift in Scale-space algorithm revealed several types of problems caused by occlusions. As illustrated in Figure 1.3, as an object passes behind an occluder, the shape of its visible regions may change beyond recognition. Shadows can cause color changes and the blobs composing the object may split. Sometimes they may vanish completely.

![Figure 1.3: A red ball rolls behind a blue pillar. The visible region on the left shrinks until vanishing completely. In every frame, the ball is partially visible. Images created in Corel Bryce 5.](image)

Feature based high-level algorithms like the one presented by Foo et al.[8], and those suggested by Lowe [21], take a shotgun approach to the object matching needed for object recognition, tracking, and 3D reconstruction. SIFT points [21] are located at places where the gradient forms a local maximum or minimum in scale-space. The immediate region around these points is used to calculate a rotationally aligned descriptor. Data for some of these descriptors may come from multiple objects. These points can still be used with a great deal of success to identify a scene [8]. However, using these features to perform operations dependent on their location can result in misinterpretations of scene geometry.
Through evolution, the human visual system has been able to overcome these problems for the general case. Human vision is not perfect. Through reverse-engineering of its visual flaws, it should be possible to achieve or surpass some of its successes. The places where the human visual model fails to properly interpret a scene are better known as optical illusions; they have been studied by the neuroscience and perception communities for as long as they have been known to exist. The illusions of particular interest to this problem are those which either split or fuse visual regions to form new shapes ([29],[34],[28],[27]).

The correlations between problems across many types of structurally dependent high-level algorithms point in a common direction to search for a solution. The differences between intrinsic and extrinsic edges must be handled. This requires object completion techniques which attempt to interpret edges in this fashion. A system which handles this difference should be able to recover surface ordering, make predictions of hidden edges, and simultaneously use this information to select edge ownership.

1.2 System Overview

It is possible to create a system capable of determining the geometric structure of a scene by solving for edge ownership. The process involved in analyzing a scene in this fashion can be broken into several distinct stages. Due both to the modular nature of the algorithm and the nature of image data, the computational processes involved are conducive to acceleration through parallelization. Each stage may be processed independently, allowing for pipelining to further accelerate video processing. The system should
lend itself well to modern massively parallel architectures and languages such as Erlang or CUDA. The following is a brief description of each of the stages necessary to extract a 2.5 Dimensional (multi-layered 2D) structural description of a scene.

1. Source - The system starts with raw pixel data. This may be partially compressed depending on the capture device used. Cheap, off-the-shelf cameras do not always provide lossless imagery. JPEG compression is frequently the compression scheme of choice.

2. Preprocessing - This data is refined with low-level image processing techniques to improve edge and feature detection. Possible uses for this stage include contrast adjustments to improve the correspondence between system output and expected human perception, reduction of a vector color space to gray scale, or conversion to a model of local energy which better identifies textures.

3. Scale-space Construction - This stage is a preprocessing step that allows the algorithm to be scale invariant. This stage may be merged with either the preprocessing or edge detection stages to reduce the
overhead of computation. Implementation should be based of the theory presented by Witkin [38], modified to normalize the information between scale layers as described Lindeberg [19]. A solid overview of scale-space theory is presented by Eaton et al.[7].

4. Edge Detection - The output of the preprocessing stage is searched for edges. These may be based on any of a number of algorithms, but should produce poly-line connected edges, preferably with junctions. Algorithms which utilize non-maximal suppression are recommended. If scale-space is not used, the Canny’s edge detector is recommended. If scale-space is used, then the algorithms described by Lindeberg [19] or Georgeson et al.[10] are decent candidates.

5. Vectorization - The edges produced by the edge detection stage are segmented through the use of polynomial fits. Higher order splines should be avoided because the system needs to be able to detect jump, crease, and bend discontinuities in a given edge. A robust estimation technique should be used to smooth over noise in the data while still recognizing discontinuities. The resulting L-junctions will be preserved for the next step. Fits should be to curves with constant concavity for use in later stages [23]. Gaps caused by noise and ill-supported edges are cleaned up at this stage.

6. Junction Analysis - Junctions between edges are processed at this stage. They are not only found, but classified into one of several types. The most important of these are L, T, and Y-junctions. The differences between varieties of junction types will be explained in detail out in a later section. This processing simplifies the more complex search in the next stage.
7. Edge Completion and Ordering - This section searches the known set of edges to find potential completion candidates. Edges connected to L or T-junctions are allowed to launch modal and amodal completions, but any unmatched edge may complete them. Edge curves detected from the original image data are considered visual. Edges generated by completion processes will be considered modal or amodal. For information on modal and amodal edges, see Section 2.1. Edge curves may be associated with either the surface on their right, their left, or marked ambiguous.

8. Surface Completion - At this final stage, all edges bordering a surface are linked and associated with a surface number. Ambiguous edges are paired off with their neighbors in an attempt to remove any ambiguity. The final results of this stage are a purely structural edge/surface representation. This stage finalizes the ownership of edges, determining which objects they are intrinsic to implicitly.

This work discusses the considerations associated with each stage of this process. Chapter 2 looks at the history of each of the stages in the system, including a section on biological and perceptually driven research. This leads to the design choices outlined in Chapter 3, which are used in the system. Chapter 4 focuses on the datasets being used to evaluate the system. This section will include qualitative explanations for why each image was chosen for testing. Finally, Chapter 5 will look at the design choices, challenges with implementation, results, and directions for further improvement.
Chapter 2

Background

The creation of a system which mimics human vision’s ability to interpret the geometry of overlapping shapes requires the integration of research from the fields of computer science, computer vision, neural science, perception, psychology, as well as robust statistics. There are many options available for each stage of the system, each with their own history of pros and cons. This chapter explores the biological foundations of vision in Section 2.1. It then continues with a consideration of edge and feature detection algorithms in Sections 2.2 and 2.3. Vectorization techniques, and a brief introduction to robust statistics are provided in Section 2.4, followed by a brief discussion of object completions in Section 2.5.

2.1 Biological Foundations

Structurally, the inner workings of the human vision system are fascinating. The human eye is a spherical pouch of transparent fluid. At the front are the cornea and the lens. These focus light through the inner medium to the back of the eye on the retina, a light sensitive patch of tissue. The pupil, a hole in the iris, is used to control the quantity of light passing into the eye.
By adjusting the shape of the lens, the eye is able to adjust the focus in the scene; by changing the size of the pupil, it is able to control the depth of field in focus. The retina is composed of layers of neurons. The layer of the retina furthest from the pupil is photo-sensitive. The neurons and blood vessels within the eye connect through a hole in the retina to the brain.

A camera is a similar system consisting of lenses and stops. Lenses, like the cornea and lens of the eye, control the focus of light. Adjusting the relative positions of camera lenses and stops along the optical axis (the center of radial symmetry in an optical system) in the camera determine what parts of the objects in a scene are in focus. The stops, like the pupil, restrict the light the image sensor is exposed to. Cameras use a variety of image sensors. Originally, analog cameras used films coated in silver nitrate or other light sensitive chemicals. Modern digital cameras have replaced the film with Charge Coupled Devices (CCD)s or Complementary Metal Oxide Semiconductor (CMOS) Sensors. Both CMOS sensors and CCDs consist of grid-like arrays of light sensitive circuitry.

Major differences between eyes and cameras exist. Cameras use a shutter to control the exposure length. When the shutter opens, light strikes the image sensor. Once the shutter closes, data can be safely stored without risk of corruption. This process resets the sensor in preparation for the next frame. Eyes, on the other hand, are continuous input analog devices. They never require a strict reset, quickly respond to the presence of light, and gradually adapts to the current light level if the image stops changing. Furthermore, the eye supports high dynamic ranges of inputs. Encoding up to nine orders of magnitude variation in input strength requires only two orders of magnitude variation in output strength [32].
The light sensors in a camera are typically laid out in rectangular grids. Eyes, however, have several distinct types of sensors (rods, cones, and some neurons themselves) and are laid out in spirals similar to how seeds sit in a sunflower. Rods are quickly oversaturated by bright light, operating best in low light conditions. They have a peak sensitivity to light with a wavelength around 500 nm (blue-green). Cones are responsible for color vision, and work only in well lit conditions. Three varieties of cones exist. Long-wave(L), mid-wave(M), and short-wave(S) cones show peak response to 560 nm(red-yellow), 530 nm(yellow-green), and 430 nm (blue-violet) light with some overlap [1]. Differences in the responses of neighboring cones to the same light result perception of colored light. If an L cone is responding to light, but M cones adjacent to it are not, the light is red [33].

A feature of eyes that is of considerable importance is the fovea. This is a dimple near the center of the retina which is almost exclusively cones. Humans align this region of the eye to areas of visual interest. The fovea provides high resolution color information to aid in distinguishing local edges, textures, and patterns. The dimple shape allows more light to strike the photoreceptors in the eye unimpeded by the layers of neurons which line the inner surface of the retina. Cameras record data uniformly across the sensor. If the camera is stationary, each pixel is equally likely to be important.

The retina is responsible for responding to local changes in image contrast. It is more sensitive to small changes in intensity if the actual intensity of the region that is changing is low. The smallest noticeable differences in brightness are often assumed to be logarithmic with a linear change in the reflectance of a surface. Research associated with the development of the Munsell color space, however, shows that this may be better modelled by a
cubic root (specifically $-1.324 + 2.217 \times R^{0.3520}$ where $R$ is the reflectance of the surface in question) [18]. This means an object with approximately 18% the luminance of a second object will appear to be half as bright as the second.

A concept well known in the neuroscience community is the center-surround receptive field structure typical of neurons in the visual system [1]. This structure consists of a large outer region and small inner region with opposite responses to stimuli. The center-surround structures in the retina are responsible for compressing information before sending it down the optic nerve to the chiasma, Lateral Geniculate Nucleus (LGN), and the visual cortex. These same structures are used to compare outputs from neighboring cone cells to distinguish the full visible spectrum of colors.

Research by Young et al. [42] demonstrates these neurons behave similarly to the Difference of Offset Gaussian (DoOG) function, a generalization of the Difference of Gaussian (DoG) function. Both the DoOG and DoG functions are linear combinations of Gaussian functions. It is for this reason that these functions have strong connections with scale-space, a technique for reducing the sensitivity of algorithms to changes in size. The DoG function has been used by Lowe [21] as an estimate of the Laplacian of Gaussian (LoG) operator to extract scale invariant features. A brief overview of scale-space is presented in Section 2.2.

The human eye uses the center-surround receptive fields to compress information from approximately 125 million neurons, to be passed down the optic nerve, a channel of about 10 million neurons. This information then reaches the chiasma, where the data from both eyes is sorted based on
whether it originated from the central or peripheral visual field. This pre-
pares the data for stereo rectification. The regrouped data is passed back to
the LGN, which continues the analysis performed in the retina. The LGN
also starts to integrate the stereo data. The process is lossy, throwing away
the true brightness levels in favor of preserving the changes in perceived
brightness. Luminance information is localized better than chromaticity
(colors). The types of data ignored during these stages are the driving prin-
ciple behind lossy image compression techniques.

Eyes are also robust against cell death and lost information. If a pixel in
a camera fails, no attempt is made to compensate for the damage. The
eye actively attempts to detect and recover from these problems. The eye
jitters constantly to detect the presence of bad data. If a chain of neurons
receives a constant input, they will gradually develop a tolerance to it. This
is likely to happen when a cell is oversaturated, undersaturated, damaged, or
dies. The chains of cells interpreting the damaged cell’s output will learn to
ignore it and fill in the gap. To understand how vital this process is to daily
vision, a significant structural difference between cameras and eyes must be
considered.

Film is developed in a lengthy post process to the actual image capture.
CCDs have external hardware for reading their data, which is sequentially
extracted from the sensor. CMOS sensors have local hardware for counting
and interpreting the amount of light captured. Eyes use neurons to process
visual data. In humans, those neurons are in the path light takes to be cap-
tured by photosensitive tissue in the retina.

Light must pass through layers of these neurons to reach the cones and rods
in the back of the eye. In addition, these neurons require nutrients to function. The blood vessels which feed neurons in the eye also lie in the light path. To enter the eye, they must pass through the optic disc, forming a blind spot in the retina. Through interpolation, the gap in the visual field created by the optic disc is hidden. Typically, the interpolation is so clean that the blind spot is not noticeable. The blood vessels themselves also cause smaller blind spots in a thin web across the retina.

There is a way to demonstrate the presence of the blind spot. In Figures 2.1, 2.2, and 2.3, there are three sets of completable objects. To demonstrate the presence of the blind spot, one eye is closed while the other focuses on the plus in the middle of one of the three figures. Holding the page level with your head, very gradually move the page toward and away from your face.

On Figure 2.1, at the proper distance, one of the dots will disappear. If your right eye is open, it will be the right one. If your left eye is open, the left one will vanish. The size of the gap that the brain compensates for is striking. The human brain performs much more complicated stitching than simply filling in a region with border color. If an object is split by the blind spot, the brain will attempt fill in the gap appropriately, using the contour information from the border. Figure 2.2 demonstrates this using an object with straight edges, while Figure 2.3 demonstrates completions with curves. The first two examples can be found in Bear et al., page 282 [1].

