

GeoStreams: An Online Geospatial Image Database

Abstract

Data products generated from Remotely-Sensed Imagery (RSI) are used in many areas, such as climatology, environmental monitoring, land use issues, and disaster management. Processing RSI data can be costly and time consuming. For the researcher, data is typically fully replicated using file-based approaches which then undergo multiple processing steps, often duplicated at many sites. For the provider, data distribution is often tied directly to the data archiving task, focusing on simple, coarse grained offerings. Many RSI instruments transmit data in a continuous or semi-continuous stream, but current techniques in processing do not utilize the streaming nature of the imagery. Recent research on continuous querying of data streams offer alternative processing approaches. In such systems data arrives in multiple, continuous, and time-varying data streams and do not take the form of persistent relations. There is potential benefit in adopting Data Stream Management System (DSMS) techniques for geospatial RSI data, but these systems typically rely on traditional relational models as basis for query processing techniques and architectures. Complex types of stream objects, such as multidimensional data sets or raster image data have not been considered.

The GeoStreams (*GeoStreams*) project investigates joining these two disciplines. The architecture allows users to formulate queries on continuous streams of RSI data. The outputs of these queries continuously return RSI data products to the user. These streams can be fed into applications to allow a continuous source of new input data from a single stream, or saved to more traditional RSI forms. As

the functionality of the RSI DSMS increases, more aspects of the applications can be formulated as part of the queries themselves.

An application focus dealing with real-time weather satellite imagery provides an important and relevant backdrop for the research. A data and query model for streaming imagery is described. New processing strategies for query processing of RSI streams are investigated. Tools for efficient processing are created and a complete system for interfacing with a satellite image stream is developed.

GeoStreams: An Online Geospatial Image Database

By

QUINN JAMES HART

B.S. (University of Arizona) 1987

M.S. (University of Arizona) 1990

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Committee in charge

2006

I'm proud of this work, but my greatest success is my family.

This is dedicated to Nikki, Connal, and Rowan.

Acknowledgments

I've been very fortunate in having tremendous support in developing this research.

First, I'd like to thank Dr. Susan Ustin, who I love and respect, for her nearly 15 years of mentoring. Susan is simply the best scholar I know and the model of what I'd like to accomplish. I also owe a great deal to my friend and adviser Dr. Michael Gertz who's enthusiasm for this project often surpassed my own. His infectious optimism has often buoyed me along when left to my own, I'd have had myself a good sulk. I hope and expect to have a career full of collaboration with both Susan and Michael.

Thanks also to Dr. Bertram Ludäscher, who despite, or maybe because, having the least exposure to my research, gave it a most thoughtful and insightful review. Dr. Nina Amenta and Dr. Vern Vanderbilt were both kind enough to plow through a disconnected research proposal and help me form it into something coherent.

I've had the pleasure of taking classes with all five of these individuals, which were high points of a great learning experience here at Davis. The University has been a tremendous place of learning and employment and I am most grateful.

I would also like to thank the National Science Foundation for their support. This work is in part supported by the NSF grant IIS-0326517 ITR: Adaptive Query Processing Architecture for Streaming Geospatial Image Data.

Most importantly, I'd like to acknowledge my family who have often had to get along with a missing father or husband when this paper was due, or that program wasn't quite working. Thanks go to Connal and Rowan, my two sons, who are so proud of me and this effort which more than anything makes me feel like I'm doing something worthwhile.

Finally thanks to Nikki, my wife. Together, we've worked equally hard on this endeavor. She's been cheerleader, manager, editor, and coach as the situations demanded. She's had to spend countless nights with the house and household while I struggled along, hunched over my computer. She's brought me good plans and good advice, coffee and comfort, more times than I can count. I wouldn't have started this without Nikki's encouragement, and I certainly could have never finished without her support.

Contents

List of Figures

List of Tables

Acronyms

CPU Central Processing Unit	ISO International Standards Organization
CQ Continuous Query	LACIE Large Area Crop Inventory Experiment
CSDGM Content Standard for Digital Geospatial Metadata	LIDAR Light Detection and Ranging
DAAC Distributed Active Archive Center	NASA National Aeronautics and Space Administration
DAG Directed Acyclic Graph	NCGIA National Center for Geographic Information and Analysis
DBMS Database Management System	NDVI Normalized Difference Vegetation Index
DCT Dynamic Cascade Tree	NOAA National Oceanic and Atmospheric Administration
DFS Depth First Search	NRC National Research Council
DSMS Data Stream Management System	NSDI National Spatial Data Infrastructure
EOS Earth Observing System	NSF National Science Foundation
ERTS Earth Resources Technology Satellite	OGC Open Geospatial Consortium
ESRI Environmental Systems Research Institute	QEG Query Execution Graph
GIS Geographic Information System	QEP Query Execution Plan
GML Geography Markup Language	QGIS Quantum GIS
GOES Geostationary Operational Environmental Satellite	QM Query Manager
GRASS Geographic Resources Analysis Support System	ROI Region of Interest
GVAR GOES VARiable	RSI Remotely-Sensed Imagery
IGFOV Instantaneous Geographic Field of View	SDBMS Spatial Database Management System

SMS1 Synchronous Meteorological
Satellite 1

STL Standard Template Library

TIROS Television and Infrared
Observation Satellite

USGS U.S. Geological Survey

UTM Universal Transverse Mercator

XML eXtensible Markup Language

WCS Web Coverage Service

WKT Well Known Text

WMS Web Map Services

WRF Weather Research and Forecasting

WWW World Wide Web

Abstract

Data products generated from Remotely-Sensed Imagery (RSI) are used in many areas, such as climatology, environmental monitoring, land use issues, and disaster management. Processing RSI data can be costly and time consuming. For the researcher, data is typically fully replicated using file-based approaches which then undergo multiple processing steps, often duplicated at many sites. For the provider, data distribution is often tied directly to the data archiving task, focusing on simple, coarse grained offerings. Many RSI instruments transmit data in a continuous or semi-continuous stream, but current techniques in processing do not utilize the streaming nature of the imagery. Recent research on continuous querying of data streams offer alternative processing approaches. In such systems data arrives in multiple, continuous, and time-varying data streams and do not take the form of persistent relations. There is potential benefit in adopting Data Stream Management System (DSMS) techniques for geospatial RSI data, but these systems typically rely on traditional relational models as basis for query processing techniques and architectures. Complex types of stream objects, such as multidimensional data sets or raster image data have not been considered.

The *GeoStreams* project investigates joining these two disciplines. The architecture allows users to formulate queries on continuous streams of RSI data. The outputs of these queries continuously return RSI data products to the user. These streams can be fed into applications to allow a continuous source of new input data from a single stream, or saved to more traditional RSI forms. As the functionality of the RSI DSMS increases, more aspects of the applications can be formulated as part of the queries themselves.

An application focus dealing with real-time weather satellite imagery provides an important and relevant backdrop for the research. A data and query model for streaming imagery is described. New processing strategies for query processing of RSI streams are investigated. Tools for efficient processing are created and a complete system for interfacing with a satellite image stream is developed.

Chapter 1

Overview

Remotely-Sensed Imagery (RSI), in particular satellite imagery, play an important role in many environmental applications and models [?]. Simple, convenient access to remote sensing data has traditionally been a barrier to research and applications. The huge amounts of data generated by the Earth Observing System (EOS) platforms have precipitated a change in this scenario, and access to data products has become substantially easier. New EOS data archives offer fine examples of more transparent data access. New open standards for distributed access to geospatial image data, like the Web Map Services (WMS) specification [?] from the Open Geospatial Consortium (OGC) have also contributed to an increase in accessibility to data products. However, access to this imagery still largely centers on choosing coarse grained, standard data products for specific regions and times. Applications that study changes in the environmental landscape require frequent, often continuous access to these data, and the temporal discontinuity in these access methods can force complicated pre-processing and synchronization steps between the data provider and the data user.

Remote sensing instruments, however, acquire data in a more stream-oriented fashion. Data is acquired continuously and transmitted to receiving stations in a continuous manner. Outside the realm of image databases, there have been recent advancements in Data Stream Management System (DSMS) applications, with new proposed query processing techniques [?, ?, ?] and research applications [?, ?, ?]. In such systems, data arrives

in multiple, continuous, and time-varying data streams and does not take the form of persistent relations. There is clearly benefit in applying techniques developed for DSMSs to streaming RSI applications.

The *GeoStreams* project [?] investigates joining these two disciplines. The architecture allows users to formulate queries on continuous streams of RSI data. The output of these queries continuously feed new RSI data products back to the user. These streams can be fed into applications to allow a continuous source of new input data from a single stream, or saved to more traditional RSI forms. As the functionality of the RSI DSMS increases, more aspects of the applications can be formulated into the queries themselves.

1.1 Motivation

A conceptual overview of the proposed system architecture for a database over streaming RSI is shown in Figure 1.1.

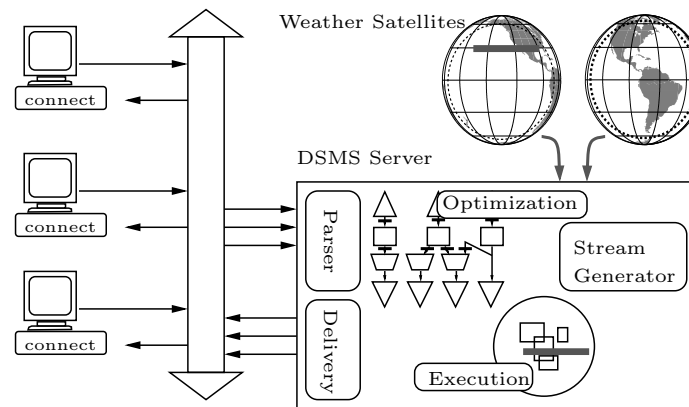


Figure 1.1: *GeoStreams* database system

Multiple users connect to a server and formulate queries to the system. The system is optimized for continuous queries on the input satellite stream of data. The queries are parsed and validated, then optimized. Optimization includes single and multi-query methods, combining queries to minimize computation time and/or the number and size of intermediate images that are created and maintained in the system. Minimizing the size of images typically reduces both memory usage and computational burden. New queries affect

the execution plan for the system, but these changes are made at specific times because the execution is continuously working on the incoming RSI stream. This stream comes from a separate module that re-interprets the raw data into a format more suitable for query processing.

Query execution is highly dependent on the structure of the incoming data. The RSI data is manipulated one row at a time. This matches the form of the satellite stream and is also convenient for multi-query optimizations. Figure 1.2 shows a notional example of how data is scanned and transmitted by a geostationary satellite.

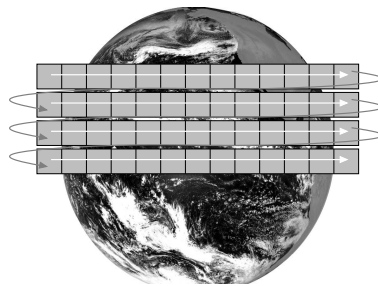


Figure 1.2: RSI data stream

The query execution includes final operators designed to return the data back to the clients, which requires persistent or synchronous connections on both the server and the client sides.

Images and image manipulation will be described with an image algebra, a rigorous and compact method for describing images, image transformations, and image analysis [?]. Image algebra is a many-valued algebra that includes *Point Sets*, *Value Sets* and *Images*. *Points Sets* are the points in space and time with defined values. *Value Sets* are the values associated with the points in the point set. *Images* can be thought of as functions mapping points to values, or as a set of point value pairs. These individual pairs are the *pixels* in the image.

Image operations are the basic building blocks for queries to the system. These operations include functional operations, image restrictions to specific point sets of interest, spatial transforms on images from one point set to another, and neighborhood operations where multiple pixels from an image are combined to a single value.

Defined operations on or among images include any operation that operates on the value set \mathbb{V} , which extends to a natural *Induced operation* on \mathbb{V} -valued images. *Image restrictions* return image subsets restricted to a given point set. Restrictions are possibly the most important of all operations, and flexible methods for defining new point sets need to be included in the query formulations. This is especially true in this model where point set restrictions define not only spatial, but also spatio-temporal limits on incoming data streams.

Spatial transformations map an image from one point set to another. Spatial transformations are used for magnification, rotation and general affine transformations. The most common spatial transformation converts the satellite point set to the point set grid requested by the queries. Each query can specify pixel locations and resolutions that differ from the input images.

One component of these transformations is to project satellite data to new coordinate systems. Data as it is received from the Geostationary Operational Environmental Satellite (GOES) satellite is in its own perspective projection. Many applications and users would prefer data to be delivered in a more standard Earth coordinate system. For example, Figure 1.3 shows a comparison of the GOES visible channel, in both its native projection, and projected to longitude-latitude (equidistant cylindrical).

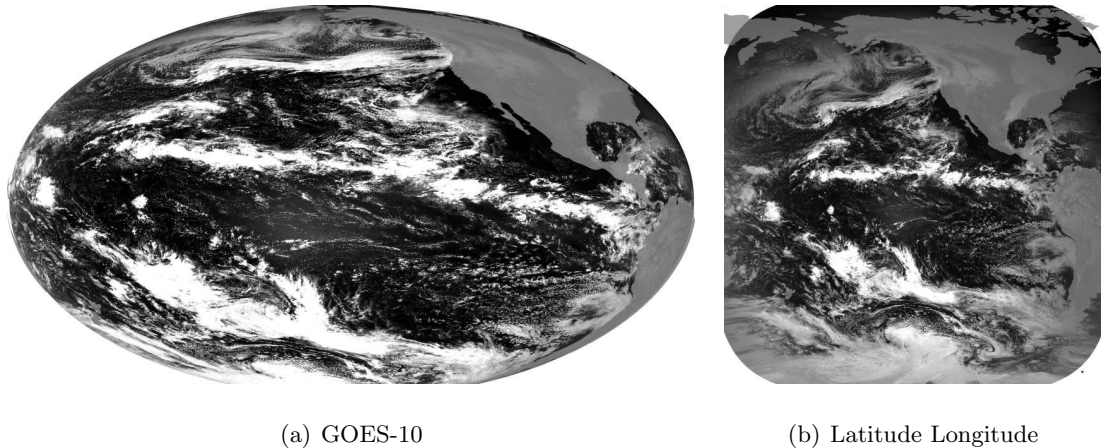


Figure 1.3: The GOES satellite projection.

Neighborhood operations allow for multiple pixels from a single image to be com-

bined to create a single pixel value in a new image. Neighborhoods allow for aggregation functions like averaging, edge detection, speckle removal, and other operations.

Queries in the *GeoStreams* project do not build on a variant of the SQL syntax, but on something closer to the image algebra representation and on specialized interfaces. The specific query declaration uses a formulation inspired by WMS specification [?]. Users specify a specific data product, coordinate system, spatial extents and pixel size via the resultant image height and width. Temporal restrictions can also be identified.

Queries can be made on other data products besides the satellite image channels. Data products like image channel ratios can be specified as long as the index itself is identified as a specific product by the server. The specification further simplifies query formulation by standardizing and simplifying both spatial transforms and restrictions to a limited but well-defined subset. More formally, these queries can be identified as a particular expression in an image algebra formulation, including a restriction, induced operation, neighborhood operation, and coordinate system transformation, where all operations are compactly described in a standard way. This simple interface does not allow for a sophisticated set of user queries, but it does satisfy the most basic requirements of serving many spatial restrictions and geometric transformations to many clients.

Query optimization attempts to limit the processing time and/or the amount of memory usage for the DSMS as a whole. Query optimization is primarily concerned with two goals: query rewriting to limit the amount of work done in the system, and finding common subsets within the queries active against the image stream, which allow results to be shared among multiple queries.

First, individual queries are rewritten to optimize their individual execution, the queries are then optimized in a multi-query fashion. Multi-query optimization centers around grouping similar query components into a single operation that works simultaneously for a group of queries. In DSMS research this has multiple conceptual definitions, including grouped filters [?] and query indexing [?]. Figure 1.4 shows a typical query index scheme for a spatial restriction operation, where rather than each query requiring its own restriction operator, a single restriction module has indexed the point sets of a number of active queries. For each continuous user query, a region, R_i , is associated that describes

the Region of Interest (ROI) for that query. For incoming RSI data, it is determined what data is relevant to what user queries and which queries can share incoming data.

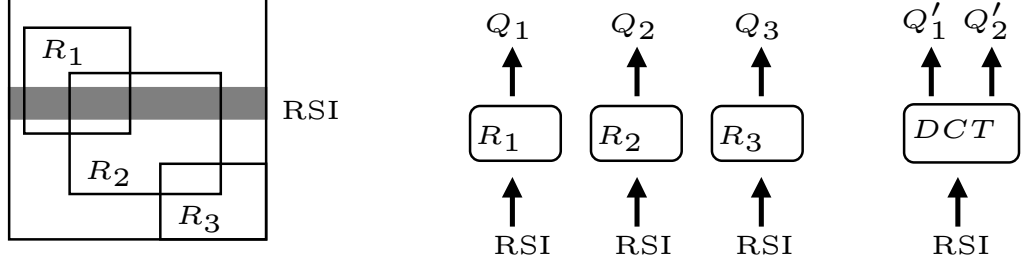


Figure 1.4: Multi-query restriction operation

By developing modules for the basic image operations that can take as input a single RSI stream and distribute results to multiple output streams, a Query Execution Plan (QEP) is developed as a number of these operators joined together for a complete system. This allows not only the pipelining of image data to operators to which the data is of interest, but it also facilitates the sharing of image data among queries that have overlapping ROIs.

Query execution is tied intrinsically to the query plan developed by the optimizer, and also by the organization of the incoming data stream. In developing the *GeoStreams* architecture, modules are developed for each of the basic image operations, which apply to more than one query in a single operation. The modules are linked together for complete query execution. As mentioned, the RSI data stream comes in an ordered row-by-row arrangement. This organization plays an important role in developing image operators and determining how modules in the query plan are arranged. By developing stream-oriented operators that work on these image rows, a system can be developed that uses minimum amounts of memory, and can deliver data in near real time.

Figure 1.4 shows an example module for processing multi-query image restrictions. For the restriction module, the Dynamic Cascade Tree (DCT) [?, ?] is proposed as a space efficient structure to index ROIs that are part of more complex queries against RSI data streams. The spatial trends inherent to most types of streaming RSI data are exploited to build a small index that is especially efficient when the incoming stream data is in close

spatial proximity. Queries can be answered very quickly if the next data stream segment has the same result as the previous query and will incrementally update a new result set when the result is different. Based on the information provided by the DCT, incoming data can be pipelined to respective query operators, providing the basis for multiple-query processing models for streaming RSI data.

1.2 Objectives

The overall goals of the *GeoStreams* project were to investigate processing strategies for query processing of RSI streams, to provide tools for efficient processing, and to develop a complete system for interfacing with a satellite image stream. The system answers continuous queries from multiple users. Specific objectives for the research included:

- Identify a data and query model that is natural for users, researchers and application developers, as well as concise and unambiguous. The models need to be simple enough to be easily understood, but with enough richness to satisfy a large class of data and query specifications. Develop a simple query syntax to represent the query model.
- Define a core set of operators that provide sufficient support for answering user queries on RSI data. These operators should naturally follow from the data and query model, and represent an implementation of those models.
- Develop query optimizations that allow a DSMS system to tailor its Query Execution Plan (QEP) to the currently active queries. Optimizations include query rewriting to optimize each individual query. Multi-query optimization techniques designed to share operators and common data among queries will also be examined.
- Create specialized operators that take advantage of the highly organized structure that is common in Remotely-Sensed Imagery (RSI) data.
- Design studies to validate query optimization techniques. These studies will test the operational parameters of individual components of the QEP. In addition, overall QEPs will be compared to similar systems running without such multi-query optimization strategies.

- Architect the design of a complete on-line system for processing continuous queries from multiple users. A processing design could be developed for use with an existing Geographic Information System (GIS) and image processing application. In addition, a ground up implementation designed specifically with the streaming nature of the incoming RSI data will be described.

1.3 Summary of Results

The research carried out to address the above objectives focuses on applications dealing with real-time weather satellite imagery. In particular, National Oceanic and Atmospheric Administration (NOAA) Geostationary Operational Environmental Satellite (GOES) satellite [?] was used as the input RSI data stream to the system, and all queries were made to this stream. The GOES program is one of the most important meteorological programs in the United States and offers an imaging instrument with five radiometric channels for a variety of applications, including cloud identification and tracking, atmospheric water measurement, absorbed solar energy, and thermal studies of clouds and Earth surfaces.

The DSMS applications developed allow multiple users to connect to a server to receive this RSI. A simple, but effective interface allows users access to most needed set of queries to the system.

Major results from this research correspond to the chapters of this work and are summarized below.

The *GeoStreams* Model, Chapter 3

A data and query model was developed based on image algebra, described in detail in Chapter 3. Additional specifications particular to the *GeoStreams* architecture are added to an image algebra to provide concrete semantics for streaming RSI imagery. Some image algebra operators were altered to allow for point sets of indeterminate length.

Queries are defined as image algebra expressions. A small subset of expressions are allowed as queries. In particular, a simple syntax using the Web Map Services (WMS)

interface was adopted for queries.

The basic query processing operators were developed from the model based in the image algebra and the allowed query expressions. In particular, operators for *induced operation*, *restrictions*, *spatial transformations*, and *neighborhood operations* were defined. Rules for expression rewriting were also introduced.

Query Processing, Chapter 4

One major feature of an effective query processor is choosing an optimal execution plan for a given query. Because queries are in image algebra, specialized rules regarding the rewriting of image algebra expressions are developed. A strategy is introduced to create an optimal or “best-effort” QEP. Both single query optimizations and multi-query optimizations are utilized.

Single query optimization deals with rewriting a query to be more efficient in its computation. For a single query, optimizations concern the ordering of the operators. Multi-query optimization adds additional savings by sharing results between queries to develop a single global QEP.

Multi-Query Optimization with Existing GIS Applications, Chapter 5

The processing improvements from single and multi-query optimization techniques is dependent on both the implementation of the individual operators and on the make-up of the current queries within the system. The savings of a multi-query QEP are quantified using an existing application framework, a real RSI image stream, and real world query parameters.

The application framework chosen is the Geographic Resources Analysis Support System (GRASS). GRASS has no notion of image streams and, like most GIS applications, works on discrete images in secondary storage.

Experimental results using predicted query patterns over the visible hemisphere of the weather satellite indicate that developing multi-query optimized plans can improve performance significantly when compared to queries executed separately.

The Dynamic Cascade Tree, Chapter 6

Multi-query optimization techniques require an operator that is able to provide restrictions for multiple query ROIs. RSI rows enter the system with high frequency and there can be many individual ROIs, and many restriction operators handled in a QEP. Also, the on-line implementation of the spatial transformation operator requires a restriction operation with many ROIs, one per output row.

To satisfy these needs, an index named the Dynamic Cascade Tree (DCT) was developed, which indexes spatio-temporal ROIs. The DCT is designed to exploit the spatial trends in incoming RSI data to provide a very efficient index. Experimental results using both random input and Geostationary Operational Environmental Satellite (GOES) data give a good insight into restrictions on streaming RSI and verify the efficiency and utility of the DCT.

Multi-Query Execution using an Online System, Chapter ??

An on-line DSMS for RSI data is designed in Chapter ??. A DSMS system designed specifically for RSI data operates on smaller discrete parts of an image directly as they arrive. Input image data is manipulated a single row at a time. This is most consistent with the data stream arrival pattern and allows for the smallest in-memory footprint of the processing system. The DSMS is divided into three main components, the Query Manager (QM), *Row Memory*, and *Query Data Management*.

The QM is the main component. It creates a Query Execution Plan (QEP) for the system, creates needed operators, and executes the plan. The QM also includes implementations of the individual operators; an implementation for general induced operations; a restriction operator designed to allow multiple restrictions to be satisfied simultaneously; a halving operator for averaging; and a spatial transformation operator for image projection.

The *Row Memory* component provides an interface to create, subset, and destroy rows of data in the system. The *GOES VARiable (GVAR) Input* handles the interface to the GVAR data stream and creates rows for input into the system. *Query Data Management* provides support for making the data available to the users, including formatting images

and distributing results.

1.4 Reference Queries

Tables 1.1 and 1.2 describe an example set of queries that might be run against an RSI DSMS system. These queries will be revisited throughout the paper and will be used to illustrate important aspects of query optimization and execution. Because of their role, the queries are chosen for their suitability as examples and not as a representative cross section. For example, the input streams are limited to a small number of channels from a single satellite, to increase in interplay between the queries.

Table 1.1: Example queries

Q	Product	ROI	Time	Projection	Resolution
A	$C1$	Mexico	Always	GOES	$\approx 1 \text{ km}^2$
B	$C1$	N. America	Always	Lat/Long	$\approx 4 \text{ km}^2$
C	$\text{NDVI}(C1, C2)$	N. America	Always	GOES	$\approx 4 \text{ km}^2$
D	$\text{NDVI}(C1, C2)$	Hemisphere	Always	Lat/Long	$\approx 8 \text{ km}^2$
E	$C1$	N. America	Time= \mathbf{t}	GOES	$\approx 1 \text{ km}^2$
F	$C1$	California	Days < \mathbf{d}	UTM	1 km^2
G	$C1$	Point \mathbf{G}	Always	N/A	N/A
H	$C1$	Pixel Values > v	Always	GOES	1 km^2

Table 1.2: GOES queries

Q	Description
A	Query A represents a user requesting Channel 1, (visible) data over a ROI covering Mexico. There are no time constraints on the query. The resolution corresponds to the default GOES data, as does the requested projection
B	Query B is another query on the visible channel, however in this instance, the user wants the data at a coarser resolution, and projected to a longitude-latitude, (equidistant cylindrical) grid.
C	This query represents a user requesting a product, in this case NDVI, which is a normalized difference index using two input channels. This query is limited to North America.
D	This query represents an NDVI request at a coarse scale and covering the entire northern hemisphere. The requested image is also projected to longitude-latitude.
E	This query requests visible GOES data, but only the image that occurs at some time, t , everyday.
F	This is similar to the query E , but also specifies an ending date, d , when the query will end.
G	This query asks for image data at a single point.
H	Channel 1, but only in the image where the <i>pixel values</i> are greater than some amount <i>v</i> .

Chapter 2

Background

The *GeoStreams* architecture draws from work in Geographic Information System (GIS), including remote sensing, image processing, meteorological satellites, and geospatial databases. In addition, ideas are drawn from Data Stream Management System (DSMS) research. In this chapter, these areas are reviewed from both a technical and historical perspective.

2.1 Geographic Information Systems

A Geographic Information System (GIS) is a system designed to create, store, edit and maintain geographic information. More generally a GIS is a combination of computer software along with geographic data that allows users to interactively query spatial data, analyze that information, and support decision making. The term Geographic Information System is something of an open ended term as it deals with any data having a geographic component. It covers a large number of disciplines. The focus here is on the development of GIS tools that have been developed in support of processing, storing, managing and distributing remote sensing data.

Remote sensing, in contrast to in-situ measurements, is the process of acquiring information about something without being in physical contact with the object. Typically, remote sensing applications involve obtaining information about the Earth from sensors at some distance from the surface. Remote sensing instruments detect reflected or emitted

energy from the surface and make inferences about the properties of the surface based on that information. The field of remote sensing is large and includes studies describing image processing methods [?], applications oriented towards environmental studies [?, ?, ?], and more general descriptions [?].

2.1.1 Remote Sensing

In a very real way, remote sensing has its roots in pre-history, as people are in possession of one the finest remote sensing devices devised, the eye. Largely because of the anthropogenic basis of remote sensing, it is a technology that has been embraced in a wide array of disciplines.

Remote sensing as a science closely follows the advent of photography. The first natural photographic image was taken in 1827. By the 1850's multiple techniques existed for sharp images with relatively short exposure times. In 1855, a patent on using aerial photography for mapping was issued, and the first aerial images were taken from a balloon in 1858. Aerial photographs may have been used for military purposes as early as the Civil War in the 1860's. By the late 1880's and into the early 1900's, aerial photography was being used for environmental studies, military campaigns, and even the first rocket launched imagery. World War I provided a boost in the science of photo interpretation, and by the 1920's, books were being written about the subject [?, ?]. World War II encouraged more sophistication in interpretation, as well as more products, including the first infrared film, which allowed for more sophisticated interpretations, especially vegetation studies. The 1960's marked the beginning of imaging satellite systems, beginning with the first meteorological satellite, the Television and Infrared Observation Satellite (TIROS) (Figure 2.1(a)), launched in 1960. These satellites were equipped with a television camera that radioed images back to Earth, offering meteorologists an important new tool. 1960 also saw the launch of the first US spy satellite, a top secret program code named CORONA. At its peak in the late 60's and early 70's, CORONA satellites were acquiring images with 2 meter resolution (Figure 2.1(b)). The photographs were parachuted back from space. The CORONA program was only declassified in 1995, and the images are now available for purchase from the U.S. Geological Survey (USGS).

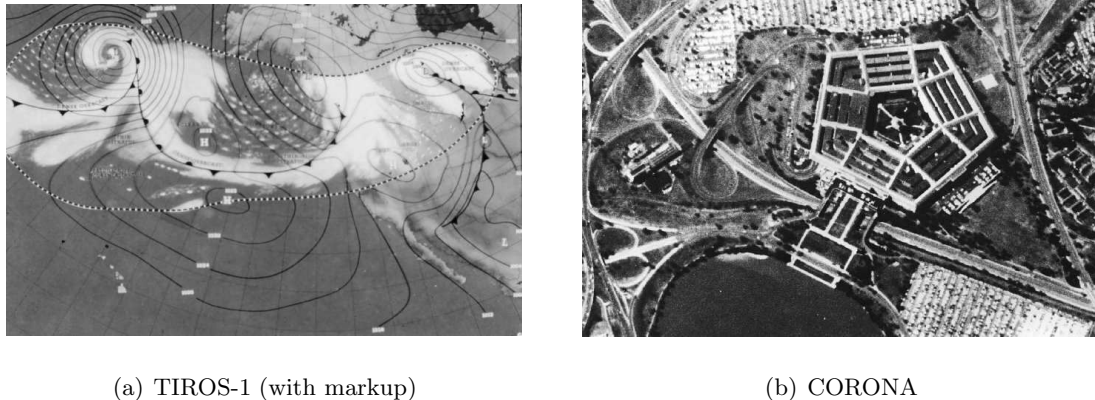


Figure 2.1: Early satellite images

TIROS heralded a new discipline of digital image processing of Remotely-Sensed Imagery (RSI). More general GIS applications were also being developed [?, ?]. At this time, main-frame computers and specialized hardware were required for any image processing.

The Nimbus meteorological program, and the Earth Resources Technology Satellite (ERTS), later renamed Landsat, environmental satellite program both began in the early 1970's, and further advanced remote sensing. The Large Area Crop Inventory Experiment (LACIE) program, started in 1974, was one of the first programs researching the application of digital remote sensing to environmental studies, in this case for determining global wheat crop yields [?, ?].

The 1980's and 1990's saw a dramatic increase in the number of satellites launched, in the image processing algorithms developed, and in the number of users of GIS and image processing software. In 1988 the National Center for Geographic Information and Analysis (NCGIA) was founded in response to a National Science Foundation (NSF) call for proposals. The NCGIA's general mission is to advance research in GIS science, with focus on modeling, spatial accuracy, cognition, and other research areas. For image processing, the U.S. Army Corp of Engineering Research Laboratories (USA-CERL), developed the Geographic Resources Analysis Support System (GRASS) from 1982 until 1995 [?, ?]. One of the first fully featured raster based GISs, GRASS was developed for management and planning applications, and has grown a wide range of tools for a variety of applications.

GRASS is currently developed in the public domain, and continues with active development, including new versions with enhanced capabilities, and new graphical front end applications, such as Quantum GIS (QGIS) [?].

In terms of general purpose GIS systems, a leader in the field is Environmental Systems Research Institute (ESRI) [?]. ESRI was founded in 1969, but started developing a core GIS computing application, ARC/INFO in 1982. Originally dealing with only point, line and polygon data, the suite of ESRI tools began incorporating image processing in the late 1980's and early 1990's. The current image processing and remote sensing applications include a wide array of processing algorithms, tight interaction with vector GIS data, and object oriented image classification [?].

The Earth Observing System (EOS), a NASA program having launched over twenty instruments designed for global monitoring of the Earth's land, sea, and atmospheric processes, is the current state of the art in remotely sensed image acquisition and also data collection and distribution. The first EOS satellite was launched in 2000.

Today, there are over 800 satellites in space, more than 150 of which have some remote sensing instrumentation. More than half of these are operated by the United States [?]. More and more users are using GIS and image processing in a growing set of diverse applications.

2.1.2 Geostationary Operational Environmental Satellite (GOES)

Kidder [?] introduces the topic of remote sensing specifically for meteorological applications, with descriptions of the current state of the art in terms of available instruments and applications. The GOES program is one of the most important meteorological programs in the United States and is a continuation of the original meteorological satellite program. The program has been active since 1974, with the launch of the Synchronous Meteorological Satellite 1 (SMS1), an experimental instrument pioneered by NASA. National Oceanic and Atmospheric Administration (NOAA) began an operational program based on this system with the launch of GOES 1. Since the launch of SMS1, there has been continuous operational satellite coverage over the Western hemisphere from these systems.

GOES satellites are geostationary. There are two main orbits for operational

satellites, sun synchronous and geostationary. In *Sun synchronous* orbits the satellite crosses the equator at the same local time with every pass. This orbit is nearly polar, and this class of satellites are called *polar orbiting* satellites. These satellites must also be in a low Earth orbit, and typically orbit the Earth about once every 90 minutes. *Geostationary* orbits, in contrast, essentially remain motionless above a point on the equator with respect to the Earth. Geostationary satellites need to be about 35,800 km above the Earth, roughly 5.6 Earth radii. There are two GOES satellites in place to cover the United States. Figure 2.2 shows these orbits, with the path of the GOESs satellites identified.

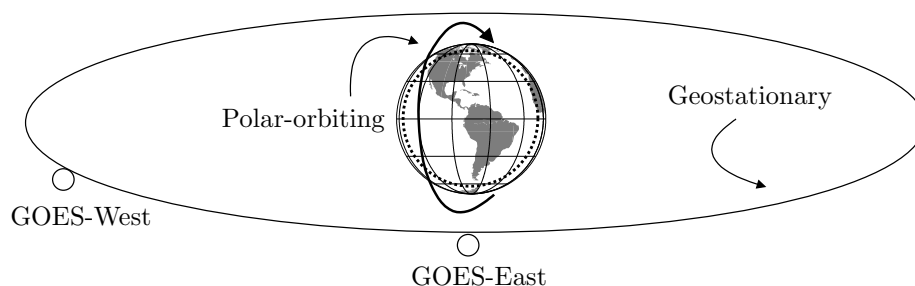


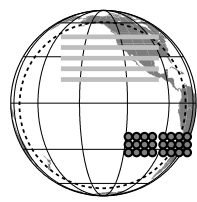
Figure 2.2: Satellite map showing GOES orbits.

Passive remote sensors detect either reflected solar energy or energy emitted from the surface itself. *Active remote sensors* emit energy, typically radar or laser energy, and measure the reflected energy from that source. The GOES satellites are passive systems.

Table 2.1 gives an overview of the GOES sensors. The satellite continuously scans a hemisphere of the Earth with two sensors, the Imager and the Sounder. The Imager acquires higher resolution images in a limited number of bands, while the Sounder has low resolution images, but with 19 bands designed for vertical atmospheric profiling. GOES-West offers a continuous stream of data for ROIs ranging from the continental United States to a hemisphere centered near Hawaii at about 135° West longitude. GOES-East is centered on the equator over South America, at about 75° West longitude.

The current GOES Imager is a five channel radiometer with one spectral band in the visible region, two in the mid-infrared and two thermal-infrared regions. The passive radiometer senses reflected solar energy in the visible, a combination of reflected and emitted energy in the mid-infrared bands, and emitted energy from the Earth in the thermal bands.

Table 2.1: GOES satellite sensors



Imager	Sounder
4 [km^2] pixels	64 [km^2] pixels
5 spectral bands	19 spectral bands
Data arrives by row(s)	Arrives by point
1-8 rows at a time	
$[2,500 - 17,000]^2$ size	

The GOES Imager uses a telescope with a two-axis scanning mirror at the entrance. The mirror is moved using servos, which in turn scan the hemisphere of the Earth. The multi-spectral channels of the Imager, simultaneously sweep an 8-kilometer swath on east-to-west and west-to-east paths, at a rate of 20 degrees (optical) east-west per second. This means the Imager can scan a 3000 by 3000 km "box" centered over the United States in about 41 seconds. The scan takes place by sweeping East to West, stepping the instrument South and then scanning back West to East.