![Figure 2.1: Demonstration of the presence of a blind spot in human vision.](image)
The brain performs interpolations like these in the presence of overlapping objects as well. The types of border interpolations the brain performs can be divided into two varieties. Modal completions are those which produce a visible response, while amodal completions can be traced and predicted, but are not visually perceived. Figure 2.4 demonstrates the difference between these types of completions by creating the illusion of a white triangle over a white background. The white triangle is bordered by well defined white on white edges, demonstrating modal completion. The shapes surrounding the triangle look like circles. The completed portions of the circles are examples of amodal completions.

The human visual system is capable of both completions and spontaneous splits. Studies suggest that these are strongly dependent on the presence
of junctions in the edge map [28]. In Rubin’s work, she demonstrates that completion processes may be launched by first-order locations where edges form L (crease) and T-junctions. These junction types do not have to contain orthogonal edges. Second-order (bend) junctions have been demonstrated to create the appearance of inter-object penetration [34].

There are strong correlations between the strength of edges completed modally and amodally, suggesting that a single mechanism is used to determine whether a pair of junctions should form a completion [29]. Perceived completions themselves seem to vary depending on whether they are modal or amodal. Amodal completions produce sharper curves than the corresponding modal completions [30]. It has been suggested that one method of determining the quality of completions is to minimize the change in curvature over the arc length [15].

As noted above, there is a plethora of research demonstrating that maps of edges are a primary feature used in human visual processing. Recognizing the significance of this may be key to solving the occlusion problem. The next challenge lies in the detection of edges and their junctions.
2.2 Edges

A fundamental image analysis technique is edge detection. Edges typically are defined as places in an image where the intensity changes drastically. Bars or ridges are locations where the image intensity is at its peak or trough. Edge detection schemes have suffered from many issues, evolving gradually in response. Many schemes are based off a convolution with a gradient kernel ([4],[19]); others attempt to find edges using transformations into alternative domains [17]. Each of these methods has advantages and disadvantages.

Most edge detection schemes begin processing through gradient analysis. In the spatial domain this is performed through a convolution of the image with a kernel. The limited nature of quantized kernels presents several problems. Smaller kernels allow for rapid image analysis, limiting the types of multi-object interactions possible within the kernel’s region of support. Kernels which are too small produce aliasing effects when used. This can be attributed to the differences between sampling rates along image diagonals and the main grid axes. Larger kernels improve rotational invariance of the gradients detected and can reduce spurious edges caused by blue-noise. Table 2.1 demonstrates several common image gradient kernels. The Scharr kernel produces results similar to the Sobel edge detector, but with improved rotational invariance.

Table 2.1: A variety of derivative kernels

<table>
<thead>
<tr>
<th>Names</th>
<th>Differences</th>
<th>Central Differences</th>
<th>Sobel</th>
<th>Scharr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernels</td>
<td>−1 1</td>
<td>−1 0 1</td>
<td>−1 0 1 1</td>
<td>−3 0 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>−2 0 2</td>
<td>−10 0 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>−1 0 1</td>
<td>−3 0 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The earliest edge detection techniques used thresholds to find the strongest edge candidates in the image. This created a strong dependency on the contrast and brightness of the image being analyzed. If a threshold was too low, it found spurious edge information, while thresholds which were too high missed major edges in the scene. One of the most famous and successful responses to this problem was proposed by Canny [4]. This paper presented the concepts of hysteresis thresholding and non-maximal suppression. In hysteresis thresholding, a high threshold is used to detect candidate edges. These edges are traced until their strength drops below the low threshold. At this point the edges are terminated.

Non-maximal suppression uses a quadratic fit along the gradient direction to localize the edges at their peak gradient. This system can theoretically find edges with sub-pixel accuracy at fine scales, providing improved stability over pure thresholding across a range of lighting conditions. Between the implicit edge tracing and the sub-pixel accuracy of the output edges, Canny’s edge detector could be considered an early mid-level algorithm. The output of the algorithm is no longer raster (pixelated) data, but a vectorized (mathematical) interpretation of the image.

Canny’s non-maximal suppression technique can be mimicked by tracing zero crossings in the second and third derivative of the image for edges and bars respectively. The LoG kernel can be used to detect these crossings. As mentioned in Section 2.1, this kernel has a close relationship to the center-surround layout of neurons in the human retina. Depth of field, soft shadows, and interactions between light and the volume of a material can produce blurry edges. The gradients of these edges can be so low that high frequency noise masks their detection. The existence of blurry, coarse-scale
edges in an image points to the need for scale-space analysis. The close association between the LoG kernel and scale-space further points to the value of the kernel’s use [21].

Scale-space is a method of introducing scale invariance into the analysis of vast grids of data. The technique takes an input, in this case the 2D image $I(x)$, which is successively blurred via convolution with the 2D Gaussian function, $G_2(x; t)$ (Equation 2.1). The Gaussian function is uniquely able to shift between scales without introducing new features [38]. The volume formed by the stack of each of the successive blurs is the scale-space volume, $L(x; t)$. This technique on its own has found powerful applications in feature detection [21] and improved blob tracking [6]. A strong foundation for scale-space techniques is presented by Eaton et al.[7].

Scale-space analysis of images is enhanced by research done by Lindeberg in his research on scale-space edges [19]. In this work, he presents a means of normalizing scale-space so edges form natural maxima in the generated volume. The original scale-space volume, Equation 2.2, is modified by multiplying each layer by a power of the variance, Equation 2.3, resulting in the normalized scale-space pyramid, $L_n(x; t)$. This technique effectively corrects the contrast sensitivity issues caused by thresholding by comparing edge strength exclusively to the local image data. It adds further support for edges which are not in focus, such as those produced by soft shadows or depth of field effects.
While the threshold problem has been corrected, there is still a normalization factor present in Equation 2.3. The value of $\alpha$ is constrained to the range $\{0, 1\}$, and can be empirically selected. Lindeberg found that choosing a value of $t^{\frac{1}{4}}$, where $t$ is the variance, allows the center of Gaussian step edges to be detected at the location of symmetry in the gradient [19]. In a study by Georgeson et al., the perceived blur of different edges and edge detectors is considered [10]. Sinusoidal edges have a local maxima in scale-space regardless of the value of $\alpha$. The study also states that the use of $t^{\frac{1}{4}}$ produces correlations between sinusoidal edges and Gaussian steps that match perceptive experimentation.

The algorithm for detecting edges, as presented by Lindeberg [19], takes a oriented gradient approach. For each pixel in an image, it is possible to determine the local gradient based on the derivatives in the $x$ and $y$ directions. Lindeberg defines a local $(u, v)$ coordinate system where $v$ is in the gradient direction and $u$ is perpendicular to that (Equations 2.4, 2.5, and 2.6). He
then defines scale-space derivatives along the \( v \) axis, \( L_{vv} \) and \( L_{vvv} \) (Equations 2.7 and 2.8). Since the sign information is most important, Lindeberg expands \( L_{vv} \) and \( L_{vvv} \) in terms of their partial derivatives and uses only the numerators \( \tilde{L}_{vv} \) and \( \tilde{L}_{vvv} \) of the resulting expressions. In addition, Lindeberg defines several partial derivatives with respect to scale. The edge detection technique searches for edges in locations where the following conditions are met:

\[
\tilde{L}_{vv} = L_x^2 L_{xx} + 2 L_x L_y L_{xy} + L_y^2 L_{yy} = 0 \quad (2.7)
\]

\[
\tilde{L}_{vvv} = L_x^3 L_{xxx} + 3 L_x^2 L_y L_{xxy} + 3 L_x L_y^2 L_{xyy} + L_y^3 L_{yyy} < 0 \quad (2.8)
\]

\[
\partial_t (G_{\gamma-norm} L) = \gamma t^{\gamma-1} (L_x^2 + L_y^2) + t^{\gamma}(L_x (L_{xxx} + L_{xyy}) + L_y (L_{xxy} + L_{yyy}) = 0 \quad (2.9)
\]

\[
\partial_{tt} (G_{\gamma-norm} L) = \gamma(\gamma - 1)t^{\gamma-2}(L_x^2 + L_y^2) + 2\gamma t^{\gamma-1}(L_x (L_{xxx} + L_{xyy}) + L_y (L_{xxy} + L_{yyy}) + \frac{t^{\gamma}}{2}((L_{xxx} + L_{xyy})^2 + (L_{xxy} + L_{yyy})^2 + L_x (L_{xxxx} + 2L_{xxyy} + L_{xyy}) + L_y (L_{xxxx} + 2L_{xxyy} + L_{yxyy})) < 0 \quad (2.10)
\]

Equation 2.7 has zero crossings at locations where edges are located in a given scale. The maxima and minima are distinguished by Equation 2.8. With the derivatives gradient aligned, local minima are thrown away. Equations 2.9 and 2.10 perform the same functions across scales. Therefore, the intersection of the zero surfaces formed by Equations 2.7 and 2.9, where Equations 2.8 and 2.10 are less than zero, are edges in scale-space.
While the benefits of using [19] to detect edges are significant, the technique is computationally intensive. The complex derivatives of the image require cascades of the kernels presented in Table 2.2. The fifth order derivatives require 7x7 filters. These may be interlaced to a limited degree, but are still expensive; they also introduce instability into the system. Lindeberg’s technique requires a progressive voxel search for edges through the entire scale-space volume. A voxel is the volumetric cell of data containing information from each of the four neighboring pixels on a given scale level, and the one coarser. The topology used by Lindeberg consists of 40 blur layers ranging in variance from 0.1 to 256 px\(^2\). To detect edges, the system must generate this volume; the derivatives for each layer must be calculated, and zero crossings traced. Without sub-sampling, using Lindeberg’s suggested topology and four derivative models requires the processing of 160 times the original image data.

Table 2.2: The fundamental derivative kernels used by Lindeberg

<table>
<thead>
<tr>
<th>1st Order</th>
<th>2nd Order</th>
<th>Second Order Cross</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1/2 0 1/2</td>
<td>1/4 -1/2 1/4</td>
<td>1/4 0 -1/4</td>
</tr>
<tr>
<td>1/4 -1/2 1/4</td>
<td>0 0 0</td>
<td>-1/3 0 1/3</td>
</tr>
</tbody>
</table>

Lindeberg further extends his discussion of scale-space in a later work [20]. In it, he proposes a method of generating a hybrid scale-space pyramid. This blends sub-sample stages with the standard blur stages found in typical scale-space volume generation. By limiting sub-sampling stages to scales where the high frequency information lost is negligible, the algorithm preserves most of the value of the full scale-space volume. There is one catch: in order to utilize this implementation, comparisons between images with
different resolutions must be possible. This step is vital to the preservation of edge data, and failure to implement it results in prematurely severed edges.

Border effects pose a second problem. Lindeberg’s edge detector effectively searches scale-space for local maxima. It does not, however, appear to be conducive to predicting edges which extend beyond the tested scales. Incorporating the border maxima into the detected edges may resolve gaps in superfine edges, as well as resolve issues at coarser scales caused by low resolution. The border effect problem is particularly prevalent at sharp corners in pixelated images. These locations are significant if the importance of L and T-junctions as described in Section 2.1 are considered.

2.3 Point Features

Point features are locations in an image with distinguishing characteristics that separate them from the surrounding regions. The types of features include corners, line ends, discontinuities in edge curvature, impulses, local extrema, and positions of peak luminance variation. Many more varieties exist, each providing dense contextual clues about the visual structures creating them. Proper clustering of extracted features can help resolve edge ownership, surface curvature, and illusory borders.

Research on these features was driven in part by 3D reconstruction through image registration techniques. Harris points [14] provided a scale dependent feature point that was robust in the presence of noise and motion. The Scale-Invariant Feature Transform (SIFT) [21] detector attempts to detect points which are not only invariant to noise, but to scale and rotation as
well. Feature descriptors are introduced by Lowe to improve the success of junction matching algorithms by adding localized support.

The feature descriptors gather information from the local region around the feature point. In SIFT points, the information is oriented to the sharpest gradient to add rotational invariance. This descriptor provides distinguishing information about the type of feature the point represents. It also contains hints which may be used to reconstruct the surface layering. The neuroscience community has studied the implications of the local region surrounding feature points. In particular, a class of point features known as junctions have been used to solve a number of geometric problems in rasterized frameworks.

Junctions are locations in an image where edges intersect. If a feature point has no incoming edges, it is an impulse. Distributions of impulses can provide clues to the locations of illusory contours as well as to the orientation of surface geometry. Junctions with one incoming gradient are line endings.

Junctions with two incoming edges may be classified in one of two ways: continuous or discontinuous. If the edges leading into the junction are smooth across it, the junction can be considered an edge point. Edge points are less significant junctions than 2-way junctions with a discontinuous derivative. Junctions at locations where two incoming edges do not smoothly complete are referred to as L-junctions. The incoming edges at an L-junction need not be orthogonal to each other. Tse and Albert discuss the existence of 2-way junctions with curvature discontinuities [34]. They note that these junction types demonstrate surface penetration.

Several varieties of 3-way junctions exist. 3-way junctions where none of
the incoming edges smoothly complete each other are referred to as Y-junctions. These tend to be indicative of three dimensional corners and visually project or recede into the image (Figure 2.6). If an edge can complete across the junction, then the 3-way junction becomes a T-junction. If two edges can complete with one other, the classification is dependent on the respective curvatures of the incoming edges. If the incoming curves are nested, the point seems to indicate an object being sliced. The inner curve is promoted to the foreground.
Figure 2.6: A well known optical illusion demonstrating the three dimensional bistability of Y-junctions. Are the red regions squares on top of boxes, below the boxes, or are they diamonds?