The imaging sensor, including the telescope, scan assembly, and detectors, is located outside the main structure of the spacecraft. Additional control electronics and power units are located internally in the spacecraft. Table 2.2 describes some of the spectral characteristics of the GOES Imager. IGFOV is the Instantaneous Geographic Field of View (IGFOV) directly below the instrument, at the nadir.

Table 2.2: GOES Imager characteristics

Channel number	1 (Visible)	2 (Shortwave)	3 (H ₂ O)	4 (IR 1)	5 (IR 2)
Wavelength [μm]	0.55-0.75	3.80-4.00	6.50-7.00	10.20-11.20	11.50-12.50
Spectral Region	Visible	Shortwave	Moisture	TIR	TIR
IGFOV [km]	1	4	8	4	4

The RSI stream is continuous weather imagery from the NOAA GOES [?]. All data from the GOES satellite is transferred via a format specific to these instruments, the GOES VARIable (GVAR) format. Figure 2.3 shows the GVAR [?] data stream.

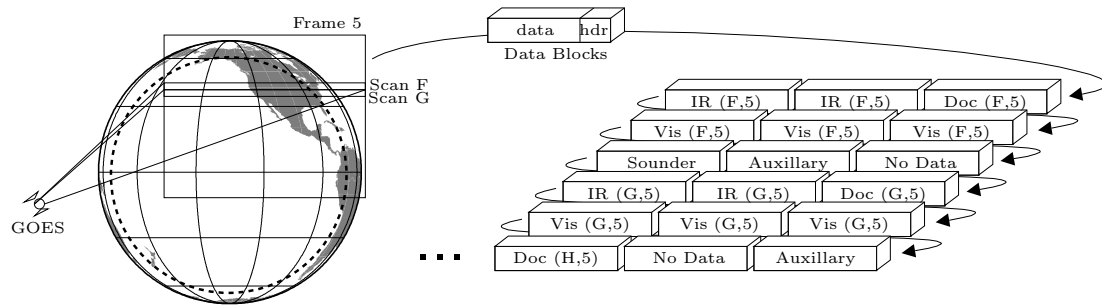


Figure 2.3: GOES GVAR data stream

The GVAR stream transmits at approximately 2.1 Mb/sec. A frame varies in size from about 100MB to 400MB, depending on the size of the region scanned. The footprint of an Imager pixel varies between spectral channels. Individual frames are made up of a large number of scans which are narrow swathes of data corresponding to the physical sweep of the instrument sensors from East to West. The scans themselves are made up of blocks of data, which are the atomic unit of transfer from the satellite to the receivers. Blocks contain from 32K to 229K bits of data, depending on the width of the scan. GOES visible channel data blocks contain 8 rows of data. Other channels from the GOES Imager come in blocks of 1 or 2 rows. The number of columns in a row varies from frame to frame. An entire frame of data is reported 8 rows at a time from North to South. New frames start from their most Northern extents.

Documentation blocks include operational parameters, such as the current location of the satellite, the current frame and scan, parameters for each instrument, and additional information for each spectral channel.

2.1.3 Geolibraries

The first, and still most important role of databases in relation to RSI, is not to manipulate the images but instead to catalog what images exist for what times and locations and organize this information for users and producers.

As expected, geospatial libraries were not an important area of research until warranted by the volume of available data. Through the 1970's and even into the early

1990's, the limited number of data providers and the method of data distribution, usually via computer tapes, limited the need for sophisticated interfaces to repositories of digital RSI data. More and more satellite instruments, coupled with the advance of the World Wide Web (WWW) have driven the need for more advanced methods of access to RSI archives.

In 1993, the National Research Council (NRC), anticipating the need for coordination of National spatial data envisioned the National Spatial Data Infrastructure (NSDI):

The National Spatial Data Infrastructure is the means to assemble geographic information that describes the arrangement and attributes of features and phenomena on the Earth. The infrastructure includes the materials, technology, and people necessary to acquire, process, store, and distribute such information to meet a wide variety of needs [?].

This led in 1994 to the founding of the National Spatial Data Infrastructure (NSDI) [?], by executive order, to establish technologies and standards necessary to encourage the use and sharing of geospatial data within a number of communities, including government, academia, and the private and non-profit sectors. Goals of the program include reducing costs and duplication and improve the quality of geographic information. NSDI committees have developed a number of standards covering the content standards for various geographical data, transfer formats, spatial accuracy, and location grids. Perhaps the most important standard from the NSDI is the Content Standard for Digital Geospatial Metadata (CSDGM) [?]. This standard describes the required metadata that should be associated with any GIS data product. It is used as a base record for many geospatial library organization structures.

In 1998, the NRC revisited the need for geospatial libraries [?]. One of the main reasons for this was that the original studies did not anticipate the profound impact that the WWW has had on data distribution and distributed systems. The panel argued the need for a common vision among geolibraries, and put forth 15 major findings, pointing toward the need for distributed systems based on WWW services. The NRC defined a geolibrary:

A geolibrary is a digital library filled with geoinformation and for which the primary search mechanism is place. Geoinformation is information associated with a distinct area or footprint on the Earth's surface. A geolibrary is distributed if its users, services,

metadata, and information assets can be integrated among many distinct locations.

Of particular interest were the extensions in service that the NRC anticipated for these repositories.

Finding 8: A distributed geolibrary would allow users to specify a requirement, search across the resources of the Internet for suitable geoinformation, assess the fitness of that information for use, retrieve and integrate it with other information, and perform various forms of manipulation and analysis. A distributed geolibrary would thus integrate the functions of browsing the WWW with those of GIS and related technologies.

Today, there are many more examples of working geolibraries. Both the USGS and National Aeronautics and Space Administration (NASA) offer exceptional geospatial data libraries. NASA's satellite data is made available via the Distributed Active Archive Center (DAAC) [?], which offers access to data products with multiple search interfaces and download capabilities.

USGS makes all maps and imagery available through several sites including the Earth Explorer [?] and the Seamless Data Distribution System [?]. These services all provide many of the required features of a geolibrary, typically over a single repository, however. Queries working over distributed, heterogeneous databases, is still an important need in the GIS community.

2.1.4 Standards

Standards fulfill an important role in making data products most useful to the widest audience. Just as the early days of remote sensing did not need common methodologies for description and discovery, many data products developed their own specific data formats. Early image formats, still quite popular today, were very simple binary representations of the image pixels. Additional information is included in separate files for inside image headers. In recent years, various standards have been introduced into the GIS landscape, and have been widely adapted in many applications. Two examples include the ESRI Shapefile for vector data and the GeoTIFF standard for images. The Shapefile format [?] describes points, lines, or polygons, along with associated attributes of the items. There is also a specification for an attached R-tree based index structure, and a coordinate space.

The GeoTIFF specification [?], published in 2000, is an extension of the TIFF standard that embeds metadata relating to georeferencing information within the file itself. Both standards have become very popular because they are relatively simple and effective formats that have grown into standards having proved their utility in multiple applications. These types of formats have typically succeeded far better than formats developed from whole cloth by committee, the Spatial Data Transfer Standard [?]. The GeoTIFF format is the de facto standard for remotely sensed imagery, although there are many, many formats in active use [?].

Some standards organizations, however, have been very successful in encouraging GIS standards, none so more than the Open Geospatial Consortium (OGC). The OGC is a consensus based organization, consisting of members from government, academia, and commercial fields [?]. The OGC has been especially effective in developing interoperability standards for Internet and web-based scenarios:

- The OGC Abstract Specification is the conceptual foundation for most OGC specification development activities and provides a reference model for the development of implementation specifications.
- Coordinate systems are defined with a Well Known Text (WKT) [?] representation, used in most OGC specifications.
- The Geography Markup Language (GML) is an eXtensible Markup Language (XML) grammar for the modeling, transport, and storage of geographic information, providing objects for describing geography including features, geometry, topology, time, units of measure and generalized values.
- The Web Map Services (WMS) and Web Coverage Service (WCS) specifications allow clients to retrieve map images for display or to provide a network interchange of geospatial image data through a standard query mechanism.

The OGC is also a major force in the creation of the International Standards Organization (ISO), in particular the ISO/TC 211 [?] technical committee on geographic information and geomatics. The ISO 19100 series of standards are incorporating the OGC

abstract specification, and a number of other OGC specifications are or will be adopted by ISO/TC 211 [?].

2.1.5 Geospatial Databases

Geospatial databases typically differ from geolibraries in that the focus is on manipulation of spatial data, as opposed to archival and retrieval. As discussed in Section 2.1.3, this distinction is becoming somewhat artificial as geolibraries expand their services. Most geolibraries use geospatial database backends.

There are many sources of information on the particulars of geospatial databases. Güting [?] and Rigaux [?] both include comprehensive introductions to the terminology of these systems. The material below was distilled in large part from this introductory material.

A *geospatial database* is a database that can store and describe *geographic objects*. Geographic objects within the database have a *geometric attribute* or *spatial extent*, which can describe in some manner the location, size and shape of an object. This description may be in two or three dimensions. Geospatial databases include extents that correspond to actual physical locations. There exists a mapping of the object's extent to an actual place. In addition to a spatial attribute, geographic objects may also have non-spatial or descriptive attributes. A Spatial Database Management System (SDBMS) allows access to the objects based on either their spatial or non-spatial attributes, or some combination of attributes. It is the geometric attribute of objects that is the defining feature of a SDBMS. For example, satellite images could be stored in an *image database*, where the focus might be on manipulating images based on the content of the images, rather than their location.

Spatio-temporal databases describe objects in both a spatial and a temporal manner. How objects are described in time, either as an additional dimension, a trajectory, a set of historical locations, or some other method, varies by their requirements and implementation and constitute an on-going research area.

One method of dividing what phenomena can be described by a spatial database is whether the model is *entity-based* or *field-based*. In an entity-based model, objects include descriptive characteristics and a geographic attribute. The geographic attribute defines a

number of points in space that make up that object. A representation of countries would be a good candidate for entity-based objects, where an individual country has a number of descriptive attributes along with an geographic attribute that defines the boundaries of the country. In *field-based* models, attributes are not associated with individual objects. Rather, attribute are considered as functions over some spatial domain and there exists one attribute value associated with any point in space. Images are field-based models. The system defined in this paper is predominantly field-based, although some aspects, like query ROIs, are entity-based.

Entity-based models can use *Tessellations* [?] to partition a space into a discrete set of cells. These cells are usually regular grids, but could be other structures like hexagons, or irregular triangular or polygonal separations. In these models the actual extent is approximated by an integral set of tessellation cells. These cells can be referenced directly, or a further decomposition, for example, a quad tree, might be used to minimize the description of the tessellation. Field-based data is also well represented with tessellations, with a modification that the attribute function is no longer defined on a continuous spatial domain, but instead on the discrete areas corresponding to the tessellation cells. All satellite imagery, in fact, all imagery in this research uses a point set definition that implies an associated tessellation of the underlying spatio-temporal space.

2.2 Data Stream Management Systems

Most research on querying and managing data streams has concentrated on either traditional data models where the data arrives in the form of simple records (tuples) or XML data. Several overviews on the recent advancements in Data Stream Management System have been published [?, ?, ?]. In such systems, data arrives in multiple, continuous, and time-varying data streams and does not take the form of persistent relations. The current interest in data stream management [?, ?, ?] has driven various new processing methods and paradigms for streaming data, such as adaptivity [?, ?], operator scheduling [?, ?], and load shedding [?, ?]. Some proposed approaches have extended to the realm of spatio-temporal databases as well, where new methods defining the spatial relationships

between queries and data streams are being investigated, in particular in the context of continuous queries over moving objects (e.g., [?, ?, ?]).

Formalisms on streaming databases are described by Arasu, Babcock, Motwani and Widom [?, ?, ?] and realized in the Stanford STREAM project, which has a goal of producing a general purpose DSMS that adds data streams into a Database Management System (DBMS) in a coherent and consistent manner [?].

Additional implementations of DSMS include Telegraph [?], based in part on PostgreSQL [?]. In Telegraph, all continuous queries are optimized with postgresql and executed with an eddy based routing to operators [?].

PSoup [?] extends some of Telegraph’s processing architecture to allow retrospective queries on data that was previously received. PSoup regards multi-query processing as a join of both query and data streams.

Aurora [?] is another DSMS that optimizes and executes queries in a process oriented, dataflow paradigm. Rather than using a declarative query interface, Aurora allows users to stitch together a stream-oriented process.

There are currently several research efforts in the context of DSMS that focus on core issues such as adaptive query processing [?, ?, ?, ?], meaning and implementation of blocking operators, including aggregation and sorting [?, ?], and approximate query processing techniques [?, ?]. Specific implementations of specialized optimizations include adaptive strategies to continually modify tuple processing [?, ?, ?, ?].

Streaming spatio-temporal data has primarily been investigated in the context of moving objects and location-aware services. Query processing and optimization aspects for streaming Remotely-Sensed Imagery (RSI) data are quite different. Streaming RSI is typical for the vast amount of imaging satellites orbiting the Earth and it exhibits certain characteristics that make it very attractive to tailored query optimization techniques.

Optimizing streaming databases has many similarities with methods for optimizing multi-queries in traditional databases, Sellis [?] offering some of the earliest examples for finding common sub-expressions. Other studies for multi-query optimizations through common sub-expressions include [?, ?, ?].

Figure 1.1 (page 2) showed an overview of a Data Stream Management Sys-

tem (DSMS) for Remotely-Sensed Imagery (RSI) data. Such data have a number of characteristics that are different from the typical streaming relational or spatio-temporal data typically described in a DSMS context. One aspect is that the incoming bandwidth of RSI data streams is very large, usually arriving as discrete parts of binary image data. Another important point is that streaming RSI data is more highly organized with respect to its spatial and temporal components than is usually assumed for more generic types of spatio-temporal data. This organization effects how data and queries can be processed in a RSI DSMS. These aspects are described in Section 3.2.1 and Chapter 4.

Chapter 3

The *GeoStreams* Model

Developing efficient optimization strategies for streaming RSI data requires a consistent and practical model describing the RSI data stream, the queries made to the system, and the operations that are performed on the stream to answer those queries.

The models developed here begin with some preliminary definitions of images and image operations, based in image algebra. From this foundation, some additional definitions specific to the *GeoStreams* project are introduced to define a data model defining images and image operations. Image algebra provides an underlying framework for manipulating images. However, in order to be used effectively for geospatial images in the streaming context of a DSMS, aspects specific to this application need to be addressed. These changes are discussed in Section 3.2. Some modifications are also required on certain operators to more easily allow for point sets of indeterminate length, as required for a DSMS.

The query model is introduced in Section 3.3. Although queries are basically image algebra expressions resulting in images, some additional information on the formulation of queries is defined. This includes the adoption of a simple syntax using the Web Map Services (WMS) interface. One major feature of an effective query processor is choosing an optimal execution plan for a given query. Because queries are in image algebra, specialized rules regarding the rewriting of image algebra expressions are introduced.

3.1 Image Algebra Definitions

An image has a fairly intuitive definition, but a more formal definition is required for the purposes of developing a consistent method describing operations on images. Image algebra is a unified theory for image transformation and analysis. It is a many-valued algebra, with multiple operators and operands. Images consist of sets of points and values associated with these points. Image algebra includes *Point Sets*, *Value Sets*, and *Images*.

A **point set** is simply a space. Individual elements of the set are called **points**. Common point sets used in image algebra include discrete subsets of an n -dimensional Euclidean space \mathbb{R}^n . Some important subsets include integer points, \mathbb{Z} , and n -dimensional point lattices, \mathbb{Z}^n .

A **point lattice** is a regularly spaced grid of points. Images corresponding to RSI datasets defined in this research all have square lattices on an integer point space as their point sets. Points in the image space are correlated with a real world locations through associations with a well defined projection system and a matrix transformation to that coordinate space (Section 3.2.1). Point sets are denoted with bold capital letters, i.e., \mathbf{X} , \mathbf{Y} , \mathbf{Z} . Points within a point set are denoted with lower case bold letters, i.e., $\mathbf{y} \in \mathbf{X}$.

A **value set** is a set and the associated operations among members of that set. Members of the set are called **values**. Value sets simply contain all the potential values associated with points in a single image. Typical value sets include integers, \mathbb{Z} and real numbers, \mathbb{R} . These value sets have the usual operations associated with their particular set. Value sets are denoted as capital letters in blackboard font like \mathbb{V} .

Images are functions mapping points to values, or as a set of point value pairs. These individual pairs are the pixels in the image.

The notation $\mathbb{V}^{\mathbf{X}}$ describes the set of all functions, $f \in \mathbb{V}^{\mathbf{X}} : f$ is a function $\mathbf{X} \rightarrow \mathbb{V}$. An image is such a function that maps from a point set \mathbf{X} to the value set \mathbb{V} . For an \mathbb{V} -valued image, $(\mathbf{a} : \mathbf{X} \rightarrow \mathbb{V})$, \mathbb{V} is the possible **range** of the image \mathbf{a} , and \mathbf{X} is the **domain** of \mathbf{a} .

Another convenient notation for an image $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$ is the **graph** or **data structure** representation, $\mathbf{a} = \{(\mathbf{x}, a(\mathbf{x}) : \mathbf{x} \in \mathbf{X})\}$. Here the pair $(\mathbf{x}, a(\mathbf{x}))$ constitutes a **pixel** of the

image. The first coordinate $\mathbf{x} \in \mathbf{X}$ is the **pixel location** and the second coordinate $a(\mathbf{x}) \in \mathbb{V}$ is the **pixel value** at location \mathbf{x} .

There are a number of operations on images that are required to answer the queries submitted to the system. The four most important operations are composition, induced operation, restriction, and neighborhood operation. These are shown in the notional diagram of Figure 3.1, and discussed below.