The significance of L and T-junctions is explored by Rubin [28]. In her work, she proposes that these junction types launch completion search processes. Geiger et al. use L and T-junctions detected in a raster image to reproduce the fusing results seen in Kanizsa’s Square and the four crosses image, as well as the spontaneous fission demonstrated by the two fish image [9]. Beyond the strict surface isolation and generation, they also succeed at layering surfaces based on this information.

Figure 2.7: Images of Kanizsa’s Square (a), the four crosses image (b), and the spontaneous splitting two fish image (c) described by Geiger et al.[9].

In the presence of a completion, T-junctions hint that the object at the head of the ’T’ is above the objects on either side of the stem. If the junction
does not form a completion, the object at the head of the ’T’ is treated as if it were in the background. If L-junctions do not complete, they are typically treated as foreground. If L-junctions complete one edge, they become T-junctions, where the stem has not completed. If both edges complete, additional information about the surfaces is needed to finalize the completion order.

4-way junctions are related to reflections, refraction, and transparency. Higher order junctions exist, but are less common. While junctions of four or more edges are potentially significant, they will not be considered in this work.

It is clear from the works by Rubin, Geiger, and Pao ([28], [9]) that using L and T-junctions to predict completions has a considerable amount of potential. They are not, however, readily detected. There are several approaches that may be taken to find junctions. One possible approach is to modify an existing point detector to improve it’s ability to distinguish of this nature. The FAST feature detector designed by Rosten and Drummond shows potential as a starting point. The algorithm performs a search along a Bresenham’s circle around each test point. If enough points can be grouped in a row, the algorithm will mark the location as a corner.

A modification of this approach searching for the number of peaks in the gradient along the circle may viably provide the information needed to classify the junction types rapidly. Work by Parida et al.[24] does just this. The algorithm creates an energy model and tests a scale dependent torus around a feature point. The local region descriptor is analyzed and the junction type is classified.

While using an independent detector to find junctions is an option, there is an alternative. If the system is already searching for edges directly, some
junctions may be located through a nearest neighbor search at the end-points. The scale of the end-points themselves provides a natural search range. Junctions located in the middle of edges require the edges be progressively fit to a model edge. Junctions are locations where the model cannot be fit. This progressive model fitting is known as vectorization.

2.4 Vectorization

Vectorization is a compressive process, which reduce the complexity of a rasterized image by replacing collections of data samples with a simpler set of equations. In the case of edges, vectorization allows for the conversion of an ordered set of points into a simple curve. The process also improves estimations based on the data by building up a data-sensitive region of support.

Early research on vectorization focused on fitting parametric polynomials to raw data. This gradually evolved into the development of splines. Splines are curves built up through the progressive concatenation of piecewise polynomial pieces of order \( n \). The \((n - 1)^{st}\) derivative of a typical spline is an impulse train. Splines therefore can be expressed using a very small amount of data [35]. Research into splines has searched for ways to minimize the amount of data needed to express complex curves. Through careful selection of knot locations (impulses in the derivatives), splines can be fit to a broad selection of shapes.

If knots are poorly selected, splines will create oscillations in the output curve. If mandates on preserving monotonicity are enforced, as with MATLAB’s implementation of Hermite splines, the splines can smooth over local
extrema. The input to the vectorizing process consists of quantized samples. This means that samples located at an actual junction’s location are very rare. For Lindeberg’s algorithm [19], the points exist at intersections between the curves and a regular grid (Figure 2.8). If the change in slope near discontinuities is smooth enough, the vectorization algorithm may not notice them. This can result in overlooking features which could have produced perceptual completions.

Figure 2.8: The relationship between an edge and its detected points.

Splines provide a powerful method for representing curves as mathematical segments, however they currently do not lend themselves well to the detection of corners. While splines may be applicable to junction detection in the future, this work restricts its vectorization techniques to the use of parametric polynomials for simplicity.

Fundamentally, vectorization is a grouping problem. Given the image $I(x)$, an edge detector locates edge points, $\tilde{P}_m(i)$, and groups collections of them into edges. Each ordered collection of points is the poly-line $\tilde{P}_m$ of length $t$ with $I$ elements. The edge detector stage produces $M$ such poly-lines, which collectively form the edge map $\tilde{P}$. An individual poly-line may consist of information intrinsic to several objects. The vectorization stage must gradually traverse $\tilde{P}_m$ and progressively build an estimation of the curve $\tilde{C}_n$. 
which best fits the points. As the strength of the fit improves, more points are added until the fit strength peaks. At this point, \( \tilde{P}_m \) is split and a second fit, \( \tilde{C}_{n+1} \), is started. This process is repeated for the full length of \( \tilde{P}_m \) for all \( \tilde{P}_m \) in \( \tilde{P} \).

The process attempts to distinguish information which is known to come from multiple sources; it must be able to support the presence of gross errors. To combat these errors, robust estimation techniques are necessary. The resulting system must be able to isolate first order discontinuities to allow for the completions described by Rubin [28]. Should it also be desirable to perform completions caused by object penetration, the fits must be able to distinguish sharp changes in curvature as well [34].

Selecting an appropriate function for performing completions exposes a trade-off between computational accuracy and ease. The data points in the image are spatially related, but have no temporal correlation. The ideal system to fit these points must effectively solve for two output variables in terms of themselves. The only functions capable of doing this are natural functions. Natural functions are functions which are uniquely describable in terms of their arc length and curvature. Commonplace examples of natural functions include the line, the circle, and the logarithmic spiral.

Smooth natural functions lend themselves well to the detection of sharp changes in curvature. Lines, as first order polynomials, are rotationally symmetric and exceedingly simple to work with. Circles are guaranteed to have unit curvature. Clothoids minimize curvature change over arc-length. Work by Kimia et al. states that clothoids are the ideal curve to use in object completion for this reason.
The clothoid is described by the Fresnel integrals. These functions are useful in optical calculations, but are not algebraically expressible. Many natural functions, including the circle, are described by transcendental functions, or functions of multiple variables. This makes integrating them into robust estimation techniques challenging. In contrast, polynomial fits are easily adaptable to robust estimation techniques.

Robust estimation is based on a reformulation of the least squares algorithm used to solve the linear system $Ax = b$. A set of orthogonal basis functions $f(t)$ and an initial prediction, $C_p$, of the model that fits the points are needed to prime the technique. For all $\tilde{P}_m(i)$ in $\tilde{P}_m$, an error metric $\epsilon_i$ is generated to determine how far off $\tilde{P}_m(i)$ is from $C_p$. This error metric is converted to a weight $w_{\epsilon_i}$ using $w(\epsilon)$. The function $w(\epsilon)$ is an even symmetric weighting function bounded between $\{0, 1\}$ and monotonically decreasing for $\epsilon \in R^+$. The values are accumulated using Equations 2.11 and 2.12, and the system can be solved via Gaussian elimination.

$$A_{uv} = \sum_{i=0}^{I} w(\epsilon_i) f_u f_v \quad (2.11)$$

$$b_u = \sum_{i=0}^{I} w(\epsilon_i) f_u P_m(i) \quad (2.12)$$

Robust estimation literature refers to this as the Iteratively Re-weighted Least Squares (IRLS) method. An M-estimator, the origin of the weight function, can be defined in one of three ways: $\rho(\epsilon)$, $\psi(\epsilon)$, or $w(\epsilon)$. The typical definition of M-estimators is by their $\rho(\epsilon)$ function. This is a positive semi-definite, even symmetric function with its origin at zero. When solving for a fit with a robust estimator, the goal is to minimize the sum of the modified residuals $\sum_i \rho(\epsilon_i)$. If $\rho(\epsilon)$ is monotonically increasing as its distance to
the origin increases from zero, the function will guarantee a unique result. If there is a unique location where the derivative of $\psi(\epsilon)$ is equal to zero, the system will approach a unique fit. The derivative of $\rho(\epsilon)$ is $\psi(\epsilon)$. The function $\psi(\epsilon)$ is referred to as the influence function of the estimator. The weight function is related to its respective influence function by Equation 2.13.

$$w(\epsilon) = \frac{\psi(\epsilon)}{\epsilon} \quad (2.13)$$

One system of equations ideal for integration into a robust estimation system are the Legendre polynomials. This set of polynomials forms an orthogonal basis between $\{-1, 1\}$. To smooth the resulting fits, the cumulative Euclidean distance between sample points can be used as the independent variable $t$ [12]. The values for $X$ and $Y$ can be solved for simultaneously using separate $b$ vectors and common accumulated $A$ matrix. The weight values used for $A$, $b_x$, and $b_y$ should be a single value for each point to allow the system to be solved once. This is justified since the $x$ and $y$ coordinates for a single point are equally valid if the point is valid.

Image vectorization offers a number of advantages over rasterized analysis techniques. The resulting structures are feature dense, requiring very few parameters to describe length and shapes of the curves. The resulting curves are also very stable; information from many point sources is accumulated into the construction of a single curve while simultaneously smoothing out noisy data and splitting at gross changes. Curves generated in this fashion can be used to predict the locations of edge junctions through the use of spatial indexing techniques. In addition, if the selected model can be solved for by the end-points alone, the same equations may be applied to completions. Bi-cubic polynomials can be uniquely described by the locations and
derivatives of their end-points for a given arc-length. This makes them ideal for representing visual edges, as well both modal and amodal completions.

2.5 Object Completion

Object completion is the process of converting object fragments viewed in an image into predictions of an objects true shape. Object fragments are defined in a view by their bordering edges. These edges are either intrinsic or extrinsic to the fragment. Intrinsic edges are defined by the objects geometry directly. Extrinsic edges are instead defined by the geometry of an occluder, which blocks the light reflecting off the first object from reaching the viewer directly.

The completion process starts by closing off the contours provided by the vectorization stage. The hybrid-illusory/visual curves that result from this process are traced. A weight based on arc-length and curvature of the edge along the loop determines if the shape is a hole in a bounding shape or the border of an object.

Closing the contours in the image is a several step process. The output of the previous stages consists of polynomial fits of the poly-line edges connected at their end-points to junctions. Each of the junctions in the image must be tested for short, smooth connections, and these connections closed. The remaining end-point vectors must be tested in the same fashion against each other using curvature and arc-length constraints to determine the quality of the respective fits. Finally, the edges need to be traced. This allows a surface to be constructed. The layering of surfaces is determined by which
polynomials fits and junctions were traced, as well as their respective convexity.

Local fits at each junction distinguish between rays emanating from the point and lines crossing it. Their purpose is to distinguish between the head and stem of a T-junction. There are several methods for determining the difference. Since a robust fit has been made of the original edge data, it is possible to solve for the model’s local derivative at the junction. A simple metric of fit quality can be determined by comparing the difference between the incoming angles to a threshold. This technique is afforded some stability by the fact that robust techniques are used in earlier stages to produce the initial fit.

While this technique can be exceedingly fast to calculate, it may not be as robust as the alternative. If the original edge data is still available at this stage, then a robust fit can be attempted across the junction point for each pair of connected edges. Should a fit of sufficient length complete, then the edges can be treated as continuous.

The problem of matching end-point vectors becomes significantly more complex at larger scales. A typical 640 × 480 image may have end-point vectors in the thousands to ten thousands. Kimia et al.[15] note the quality of a fit is determined partially by the distance between the end-points, and partially by the integral of the change in curvature along the fit. This is minimized by the clothoid, the arc length of which, they suggest, may be used to calculate the validity of a given fit. Clothoids are built using the Fresnel integrals (Equations 2.15 and 2.16), which are trancendental functions
related to the error function (Equation 2.14).

\[ \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-(t^2)} dt \]  
\[ (2.14) \]

\[ S(s) = \int_0^s \sin\left(\frac{\pi}{2} \xi^2\right)d\xi \]  
\[ (2.15) \]

\[ C(s) = \int_0^s \cos\left(\frac{\pi}{2} \xi^2\right)d\xi \]  
\[ (2.16) \]

The relationship between clothoids and the error function means they are inexpressible in closed form, although they can be estimated through their taylor expansions. The arc-length of a clothoid segment can be found using Equation 2.17, which states that the change in length is equal to the change in angle divided by the average curvature [15]. To avoid the complexity of the calculations required to find the clothoid segments directly, they can be simulated with a polynomial fit. The quality of the particular fit may still be calculated based on the angles and curvature at the end points, preserving the quality of the selected fit.

\[ s = \frac{2(\theta_0 - \theta_1)}{\kappa_0 + \kappa_1} \]  
\[ (2.17) \]

\[ \kappa = \frac{1}{r} \]  
\[ (2.18) \]

\[ s = \frac{\theta_0 - \theta_1}{\kappa} \]  
\[ (2.19) \]

The arc-length of a clothoid is closely related to the arc-length of a circular arc-segment, which is solved for in Appendix A. An approximation for the arc-length is proposed which ensures that if the end-point vectors being fit are co-circular, the fit is co-circular as well. If end-point vectors \( \vec{A} \) and \( \vec{B} \) are tangent to and located on the perimeter of a circle, the arc-length between them can be found using Equation 2.20, where \( \theta \) is either \( \arccos \frac{\vec{A} \cdot \vec{AB}}{\|A\|\|AB\|} \) or
arccos \left( \frac{\vec{B} \cdot \vec{AB}}{\|\vec{B}\|\|\vec{AB}\|} \right)

To extend this measure to more complex curves, Equation 2.20 is modified such that it is based on the average curvature at each endpoint, and the average of the absolute angle away from a linear fit (Equations 2.21 and 4.1). In the case that the fit is linear, an arc-length of the chord-length must be returned.

\[ l_{\text{circle}} = \frac{\|\vec{AB}\|\theta}{\sin \theta} \quad (2.20) \]

\[ l_{\text{curve}} = \|\vec{AB}\| \frac{\theta \kappa + \theta \bar{B}}{2 \kappa + \kappa \bar{B}} \quad (2.21) \]

\[ = \|\vec{AB}\| \frac{\arccos \left( \frac{\vec{A} \cdot \vec{AB}}{\|\vec{A}\|\|\vec{AB}\|} \right) + \arccos \left( \frac{\vec{B} \cdot \vec{AB}}{\|\vec{B}\|\|\vec{AB}\|} \right)}{\sin \theta \vec{A} + \sin \theta \vec{B}} \quad (2.22) \]

\[ = \|\vec{AB}\| \frac{\arccos \left( \frac{\vec{A} \cdot \vec{AB}}{\|\vec{A}\|\|\vec{AB}\|} \right) + \arccos \left( \frac{\vec{B} \cdot \vec{AB}}{\|\vec{B}\|\|\vec{AB}\|} \right)}{\|\vec{A} \times \vec{AB}\| + \|\vec{AB} \times \vec{B}\|} \quad (2.23) \]

Using Equation 4.1 to solve for the arc-length provides several direct benefits to the system. First, it ensures that if a curve fit is circular, the estimate will be correct. Secondly, it provides a convex bound (a circle with radius equal to the measured arc-length) outside of which end-points need not be tested for fits. This quick bounding test can prune the number of complex calculations (such as the arc cosines) which need to be performed. This acceleration can be achieved through careful use of spatial-indexing structures like Binary Spatial Partitioning trees.

Spatial indexing methods were originally developed to provide acceleration to 3D graphics algorithms. They are ideal for reducing the amount of execution time for rendering geometry on a screen from a given viewpoint to a \( O(n) \) traversal of the tree. For a starting list of vectors, they are constructed
by splitting the image at an arbitrary end-point vector. Any vectors starting in front of the root vector are moved into one list, while those behind are moved into a second list. This technique is repeated recursively on each of these lists, appropriately setting the root of the front and back trees to improve balance. In a perfectly balanced tree, it takes $O(\log n)$ comparisons to reach a leaf node from the root.

The usefulness of a tree derives from the reduced cost of determining which side of a branch has likely candidates over checking each individual candidate as a fit. A balanced tree which is able to produce initial guesses which are close to the correct fit will likely produce a limiting metric that can guarantee the far side of a split need not be tested. If the distance between the splitting line and the vector being tested is greater than the arc-length of the prospective fit, then it is impossible for a shorter fit to exist at a longer distance.

To reduce the amount of error in the image, fits are preserved if and only if the best fit for two vectors is each other. Edges which are not closed introduce problems into the system. To remedy this, multiple passes on the vector graph are made, pruning completed fits. This will be discussed further in Chapter 5. Once an edge loop is closed in a smooth fashion, it is treated as a single, contiguous surface. This means that each edge loop is intrinsic to the same object, which is either bounded by, or bounding the gap formed by the edge.

Convexity of edge segments and at junctions provides a visual clue to help determine whether a loop is bounding a hole or an object [23]. To solve for the convexity of a shape, the collected borders of the surface are traversed. Since each poly has an estimate of arc-length provided from the
earlier stages, this can be multiplied by the sign of the convexity and accumulated across the curve.
Chapter 3

System Design

Many design choices go into a system as complex as this one. Decisions as fundamental as the choice of language, or as complex as the use of robust statistics, can have drastic implications on the flow of development and success of a project. Section 3.1 focuses on the reasoning behind the choice of programming language. This is followed by sections 3.2, 3.3, and 3.4 which look into the edge detector, vectorization, and completion stages of the system respectively. Finally, Section 3.5 discusses the format of the data as it flows through each stage.

3.1 The Language

Originally, the system was to be implemented in MATLAB. A powerful IDE and language, MATLAB is designed to simplify matrix arithmetic. The language provides an over-abundance of graphical support and mathematical operations. The plethora of functions available can make the implementation of systems based heavily on convolutions and complex matrix math exceedingly simple.
While the tools available in MATLAB reduce the complexity of matrix manipulations, the environment is a heavy memory consumer. The limited flexibility of MATLAB’s memory management exposed the memory constraints of the system prematurely. Even for images of modest sizes (640 × 480), execution of the Lindeberg Edge Detector alone could easily consume over a gigabyte of RAM. This problem was mitigated in part by splitting the analysis of the image into smaller steps, but further challenges arose once the system was no longer operating on arrays of image data. MATLAB is inefficient at executing loops. Attempting to traverse a scale-space volume using a pure MATLAB source could take minutes a frame. This slowdown made testing the system problematic.

To resolve this issue, MEX files, a C-based MATLAB support executable, were implemented to enhance different aspects of the code. While this greatly accelerated some of the key areas, it was decided that the sensible approach required dropping the MATLAB code in favor of a system written purely in C. The lack of graphical and matrix commands was supplemented with an external API. This drastically improved memory management and speed issues.

The API of choice was OpenCV, a powerful library of functions provided by Intel. The API supports a wide number of formats and built-in functions, including some limited scale-space support. OpenCV’s format for storing image data is fairly flexible. They supply API necessary for not only rendering images on the screen from any bit and channel depth supplied, as well as basic drawing commands for lines, circles, and simple polygons.

While extremely useful for using pre-built functions, it was not easy to adapt to the implementation of low-level functionality. The scale-space
implementation provided by OpenCV forces a blur with the separable kernel. Their implementation of sequences does not allow for their direct deallocation. They also do not provide an optimized means of applying arbitrary separable filters.

The flexibility of the C programming language, in addition to its light weight, make it, and its derivatives, ideal languages to work with. The ability to reference information via pointers is particularly useful for keeping track of the relationships between the outputs of the different stages of processing.

One particularly useful feature of the C programming language is a powerful pre-processor. Through careful use of this, the code in each stage has been parameterized to allow functional portions of each of the stages to be disabled on the fly for testing. This also allows variations of the code to be compared against one another.

### 3.2 The Edge Detector Stage

The greatest challenge during the implementation of this work has been implementation of the edge detector stage. Minute changes in the implementation drastically affect the uniformity of the response of the edge detector across different scales and rotational inputs. The final choice for the edge detection algorithm was the Lindeberg Edge detector [19]. This algorithm is designed to detect edges at unique maxima in a scale-space volume. The technique localizes edges at their peak change in intensity, producing results which are designed to be invariant to scale. The output edges tend to be located in similar locations to the zero-crossings detected by the LoG operator. To accelerate the processing and reduce memory constraints on
the system, hybrid scale-space pyramids as described by Lindeberg’s later work were integrated into the approach [20].

The first step toward implementing the Lindeberg algorithm was to prepare the scale-space filtering module. This was implemented, taking considerations from both Lindeberg’s works on edge detection and hybrid scale-space pyramids, as well as work by Eaton et al. clarifying scale-space considerations [19] [20] [7]. Witkin and Lindeberg both discuss the significance of the Gaussian kernel in the creation of scale-space pyramids ([38], [20]). The Gaussian kernel uniquely removes high spatial frequency information without creating new features at higher scales for an arbitrary image. Lindeberg suggests that the kernels in Equations 3.1 and 3.2, and further convolutions of them, may be used as drop-in discrete replacements for the Gaussian kernel. For both kernels, the value of ∆t equals the variance of the kernel. For the rotationally symmetric kernel, λ adjusts the kernel’s angular response. Lindeberg states that when λ = \( \frac{1}{3} \), the kernel is optimally rotationally symmetric [20]. If ∆t = λ = \( \frac{1}{3} \), the rotationally symmetric kernel becomes separable. These kernels support variances in the range of [0, \( \frac{1}{2} \)].

\[
\begin{bmatrix}
\frac{\Delta t^2}{4} & \frac{\Delta t(1-\Delta t)}{2} & \frac{\Delta t^2}{4} \\
\frac{\Delta t(1-\Delta t)}{2} & \frac{\Delta t(1-\Delta t)}{2} & \frac{\Delta t(1-\Delta t)}{2} \\
\frac{\Delta t^2}{4} & \frac{\Delta t(1-\Delta t)}{2} & \frac{\Delta t^2}{4}
\end{bmatrix}
\]

(3.1)

\[
\begin{bmatrix}
\frac{\lambda \Delta t}{4} & \frac{\Delta t(1-\lambda)}{2} & \frac{\lambda \Delta t}{4} \\
\frac{\Delta t(1-\lambda)}{2} & 1 - (2 - \lambda) \Delta t & \frac{\Delta t(1-\lambda)}{2} \\
\frac{\lambda \Delta t^2}{4} & \frac{\Delta t(1-\lambda)}{2} & \frac{\lambda \Delta t^2}{4}
\end{bmatrix}
\]

(3.2)

The Generation of scale-space itself is then straightforward. To reduce the amount of information stored in memory at any given point in time,
the blurring and sub-sampling steps were interlaced with the edge detection process. The algorithm described by Lindeberg uses four metrics to determine the strength of an edge across the image space and across scale respectively (Equations 2.7, 2.8, 2.9, and 2.10). These equations mix first, second, third, and fifth order derivatives. Both even and odd order derivatives must be aligned to the same pixel. For this reason, the first and second derivative kernels used to calculate these are forced to have odd dimensions. Lindeberg recommends the use of the central differences operator [19], but it should be noted that Eaton et al., in a later work recommend against its use in scale-space calculations since the results after sub-sampling do not match what is expected [7].

To prime the edge detection loop, the values for each of these metrics are calculated for the fine image. Each loop iteration blurs the fine image to produce the next coarser scale. The values of $P_t, P_{tt}, \tilde{L}_{vv}$ and $\tilde{L}_{vvv}$ are calculated for the new coarse image, and then all eight inputs (four metrics for two layers of scale-space) are dropped into a routine which performs edge detection on a per-voxel basis. Once the edges in a given layer have been detected, the fine and coarse arrays are swapped, and the process repeated where the coarse scale from the previous iteration is the new on the new fine scale.

To simplify data manipulation, a variable pointer (vPointer) with two data members was created. The members of the vPointer consist of a typical pointer to a location in memory, as well as a type descriptor. The addition of the type descriptor allows the system to distinguish between the endpoints of the same edge. It also allows the same structure to distinguish between edges of poly-lines, polynomial fits, junctions, and vertices, allowing the edge detection stage to interact with structures of different types
while searching for edges and their vertices.

This system allows edge data to propagate along the front faces of each of the voxels. The system will detect four distinct voxel types. Voxels either contain no edge information, are brand new data, connect to old data, or are a mix. If a voxel is connected to two sources of edge data, including the information propagating from the prior cell, row, or tier, the voxel is treated as a part of an edge. If a voxel is connected to one edge source, or three or more sources, it is treated as a vertex. Edge cells connecting to new data will be extended with new sample points at the appropriate end. Vertices propagate connectivity information only.

Voxel tracing proceeds linearly across each cell of each row and each row of each tier of the scale-space volume. The process within a voxel is implemented to match the description by Lindeberg [19]. The values for $P_{tt}$ and $\tilde{L}_{vvv}$ are sign tested at each corner of a voxel. If both are negative within the voxel, the stage will attempt to find a zero crossing edge. Once a voxel has been validated, each of the 9 sides of the voxel touching an advancing face are tested for a zero crossing for both $P_t$ and $\tilde{L}_{vv}$. If two zero-crossings are detected on a face for both $P_t$ and $\tilde{L}_{vv}$, the system will connect the pairs with lines and test for an intersection point between the two surface metrics within the face.

When an invalid cell (one where $P_{tt}$ or $\tilde{L}_{vvv}$ are positive for all corners) is encountered, the program will terminate all incoming edges at individual junctions without starting new ones. Separate functions close the last cell face of each row, the last row of each tier, and the last tier of the volume in the same fashion respectively.

The final piece necessary for implementation of the modified Lindeberg
The edge detector is hybrid scale-space support. The integration of the two is not described by Lindeberg. In addition to the normal blur term needed to support Equations 2.9 and 2.10, the system requires a coarseness term $h = 2^n$ where $n$ is the number of times the image has been sub-sampled. When blurring at coarser scales, the blurring effect of a Gaussian filter with variance $v$ is equal to $v \times h^2$. This allows $P_{tt}$ and $\tilde{L}_{vvv}$ to evolve correctly when sub-sampling.

When sub-sampling the image, it is vital that the system compare the pre-sub-sampled image either to the sub-sampled variant directly, or to a re-up-sampled copy to avoid losing scale ranges in the scale-space. In addition, the prior-tier created immediately before a sub-sampling step must be sub-sampled appropriately. This was done in this work by performing a voxel trace through the tier for every voxel in the destination image.

Voxels clustered in groups of four were tested for the presence of edge or vertex connections. If one of these super-voxels contained a single edge or vertex connection, that connection was forwarded through. If more connections were present, and at least one was a vertex, all the edges were merged into the vertex with the most connections. If no vertexes were present, a new one was made, and the ends of the edges in the super-voxel were joined to it and closed if appropriate. Voxels on the border of the image were merged into their adjacent super-voxel, with a maximum of a $3 \times 3$ voxel super voxel on the bottom right corner of the image. This voxel alone has the potential of holding up to 13 connections at once.