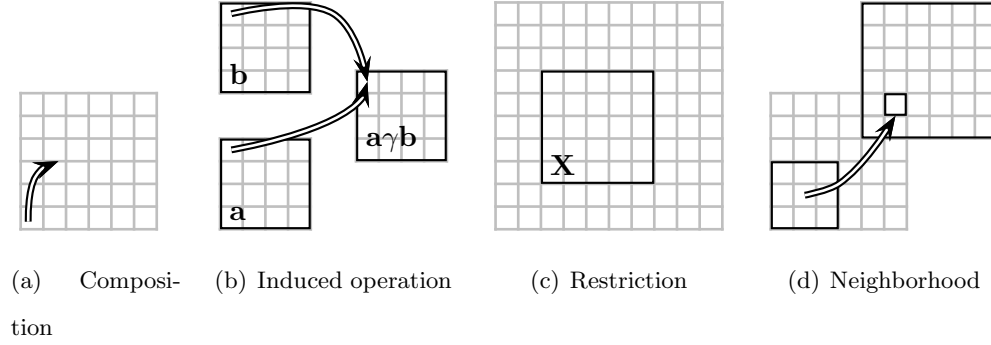


Figure 3.1: Image algebra operations

Since images are defined as functions, standard functional operations apply as well. A **composition** is the nesting of two or more functions to form a single new function. There are two orders on the nesting of the functions.

Given a function $\Gamma : \mathbb{V} \rightarrow \mathbb{W}$, the composition, or value transform, $\Gamma \circ \mathbf{a}$, changes an image in $\mathbb{V}^{\mathbf{X}}$ to an image in $\mathbb{W}^{\mathbf{X}}$ defined as:

$$\Gamma \circ \mathbf{a} \equiv \Gamma(\mathbf{a}) = \{(\mathbf{x}, \Gamma(a(\mathbf{x}))) : \mathbf{x} \in \mathbf{X}\}$$

Given a function $f : \mathbf{Y} \rightarrow \mathbf{X}$ between two point sets, the composition, or spatial transform, $\mathbf{a} \circ f$, changes an image $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$ from $\mathbb{V}^{\mathbf{X}}$ to $\mathbb{V}^{\mathbf{Y}}$ where

$$\mathbf{a} \circ f \equiv \{(\mathbf{y}, a(f(\mathbf{y}))) : \mathbf{y} \in \mathbf{Y}\}$$

The composition $\Gamma \circ \mathbf{a}$, can be used to define operations on the values of an image. A natural set of induced operations on images are defined. *Any operation that operates on the value set \mathbb{V} , defines a natural induced operation on \mathbb{V} -valued images.* If γ is some binary operation on \mathbb{V} , then for $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{V}^{\mathbf{X}}$, $\mathbf{a}_1 \gamma \mathbf{a}_2 = \{(\mathbf{x}, \gamma(a_1(\mathbf{x}), a_2(\mathbf{x}))) : \mathbf{x} \in \mathbf{X}\}$. If $\gamma()$

is an n -valued operation on values v_1, v_2, \dots, v_n , then $\gamma(a_1, a_2, \dots, a_n)$ is a similarly defined induced operation.

The functional composition, $\mathbf{a} \circ f$, where f is applied to the image point set is often used to project images to new geographic coordinate systems. For consistency to other RSI frameworks, these will be described as spatial transformations as they map an image from one point set to another. Note that the definition of a spatial transform implies that the image is defined for an arbitrary point located at $f(\mathbf{y})$. Section 4.2.3, describes how an RSI image may be defined on arbitrary points by, for example, interpolation methods on the original image.

Other basic concepts of function theory can be applied to images. In addition to the previously described domain, range and composition, these include *restriction* and *extension* operators. Restrictions can be defined on both the point set and the value set of an image.

If $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$, then the **point restriction**, $\mathbf{a}|_{\mathbf{Z}}$ of \mathbf{a} to a subset $\mathbf{Z} \subseteq \mathbf{X}$, is defined as

$$\mathbf{a}|_{\mathbf{Z}} \equiv \mathbf{a} \cap (\mathbf{Z} \times \mathbb{V}) = \{(\mathbf{x}, a(\mathbf{x})) : \mathbf{x} \in \mathbf{Z}\}$$

If $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$, then the **value restriction**, $\mathbf{a}|_{\mathbb{S}}$ of \mathbf{a} to a subset $\mathbb{S} \subseteq \mathbb{V}$ is defined as:

$$\mathbf{a}|_{\mathbb{S}} \equiv \mathbf{a} \cap (\mathbf{X} \times \mathbb{S}) = \{(\mathbf{x}, a(\mathbf{x})) : \mathbf{x} \in \mathbf{X}, a(\mathbf{x}) \in \mathbb{S}\}$$

The query model makes extensive use of the restriction operator. This is the primary mechanism to create image subsets that satisfy requests for the specific Region of Interest (ROI) for a query. The extension of an image, \mathbf{a} , can extend the image to an expanded point set beyond the original space of the image. By defining a function, \mathbf{b} , extensions can be used to grow images to expanded domains.

The **extension** of $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$ to $\mathbf{b} \in \mathbb{V}^{\mathbf{Y}}$, where \mathbf{X} and \mathbf{Y} are subsets of the point set \mathbf{Z} is defined as:

$$\mathbf{a}|^{\mathbf{b}}(\mathbf{x}) = \begin{cases} a(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbf{X} \\ b(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbf{Y} \setminus \mathbf{X} \end{cases}$$

Although the extension operator is not included explicitly in the query model below, refined models could use this to allow users to choose some additional parameters on queries. This is discussed in more detail in Section 3.3.4.

Finally, **neighborhood operations** *allow for multiple pixels from a single image to be combined to create a single pixel in a new image*. Although the neighborhood operation is conceptually simple, its definition is somewhat convoluted. For the summation operation of neighborhood \mathbf{n} over image \mathbf{a}

$$\mathbf{a} \oplus \mathbf{n} = \sum_{\substack{\mathbf{y} \in \mathbf{Y} \\ \mathbf{x} \in n(\mathbf{y})}} a(\mathbf{x})$$

Where the image \mathbf{n} has a point set, \mathbf{Y} , and each value of \mathbf{n} is itself a point set, $\mathbf{X}' \subset \mathbf{X}$. For every point $\mathbf{y} \in \mathbf{Y}$, \mathbf{n} describes a set of points whose values will be aggregated. Typically, the points sets \mathbf{X}' are small uniform regions around a single point, the 3×3 grid shown in Figure 3.1 for example.

Neighborhood operations are used for averaging satellite images into lower resolution pixels. Users often desire images at a coarser resolution than the raw satellite data when investigating or modeling large areas. The goal of the neighborhood operation is to model what the radiance of RSI image would be if it were acquired at this coarser scale. No methodology can replicate this goal exactly, primarily because, in general, the final point sets do not align exactly with the boundaries in the original image point set. This is especially true with queries in different coordinate systems, where the resolution for a point set transformed to the GOES reference frame is not even necessarily constant.

Section 3.3 will show how all the operations can be combined to allow for the specification of queries against the RSI data stream in the *GeoStreams* environment. First, however, the image algebra formulations require some modifications for use with streaming image data.

3.2 Data Model

The image algebra definitions above go a long way toward a definition of a complete data model. However, a few modifications and extensions to the model need to be included to cover some aspects particular to Remotely-Sensed Imagery (RSI). The aspects include additional features to reference image point sets to real world locations and an explicit ordering associated with the points in a point set. This ordering allows operations to execute on the incoming data stream. These modifications lead to the definition of a *GeoStreams* image. In addition, the ordering of the point set allows a streaming image to be broken down into smaller images. Two special subsets, the frame and row, will be explicitly defined. A single row of data will be the atomic element passed among operators during query execution. Rows, images, and frames are the primary components of a reference schema for the data streams in the system.

3.2.1 Streaming Geographic Point Sets

In order to support streaming geospatial images, the definition of a point set needs to be modified to explicitly include a frame of reference for the point sets corresponding to Earth coordinates, and to include an explicit ordering on the points. These modifications allow for images to be compared in terms of their underlying geographic coordinate system and for operations to proceed on streaming RSI data.

GeoStreams queries on the RSI data either explicitly or implicitly reference a particular coordinate system. However, the physical representation of the images are in a simple grid space. To provide a transformation between these two spaces, point sets need parameters that describe the translation between the raster and model coordinate systems. The implementation used in the *GeoStreams* system represents this transformation with two parameters, the *projection definition* and the *matrix transformation*. This representation is similar to the representation of the OGC GML standard [?, ?]. It also works well with the OGC WMS specification, which will be the basis of *GeoStreams* queries described in Section 3.3.2.

The *projection definition* is a standard representation described by the OGC Well

Known Text (WKT) [?]. These are text representations for all standard Earth projections. Figure 3.2 is an example of a complete definition of a particular coordinate system. The figure shows that each parameter in the WKT definition is assigned an authority and an identifier. Some GIS standards require that systems know all definitions of certain authorities. In that case, the identifiers alone can be used as specifications. In the case of Figure 3.2, the shorthand notation of the projection definition is simply “epsg:26941”.

```
PROJCS["NAD83 / California zone 1",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980",
        6378137,
        298.257222101,
        AUTHORITY["EPSG","7019"]],
      AUTHORITY["EPSG","6269"]],
    PRIMEM["Greenwich",0,
      AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.01745329251994328,
      AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4269"]],
  PROJECTION["Lambert_Conformal_Conic_2SP"],
  PARAMETER["standard_parallel_1",32],
  PARAMETER["standard_parallel_2",42],
  PARAMETER["latitude_of_origin",37.5],
  PARAMETER["central_meridian",-100],
  PARAMETER["false_easting",2000000],
  PARAMETER["false_northing",500000],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]],
  AUTHORITY["EPSG","26941"]]
```

Figure 3.2: OGC WKT definition, epsg:26941

The definition describes the PROJECTION, and all the details of the associated PARAMETERS. The geographic coordinate system, GEOGCS, is the most complex parameter, which includes the SPHEROID, and DATUM, as well as the prime meridian. Even without extensive knowledge of coordinate systems, the above definition is readable. This particular projection, “California zone 1”, is in the Lambert conformal conic projections, with parameters; standard parallels of 32 and 42, a latitude of origin at 37.5, and central meridian at

-100 degrees longitude, and so on.

By convention images are typically defined in a row/column coordinate system, with the origin in the upper left hand corner of the image. The method of converting these coordinates to a real world coordinate system is by using a *transformation matrix*. This matrix formulation allows for a general affine transformation between the raster and model space. The affine transform consists of six coefficients that map row/column coordinates into the model space, as shown in Figure 3.3(a). Consistent with many other GIS systems, *GeoStreams* model simplifies this transformation and uses only the scale and bias form of the transformation, where s_x and s_y are the scale factors, or length and width of an individual pixel, and b_x and b_y correspond to the location of the upper left hand image pixel in real world coordinates. This is shown in Figure 3.3(b).

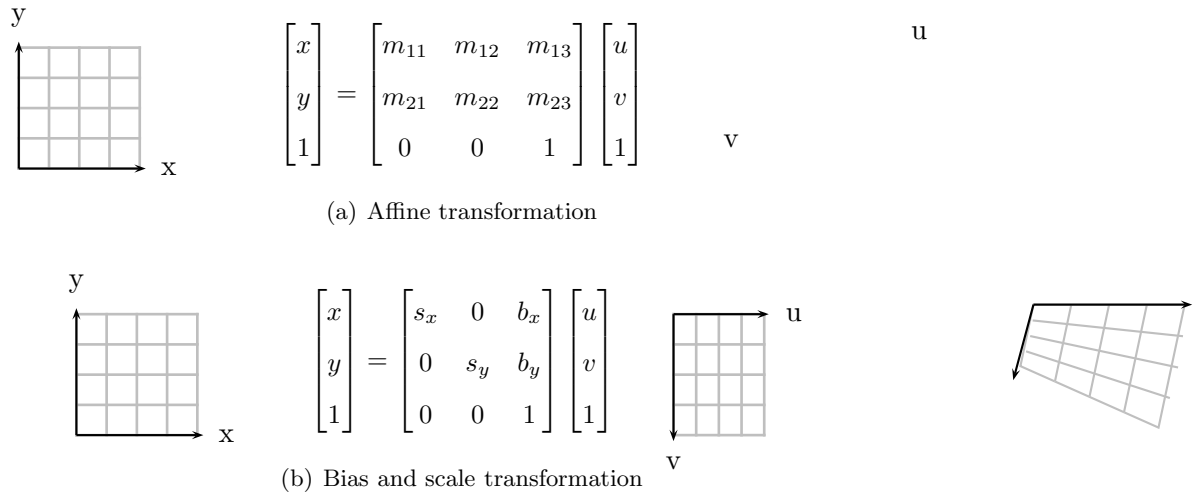


Figure 3.3: Image transformations

The streaming nature of the incoming RSI imparts an ordering on the points in the point set. This ordering allows image operations to proceed on the image stream, since it can be determined whether a particular point will arrive. It allows pixels from different images to be aligned, and it allows for specialized organization of the data. The point sets in the *GeoStreams* system have a well defined ordering associated with the set.

Point set orders depend on the remote sensing instrument. Different instruments collect their data with different methods and impose different structures on the points sets.

For example, some instruments have non-uniform point set structures and can only be ordered in time. Sensors such as Light Detection and Ranging (LIDAR) [?], which take point measurements at discrete time steps, create such point sets. Other instruments have an underlying point lattice for their images. This gridded structure may have many points acquired at the same time, but in this constant time the images have additional structure to define a point set ordering. Figure 3.4 shows an example of non-lattice and lattice point sets associated with different image acquisition schemes.

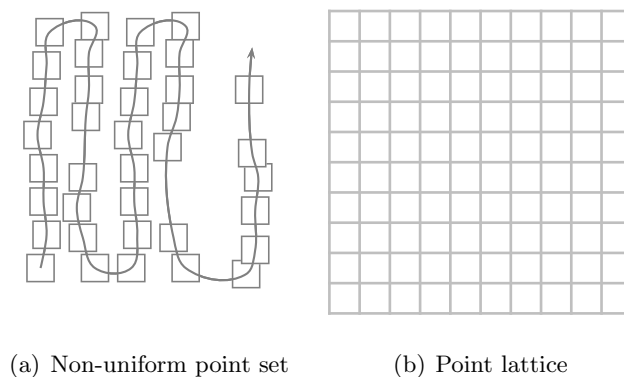


Figure 3.4: Different point set types

Even for sensors delivering images on point lattices, exactly how data arrives with respect to time varies for different RSI data streams. Image data generally arrives as discrete contiguous packets of data. The packets may be individual pixels or organized sets of pixel data, depending on the instrument. Figure 3.5 illustrates some common pixel organizations; image-by-image, row-by-row, and pixel-by-pixel.

Airborne cameras obtain imagery in an image-by-image manner. Entire frames are collected simultaneously and potentially transferred as such. The streaming nature comes from the continuous acquisition of new frames. Many satellite instruments, such as the GOES Imager, obtain RSI data in more of a row-by-row fashion, where strips of image data arrive at a time. Although conceptually the data collected by GOES may be represented as sets of more complete frames, the images are actually received incrementally in a *row-scan order* where pixels are delivered a few lines at a time. Other types of sensors gather data on a pixel-by-pixel basis. These include imaging sensors with large pixel sizes such as the GOES Sounder and some active sounding instruments.

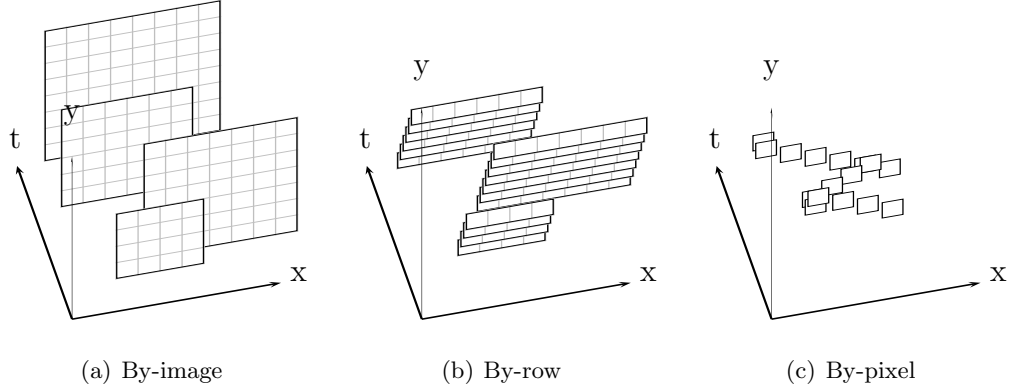


Figure 3.5: Pixel organization for different RSI instruments

Regardless of the way the pixels are streamed from the instrument, within a packet of data the spatial organization of the pixels is well defined. The ordering of the packets themselves is also well defined, and consecutive data packets from an RSI data stream are usually in close spatial proximity. Data from an RSI stream usually arrive only at one or a small number spatial locations for a given instrument. For example, in GOES, the Imager and Sounder instruments provide two separate imaging streams.

The ordering that will be used in the *GeoStreams* system will be row-scan order. Points are first ordered in time as they arrive from the instrument. When multiple points have the same associated time, they are ordered by their row coordinate, and finally by their column coordinate. These coordinates are those defined in the raster space, and not their associated geographic location.

For a 3-d point lattice, \mathbf{X} , consisting of two spatial dimensions, r , c , and time t , **row-scan order** is the order, where for $\mathbf{x}, \mathbf{y} \in \mathbf{X}$:

$$\mathbf{x} \leq \mathbf{y} \equiv \begin{cases} \mathbf{x}_t < \mathbf{y}_t & \text{if } \mathbf{x}_t \neq \mathbf{y}_t \\ \mathbf{x}_r < \mathbf{y}_r & \text{if } \mathbf{x}_t = \mathbf{y}_t \text{ and } \mathbf{x}_r \neq \mathbf{y}_r \\ \mathbf{x}_c \leq \mathbf{y}_c & \text{if } \mathbf{x}_t = \mathbf{y}_t \text{ and } \mathbf{x}_r = \mathbf{y}_r \end{cases}$$

Even with a defined row-scan ordering, the image can be partitioned in various ways as it is input into the DSMS. All partitioning schemes in Figure 3.5 could be arranged

with a row-scan point set ordering. One way an image can be partitioned is using each pixel separately; that is, each tuple is the point and value $(\mathbf{x}, a(\mathbf{x}))$. For non-lattice point-sets, this is often the only possible method. The advantage of partitioning images in the DSMS as individual pixels is that it is the most general solution and can represent any point set. The main disadvantage is that there is a large overhead in representing each individual point in the point set.

Sets of pixels could also be partitioned as complete 2-d sets. In this case, however, in order to maintain this same organization for restrictions, each satisfied restriction would need to create completely new images, and no data could be shared. There would be many redundant pixels replicated in the system. Alternatively, incoming frames could be partitioned in such a way as to maximize the potential to share data. For row scan images, that partition is on each individual row. Each new tuple corresponds to intrinsic discontinuities in the original point lattice.