The decision to fuse the border voxels into the main voxels of the image was made based on the behavior of the sub-sampling process. If an image $I(x)$ has a height of 15 and a width of 16. Since there is one less voxel in
each dimension than the pixel with, this corresponds to a voxel plane \( V(x) \) with a height of 14 and a width of 15. Sub-sampling the even values should return exactly half the original size. The result of sub-sampling the odd values can be solved for by noting that an image with a dimension \( D \) has \( D - 1 \) voxels along that dimension. The height of the sub-sampled image \( H = 7 + 1 \), while the width of the sub-sampled voxel plane \( W = 8 - 1 \), therefore the resulting surfaces from the sub-sampling process should have resolutions of \((8, 8)\) and \((7, 7)\) respectively. Odd pixel widths should be rounded up, while odd voxel widths should be rounded down.

Lindeberg’s work on hybrid pyramids [20] states that the most the image can be sub-sampled at a given stage is proportional to the standard deviation of the total blur up to that point \( h_{\text{max}} = \rho \sigma_{\text{net}} \). He defines the base blur implicit in an arbitrary image to be \( t_{\text{start}} \) and the blur applied to the image by the blur kernel at the finest tier to be \( \Delta t_{\text{cycle}} \). If the total blur between sub-sampling operations is \( \Delta t_{\text{stage}} = h^2 J \Delta t_{\text{cycle}} \) where \( h \) is the coarseness of the tier and \( J \) is the number of times the blur is applied between sub-samples, then the system forms a series for each tier. Solving this equation for \( \rho \) and \( t_{\text{start}} \) yields Equations 3.4 and 3.5.

\[
2^{2(h-1)} = \begin{cases} 
\rho^2 t_{\text{start}} & h = 1 \\
\rho^2 (t_{\text{start}} + \sum_{n=0}^{h-2} 2^{2n} \Delta t_{\text{cycle}}) & \text{otherwise} 
\end{cases} 
\]  

\[
\rho^2 = \frac{3}{\Delta t_{\text{cycle}}} 
\]  

\[
t_{\text{start}} = \frac{\Delta t_{\text{cycle}}}{3} = \frac{1}{\rho^2} 
\]  

Equations (3.3), (3.4), and (3.5)

This provides the final piece needed to detect edges. The image should only be sub-sampled when the net blur multiplied by \( \rho \) increases beyond the next
coarseness threshold. Once the edges have been extracted from their peaks in scale-space, one question remains. What can be done with them?

3.3 The Vectorization Stage

The output of the edge detector stage consists of poly-lines in a three-dimensional volume. While this data provides a strict representation of the edges in the image, it is not entirely useful to a system that requires the ability to find the local derivative or curvature. To do this, the data must be fit to a model. This is done through the use of a robust statistics based fitting mechanism.

The robust fitting mechanism implemented here follows the IRLS guidelines described by Hampel et al. and Maronna et al. ([13], [22]). The basic principles behind the theory have been laid out in Section 2.4. Due to the simplicity of providing drop in replacements for the weight function, support for Least Squares (LS), Fair, Geman-McClure, Cauchy, and Tukey robust estimators was added. For a useful source of information regarding these different types of estimators, including functional and graphical representations of their respective $\rho(\epsilon)$, $\psi(\epsilon)$, and $w(\epsilon)$ functions, refer to [43].

The Tukey estimator is of special interest, since its influence function is re-descending. Re-descending estimators sacrifice unique responses to improve the estimators breakdown point (tolerance to the presence of outliers). When the Tukey estimator detects outliers beyond a certain range, it will ignore them completely. If a predicted fit is too far from the actual data,
Table 3.1: A variety of M-Estimators

<table>
<thead>
<tr>
<th>Names</th>
<th>$w(\epsilon)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS</td>
<td>1</td>
</tr>
<tr>
<td>Fair</td>
<td>$\frac{1}{1+</td>
</tr>
<tr>
<td>Geman-McClure</td>
<td>$\frac{1}{(1+x^2)^2}$</td>
</tr>
<tr>
<td>Cauchy</td>
<td>$\frac{1}{1+x^2}$</td>
</tr>
<tr>
<td>Tukey</td>
<td>$\begin{cases} 0 &amp; \text{if }</td>
</tr>
</tbody>
</table>

re-descending estimators risk dropping the data completely. Despite this detectable case, their ability to completely ignore gross outliers can produce the most accurate results of any of the robust estimators.

The Least Squares estimator is the only one that responds linearly to the residual. The influence functions of all the others have a central range where the influence of a small residual is considerably greater than that of a large one. The range of these regions can be tuned by a constant to adjust the sensitivity of the algorithm to data that does not perfectly match a given fit. This sensitivity is necessary for the IRLS algorithm to work. If the range of sensitivity is too large, the algorithm will accept outliers into the system. If the sensitivity is too small, it will converge prematurely.

This threshold should also vary with scale. The coarser the scale a point is detected at, the lower the accuracy of its location. This means that it should be penalized less for being at greater distance from a fit than a point at a fine scale should. The algorithm can be further weighted so that the accuracy of fine points is assumed to be greater than that of coarse ones by multiplying the output response of the weight function by the inverse of the scale. In this case, if a line consists of points at a single scale, they are equivalently weighted, but if it connects to finer points as well as coarser ones, the fit will
focus on the fine points to a greater degree than the coarse ones.

The robust fitting mechanism builds a fit by assuming every point on a given source poly-line is valid to some edge in the scene. This formulation changes the problem from the detection of outlying points which do not belong on any line to the detection of outlying points which do not belong on the same fit curve as the current point.

Several steps are involved in the robust fitting process. The fits are made against either parametric second or third order polynomials. The second order polynomials are preferred as they cannot support inflection points or sharp corners. These polynomials are therefore guaranteed to have a constant sign of curvature, which may be used to help determine surface ordering as described by Pao et al.[23]. Generated fits are time-scaled to $t = \{-1, 1\}$ during the fitting process for simplicity. Legendre Polynomials (Equation 3.6) are orthogonal across this range, and can be used to produce the actual fit. The fit is tested against points outside this range to generate the weights used to produce the next scaled fit. Only once a fit is finalized is it converted to a monomial form with a time scale of $t = \{0, t_{max}\}$ where $t_{max}$ was the accumulated Euclidean distance between adjacent points along the poly-line segment being fitted.

$$\begin{align*}
1 & \quad n = 0 \\
x & \quad n = 1 \\
\frac{1}{2}(3x^2 - 1) & \quad n = 2 \\
\frac{1}{2}(5x^3 - 3x) & \quad n = 3 \\
\frac{1}{8}(35x^4 - 30x^2 + 3) & \quad n = 4
\end{align*}$$ (3.6)
In addition, the fitter modifies weights through a cumulative minimum algorithm. As the fit is produced, if the weight ever drops, no points after that may have stronger support than that one. This prevents fits which breach gaps in line data. Figure 3.1 shows an example of a problem that is resolved by this successive minimum technique. The blue line is an initial fit based of the first two points. The brighter the blue on the dots, the stronger their influence on the line fit. Red dots have negligible influence. With cumulative minimums in place, the blue dots on the far side of the gap are ignored.

![Figure 3.1: Demonstration of fit sensitivity across gaps without cumulative minimum thresholding](image)

The algorithm will fail if the accumulated matrix is singular. When this happens, there are several options available. The solution used by this work reduces the polynomial order of the attempted fit. The prediction that bailed out is used to generate the new weights for the simpler model. Should the fit order ever reach zero, the system will return the initial two point linear fit and move on.
3.4 The Completion Stage

The completion stage is designed to perform several tasks. First, it extracts the end-points from the vectorized polynomial fits provided as an input. These end-points are locally tested against one another for trivial fits to prevent interference during later processing. Matched end-points are pruned. The remaining end-points are searched against each other for best possible candidates for closure. Finally, fits are generated between matched end-points. Typically, a matching procedure with $n$ vector inputs requires $O(n^2)$ time to compute. If the weights used to measure the validity of the fits are related to the arc-length of the fits, then the global matching process can be accelerated close to $O(n \log n)$.

The end-point extraction step generates an end-point vector at each of the end-points of a polynomial fit by calculating the derivatives of the fits at the end-points directly. Starting from a bi-cubic polynomial fit (Equation 3.7), the end-point derivatives are defined by Equation 3.8. Before passing the end-point vectors out to be globally match, a local matching operation is used to determine if any pair of end-vectors would form a smooth edge across the surface.

\[
\begin{bmatrix}
x \\
y
\end{bmatrix} = \begin{bmatrix}
x_0 + x_1 t + x_2 t^2 + x_3 t^3 \\
y_0 + y_1 t + y_2 t^2 + y_3 t^3
\end{bmatrix} \tag{3.7}
\]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
x_1 + 2x_2 t + 3x_3 t^2 \\
y_1 + 2y_2 t + 3y_3 t^2
\end{bmatrix} \tag{3.8}
\]

Once the end-point vectors have been extracted from the edge fits, they are locally tested for smoothness through careful use of the dot product operator and a threshold. The purpose of the local fitting step is to identify
junctons which arise from limitations on the bi-cubic’s ability to approximate a smooth curve, as well as to prevent forming fits between vectors at the head of T-junctons. While crude, the search at this step is inexpensive in comparison to the calculation needed to find arc-length, and so vectors pruned at this stage will often save time later.

The unmatched edges remaining after the local fit process completes are placed into a Binary Spatial Partitioning (BSP) tree. BSP trees are spatial indexing structures designed to simplify searches for nearest neighbors. To generate the tree, each vector is treated as the normal for a line in the image space. All vectors in front of a root vector’s line are placed in the front tree. All vectors behind the root vector’s line are placed in the back tree. The process is performed recursively on the list of vectors in front of and behind the root vector.

BSP trees accelerate searching in two ways. The first, trivial way encourages vectors close to the target vector to be searched first. Only those vectors which are on the same side of all parent vectors will be down the same branches of the BSP tree. The calculation to determine vector ordering is a sign test on the dot product between the root vector and the vector pointing from the root to the target.

The second benefit of BSP trees arises from using the arc-length of the fit curve to determine the quality of the fit. If the arc-length is used, it is guaranteed that no vector outside of a circle centered on the target vector with a radius equal to the fit’s arc-length can possibly have a shorter arc-length. Since vectors split the image into hemi-planes (half-planes), determining if it is possible for a vector on the far side of a split to be a viable fit is equivalent to determining if the distance between the target vector and the root
vector splitting the image is greater than the fit’s arc-length.

The benefits of BSP trees can only be realized if they are relatively balanced. To achieve this, this system runs a comparison of all vectors in a branch on the first $n$ vectors in that branch, incrementing a weight if the vector is in front, and decrementing if it is behind it. At the completion, the vector who’s weight is closest to zero is used as a pivot, and the vectors are split and the process continues.

![Figure 3.2: A small sample BSP space. The green vector is a local root node. It splits the space into two hemi-planes. In the front hemi-plane is the blue vector, which has a fit length that passes the hemi-plane border. The red vector in the back hemi-plane, on the other hand can safely ignore things in the front hemi-plane since its fit is shorter than the distance to the border.](image)

Once the end-point vectors have been properly accumulated into the BSP tree, the global fitting may proceed. Global fitting steps through each vector in the BSP tree and searches for it’s best match. The arc-length is found using Equation 4.1 as described above. After all the vectors have been matched against the main tree, those which mutually agree on the match are pruned and fitted. The remaining edges are iteratively pruned until an equilibrium is reached.

Once the end-point vectors have been matched, the system must generate a
polynomial fit. This can be done using Equations 3.7 and 3.8. Using the arc-length prediction as the value for $t$, and the known values for the derivatives at the end-points, it is possible to derive the value of the coefficients for the bi-cubic. The derivation of the fit is shown below. For simplicity, let the vector function $V(t) = \frac{x(t)}{y(t)}$. If the end-points of the function are set at times 0 and $t$, then their locations are defined by Equations 3.9 and 3.10 and derivatives by Equations 3.11 and 3.12 respectively.

\[
V(0) = V_0 \quad (3.9)
\]
\[
V(t) = V_0 + V_1 t + V_2 t^2 + V_3 t^3 \quad (3.10)
\]
\[
V'(0) = V_1 \quad (3.11)
\]
\[
V'(t) = V_1 + 2V_2 t + 3V_3 t^2 \quad (3.12)
\]

the coefficients $V_n$ must be solved for in terms of the values at $V(0)$, $V(t)$, $V'(0)$, and $V'(t)$. To do this, the derivatives are multiplied by $t$, and the problem becomes a simple linear equation.

\[
V_0 = V(0) \quad (3.13)
\]
\[
V_1 = V'(0) \quad (3.14)
\]
\[
V_2 = 3V(t) - 3V(0) - (V'(t) + 2V')t \quad (3.15)
\]
\[
V_3 = 2(V(0) - V(t)) + (V'(t) + V')t \quad (3.16)
\]

3.5 The Interfaces

The final goal of the system is to be able to not only handle simple geometric black and white figures like those used in many of the test images, but to support arbitrary images as input. Through OpenCV, the system supports
loading common image formats including the Portable Network Graphics (PNG) and Joint Photographic Experts Group (JPG) formats. These formats are both typically three channel color (red, green, blue) images, and therefore are converted internally using the JPG definition for image luminance (the Y channel in the YCbCr color-space). This effectively forces the system to treat all images as if they were purely grayscale. With the exception of this preprocessing, the outputs of each stage is accumulated in memory. This allows later stages to reuse information from all earlier stages to improve structural estimations.

The edge detector stage performs operations on floating point grayscale data. The initial conversion from 8-bit per channel, three channel rgb produces quantized gray levels within the new floating point data. The edge detector is sensitive enough that the resulting step edges will be located. The detector progressively blurs and searches the space, avoiding excess memory usage. The output of this stage is a link-list of poly-lines with a structural tag connecting each edge to vertex object. Each vertex contains a list of pointers back to all connected edges. The vertex objects are stored in a second link-list.

The vectorization stage creates a collection of polynomial fits and junctions using only the detected edge data. The fits are a small structure with time, $x$ and $y$ coefficients, as well as pointers to the parent edge including start and end indices for related data points, and connections to adjacent junctions. Junctions connect back to parent vertices, but instead of lists of pointers to their corresponding fits, they connect via a fixed array with a count of entries. The stage generates junctions which do not correspond to any vertices at locations where changes in curvature of the edges being fitted strain the model used by the robust fitter to vectorize beyond their ideal fit.
The final completion stage uses polynomial fits derived by the vectorization stage, along with their links to associated junctions. The system extracts end-point vector locations and derivatives from the polynomial fits directly, and then proceeds to match only those which are not associated with the same junction. This avoids trivial fits at locations which are known to connect. The stage generates a second collection of illusory fits which are identical structurally to the polynomial fits which represent the vectorized edge map.
Chapter 4

Test Cases

A system cannot stand on its design choices alone; it needs to be validated. This chapter presents several banks of test images. The sets are designed to highlight the functionality of each of the different stages in the code. In addition to raw images, one set of tests allows the edge detector stage to be bypassed completely, to demonstrate the operation of the later stages without its influence. Each of the following sections presents one test bank, describes its purpose, and discusses the results of the test.

4.1 Edge Detector

The edge detector is an implementation of the design described by Lindeberg [19], modified to support the hybrid scale-space pyramids discussed in his later work [20]. The sampling frequency considerations mentioned by Eaton et al. are taken into account, and the majority of the tests are designed so the image is blurred to a standard deviation of two pixels at the current coarseness before sub-sampling to avoid data loss.

The detector has been written with many optional arguments to demonstrate
the effects that each has on the execution.

4.1.1 The Gaussian Step Test

The aspect of the edge detector of greatest interest is its response to edges at a wide variety of scales. To facilitate this in a controlled fashion, the images in Figure 4.1 were designed and generated in MATLAB. Each test image consists of a Gaussian step edge passing from the left to the right side of the image. The standard deviation of the edge varies between 1 px and 128 px from the top to the bottom of the image respectively. The code used to generate them has been provided in Appendix B.

![Figure 4.1: The Gaussian test images. Each image varies in standard deviation of blur from 1 px to 128 px as the image is traversed from top to bottom respectively.](image)

The ideal output of the system consists of a single vertical bar passing from the top center screen to the bottom uninterrupted. In addition to this, *wings* to the sides of the bar are expected to arise from the quantization of the image. Since a typical image only supports 256 gray levels, the system will detect fluctuations near the border between levels as either side of a \( \frac{1}{256} \) of
the dynamic range of the image. The source image is converted to floating point immediately at the start of execution. It is then manipulated in floating point form to avoid introducing additional quantization side effects.

This test reveals a great deal about the nature of the edge detector’s dependence on scale. Different patterns in breaks in the line point to different sources for errors. Irregularly spaced point breaks may occur if an edge passes through a grid point. This is more likely to occur at finer scales, where quantization still plays a significant role. Regularly spaced breaks on the linearly changing Gaussian edge (Figure (c)) point to an implicit problem arising during typical blurring. This is further supported if they occur with sub-sampling disabled. Incremental breaks which cover wider ranges at larger scales are indicative of skips caused by the sub-sampling itself. A large gap at the end of the line shows the limit of the system’s ability to detect coarse edges.

4.1.2 Subsystem Effects

There are three main portions of the edge detector: the edge tracer, the sub-sampler, and the re-up-sampler. The system is designed to be run with or without sub-sampling. If sub-sampling is enabled, the re-up-sampler may be disabled. The effects of each of these have been included below. It takes as much processing power and time to analyze an image with $n$ blurs per sub-sample as it does to analyze an image with $2n$ blurs without sub-sampling.

The differences between the images are clear. Processing the image 16 times with a blur variance of $\frac{1}{2}$ should theoretically be able to find edges with a standard deviation of up to $2\sqrt{2}$ px. Since Figure (a) demonstrates a linear
change in standard deviation from 1 px to 128 px across 256 steps, only the first three pixels of the line should be detectable at all. The hybrid pyramid is bounded by the dimensions of the image itself, rather than the number of blurs. This corresponds to the presence of significant patches of scale-space with viable responses. The improvement of the edge detector with the addition of the re-up-sampling step shows a similar jump in quality. There are still gaps in the detected scales, which require further attention. It is possible that modification of the re-up-sampling and the down-sampling code could close these gaps.
4.1.3 Adjusted Variance, Fixed Blurs per Sub-Sample

This test set focuses on the effect adjusting the variance of the blur kernel has on the detected scales in scale-space. The linearly blurring Gaussian step edge is processed using blur kernels with variances of $\frac{1}{2} px^2$, $\frac{1}{3} px^2$, and $\frac{1}{4} px^2$. The results are in Figure 4.3. As the kernel’s variance decreases, finer scales in scale-space are detected. For both the $\frac{1}{3} px^2$, and $\frac{1}{4} px^2$ kernels, a gap in the finer scales has landed on the scales in which the quantization wings are detected. For this reason, the wings are mostly ignored.

Figure 4.3: Each image in this set was blurred 8 times per sub-sample with re-up-sampling enabled.
### 4.1.4 Fixed Variance, Adjusted Blur Count Per Sub-Sample

For a fixed variance kernel, the number of blurs per sub-sample will drastically change the output of the detector. Eaton et al.
[7] states that an ideal hybrid scale-space pyramid will blur each tier to a standard deviation of 2. At this point, Nyquist’s theory states that the aliasing due to sub-sampling should be negligible.

![Figure 4.4: Each image in this set was blurred with a $3 \times 3$ Gaussian filter with variance of $\frac{1}{2} px^2$, and sub-sampled with re-up-sampling enabled.](image)

In Figure 4.4, the cleanest edges are those produced by using a blur filter with variance $\frac{1}{2}$ 8 times. Since variances are additive across convolutions, this adds up to a variance of 4, which matches the target standard deviation of 2. The effects of blurring close to this amount are examined in Figure 4.5. The differences between blurring 7 or 8 times appear to be almost negligible, while over blurring shows a marked drop in detected scales.
Figure 4.5: This looks at the variance around the ideal blur per sub-sample described by Eaton et al.[7]. The base image is the quadratically increasing Gaussian step to highlight smaller scales.

4.1.5 Matched Variance and Blurs per Sub-Sample

To determine if there are benefits to be had using more iterations with a finer blur kernel, the system was tested with both the blur and the number of blurs varying in unison. The gaps in scale-space appear to jitter, but do not show a clear trend in any direction.

4.1.6 Rotational Versus Separable Kernels

In Lindeberg’s work on hybrid pyramids [20], he presents two blur kernels. The first are separable, allowing for accelerated processing. The second are
Figure 4.6: This test bank blurs each image to a standard deviation of $2\sigma$ before sub-sampling with a blur kernel of the specified variance. rotationally symmetric. These are compared in Figure 4.7. It is noted that when the blur is $\frac{1}{3}$ and the rotational kernel is ideal, both kernels are equal. Using the rotationally symmetric kernel appears to remove the distinguishing characteristics of the output from the system using various blurs with the recommended number of blurs per sub-sample.

4.1.7 Differential Kernel Selection

For the sake of testing alternatives, three differential kernels were implemented in the system. These are the central differences operator, the Sobel
Figure 4.7: A comparison of the edge detector output using rotationally symmetric versus separable blur kernels. Each image is blurred to a standard deviation of 2 px before subsampling.

either Sobel, nor Scharr detected the primary edge at all. It is for this reason that the central differences operator is recommended for edge detection in scale-space.
4.2 Polynomial Edge Fitting

The vectorization stage takes poly-line inputs and converts them to polynomial curve segments. The fitter has four adjustable parameters: the extension threshold, the garbage threshold, the weight threshold, and the M-estimator used. The system automatically adjusts its fit order as the strength of fit edges increases enough to support a curve. The following tests look at the system’s performance as a whole. In addition, the effect of each of the four parameters on the system’s performance is briefly analyzed.

To test the system without side-effects caused by noise or quantization in the edge detector, a bypass was introduced into the system to allow manual
injection of edges defined in a similar fashion. The output of the fake edge generator has consists of sample points at a single scale of 1px, uniformly spaced along the curve. The shapes have been carefully constructed to avoid zero order discontinuities that would provide hints to the fitter of the location of junctions. Final tests are performed directly on the output of the edge detector stage to demonstrate its ability to operate on real data.

4.2.1 Approximating Simple Shapes

The simplest shapes to test edge fitting on are polygons and circles. Polygons provide a controllable means of determining the finest first order junctions extractable from an image. The robust fitting technique is theoretically able to recognize junctions on a polygon with an arbitrary number of sides, provided the sides have enough length to support the split hypothesis. To test the limits of the robust fitting mechanism, five images of simple convex shapes are presented (Figure 4.8).

The original image sizes are 500 × 500 pixels. The distance from the center to the furthest points on the perimeter is 188 pixels. Each of the edges of the polygons have 100 evenly distributed sample points. For polygons, the edges are generated as a continuous detected edge, with a single break at the top center. The circle is the exception to this, with its break at the right. For this test, the edge data was fitted with up to bi-cubic polynomial fits. The threshold values for the system were 1, 8, and $\frac{1}{2}$ for the extension, garbage, and weight thresholds respectively. For clarity, the output of the completion stage is shown.

As is demonstrated by the output in Figure 4.8, the system performs ideally.
For all polygon test shapes, the system correctly identified the locations of every junction. The circle is represented by a smooth chain of five bi-cubic arc segments. The most striking of these examples is the dodecagon.

![Image](image_url)

Figure 4.9: Five simple convex shapes. From left to right: A square, pentagon, octagon, dodecagon and circle. From top to bottom: The generated edge with a single vertex (at top for all except the circle, which is at right), polynomial fits, and junction closures.

### 4.2.2 Junction Extraction In Polygons

It is clear that the system is able to identify even glancing angles if the surrounding edge structure is sufficient to support them. The obvious question is, given the constraints used in the last test, what is the most complex polygon which produces a full junction match. To test this, the order of the polygons was doubled. At a resolution of 500 px square, using cubic polynomial fits, the system is able to accurately detect the presence of all
junctions on a 24 sided polygon. A few junctions are lost on the next larger, and the surface is treated like a circle for 26 sided polygons.

Figure 4.10: A demonstration of the limit of junction detectability in the estimator. The polygons from left to right are 24, 25, and 26 sided. All junctions are found on the 24 sided figure. The top right corner is lost on the 25 sided figure. Only seven junctions are detected in the 26 sided figure, one of which is caused by a vertex on the edge.
4.2.3 Fitting To Extracted Edges

While the success of the robust fitter on generated edge points is significant, it is meaningless if the fitter will not process extracted edges to a close degree of precision. For this test bank, the same shapes used in Figure 4.8 were built as black objects on a white background. The images were then passed through the edge detector with a kernel variance of $\frac{1}{4}$, and 4 blurs per sub-sample. The resulting images are presented in Figure 4.11.

Figure 4.11: Four simple convex shapes. From left to right: A square, pentagon, octagon and dodecagon. From top to bottom: The original image, the extracted edge, the polynomial fits, and resulting junction closures.

At first glance, it seems as if the robust fitting process performed markedly
worse on raw image data; on closer examination, the missed corners are missing in the edge detector output as well. The robust fitter performs well in the presence of these gaps, fitting the short segments as well as possible, and detecting junctions with at least half an edge length of support on either side. This is demonstrated by the dodecagon, whose solid junctions have all been detected correctly.

4.2.4 The Thresholds

Three varieties of thresholds are used by the robust fitter to tune different parameters of the system. The extension (X) threshold determines how many times a baseline weight a fit must accumulate before attempting a higher order fit. The weight (W) threshold specifies the minimum weight a point must have to be considered a part of a fit. The garbage (G) threshold sets the minimum length in standard deviations a line is allowed to be if a fit is to be generated using it. To analyze the effects of modifying the thresholds on the vectorized output, the trefoil in Figure 4.12 was selected. This image is computer generated, and so has a very high signal to noise ratio. It also an example of a shape that is its own background.

(a) Source image  
(b) Edge detector output

Figure 4.12: A trefoil: a three dimensional knot. Rendered in Corel Bryce 5.
The X threshold is the most interesting of the three. It allows the system to automatically increase the current fit if enough new data has been added to be worth checking. The value for the threshold is difficult to choose appropriately. The value typically used in this implementation is 1, requiring the data double in length before attempting to increase the fit.