For row partitioned data, restrictions can be performed by reusing the input image data. The key is that no discontinuities exist inside the row of image data, so that the data needs no reorganization for restrictions. Rows for the new restrictions can use pointers into the existing image data. Figure 3.6 illustrates multiple restrictions using common image data. By retaining the original image data, all matching restrictions can point into the same image data object.

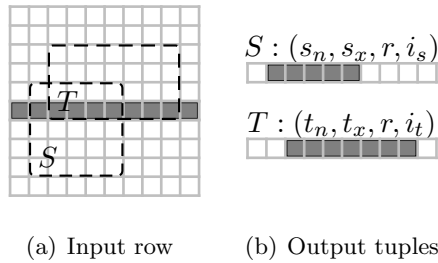


Figure 3.6: Restrictions on row tuples

With row partitions, only one pixel per row needs to be identified in the geographic point set to define the points of the other pixels. This is the method that is used in the *GeoStreams* architecture. Chapter 6 is devoted to a more detailed description of a restriction operator, the Dynamic Cascade Tree (DCT). This operator is specially designed to handle

many combined restrictions, and when the data arrives partitioned by row, the DCT can share data among many operations.

Later, when operators and operator costs are defined, it will be assumed that the data comes in row-scan order and in addition, a single packet, called a *row tuple*, of data that contains a complete single row of pixel values. This corresponds to the actual strategy used in the GOES data stream from the satellite.

The combination of geolocation and point set ordering leads to a definition of the geographic point set.

*A **geographic point set** is an ordered point set along with a projection definition, and a matrix transformation that locates the image in some Earth coordinate space.*

Unless otherwise specified or unclear, the term point set will be used as a shorthand notation for the geographic point sets. In general, there are no restrictions on the shape, orderliness, or values of points within a point set. However images in the *GeoStreams* project are all defined with 3-dimensional point lattices. Practically, this allows for standard geographic location methods and image formats.

3.2.2 Value Sets

Unless otherwise specified, the *Value Set* for an individual image will be the set of integer, \mathbb{Z} or real numbers, \mathbb{R} , corresponding to the radiometric intensity of the surface. One item specific to multi-band RSI data needs clarification. Many RSI instruments transmit images of the same area in multiple bands. Physically, many image formats arrange the data from these bands together, and a reasonable representation of satellite imagery could be a combination of a single point set, with a multi-valued value set corresponding to these multiple bands. The value sets in these cases would be values in \mathbb{Z}^n or \mathbb{R}^n , where n is the number of bands. Although a reasonable and convenient method, this representation is not used here, primarily because for the GOES Imager, each band has its unique associated point set. In the case of the GOES Imager, having multi-valued value sets would require some additional implicit processing to transform images to new point sets, which is not desired.

3.2.3 GeoStream Images

GeoStream images can now be defined based on the above point and value set descriptions.

*A **GeoStream Image**, \mathbf{g} , is an image having a geographic point set, \mathbf{X} . \mathbf{X} may contain a possibly infinite number of points as the temporal dimension of \mathbf{X} can grow indefinitely.*

The temporal dimension has still not been completely defined for the *GeoStreams* stream. This is also a function of the sensor instrument. The GOES instrument sends unique timestamps on each row of data. There is also a timestamp associated with each frame of GOES data. Each frame also has an integer identifier with the same order as the timestamps. The *GeoStreams* system uses this identifier for its time reference. Often, a single frame from the geostream image is of interest.

*A **GeoStream Frame**, \mathbf{i} , of a geostream image, \mathbf{g} , $\mathbf{i} \subset \mathbf{g}$, is an image whose point set has the same temporal value for all points. In image algebra terms, a frame is a restriction on the image \mathbf{g} , $\mathbf{i} = \mathbf{g}|_{\mathbf{Y}}$, where $\mathbf{Y} = \{\mathbf{x} : \mathbf{x} \in \text{domain}(\mathbf{g}) \text{ and } \mathbf{x}_t = t\}$ for some time t . A convenient notation for a geostream frame is $\mathbf{g}|_{@t}$, where t is the time of interest.*

When discussing implementation issues, the atomic tuple in the *GeoStreams* system will be a single row of data. Rows will be shown to have some nice properties for this purpose.

*A **GeoStream Row**, \mathbf{r} , of a geostream image, \mathbf{g} , $\mathbf{r} \subset \mathbf{g}$, is an image whose point set has the same temporal and row values for all points. In image algebra terms, a row is a restriction on the image \mathbf{g} , $\mathbf{i} = \mathbf{g}|_{\mathbf{X}}$, where $\mathbf{X} = \{\mathbf{x} : \mathbf{x} \in \text{domain}(\mathbf{g}), \mathbf{x}_t = t, \mathbf{x}_r = r\}$ for some r and t .*

As frames and rows are simply restrictions on an image, they are images as well and all image operations apply to these entities.

A reference schema follows directly from the definitions of *GeoStreams* images, frames, and rows. The reference schema describes the fundamental aspects of the GOES Imager data stream, and any images that are created in the stream. All data from a single frame share a common time, and frame identifiers are used as values in a temporal

domain when accessing image data. Newly created images inherit the geographic point set information, regardless of whether the operation is applied to an image, frame, or row.

Figure 3.7 shows the reference schema that is used to describe the *GeoStreams* image streams. There are separate *GeoStreams* images for the five incoming spectral channels from the GOES stream and also separate streams for each created images. All images have definitions for frames and rows.

In addition to the images, a relation containing separate information regarding the individual frames is also included. Information in this relation includes the actual time of the acquisition and information about the constraints on domain of that incoming frame. Some information regarding the overall size of the individual frames is also included.

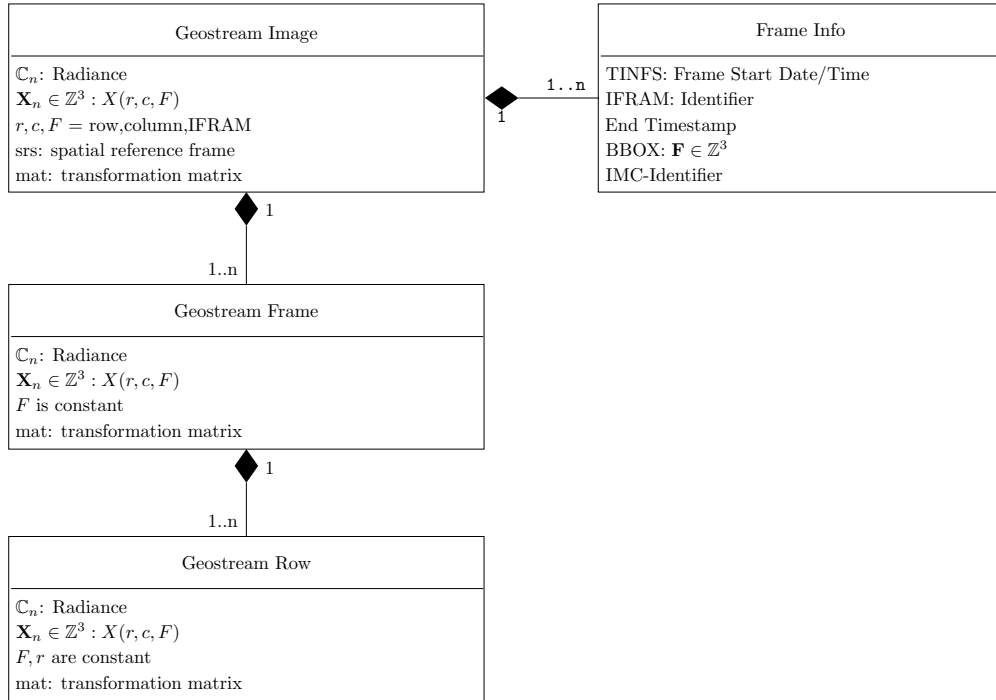


Figure 3.7: Reference schema for GOES data

3.2.4 Induced Operator Modification

Allowing point sets in a streaming system to have an infinite number of points can lead to some complications for some image algebra operators, in particular the induced

operations, such as $\mathbf{a} \gamma \mathbf{b}$. In image algebra, these induced binary operations are defined only when $\mathbf{a}, \mathbf{b} \in \mathbb{V}^{\mathbf{X}}$. For streaming data, the point set is not known in advance and a processing system could not verify the point sets were indeed the same. The basic induced image operations are therefore inconvenient for a DSMS. Instead, a modified induced operation is introduced to allow for greater latitude in query execution. The idea is to perform an operation on an image when permitted, in this case when both images share a point.

To compare with the traditional image algebra formulation, a new notation is temporarily introduced below. For two images, $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}, \mathbf{b} \in \mathbb{V}^{\mathbf{Y}}$ and a binary operation γ on \mathbb{V} ,

$$\begin{aligned} \mathbf{a} \gamma_{\cap} \mathbf{b} &= (\mathbf{a} \gamma \mathbf{b}) \cap (\mathbf{X} \cap \mathbf{Y}) \times \mathbb{V} \quad \text{or} \\ &= \{(\mathbf{x}, a(\mathbf{x}) \gamma b(\mathbf{x})) : \mathbf{x} \in \mathbf{X} \cap \mathbf{Y}\} \end{aligned}$$

Figure 3.8 shows the results of this induced operation on two images with different point sets.

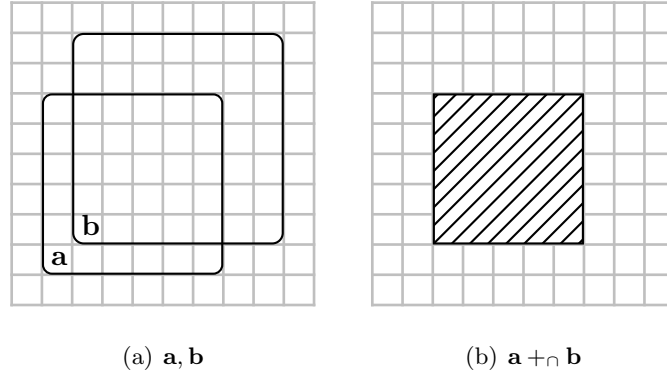


Figure 3.8: Induced operation redefinition. $\mathbf{a} + \mathbf{b}$ is not defined, where $\mathbf{a} +_{\cap} \mathbf{b}$ is shown

An important feature of this operation that will be used when optimizing queries is that restrictions can be distributed across this form of induced operation. The original induced operation disallows distributive rules in relation to restrictions. For example, $\mathbf{a}|_{\mathbf{A}} + \mathbf{b}|_{\mathbf{A}} \neq (\mathbf{a} + \mathbf{b})|_{\mathbf{A}}$ since the left hand side holds for $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}, \mathbf{b} \in \mathbb{V}^{\mathbf{Y}}$ while the right side generally does not. The redefined operation $\mathbf{a} +_{\cap} \mathbf{b}$, however, does allow restrictions to be distributed.

In formulations below, the modified intersection operation will be the default in-

duced operation on images used in the system. These operator definitions also have the nice property that arbitrary \mathbb{V} -valued images can be combined. Query operator rewrite rules, further described in Section 3.3.3, will take advantage of this formulation.

It is ordering of the geographic point sets that allows images to be combined in induced operations or other operations. Because point sets are ordered, operations are able to determine whether a point belongs in the union of two point sets by investigating the current point in the streams. As some new point from an image stream, **a**, arrives in one stream, any points in another stream ordered before this point can be removed as not belonging to the union of the point sets, as that point is guaranteed not to be in stream **a**.

Neither image restrictions nor extensions are affected by the indeterminate nature of streaming point sets. Though the entire point set for an image may not be known, since image arrives in an order, enough local information can be used to determine whether points in an image are involved in restriction and extension operations. As will be seen in Chapter ??, compositions or spatial transforms might require that more than the current point in an image stream needs to be cached to complete these operations, but they do not require knowledge of the entire point set and are therefore also usable without modification.

3.3 Query Model

Queries in the *GeoStreams* system will be based on a particular image algebra expression. An individual query will follow an image algebra template with various parameters specified for the query.

The specific query declaration uses a formulation inspired by WMS specification. Users specify a specific data product, coordinate system, spatial ROI, temporal extents, and pixel size. The WMS queries have equivalent image algebra formulations which are shown for the reference queries.

Query optimization will be discussed in detail in Chapter 4, but an important aspect of optimization is taking advantage of rules for rewriting a given expression to an equivalent form. Some algebraic properties of image algebra are discussed in Section 3.3.3. Image restrictions are once again a focus of Section 3.3.4 in combination with extensions,

where a number of potential image results from queries are discussed and formulated.

3.3.1 Reference Queries

With a data model in place, queries are represented as simple expressions. Table 3.1 revisits the example queries of Section 1.4 (page 11) and writes those queries as image algebra expressions. As can be seen, the expressions for these simple queries are quite compact.

Most of the queries follow a particular format, being a combination of induced operations, a neighborhood operation, a restriction, and finally a spatial transform. The examples above use a rather loose notation in defining the parameters of each query, the restriction ROIs, and the different spatial transforms used. One problem with the current image algebra formulation is there is not a simple standard method to represent these aspects of the queries. The WMS specification is used to complete these query specifications, with a method of describing these parameters.

3.3.2 WMS Queries

The goal of the *GeoStreams* system is to provide users a convenient interface to RSI data, including an easy to use query interface. Currently, the most common network application interface to general purpose GIS data is through the OGC Web Map Services (WMS) [?]. The *GeoStreams* project adapts the WMS specification as a query specification. The WMS implementation specification provides an interface to support the creation and display of registered views of information that come simultaneously from multiple remote sources. The implementation allows a client to access maps from any WMS server. Clients can also interrogate servers to determine what types of data products are available from the server. The WMS defines a method to convey information about what types of RSI a service can provide and a method to convey a request for a particular RSI data product. WMS queries are simple URL requests. An example query looks like the following:

```
http://geostreams.org/wms?VERSION=1.3.0&REQUEST=GetMap
&LAYERS=NDVI&CRS=epsg:4326&BBOX=-125,19,-65,49&WIDTH=5000&HEIGHT=5000
```

Table 3.1: Reference GOES queries

Q	Image Algebra	Description
A	$C1 _{MX}$	Query A represents a user requesting Channel 1, (visible) data over a ROI covering Mexico. There are no time constraints on the query. The resolution corresponds to the default GOES data, as does the requested projection
B	$C1 \oplus N_{4 \times 4} \circ LL _{NA}$	Query B is another query on the visible channel, however in this instance, the user wants the data at a coarser resolution, and projected to a longitude-latitude, (equidistant cylindrical) grid.
C	$\frac{C2 \oplus N_{4 \times 4} - C1 \oplus N_{4 \times 4}}{C1 \oplus N_{4 \times 4} + C2 \oplus N_{4 \times 4}} _{NA}$	This query represents a user requesting a product, in this case NDVI, which is a normalized difference index using two input channels. This query is limited to North America.
D	$\frac{C2 \oplus N_{8 \times 8} - C1 \oplus N_{8 \times 8}}{C1 \oplus N_{8 \times 8} + C2 \oplus N_{8 \times 8}} \circ LL _{HEMI}$	This query represents an NDVI request at a coarse scale and covering the entire northern hemisphere. The requested image is also projected to longitude-latitude.
E	$C1 _{NA} _{@t}$	This query requests visible GOES data, but only the image that occurs at some time, t , everyday.
F	$C1 \circ UTM _{CA} _{<d}$	This is similar to the query E, but also specifies an ending date, d , when the query will end.
G	$C1 _G$	This query asks for image data at a single point.
H	$C1 _{C1(x)>v}$	Channel 1, but only in the image where the <i>pixel values</i> are greater than some amount v .

A selected set of the WMS parameters are described in Table 3.2. The key parameters are: **LAYERS**, which specifies the desired product; **CRS**, which specifies the output coordinate reference system; **BBOX**, which specifies the bounding box of the query; and **WIDTH** and **HEIGHT**, which combined, specify the resolution of the image. **FORMAT** is a comma limited set of graphic file identifiers, for example, GeoTIFF. Currently, no common existing format supports a geostream image as output.

Table 3.2: Selected WMS *GetMap* parameters

Parameter	Description
LAYERS=layer	one or more map layers.
CRS=namespace:identifier	Coordinate reference system.
BBOX=minx,miny,maxx,maxy	Bounding box corners in CRS units.
WIDTH= <i>c</i>	Width is <i>c</i> pixels.
HEIGHT= <i>r</i>	Height is <i>r</i> pixels.
FORMAT=output_format	Output format of map. (STRING)
TRANSPARENT	Background transparency of map. (TRUE or FALSE)
BGCOLOR=HEX	Hexadecimal color for the background.
TIME=time	Time value of layer desired.

The service description component of the WMS interface is the *GetCapabilities* command. A client issues a *GetCapabilities* request to the server, and the server responds with a description of the service. The description includes what layers are available from the service and their spatial extents, what particular spatial reference systems are available, what output formats, and a number of other parameters. Using the WMS implementation allows the server to limit the types of queries that are allowed by users of the system.

By limiting the requests to a finite number of specific products the server is able to limit the types of induced operations that the user can specify. Using the WMS implementation the user is only able to specify particular layers defined by the system. The server builds a limited subset of allowed induced operations by creating specialized layers that encompass these operations.

In a similar manner, the available formats and coordinate reference systems can also be specified as a limited set of choices to the user. The capabilities of a server can be thought of as a template on the allowed queries to the system. Figure 3.9 shows such a template designed with the reference queries from Table 1.1 (page 11). Here, six different layers, including all image channels and the Normalized Difference Vegetation Index (NDVI) formula are allowed. Also allowed are three coordinate system transformations; the original GOES projection $\circ G$, various Universal Transverse Mercator (UTM) zones, $\circ UTM$, and longitude latitude, $\circ LL$.

$$\text{http://...LAYERS} = \left\{ \begin{array}{c} C1 \\ C2 \\ C3 \\ C4 \\ C5 \\ NDVI \end{array} \right\} \&\text{CRS} = \left\{ \begin{array}{c} LL \\ UTM \\ GOES \end{array} \right\}$$

$$\&\text{WIDTH} = any \&\text{HEIGHT} = any \&\text{BBOX} = S, E, W, N$$

Figure 3.9: WMS queries

This simple interface does not allow for a sophisticated set of user queries, but it does investigate the most basic requirements of serving many spatial restrictions and geometric transformations to many clients. The interface allows users to specify specific data products, coordinate systems, and spatial extents. Temporal restrictions can also be identified. Queries like the reference queries in Section 1.4 can be specified, as long as the values of the resultant query are identified as a product in the server. The WMS specification further simplifies query formulation by standardizing and simplifying both spatial transforms and restrictions to a limited but well-defined subset. In general, the WMS specification limits queries to the form, $\mathbf{a} \oplus N \circ f|_X$, where \mathbf{a} , N , f , and X are specified in a simple standard way.

3.3.3 Rewrite Rules for Image Expressions

An important part of the query model is that it lends itself to optimization techniques by the system. When optimizing a query, it is necessary to rewrite some expressions into equivalent forms that may improve performance. In this section some of the most important rules involved in rewriting expressions will be discussed.

There are a number of rules regarding the restriction operation. First, a number of identity relations on images are defined:

$$\mathbf{a} \equiv \mathbf{a}|_{\text{domain}(\mathbf{a})}$$

$$\mathbf{a} \equiv \mathbf{a}||_{\text{range}(\mathbf{a})}$$

$$\mathbf{a} \equiv \mathbf{a}|\mathbf{a}$$

These identity relations simply show that there are obvious restrictions that do not change an image. In particular, the restriction identity $\mathbf{a} = \mathbf{a}|_{\text{domain}(\mathbf{a})}$ is useful, since restrictions have other properties, like the ordering of multiple restrictions, which prove useful in optimizing queries.

$$\begin{aligned} \mathbf{a}|\mathbf{Y}|\mathbf{X} &= (\mathbf{a} \cap (\mathbf{Y} \times \mathbb{V}))|\mathbf{X} \\ &= (\mathbf{a} \cap (\mathbf{Y} \times \mathbb{V})) \cap (\mathbf{X} \times \mathbb{V}) \\ &= (\mathbf{a} \cap (\mathbf{Y} \cap \mathbf{X}) \times \mathbb{V}) \\ &= \mathbf{a}_{\mathbf{X} \cap \mathbf{Y}} \quad \text{or} \\ &= \mathbf{a}|\mathbf{X}|\mathbf{Y} \end{aligned}$$

One result of the above properties is that other restriction operations can safely be placed in expressions if their point sets are inclusive of the other restrictions, that is $\mathbf{a}|\mathbf{Y}|\mathbf{X} = \mathbf{a}|\mathbf{X}$ if $\mathbf{X} \subset \mathbf{Y}$. It also implies that restrictions can be added into expressions and distributed through other operations as seen next.

The restriction operation is associative and right distributive over induced operations with the induced function definition introduced in Section 3.2.4. For images $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$

and $\mathbf{b} \in \mathbb{V}^{\mathbf{Y}}$, using the previously defined rules on restrictions, this can be shown.

$$\begin{aligned}
 (\mathbf{a} \gamma \mathbf{b})|_{\mathbf{Z}} &= ((\mathbf{a} \gamma \mathbf{b}) \cap (\mathbf{X} \cap \mathbf{Y}) \times \mathbb{V}) \cap \mathbf{Z} \times \mathbb{V} \\
 &= ((\mathbf{a} \gamma \mathbf{b}) \cap (\mathbf{Z} \cap \mathbf{X} \cap \mathbf{Y}) \times \mathbb{V}) \\
 &= \mathbf{a}|_{\mathbf{X} \cap \mathbf{Z}} \gamma \mathbf{b}|_{\mathbf{Y} \cap \mathbf{Z}} \\
 &= (\mathbf{a}|_{\mathbf{Z}} \gamma \mathbf{b})|_{\mathbf{Z}}
 \end{aligned}$$

Spatial transformations also have an identity function with respect to a restriction.

For an image, $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$, and function $f : \mathbf{Y} \rightarrow \mathbf{X}$,

$$\mathbf{a} \circ f = \mathbf{a}|_{\text{range}(f)} \circ f$$

This simply says that a restriction on \mathbf{a} over the range of the function f does not affect the results of the composition, since those are the only pixels used in the composition. This is shown pictorially in Figure 3.10.

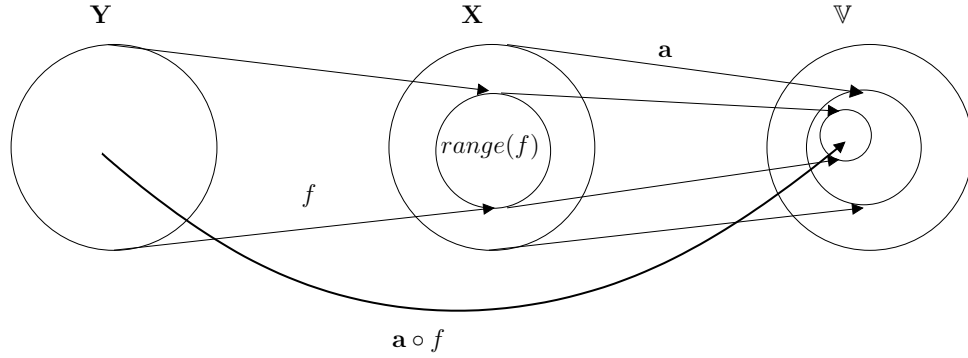


Figure 3.10: Diagram showing $\mathbf{a} \circ f = \mathbf{a}|_{\text{range}(f)} \circ f$

Restrictions are not left distributive over spatial transforms. But transforms of the point set can be distributed over a composition. For an image, $\mathbf{a} \in \mathbb{V}^{\mathbf{X}}$, and $f : \mathbf{Y} \rightarrow \mathbf{X}$, if $\mathbf{W} \subseteq \mathbf{Y}$, then

$$\begin{aligned}
 (\mathbf{a} \circ f)|_{\mathbf{W}} &= \mathbf{a} \circ (f|_{\mathbf{W}}) \\
 &= (\mathbf{a}|_{\text{range}(f|_{\mathbf{W}})} \circ f)|_{\mathbf{W}}
 \end{aligned}$$

Figure 3.11 shows this is very similar to the previous result.

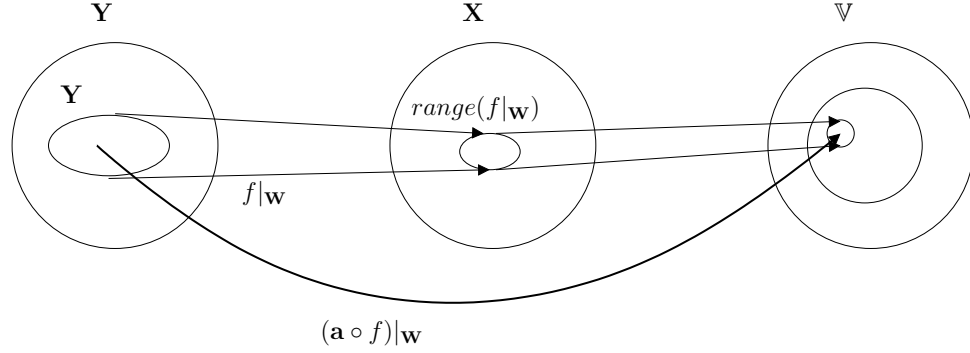


Figure 3.11: Diagram showing $(\mathbf{a} \circ f)|_{\mathbf{W}} = (\mathbf{a}|_{\text{range}(\mathbf{W})} \circ f)|_{\mathbf{W}}$

The combination of a neighborhood operation with a restriction is similar to the above combination of restriction and composition. The difference being that a point set $\mathbf{Z} \subseteq \mathbf{X}$ needs to include all points used in the creation of each of the aggregation points in the restriction on $\mathbf{W} \subseteq \mathbf{Y}$, where, as before, \mathbf{n} is an image where values of \mathbf{n} are point sets, $n(\mathbf{y}) \subseteq \mathbf{X}$.

$$\begin{aligned} (\mathbf{a} \oplus \mathbf{n})|_{\mathbf{W}} &= \{(\mathbf{y}, \sum_{i \in n(\mathbf{y})} a(\mathbf{i})) : \mathbf{y} \in \mathbf{Y}\}|_{\mathbf{W}} \\ &= \{(\mathbf{y}, \sum_{i \in n(\mathbf{y})} a(\mathbf{i})) : \mathbf{y} \in (\mathbf{Y} \cap \mathbf{W})\} \end{aligned}$$

Note that the only values used in \mathbf{a} , are those included in the summations from values of $\mathbf{n}|_{\mathbf{W}}$. These points can be collected in a new pointset $\mathbf{Z} = \bigcap \{x : x \in n(\mathbf{y}), \mathbf{y} \in \mathbf{W}\}$. Then, as also shown in Figure 3.12,

$$(\mathbf{a} \oplus \mathbf{n})|_{\mathbf{W}} = (\mathbf{a}|_{\mathbf{Z}} \oplus \mathbf{n})|_{\mathbf{W}}$$

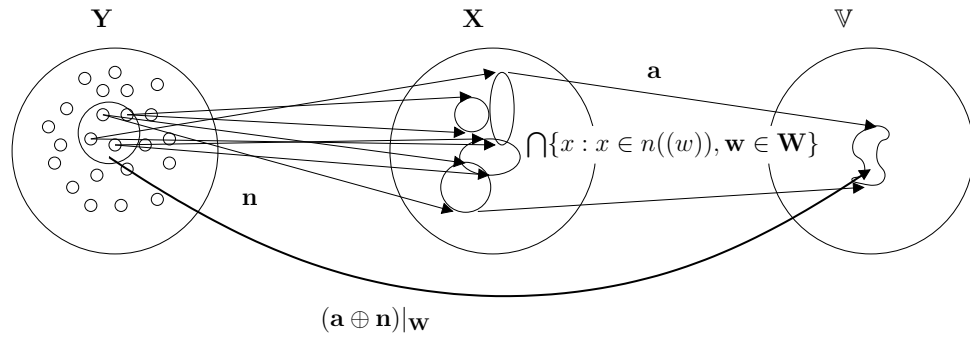


Figure 3.12: Diagram showing $(\mathbf{a} \oplus \mathbf{n})|_{\mathbf{W}} = (\mathbf{a}|_{\mathbf{Z}} \oplus \mathbf{n})|_{\mathbf{W}}$ where $\mathbf{Z} = \bigcap \{x : x \in n(\mathbf{y}), \mathbf{y} \in \mathbf{W}\}$

These rewriting rules will be used in Chapter 4, where cost models are introduced to estimate the benefits of certain query rewriting strategies.

3.3.4 Query Restriction Types

One problem that remains with the query model has to do with how to fill in missing points in a query when not all points exist in the image stream. For example, consider three individual image frames that are being restricted by a query. The three frames are shown in Figure 3.13 labeled \mathbf{q} , \mathbf{r} , and \mathbf{s} . The ROI for the query is labeled \mathbf{Q} . Figure 3.13 shows the restriction that has been discussed above. This returns pixels with points that are in both the query point set and the image point set. This means that the result point set for the query can change from frame to frame.

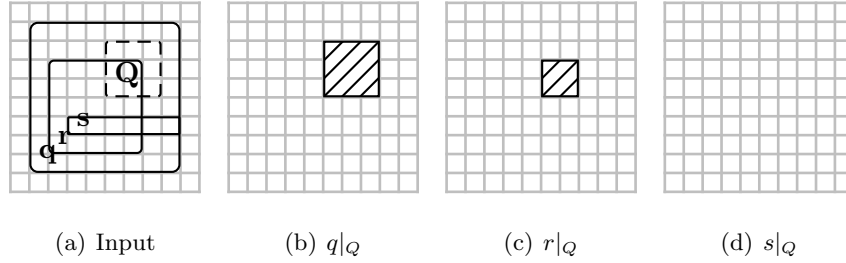


Figure 3.13: Example image restriction

There are other potential ways to interpret the WMS parametrized query. For example, the system could return results only from frames that completely contain the query ROI. Figure 3.14 shows this example by including a test verifying all points in the query pointset are also in the incoming image. Such a test is not currently represented by the query model.

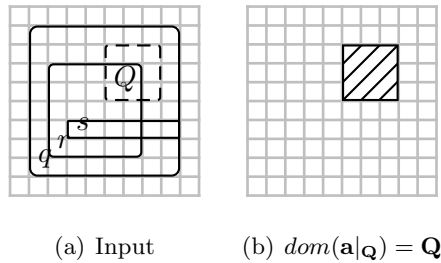


Figure 3.14: Complete image subset

Figure 3.15 shows how queries could use an image extension operation to compel each frame to have a common point set. In this example, \mathbf{o} is taken as an image that covers all of \mathbb{Z}^2 , with a null value for all points. Unlike the domain restrictions, the extension of a point set onto \mathbf{o} will always return an image with point set \mathbf{Q} . This is an important modification because images that share a common point set can be combined to form higher dimensional grids. For example, $g|_{\mathbf{Q}}^{\mathbf{o}}$ would return a set of images with the same point set.

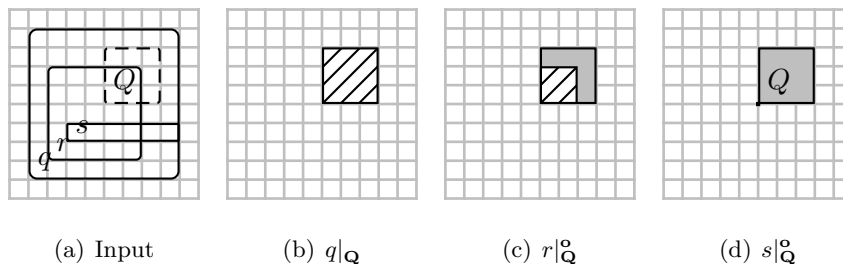


Figure 3.15: Image extension example

It would be perfectly reasonable to assume that the queries represented by the WMS syntax contain an implicit extension operator. In fact, the standard contains two parameters `BGCOLOR` and `TRANSPARENT` that can allow the user to specify an extension to the query expression. Currently, however, the simplest, non-extension restrictions are adopted for all queries in the system.

3.4 Related Work

To provide a transformation, point sets need parameters that allow translation between the raster and model coordinate systems. There have been a number of methods to describe this transformation [?, ?, ?, ?]. There are also reviews of various scenarios for transformation from image to space coordinate positions. The OGC abstract specification contains a special topic on Earth Imagery containing descriptions of geometry models [?], which describes image transformations in more detail. The OGC also publishes a introduction to image geometry models [?]. The implementation used above follows that of the Geotiff standard [?] and also closely follows the representation of the OGC GML standard [?, ?, ?].

In terms of a query language, besides the OGC WMS implementation, there exists a more expansive interface specification, the Web Coverage Service (WCS) [?]. This specification extends the WMS interface to request other image formats rather than the more picture oriented WMS specification. The WCS offers more control on the format of the retrieved images, and some more sophistication in the queries, including more specificity on multi-band images and requesting explicit interpolation routines. The above queries could be formulated equally well with the WCS. One added bonus being that queries could explicitly define an interpolation strategy to use for spatial transforms. In a working environment the WCS is probably a more valuable interface. The WMS is maintained here primarily because of its simplicity.

Gertz et al. [?] also describe the *GeoStreams* project data and query model with a similar emphasis on its relationship to image algebra. Güting has introduced the ROSE algebra, another algebra defined on a point lattice [?] focused more on vector GIS data. Marathe [?] describes processing techniques used on RSI.

Chapter 4

Query Processing

After having defined an image algebra based data model and query semantics for the *GeoStreams* model, and having defined a core set of operators for the system, this chapter is concerned with developing optimization strategies for the creation of a Query Execution Plan (QEP). The overall objective is to determine an optimal, or at least good global QEP that answers all queries in the DSMS. Both single query optimizations and multi-query optimizations will be discussed.

Because of the nature of the system, where multiple, perhaps completely disjoint, query plans are being executed, the efficiency of the system is dependent both on the strategy of execution and the order of component execution of that strategy as well. This is shown in the difference between the Query Execution Graph (QEG) and the QEP. These two terms are defined below:

A Query Execution Graph (QEG), is an acyclic directed graph $QEG = (V, E)$ that represents the operations to execute all active queries within the Data Stream Management System (DSMS). The nodes, V , correspond to individual operators within the system, and the edges E , correspond to data paths from one operator to another. Operator costs are associated with nodes and edges carry the defined point sets and data between operators.

A Query Execution Plan (QEP) is a combination of a QEG, plus an ordering on the nodes of V , which satisfies the partial ordering imposed by a topological sort of the QEG.

Defining efficiency requires a definition of a cost model for the QEP. Section 4.1 defines a few potential cost models. A graphical representation of these costs will allow for all cost models to be visualized. Understanding the costs of different optimizations requires some estimates of the costs of individual operators. Operator costs are also introduced in this section. Sometimes the ordering in the QEP can affect the total cost of a QEG. This is the case only when memory constraints are included when calculating the cost of a particular plan. One result from the examination of a number of cost models is that optimizations are often complementary for a number of models. A cost model based on execution time is chosen as a representative optimization model, and used in subsequent sections.

Section 4.2 introduces some optimizations for single queries. These optimizations primarily deal with rewriting a query to be more efficient in its computation. Within a single query, optimizations will be concerned with the ordering of the operators. Just as in relational algebra, the image algebra formulation describes what output images are formed, but these formulations can be rewritten to produce multiple paths that perform the operation. Section 4.3 builds on these optimizations to perform query optimizations considering all queries in the system. Where single query optimization develops operator ordering, multi-query optimization adds additional savings by sharing results between queries to develop multi-query QEGs.

4.1 Cost Models

In traditional databases, the usual cost model for query processing algorithms are based on access to secondary storage, typically counting the number of pages accessed for a particular query. In the case of the DSMS, however, processing is assumed to occur in main memory. Most in-memory cost models attempt to minimize the total computation cost associated with a QEG, where the computation cost is usually simplified to a single operation, counting the number of comparisons in a join, for example. This is a natural cost model for a DSMS as well, since the interest is in pushing data through the system fast enough to keep up with the incoming stream. Rather than counting a single operation, for

images, the computation cost might be the average computation performed on a pixel.

There are other considerations as well. Computation cost does not account for the potential limit on the total memory available to the system. Since main memory is limited, if total size of the processing system gets too large, then memory will need to be swapped to secondary storage, significantly modifying the costs of executing the QEG. Therefore, other cost models might consider cost models that monitor and limit the amount of memory used by the system.