![Figure 4.13: The result of varying the X-threshold on fits.](image)

Figure 4.13 demonstrates a range of extension threshold values with garbage and weight thresholds held at their baseline values. Table 4.1 lists the number of edges found for each of these respectively. For this image, the number reaches a local minimum between values of 1 and 4. For values above 8, the location of junctions in close proximity to one another were detected, rather than smoothed over. For values considerably higher than 8, the system reduces to a robust line fitter.

<table>
<thead>
<tr>
<th>X Threshold</th>
<th>0.125</th>
<th>0.25</th>
<th>0.5</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segments</td>
<td>75</td>
<td>75</td>
<td>73</td>
<td>72</td>
<td>71</td>
<td>73</td>
<td>77</td>
<td>86</td>
<td>93</td>
</tr>
</tbody>
</table>

The W Threshold determines the minimum strength of a fit at a point needed for that point to be considered part of the fit. It is bounded by the range \( \{0, 1\} \). If the W threshold is too high, the system will not generate fits. If the threshold is too low, any point near the line will be included in the fit. Figure 4.14 demonstrates the effects of modifying the W threshold.
The number of fits produced by the system is stable and close to 70 for these test images while the W-threshold is between 12.5% and 65%. Thresholds greater than 65% show a marked increase in the number of generated fits. As the purpose of the W-threshold is to prevent connected edges from independent sources from sharing data, values below 65% are recommended.

The G-threshold is the most dangerous of the thresholds. It distinguishes between edges which have too little support to be used for fits. The threshold provides a significant speedup, preventing particulate edge fragments from being manipulated by the robust fitter, while simultaneously reducing the number of end-point junctions to be fit by the later completion stage. The problem exists in determining which edges should be kept and which thrown away. The threshold will also throw away dotted lines generated by an edge weaving in and out of the scales the edge detector is responsive in.

Figure 4.15 illustrates the damage that the G-threshold does to the edge map. Best results tend to fall in the range of 2 to 8 pixels, where 2 pixels will
preserve more image data. Values less than two result in the image flooding with noise. Ideally, this threshold will be replaced in later implementations with a system for fusing trivial edges into significant data sources.

4.2.5 The M-Estimator

The careful choice of an M-Estimator determines the speed and accuracy that a fit will produce. Five different M-Estimators have been implemented for use by the robust fitter in the vectorization stage: Least Squares, Fair, Geman-McClure, Cauchy, and Tukey. Least Squares effectively finds the average of the system. Tukey is unique among the group since it is a re-descending estimator.

The test edge used to compare the relative effect each of the M-Estimators has on the system is a closed semicircular segment. The poly-line starts and ends at the center of the circular segment. This shape tests the system in two ways: rate of convergence, and accuracy. Because the circular segment is non-polynomial, it must be approximated by several polynomial segments. The fewer the iterations needed to stabilize, the better. Additionally, the corners where the circular arc meets the diameter force the system to locate actual junctions. These corners are right angles; an estimator unable to properly detect them cannot be applied to this task.

![Figure 4.16: The test edge for comparing various M-Estimators.](image)

Figure 4.16: The test edge for comparing various M-Estimators.
The results of running these M-Estimators against Figure 4.16 are shown in Figure 4.17. The number of iterations to convergence is shown in Table 4.2. The most glaring result is the performance of the Least Squares technique. The weakness of the Least Squares algorithm is its low breakdown point zero. This means that the addition of a single gross error to a well formed fit is enough to render the fit irrelevant. This highlights the importance of using robust estimators to solve the discontinuity detection problem.

Table 4.2: Steps to convergence of several M-Estimators

<table>
<thead>
<tr>
<th></th>
<th>Least Squares</th>
<th>Fair</th>
<th>Geman-McClure</th>
<th>Cauchy</th>
<th>Tukey</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segments</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Average Steps</td>
<td>1</td>
<td>13.57</td>
<td>12.25</td>
<td>13.43</td>
<td>11</td>
</tr>
<tr>
<td>Total Steps</td>
<td>1</td>
<td>95</td>
<td>98</td>
<td>94</td>
<td>77</td>
</tr>
</tbody>
</table>
The remaining four algorithms each split the circular arc into regularly spaced intervals. Fair, Geman-McClure, and Tukey break the curve at approximately the same places, with Geman-McClure focusing the edges to a slightly greater degree. In contrast, Cauchy produces considerably less accurate fits, requiring approximately the same number of iterations to cover the curve as either Fair or Geman-McClure. A sharp improvement over the other algorithms is demonstrated by Tukey in this area. To converge on the final curve requires almost a fifth fewer iterations than any of the other three.

The Least Squares algorithm has been left out of the convergence discussion because it is unable to detect when its fit strength is weakening. The estimator is guaranteed to converge to a value for any set of points; adding points to a Least Squares system incrementally will never prevent corruption of the fit.

### 4.3 Curve Completion

The completion stage is designed to detect edges in a scene and attempt to close them. To do this, it performs a matching search of the end-point vector space for ideal matches. To allow execution in a reasonable amount of time, the technique is accelerated through the use of spatial indexing, which allows pruning irrelevant junctions from the search space. The algorithm has three steps: a local fit, the tree generation, and the global fit.

Local fits use a threshold to determine whether completion processes should be launched. The theory is a junction on a non-polynomial, smooth surface, like the perimeter of a circle, will not launch completion processes in the same fashion as an L or T-junction. The remaining edges are matched using
the arc-length metric from equation 4.1. The following test images demonstrate the operation of the curve completer in different scenarios. The section on fusing surfaces discusses possible modifications to the completion metric.

4.3.1 Polygonal Curve Closure

Before testing the system on objects which actually form completions, the system was tested on convex polynomial curves. This illustrates two aspects of the detector. The closures on a detector never consider a corner to be perceptually more significant than a completion. Second, the completions for symmetric, convex fits are accurate and logical. This means that the expected output from each of the polynomial fits is a rosette of loops with linear insides and curved outsides. The fewer the angles, the more extreme the curvature of the fit. The test results for these objects are presented as the bottom row of Figure 4.8.

4.3.2 Edge Closure Limits

The next test looks at when a closure is produced, and when it is not. To test this, a figure consisting of two squares is generated. Each of the squares consist of a single poly-line starting at the top and wrapping around each of the four corners. Fits made to the system are cubic, and completions are predicted. The arc-length of the rosette completions is equal to \( \frac{\pi}{2} \) (approximately 1.57) times the length of a side. Theoretically, the edges of
the squares should complete across the gap if the gap is less than this metric. Figure 4.18 shows completions formed between squares at varying distances. As expected, the completions follow these limits.

![Figure 4.18: The result of varying the W-threshold on fits. The distance between each pair is measured in the length of the side of a square.](image)

### 4.3.3 Fusors

When attempting to fuse illusory contours across a figure like Kanizsa’s triangle, the system does not behave as expected. Rather, it shows preference for arcs with inflection points (change in curvature sign). Figure (a) shows the algorithm fitting against a generated Kanizsa’s square. The circular components do not complete as expected. By incrementally increasing the weight against sinusoidal fits, the system is able to be coerced into a full completion. The discrepancy demonstrates a need for additional research into curve completions.

In the figures below, different values for $\psi$ are used to adjust the weighting of non-circular curves. The value of $\psi$ is always chosen so when $\vec{A}$ and $\vec{B}$ are co-circular, $\psi = 0$. The intermediate stages between $\psi = 0$ and $\psi = 4(\sin \theta_A + \sin \theta_B)^2$ show that modifying the fit function in this fashion does not improve the quality of the fit, but rather, that a second metric beyond arc-length of the smoothest curve is needed to identify completions. The
'S’ shaped completions in the mean-curve fit could be trivially ruled out with tests for intersections with known borders.

\[ l_{adjcurve} = \left\| \overrightarrow{AB} \right\|^2 \frac{\arccos \left( \frac{\overrightarrow{A} \cdot \overrightarrow{AB}}{\left\| \overrightarrow{A} \right\| \left\| \overrightarrow{AB} \right\|} \right) + \arccos \left( \frac{\overrightarrow{B} \cdot \overrightarrow{AB}}{\left\| \overrightarrow{B} \right\| \left\| \overrightarrow{AB} \right\|} \right) + \psi}{\left\| \overrightarrow{A} \times \overrightarrow{AB} \right\| + \left\| \overrightarrow{AB} \times \overrightarrow{B} \right\|} \]  \quad (4.1)

Figure 4.19: The result of varying the W-threshold on fits.

4.3.4 Splitters

The final test to run the system against is on figures which are expected to split. Two figures have been designed to test this, both to be processed by
the edge detector stage before fitting and closing. The first is a two-fish image similar to the one described by Geiger et al.[9]. The second is the silhouette of two overlaid circles. The splitting figures are supposed to split into distinct objects. In both cases, the figures do not completely separate. The two-fish image does form a completion across one of the fish tails. The error introduced by the edge detector once again prevents a complete separation of the two images.

4.4 Complex Test Images

With the sensitivity of the edge detector stage to each of its configurable parameters, it is difficult to evaluate the performance of this system on complex natural test images. In spite of this, running the system on more complicated image sets exposes some challenges that will be present regardless of the implemented edge detector stage. The purpose of the final system is to provide support for high-level algorithms operating on natural images for use in a wide variety of fields including navigation, motion capture, and 3D object extraction. To provide a baseline, the results of running the system

![Two-Fish Image](image1)

![Two-Balls Image](image2)

Figure 4.20: A demonstration of the system processing splitting figures.
against test images for each of these is supplied here.

![Image](a) Room(1) ![Image](b) Room(7) ![Image](c) Auto

Figure 4.21: A demonstration of the system processing complex figures.

To avoid the scale-space gaps produced by sub-sampling, each of the examples in this section was processed with sub-sampling disabled. The edge detector stage processed the image with a blur kernel variance of $\frac{1}{2}$, an assumed initial blur of 3, and 16 blur levels for the volume. The maximum robust fit order was 2 using a standard Tukey M-Estimator. The X, G, and W thresholds for the estimator were 1.98, 4, and 0.5 respectively. The vector stitching algorithm used the mean curvature approximation described in Section 2.5. Images tested against were provided by the Signal Analysis and Machine Perception Laboratory at Ohio State [3].

The test demonstrates several key points which must be considered. The presence of noise in the output of the edge detector stage can easily disrupt the completion stage in its current shape. This problem is exacerbated by the fact that gaps exist within the scale-space volume, resulting in edges which are not traced completely. This is particularly obvious in Figure (a),
where the seat of the chair prefers to complete with the noise over itself. Edge detectors and feature detectors will often produce positive results in unexpected places. A metric which is able to project through noisy data to detected edges further into the image is needed to overcome these false positives.
4.5 Comparison to Prior Work

There is one work that approaches this problem in a similar fashion to the method presented here. The work by Geiger et al.[9] uses known junction locations in rasterized images in combination with an energy function to perform occlusion completions. Multiple hypothesis are used to determine the type of completion that is present at the located junction. Object surfaces detected are represented as a rasterized gray-scale image map where white demonstrates the presence of the surface and black, the absence.

The algorithm makes no attempt to detect the edges themselves, nor does it operate on the output of an algorithm designed to detect these specific types of edges. The use of known junction locations means that the algorithm is effectively dealing with purely black and white imagery. While this initially does not appear useful in the real world, the algorithm may quickly be enhanced to incorporate gray-scale and color imagery through the use of a junction detector like the one presented by Parida, Geiger, and Hummel [24] around the same time as the work by Geiger et al.[9]. As the authors were aware of both results, it makes sense to consider their integration.

The vectorization approach used in this thesis enhances the quality of the end-point derivatives located at each of the junctions in a given image. The additional support provided by the detected edge, and the ability of the robust estimator to split edges at junction locations means that more regional information may be used to calculate end-point vectors accurately. While this is an improvement over the end-point vectors in the junctions provided by Parida et al.[24], the accuracy of the edge detector stage, in finding the
location of the junctions themselves, is currently not close to that demonstrated by Parida et al.[24]. This hinders the ability of later stages to perform as well as the technique reported in Geiger et al.[9] on similar images.

Since Geiger et al.[9] bypass the process of detecting junctions to demonstrate the operation of their portion of the algorithm, it is reasonable to use the output of the edge bypass stage to compare functional differences between the algorithms. The algorithm presented in this thesis currently makes no attempt to sort through salient surfaces. It searches the image edge map for junction locations and uses those to find likely edge completions. Both our approach and Geiger’s et al.[9] obtain accurate location of junctions if the vectorization given accurate edge information.

Looking only at the ability of the system to locate viable edges, the work by Geiger et al.[9] exceeds the performance of our work. Geiger’s approach is able to complete junctions in a logical fashion which avoids edges penetrating known surfaces implicitly. The pure edge based approach presented here raises a few problems. Based exclusively on the arc-length of a clothoid as suggested by Kimia et al.[15] (see Section 2.5), it is noted that the circles in Kanizsa’s square cannot complete without additional information about surface structure. It is also impossible for the junctions on the two fish shape to form a completion of the tail of one fish across the body of the other if the width of the body is more than π times the width of the tail. In this case, the tail’s end-points form a nearly perfect circular completion with themselves. If a two fish image visually completes to a person in this instance, it invalidates the use of a clothoid length and mandates the use of an alternative metric.