Four potential cost models for evaluating different QEGs are described in Table 4.1. Each cost model has a potential for validity in the system. While developing an implementation of the *GeoStreams* DSMS, the emphasis will be on minimizing execution time. In this chapter, examples will compare QEGs using different cost models. Optimization techniques are broadly applicable between models, and minimizing costs in any one model typically lowers the cost of the other formulations.

Table 4.1: Cost models

Model	Description
<i>Execution</i>	Minimize the total CPU time of the query
<i>Creation</i>	Minimize number of new tuples created, or equivalently the total amount of memory used
<i>Max-Size</i>	Minimize the maximum instantaneous memory usage
<i>Size-Time</i>	Minimize the memory usage of tuples in the system integrated over the time they are required in the QEP.

For the multi-query models of Section 4.3, the QEG alone does not contain enough information to determine the cost of operations and the ordering defined by QEP is also required. However, some costs can be calculated from the QEG alone. For example, the *Execution* cost model can be calculated with a Depth First Search (DFS) through the QEG. For each node, the cost of the operation can be calculated using only the input point set, which can define the size of the incoming data, and the node cost. Summing all node costs gives the total execution cost for the QEG. The *Size-Time* cost is most effected by the

ordering of operations within the QEG. In this section, a specific QEP is assumed, the operator ordering is shown explicitly in the graphical representation of the query plan.

To develop a complete cost model, costs for the individual operators need to be defined. Table 4.2 shows some basic costs for individual operators. Again, it is assumed that an image arrives in row-scan order. The complexity of the operators are given in terms of m , the number of rows sent to the operator. Because the constants, C for processing, and S for size, can vary significantly and are important in cost models, they should not be ignored in these formulations, and constants are given as functions of n , the number of pixels per row tuple, and q , the numbers of queries in the system. h indicates the size of the window in a neighborhood operation. Also, m' indicates the number of resultant rows in the new point set from a transformation. For repeating frames of data $m' = km$, so this is the same order as m , and the extra notation is given mostly as a hint to the change in point sets.

Table 4.2: Operator costs

Operator	CPU	Size	Selectivity	Create
$ x$	$O(C m)$	$O(S)$	$\leq m$	0
$a \gamma b$	$O(C_\gamma(n)m)$	$O(S)$	m	m
$\circ f$	$O(C_{of}(n)m)$	$O(S_{of}(n^2)m')$	m	m'
$\oplus N_{h \times h}$	$O(C_\times(hn))$	$O(S)$	m	m/h
Multi-query				
$ x$ (DCT)	$O(\approx C(\lg q)m)$	$O(S_{DCT}(q))$	$\leq mq$	0

h : neighborhood window

m : incoming rows

$m' \propto m$: new rows

n : pixels per row

q : numbers of queries

The *Cost* column identifies the Central Processing Unit (CPU) cost of each operation. *Size* indicates the running size of the operator. The *Selectivity* column defines

whether the operation has any selectivity. An important consideration for image algebra queries is that all operators with the exception of the spatial restriction operator require creation of new row tuple data into the data stream. The cost of creating new row tuples has an additional overhead and will be modeled explicitly in the cost models. Operator costs are a function of implementation, and the costs below are further discussed in Chapter ??.

Except for the restriction operator, which can operate on complete rows, each operation requires access at the individual pixel level, and therefore requires times proportional to the number of pixels, n , in the row tuple. The operator size for transformations is constant over many rows, but can be quite large, in relation to other operator sizes, as much as a single image frame.

The table also contains a multi-query restriction operator. While the first restriction operator performs a restriction on a single query, the multi-query restriction operator simultaneously performs restrictions on many queries. Use of this operator is deferred until Section 4.3. As with the other entries in the table, the values are given without proof. However, where the cost of the single operators are fairly self explanatory, the multi-query restriction is not. These values are dependent on the implementation, and Chapter 6, which defines the implementation used for the *GeoStreams* project, describes in detail these costs.

These individual operator costs can be combined together to determine the overall cost of a query. In this example and throughout this paper, serial execution of the operators is assumed. Although parallel processing is very viable for image processing, developing cost models is considerably complicated and obfuscates the image algebra operator costs. Figure 4.1 shows a pictorial representation of the QEG for a simplification of query **C** (Tables 1.1 and 3.1). These figures are meant to give a feeling of the CPU and memory usage of a single query in the *GeoStreams* system.

The x -axis represents time, while the y -axis represents memory usage. Individual blocks show the lifetime of each row tuple within the DSMS. The blocks are shown with two shades, the hatched area represents the cost of creating a new row tuple. The vertical bars distinguish the operators that are being executed. Restriction operations only direct row tuples and have no data creation costs.

In this figure, the total execution time of the system is equivalent to the length of

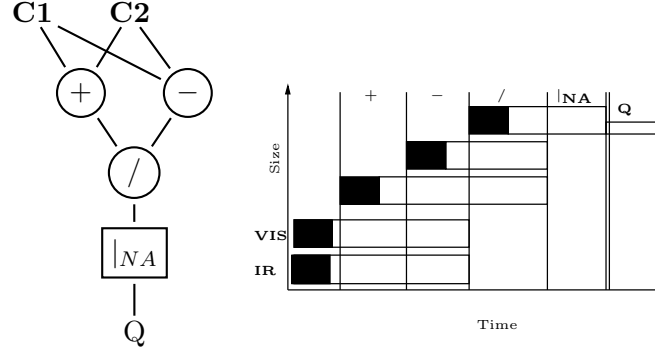


Figure 4.1: Figure representation of QEG for $\frac{C2-C1}{C1+C2} |_{NA}$. Dark areas identify tuple creation and light areas identify tuple creation.

the x -axis. The total cost can be expressed as $3C_M + 3nC_\gamma + C_\perp$, where C_M represents row creation costs, C_γ is average cost for per-pixel operations, n is the number of pixels in the row and C_\perp is the cost for a restriction.

One cost associated with memory use is to measure the total amount of new row tuples created which is measured by the y -axis as shown in the figure. Since every binary operation creates data, the cost of the operation would be $3C_M$. This cost could also be reasonably modeled in the previous execution cost model. Since creation costs are explicitly included, minimizing tuple creation can use the same scheme, assigning zero costs to non-creation operations. This would be a valid simplification if creation costs dominate.

Two other cost models are apparent. First, if memory is the limiting factor, then a QEG might aim to limit the total amount of memory used at any given time. In the figure, the instantaneous use of memory is the total height of all bars at any given time. The maximum value in this instance is $4C_M$, during the second operation.

Finally, a cost model might attempt to limit the total amount of memory used over the lifetime of the QEG. This is the product of each memory allocation over its lifetime in the QEG. In Figure 4.1, this corresponds to the sum of the areas of each bar in the QEG. Adding all bars in this example shows this cost to be $10C_M + 10nC_\gamma + C_\perp$.

4.2 Single Query Optimizations

Evaluating a query using any of the previous cost models depends on the method used to process the query. Rewriting a query expression can result in a more efficient QEG. These ideas can be demonstrated with the previous simplification of Query **C**, with the image algebra formulation $\frac{C2-C1}{C1+C2}|_{NA}$. This can be realized with a number of different QEGs, first by using the distributive property of restrictions over induced operations shown in Section 3.3.3 (page 47), to move the restrictions to the input channel streams.

$$\begin{aligned} \frac{C2 - C1}{C1 + C2}|_{NA} &= \frac{(C2 - C1)|_{NA}}{(C1 + C2)|_{NA}} \\ &= \frac{C2|_{NA} - C1|_{NA}}{C1|_{NA} + C2|_{NA}} \end{aligned}$$

Figures 4.2 and 4.3 show two other formulations. One plan simply merges all induced operations (Figure 4.2), the other (Figure 4.3) distributes restrictions over the inputs, and merges the induced operations. The various costs for the model described in Section 4.1 are summarized in Table 4.3 for all three formulations. All variables have been introduced except for s , which is the selectivity of the restriction operator.

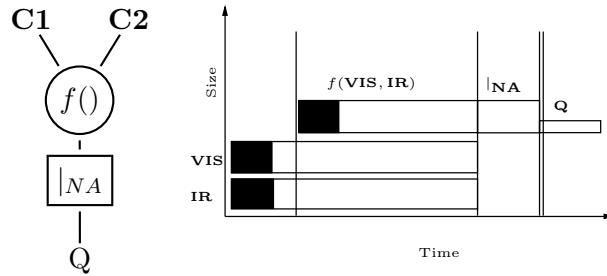


Figure 4.2: QEG for query **C** with merged algebraic operations

Table 4.3: Query cost summary for $\frac{C2-C1}{C1+C2}|_{NA}$

Plan	Execution	Creation	Max Size	Size-Time
Figure 4.1	$3C_M + 3nC_\gamma + C_l$	3	4	$10C_M + 10nC_\gamma + C_l$
Figure 4.2	$C_M + 3nC_\gamma + C_l$	1	3	$3C_M + 3nC_\gamma + C_l$
Figure 4.3	$sC_M + 3snC_\gamma + 2C_l$	s	$\max(2, 3s)$	$3sC_M + 3snC_\gamma + (3 + s)C_l$

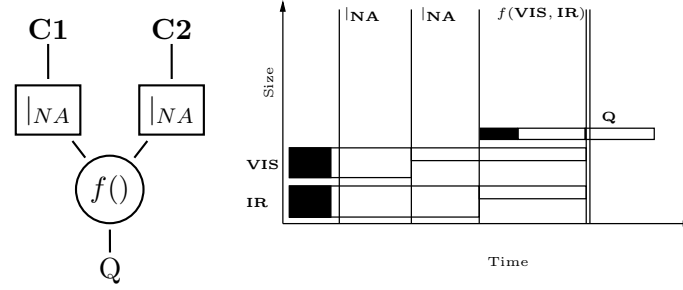


Figure 4.3: QEG for query **C** with distributed restrictions

Determining each individual cost is fairly simple to do given the graphical representation of the query plans. For the Execution cost, the time, or x-axis, is calculated for the plan. The costs measured between each set of bars is based on the operation, which is identified at the top of the graph. In addition, the tuple creation costs, identified by the dark regions in the graphs, also need to be added in the cost. The creation costs of the original tuples is not included in the cost. Using Figure 4.1 as an example, there are three new tuples created, $3C_M$. There are also three induced operations over n pixels per row, $3nC_\gamma$, and one final restriction, C_\perp .

The *Creation* cost simply counts the new tuples created. The *Max-Size* cost, counts the maximum number of tuples being used at any one time. This is a measure of the instantaneous use of memory.

For the *Size-Time* cost, the time is summed for each row tuple in the plan. Again, the costs measured between each set of bars is based on the operation, which is identified at the top of the graph. Now all the tuples active in the plan at that time are included in the cost.

As with the *Execution* cost, tuple creation costs are also included. Figure 4.1 will be used as an example. The first time a tuple is created, there are three total tuples in the system, so that cost is $3C_M$. The second time, there are four active tuples, and the final creation has three active tuples. The total cost is then $10C_M$. The number of tuples active for the induced operators are 3, 4, and 3, so that cost is $10nC_\gamma$. Finally, there is one active tuple on the final restriction so that cost is C_\perp . A similar strategy is used for determining the costs of other plans. When there is some selectivity involved in an operation, this limits

the number of tuples in subsequent steps, and affects the costs as well. For example in Figure 4.3, moving the restrictions toward the incoming channels means that only s tuples make it through the restriction operators and so only sC_M tuples are created. That also limits the number of rows in the induced operation step as well, so that has a cost of $3snC_\gamma$. The constant 3 is retained even though there is only one operation. In Figure 4.3, there are 2 restrictions, so the total execution time is $sC_M + 3snC_\gamma + 2C_\perp$.

Some simple optimization techniques present themselves. First, merging induced operations is always a good idea for optimizing single queries, as intermediate row tuple creation steps are not used. Also, while the cost of the merged induced operation is considered equal to the sum of the per-pixel costs all operations, it is more of an upper bound on this cost, since merging operations also limits the number of pixel accesses that take place.

Other optimizations depend on the individual operator costs. If row allocations and restrictions are not expensive, that is $C_M \approx C_\gamma \approx C_\perp$, then for row tuples with $n \gg 1$, the cost model simplifies to counting only induced operations. In this case distributing restrictions always saves cost. If instead allocations are very expensive, i.e., $C_M \gg C_\gamma$, then the cost models reduce to counting allocations. Again, distributing restrictions makes sense.

In fact, the only time that distributing restrictions does not save cost is if either C_\perp is very high, or if the selectivity on the restriction is nearly 1, as shown in the comparison of execution time costs shown below.

$$\begin{aligned}
C_M + 3nC_\gamma + C_\perp &< sC_M + 3snC_\gamma + 2C_\perp \\
&\equiv C_M + 3nC_\gamma + C_\perp < s(C_M + 3nC_\gamma) + 2C_\perp \\
&\equiv 1 - \frac{C_\perp}{C_M + 3nC_\gamma} < s \\
&\equiv 1 - s < \frac{C_\perp}{C_M + 3nC_\gamma}
\end{aligned}$$

This shows that in systems where restriction costs are not high, distributing restrictions will almost always result in optimal solutions. These optimizations are with respect to the execution time cost model, but examination of Table 4.3 shows this plan is optimal for

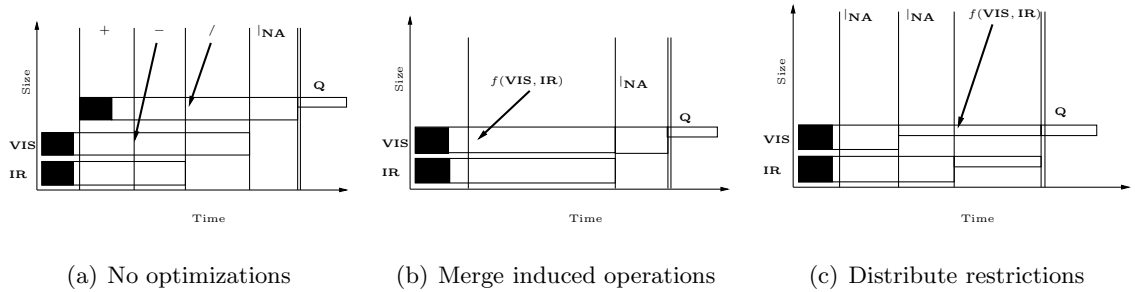
Table 4.4: Cost summary for query **C** with tuple reuse

Plan	Execution	Creation	Max Size	Size-Time
Figure 4.1	$C_M + 3mC_\gamma + C_\perp$	1	3	$3C_M + 8nC_\gamma + C_\perp$
Figure 4.2	$3nC_\gamma + C_\perp$	0	2	$6nC_\gamma + C_\perp$
Figure 4.3	$3snC_\gamma + 2C_\perp$	0	2s	$2snC_\gamma + (3 + s)C_\perp$

all plans. This is mainly because minimizing tuple creation costs will also limit the amount of memory used by the query.

4.2.1 Memory Management

Normal memory allocation requests in a program can be very expensive in relation to other costs, and in designing a DSMS, it is apparent that keeping tuple creation costs low by developing a specialized memory management system can reduce query costs. This is discussed in Section ???. However, other memory costs could be affected by modifications of the operators themselves. For induced operations, rather than output tuples being created, the input tuples could sometimes be reused to hold the output values. This results in savings for most cost models. Figure 4.4 shows QEG diagrams utilizing tuple reuse, and Table 4.4 summarizes the new costs.

Figure 4.4: QEGs for $\frac{C_2 - C_1}{C_1 + C_2} |_{NA}$ with reuse

Tuple reuse requires more sophisticated execution plans that order operations such that tuples are only marked for reuse inside operations when they are no longer needed for other operations. In the implementation described in Chapter ??, a memory management system based on reference counting is used, and the optimizer does not reuse tuples within

operations. One practical reason for this, also discussed in Chapter ??, is that some operations hold input tuples for a short time waiting for more input. If these queues are hidden from the optimizer, it can't be sure of when tuple data can be overwritten.

4.2.2 Neighborhood Operations

The complete formulation of query **C** also contained a neighborhood operation, $\oplus \mathbf{N}_{4 \times 4}$. How neighborhood operations are included into the single query optimizations is also an important question. There is less ability to rewrite the neighborhood operations within an operation, because they cannot be distributed across induced operations or spatial transforms. As an example, it is clear that for some single valued induced operation, f , on image **a** with an associated neighborhood summation, $f(\mathbf{a}) \oplus \mathbf{N} \neq f(\mathbf{a} \oplus \mathbf{N})$. Figure 4.5 shows a concrete example of this, using query **C**, where the result for a single pixel of data changes with the ordering of the operations. Section 3.3.2, described the formulation of the queries in the *GeoStreams* system. These queries included specification of an implicit neighborhood averaging operation. For all queries that operation is realized directly on the input image channels from GOES. Moving neighborhood operations to the input channels ensures the query results are most consistent with the incoming RSI data.

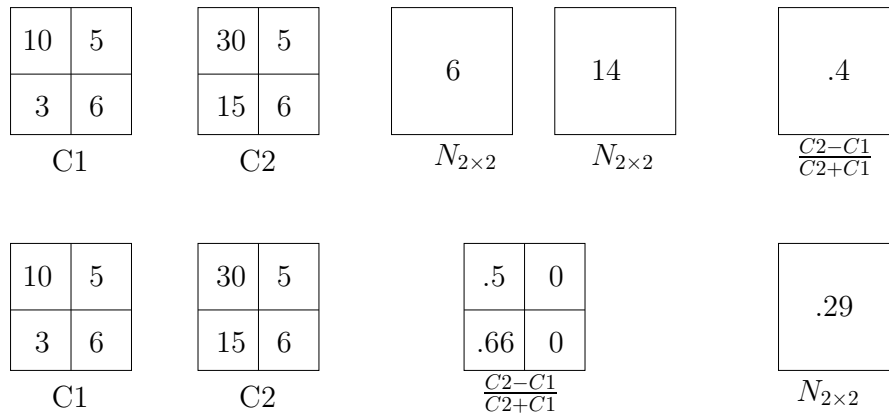


Figure 4.5: Neighborhood operation order

As discussed in Section 3.3.3, it is possible, however, to distribute a restriction operator through a neighborhood operation when the neighborhood is fixed in size, by increasing the point set to include any pixels needed to perform the neighborhood operation

for each pixel in the final point set. For the averaging operation included in the query, this simply means expanding the point set on each edge by the size of the averaging window.