The final difference between the two approaches is related to T-junctions as
described by Rubin [28]. Provided that the system presented here detects
the presence of 3+ way junctions properly, it will attempt to perform com-
pletions on them with other junctions in their vicinity. The work by Geiger
*et al.*[9] makes no provision for greater than 2-way junctions at the input of
the system, nor does it deal with higher order junctions. Currently, in our
approach, there is nothing to prevent Y junctions from forming completions
of their own. It is not clear whether this is a significant drawback.
Chapter 5

Conclusions

Bringing together research results from several fields, this work provides the foundation for the development of practical computer vision systems that handle the object completion problem caused by occlusions. Solving this problem has the potential to resolve a large number of problems, currently caused by occlusion, that limit the usefulness of computer vision systems in practical application. The work presented here can be improved in a number of ways including the implementation of specific stages and the integration between them.

The implemented system, like most high level algorithms, worked only as well as its weakest link. In this instance, it was the edge detection process. The benefits of scale-space are dependent on a large number of factors. If the scale-space pyramid is not scale invariant it may not be worth the overhead. The cost of utilizing a scale-space pyramid without sub-sampling is too great to be a viable source of real-time data. The differences between images with and without the up-sampling process were significant; the gaps filled by this process were smaller than those that remained after it. The cost of approximating the full set of scales in a pyramid without sub-sampling is too great, but these gaps in hybrid scale-space pyramid responses undermine
the value they provide to the system.

The techniques presented by Lindeberg [19] provide an unusual approach to searching for edges in the scale-space volume. The algorithm was chosen specifically because it generated poly-line output in an intuitive fashion. While the location of the strongest edges is logical, the tendency for the algorithm to split edges pre-maturely is damaging to the later processes. The edge detection stage must be extended for junction detection to fuse edge pieces.

The spatial indexing technique applied to the contour completion process might provide a mechanism for guaranteeing closure of edges which are trivially close to one another (e.g., less than one standard deviation at the detected blur level). Edges which terminate at a 1-way junction do not provide the structural cues present in L and T-junctions. The techniques presented by Rubin, Geiger, and Pao ([28], [9], [23]) are reliant on the feature dense nature of junctions. Closing the edges in the image in a meaningful fashion is vital to the viability of the object stitching algorithm. Once spatially related edges are joined, at least across junctions, modifications to later processing stages may be able to retroactively generate fits.

The robust estimation technique shows potential in this area. Its current implementation provides a powerful way of analyzing images using low-level features. The simplicity of the technique and the quality of the results is well worth the overhead of the iterative approach. Fitting polynomial models to raw edge data by locking down one end-point successfully detects the locations of junctions in 12 sided polygons taken from raw image data. Testing the algorithm on computer generated edges, the system was able to detect junctions on 25 sided polygons almost perfectly. There are, however,
three distinct weaknesses in the current implementation: multi-source fits, under-supported fits, and traversal order dependence.

The robust edge detector is able to take a single input curve and split it into multiple fits. It is unable to fuse multiple input curves into a single fit. To resolve this problem, the fitter must be modified to support multiple data sources. An implementation which supports could be applied across junctions to determine if edges smoothly complete. This technique overcomes problems caused by sharp curvatures, and may be applicable to detecting and analyzing second order junctions. It also rectifies the presence of many under-supported edges in the output of the edge detector, as related edge pieces might be fused into a single edge despite the presence of junctions caused by the edge detection process. This could also smooth detected loops.

Under-supported edges result in poor predictions of end-point tangent vectors. This is damaging to the completion stage. This implementation attempts to solve the problem with the G thresholds, which prunes edges which are too short. The G threshold is very unstable, but seems to be a necessary evil to overcome noise caused by image compression or quantization. A revision of this concept would ideally be a non-destructive solution. The shorter an edge is, the more room for error there is in an end-point fit. The robust estimators provide an upper-bound to the variance, while the X threshold prevents premature second order fits. This means that it should be possible to develop an error metric which eases the curvature restrictions of the vectorization stage to resolve the lost completions.

The final major weakness of the robust estimator is its fixed traversal order. This produces fits which are heavily biased to the start of an edge. The last
portion of a non-polynomial arc to be fitted is guaranteed to have the small-
est region of support of the fitted segments. This also means that the tangent
vector at this end-point has the greatest error. A robust fitting technique
which traverses back and forth may find junction locations more accurately
than the current implementation, as well as improve the support for the last
fit section of non-polynomial arc sections.

In Collins’ work on blob tracking [6], it is noted that instability in automated
systems arises when methods for increasing the scale or quality of a fit are
not complimented by means of reducing it again when appropriate. The
X Threshold described in this system may suffer from this instability. A
complimentary technique for reducing the order of fits back to first order is
necessary. This could resolve oscillations in data due to sub-pixel noise, and
should produce cleaner fits for long edge segments.

The algorithm used to complete curves requires the most further research.
Kimia et al.[15] state that the arc-length of a clothoid provides a measure
which may be used to determine completion properties. It is not clear that
the perceptual relationship between edge length and end-point fusion is
valid without a secondary metric to determine which fits are actually bet-
ter. The arc-length approximation provided by this paper favors completion
curves which intersect the chord between the end-points. This causes the ex-
pected circular completions on Kanizsa’s square to fail without additional
biasing to overcome the preference.

Using a bi-cubic model for edge completions provides the flexibility needed
to connect an arbitrary pair of end-point vectors in a scene. Bi-cubic splines
may be uniquely solved for provided end-point locations and derivatives. If
an estimate of the arc-length is used for the range of independent variable,
the end-point vectors do not need to be normalized prior to generating a fit. Since the Euclidean distance between sample points is used to generate the fits, the relationship between the length of independent fits is still valid.

There is a simple perceptual test which could determine whether the metric of minimizing the change in curvature over arc-length is the metric which determines edge completions. If a Kanizsa’s square is constructed where the gap between adjacent pacmen is more than \(\frac{3\pi}{2}\) times the radius of the pacman, it should not complete as a square. In this case, the arc-length of the circular completion of the pacmen with themselves is shorter than the distance between two adjacent pacmen. If the completion is dependent on the change in curvature, both completions have none; they should be distinguishable by arc-length alone.

If this is true, the complexity of using clothoids to generate fits need not be braved. The completion system should be able to determine the likelihood of a completion being viable based entirely on the relative position and derivative of end-point vectors. The estimate of that measure created in this work produces visually pleasing fits. It does not, however, always generate fits that are perceptually ideal.

The computational expense of finding the effective arc-length for every vector pair in the final image can be mitigated successfully through the application of spatial indexing structures. If the metric determining the plausibility of a fit is based on the arc-length of the fit, no point outside a circle with radius equal to the arc-length of the current best fit need to be tested.

These are the same principles applied to the acceleration of ray-tracing and rasterizing algorithms in computer graphics. The implementation of the spatial indexing system in this work was a BSP tree. This approach only
considered the spatial ordering of end-point vectors. More complex spatial indexing structures which incorporate the relative angles of the end-point vectors may provide an additional speedup.

The computer vision industry is very close to being able to solve the problems studied in this work. Much of the theory needed to solve the object completion problem is close to where it needs to be to make this technique a reality. This implementation, while unable to accurately predict occluded shapes to the degree achieved by Geiger, Pao and Rubin [9], brings an alternate approach to solving the completion problem to the table. This approach to solving for object completions presents a lot of promise.
Bibliography


[7] Ross S. Eaton, Mark R. Stevens, Jonah C. McBride, Greydon T. Foil,


[40] Christopher R. Wren and Alex P. Pentl. To appear: Journal of image and vision computing special issue on face and gesture recognition dynaman: A recursive model of human motion.


Appendix A

The Chord to Arc-length Relation

The arc-length $l$ of a circle can be found as a function of the chord length $\|D\|$ and the cosine of the angle between the tangent vector and the chord $\phi$. To demonstrate this, let $\hat{A}$ and $\hat{B}$ be two tangent vectors on the perimeter of a circle with a center at $\hat{C}$. The tangent vectors are located at points $\hat{A}$ and $\hat{B}$ respectively. Let $\kappa$ equal the curvature of the circle, which is equal to the inverse of $r$, its radius. If $\kappa$ equals zero, the circle degenerates to a line. In this case, the arc-length is equal to the Euclidean distance between the points, which is trivially in terms of the chord length $\|D\|$. If $\kappa$ is non-zero, $l$ is the product of the central angle which subtends the chord, and the radius (Equation A.4).

\[
D = \hat{B} - \hat{A} \quad (A.1)
\]

\[
\phi = \vec{A} \cdot \vec{D} = \vec{B} \cdot -\vec{D} \quad (A.2)
\]

\[
= \frac{\theta}{2} \quad (A.3)
\]

\[
l = \begin{cases} 
\|D\| & \kappa = 0 \\
\frac{\theta}{|\kappa|} & \text{otherwise}
\end{cases} \quad (A.4)
\]
Figure A.1: A circle with tangent vectors $\vec{A}$ and $\vec{B}$, with angles $\phi$ and $\theta$ defined.

Because $\hat{A}$ and $\hat{B}$ are on the perimeter of a circle when $\kappa$ is non-zero, $\hat{A}$, $\hat{B}$, and $\hat{C}$ form an isosceles triangle (Equation A.5).

$$r = \|\vec{CA}\| = \|\vec{CB}\| = \frac{1}{\kappa} \tag{A.5}$$

To begin, solve for the radius with respect to the chord length through the law of cosines $c^2 = a^2 + b^2 - 2ab \cos(\theta)$. Since the angle between the tangent vector and the chord is equal to half the angle subtending the chord, simple trigonometric equalities may be used to further reduce the system to a function of $\phi$ and $r$.

$$\|\vec{D}\| = \sqrt{r^2 + r^2 - 2rr \cos(\theta)} = 2r \sqrt{\frac{1}{2}(1 - \cos(\theta))}$$

$$= 2r \sin\left(\frac{\theta}{2}\right) = 2r \sin(\phi) \tag{A.6}$$

$$r = \frac{1}{\kappa} = \frac{\|\vec{D}\|}{2 \sin(\phi)} \tag{A.7}$$
Finally, the equations relating the radius to the chord length, and $\phi$ to $\theta$ are plugged into Equation A.4. This results in Equation A.8. This can trivially be massaged into a function of the tangent vectors and the chord themselves, which is presented in Equation A.9. Since the limit of $\text{sinc}(\theta)$ as $\theta$ approaches 0 is 1, the case where $\kappa = 0$ is the limit condition of Equation A.9 (Equation A.10).

\[
l_{\kappa \neq 0} = \frac{\theta}{\kappa}
= 2\phi \left| \frac{\|D\|}{2 \sin(\phi)} \right|
= \left| \frac{\|D\|}{\sin(\phi)} \right|
= \left| \frac{\|D\|}{\text{sinc} \, \phi} \right|
\]

\[
l = \begin{cases} 
\|D\| & \kappa = 0 \\
\|D\| \arccos \left( \frac{\vec{A} \cdot \vec{D}}{|\vec{A} \times \vec{D}|} \right) & \text{Otherwise}
\end{cases}
\]

\[
\|D\| = \lim_{\phi \to 0} \frac{\|D\|}{\text{sinc} \, \phi}
\]
Appendix B

Gaussian Steps

To test the stability of a scale-space edge detector across scales, this series of test images was produced. Each of the images consists of a Gaussian step edge with standard deviation increasing along the length of the edge as it stretches from the top to the bottom of the image.

Figure B.1: Three Gaussian step images used to test the edge detector’s responsiveness at different scales. The images are designed to produce a vertical edge down the center of the image. An ideal edge detector will return an uninterrupted edge at this location. Two additional edges are expected at the border where the gradient stops changing.

The images presented here were generated using the following MATLAB code with a low variance of 1, a high variance of 128, and 512 steps. The output images are linearly, quadratically, and exponentially spaced.
function [ Slin, Squad, Sexp ] = gaussstepgen( lowvar, steps, highvar )
% gaussstepgen Generates images of a variable gaussian step edge.
% Generates an image of a gaussian step function increasing at
% a linear, quadratic, and exponential rate.
% This simplifies the analysis of the quality of an edge-detector at
% various scales.

% The gaussian function used to generate each point in the original image
gauss = @(X,Y) (1./(Y.*sqrt(2*pi))).*exp(-(X.^2)./(2.*(Y.^2)));

% The range, scaled properly for exponentially increasing scale
lex = log2(lowvar);
hex = log2(highvar);
stex = (hex - lex) / steps;

% The range, scaled properly for a quadratically increasing scale
lq = sqrt(lowvar);
hq = sqrt(highvar);
sq = (hq - lq) / steps;

% The range, scaled properly for a linearly increasing scale
slin = (highvar - lowvar) / steps;

% Calculate out the Y coordinates for each of these models
[X,Yex] = meshgrid(-255:1:256, 2.^lex:stex:hex);
[X,Yquad] = meshgrid(-255:1:256, (lq:sq:hq).^2);
[X,Ylin] = meshgrid(-255:1:256, lowvar:slin:highvar);

% Calculate out the gaussian for each. This is where this gets
% interesting.
% Note, we no longer need any of the Xs or Ys after this step.
Gex = gauss(X,Yex);
Gquad = gauss(X,Yquad);
Glin = gauss(X,Ylin);

% Create an edge which is wide enough to encompass the full width of the
% gaussian.
E = [ones(1,513), zeros(1,513)];

% Generate our integrated step edge through convolution.
Sexp = conv2( Gex, E, 'same' );
Squad = conv2( Gquad, E, 'same' );
Slin = conv2( Glin, E, 'same' );