Figure 4.6 shows the QEG of query **C**, including the neighborhood operation. Two sets of restrictions are required in this formulation, an initial restriction to limit the neighborhood operation and a final restriction to limit the output point set of the neighborhood operation to the query specification. In a calculation equivalent to that described in Section 4.2, including the initial restriction almost always results in cost savings. In the figure \mathbf{NA}' indicates the \mathbf{NA} point set expanded to allow the neighborhood operation. The figure also shows that the second set of restrictions have a selectivity of nearly $s = 1$. This is because the bounds on the initial selection on the original point set is very close to the final coarser point set. This high selectivity implies that this restriction could be moved after the induced operation, or even removed for some queries, as will be seen in the next section.

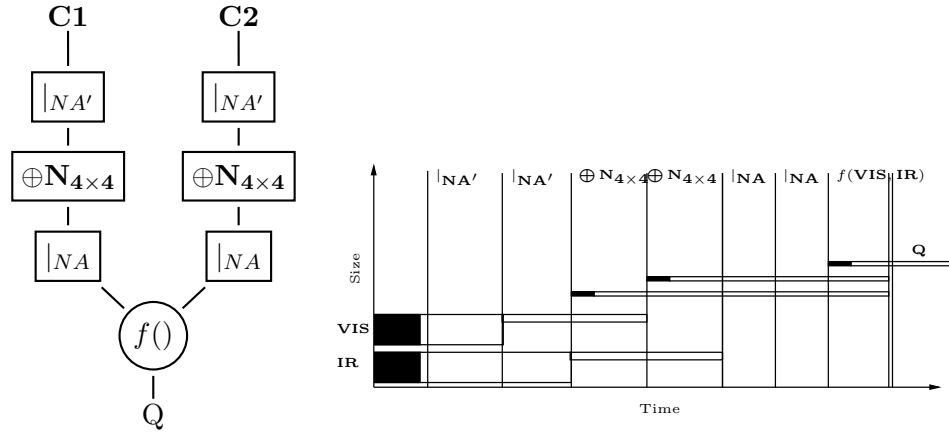


Figure 4.6: QEG for query **C** with neighborhood operation

Table 4.5 shows the associated costs for this query. The neighborhood operation reduces costs for later operations in that for every four rows that are input to the $\oplus\mathbf{N}_{4 \times 4}$ operation, only one row is output and with only one fourth of the number of pixels in the row. This also means that although there are more operators with a neighborhood operation, there are also savings in the execution time due to the changes in the point set.

No requirements on the specific way the neighborhood operations are implemented is assumed in the above formulations. In Section 4.3.1, the neighborhood operation is decomposed into a set of operators as a way to increase sharing of intermediate data in a

Table 4.5: Cost summary for query **C** with neighborhood operation

Model	Cost
Execution	$2s/hC_M + 2s/hC_{N_{4 \times 4}} + 3(s/h)(n/h)C_\gamma + 4C_\perp$
Creation	$\approx 3(s/h)$
Max Size	$\max(2, 3s)$
Size-Time	$3(s/h)C_M + 3(s/h)(n/h)C_\gamma + (3 + s + 4(s/h))C_\perp + (3s + 3(s/h))C_{N_{4 \times 4}}$

multi-query operation. The implementation described there does not affect the constraints to creating a QEG described in this section.

4.2.3 Spatial Transforms

The final operation that is available in the query interface of Section 3.3.2 is composition, or spatial transform. An example of this is Query **D**, which has the same formulation as Query **C** but with an additional step of projecting the data to a new coordinate system. This more complete query is the final example.

The most common spatial transformation converts the satellite point set to the point set grid requested by a query. Each query can specify pixel locations and resolutions that differ from the input images. Transformations perform this conversion. The transformation definition requires that the image **a** be defined at arbitrary points and not only at the original satellite pixel locations. This can be accomplished by extending a physical image to one where all points are defined in some interpolated manner from the original satellite values. Figure 4.7 shows this interpolation technique. In this example, bi-linear interpolation extends an image to supply arbitrary pixel values based on a linear combination of the surrounding 4 pixels in the image [?]. Other methods common in remote sensing include nearest-neighbor techniques, where only the closest pixel value is used, and bi-cubic convolution [?], where a window of nine pixel values is used.

Section 3.3.3 describes how spatial transforms can be represented as a restriction operator followed by a composition operator, by performing a reverse transform of the final point set. This allows a restriction to be added into the single query. For a WMS based

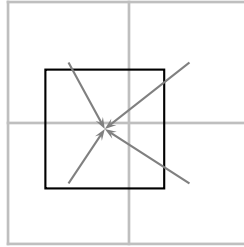


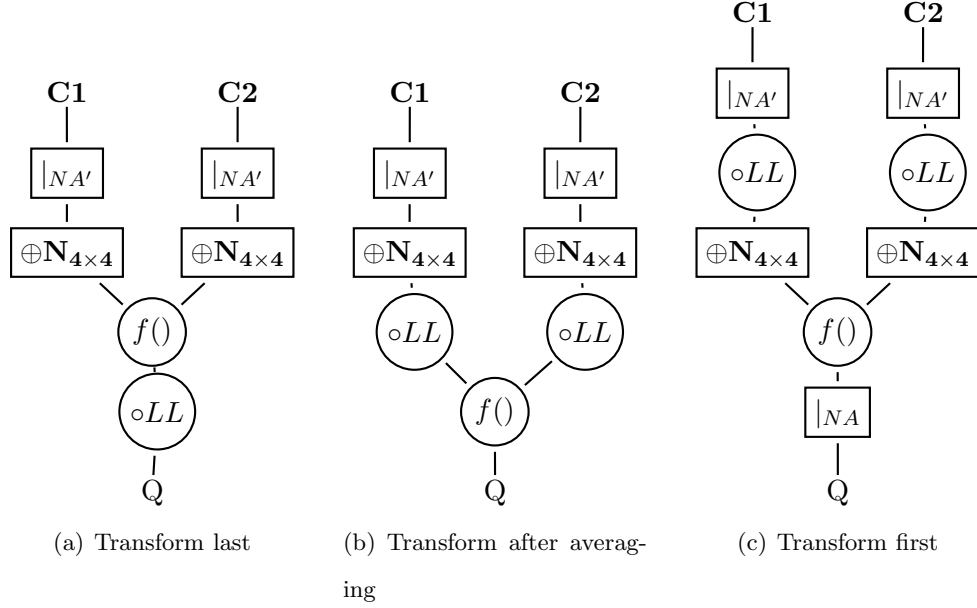
Figure 4.7: Image interpolation for spatial transforms, $\mathbf{a} \circ f$

query, the only restriction specified is in terms of the final coordinate system, so this step is required to determine a restriction operator for the query.

The WMS based query formulation does not explicitly define the order of operations, and there is some flexibility in interpretation. In general, the best practice is to carry out all averaging and induced operations before any spatial transformations. This insures retaining radiometric values as close as possible to those measured by the RSI instrument. In practice, however, the transformation is sometimes a first step in processing RSI data. This is primarily because of the advantage of working in a standard coordinate system, particularly when combining RSI data with other GIS data sets.

The required implicit interpolation step for performing a transformation works best when the composed point set has a comparable resolution to the input point set. This means that wherever the spatial transform is performed, the resolution of the resultant point set should match the input point set resolution as closely as possible. If the transformation is performed as the last step, then the resolution of the final point set is matched to the averaging step in the GOES coordinate system. If the transform is performed first, the point set is matched to the incoming channel point set resolution.

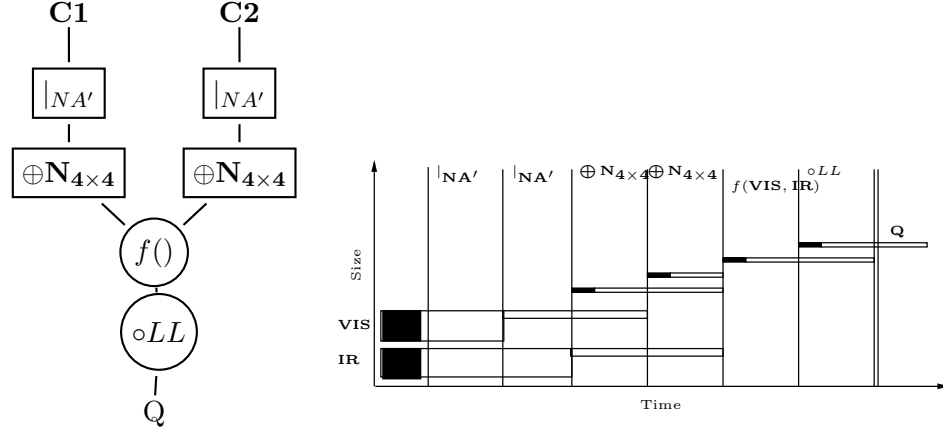
A few other processing orders are possible as well. Figure 4.8 shows some QEGs for Query **D** with the spatial transformation in different locations. The figures show a removal of the last restriction in query **C**. Unlike the previous example, this restriction is not required to match the point set requested by the WMS query. That role is instead handled with the transform step. Since the selectivity of these restrictions are nearly one, removing these operators does not significantly increase the number of rows in entering the final operation.

Figure 4.8: Potential QEGs for query **D**

In terms of defining costs for each of these QEGs, it is clear from the formulations that the spatial transform is simply an added cost to the previous model for query **C**. Also, Table 4.2 shows the cost is a function of the number of rows sent through the operator. Though the order of the operators does affect the point sets that are in the plan—for example moving the spatial transform to the front of the QEG does imply all point sets after this are in the new coordinate system—the requirement that resolutions match between point sets means that the number of rows in the point sets does not change dramatically from one coordinate system to the next.

All of the above considerations imply that locating this operator at the end of the QEG will result in the least cost, or at worst a reasonable approximation. This limits the number of row tuples entered into the operator, and also reduces the constant on the size of the operator as well. In addition, Section 4.3 uses this location for the to improve sharing of data for multi-query optimizations. Figure 4.9 shows the graphical cost model for this formulation.

Table 4.6 shows the final equations for the cost of a complete query. This shows the optimization plan for any general query using the WMS query formulation.

Figure 4.9: QEG for query **D**Table 4.6: Cost summary for query **D**

Model	Cost
Execution	$4s/hC_M + 2s/hC_{N_{4 \times 4}} + 3(s/h)(n/h)C_\gamma + 2C_l + (s/h)C_{\circ LL}$
Creation	$\approx 4(s/h)$
Max Size	$\max(2, 3s)$
Size-Time	$4(s/h)C_M + (3s + 3(s/h))C_{N_{4 \times 4}} + (3 + s)C_l + 3(s/h)(n/h)C_\gamma + (s/h)C_{\circ LL}$

4.2.4 Building a Single Query Execution Plan

Building upon the previous sections, a simple methodology can be defined for single query optimization in a very straightforward manner. The methodology does not change from query to query. It is a heuristic approach only in the sense that some particular queries could have a moderately smaller cost by eliminating restrictions or reordering operations. For example, a query involving 5 channels on nearly the entire hemisphere might be slightly faster if the entire hemisphere were processed and only one restriction applied to the final result, as opposed to applying 5 restrictions to no or little effect. Even in such cases the heuristic is a reasonable alternative.

Section 4.3.2 will develop a specialized realization of the neighborhood operation for multi-query optimization. One result of this will be that all queries, even those without a specified coordinate system, will end with a transformation step even if the requested coordinate system is in the original GOES system. Therefore, the steps below are valid for

all input WMS queries.

The following rules are used to optimize a single query:

1. The specification of the point set in the WMS query is defined in the final coordinate system. This point set is transformed to the GOES perspective coordinate system including consideration of the pixel resolution. This transform also defines the query restriction in the GOES coordinate system.
2. If the query is for a specified derived product available through the WMS interface, all input image channels required for that product are identified.
3. The query restriction is applied to all identified raw image channels.
4. The input channel(s) are averaged to the resolution appropriate for the query, as specified by the GOES coordinate point set.
5. If the query is a derived product, a processing node is added to perform that calculation. All image channels for the product are directed through this node.
6. The resultant image is transformed to the final coordinate system and point set as requested by the query.

The resultant QEG will always resemble the organization shown in Figure 4.9.

4.3 Multi-Query Optimization

In a typical DBMS, once the individual queries are rewritten to optimize their individual execution, they are executed without regard for other queries in the system. For the DSMS, however, queries are often long running and it makes sense to optimize on multiple active queries in the system. This is especially true for queries on RSI where there exist opportunities to share operators as well as intermediate results among queries that share common products and point sets.

As an example, multi-query optimization techniques are explored for the first four example queries from Table 1.1, replicated below in Table 4.7. The first four queries

are chosen since they share many components and offer a good example for multi-query optimization.

Table 4.7: Example queries

Q	Product	ROI	Time	Projection	Resolution
A	C1	Mexico	Always	GOES	$\approx 1 \text{ km}^2$
B	C1	N. America	Always	Lat/Long	$\approx 4 \text{ km}^2$
C	NDVI(C1, C2)	N. America	Always	GOES	$\approx 4 \text{ km}^2$
D	NDVI(C1, C2)	Hemisphere	Always	Lat/Long	$\approx 8 \text{ km}^2$

Figure 4.10 shows these queries all rewritten to the processing order derived from the single query optimization of Section 4.2. Determining costs for this set of queries would entail summarizing the costs for each individual query. In the case of *Execution* and *Creation*, this cost is independent of the order of execution. The *Max-Size*, and *Size-Time* cost models are dependent on the order of processing steps, as both are affected by the active number of tuples from the streams in the system.

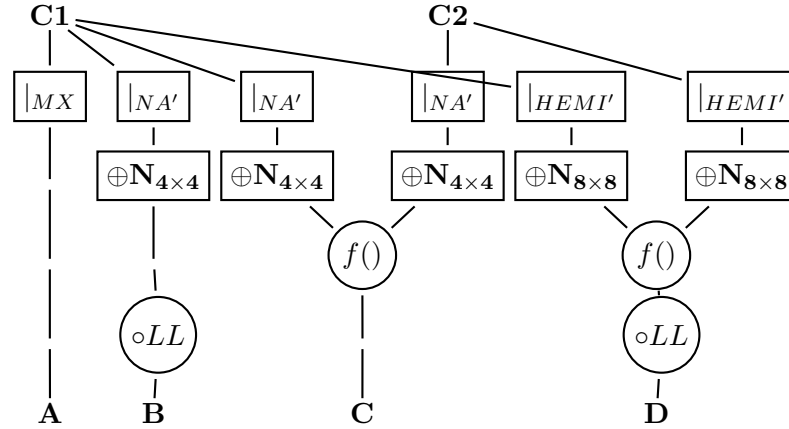


Figure 4.10: Unoptimized QEG for queries A, B, C, and D

Before investigating these costs, however, a few optimizations will be applied to this set of queries. The first thing to notice about the above QEG is that all queries start with a restriction operator on one or more input image channels, all having the same input stream. A more efficient method is to design a restriction operator developed for multiple queries. Table 4.2, anticipated a cost savings in combining restrictions. Chapter 6 will

develop an index, the Dynamic Cascade Tree (DCT), developed specifically for restriction operations on streaming RSI data. Leaving the implementation until that chapter, the previous set of queries can be rewritten using two shared restriction operators. Figure 4.11 shows the new QEG with this addition. A single restriction operation is now used and multiple streams are output, each with their individually defined restrictions. As in the case of a single restriction, the *row-scan* ordering of the data allows for all restrictions to be implemented without the creation of any new row tuples of data.

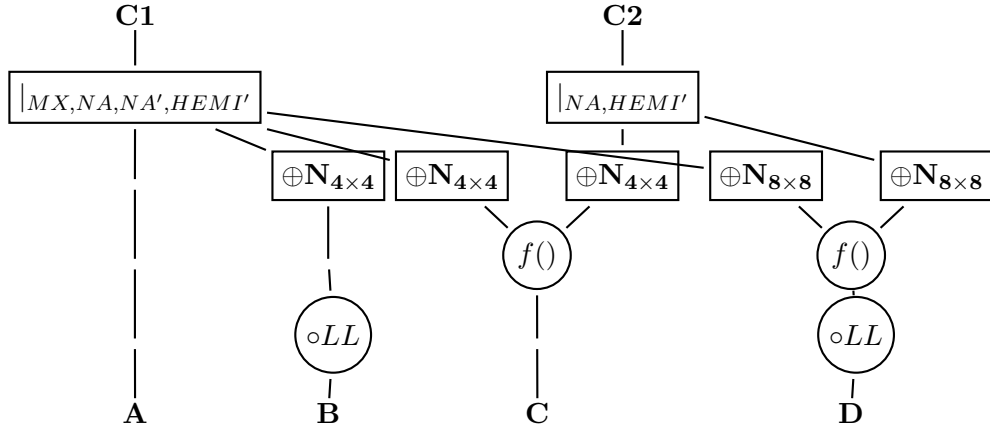


Figure 4.11: QEG for queries **A**, **B**, **C**, and **D**, with shared restriction

Figure 4.11 now shows multiple queries going into very similar neighborhood operations, covering similar geographic regions. Significant savings could be obtained by sharing the intermediate results of these neighborhood operations. However, each query has its own averaging window size, and although these may be very similar, there is no guarantee they are the same for any pair of queries. In order to allow more sharing of intermediate operations, the neighborhood operation needs to be specialized.

4.3.1 Multi-Query Neighborhood Operations

Up to this point, the actual implementation of neighborhood operations has not been discussed. Implementation becomes an issue in the context of multi-query optimizations since it is desirable to share intermediate results between neighborhoods with different point set resolutions. The strategy chosen here is to develop a fixed number of image resolutions that are created by progressively averaging the RSI images in 2×2 pixel grids. An

important component of this strategy is that these fixed resolutions are the best estimations of what the image radiance for the RSI data would be at coarser scales. These particular resolutions will not in general match the resolution needed by queries. In order to calculate image values for the point set of an individual query with its associated pixel resolution an additional spatial transformation step is included. The closest averaged grid smaller than the query resolution is chosen and, as with the previous discussion of the spatial transform, bi-linear interpolation is used to determine the values for the new point set.

Figure 4.12 shows both neighborhood operations and interpolation techniques. The interpolation matches that discussed in Section 4.2.3. Involved in the neighborhood operations is an implicit transformation to a point set which is one fourth the size of the original. The pixel locations are also moved to the new center of the averaged pixels.

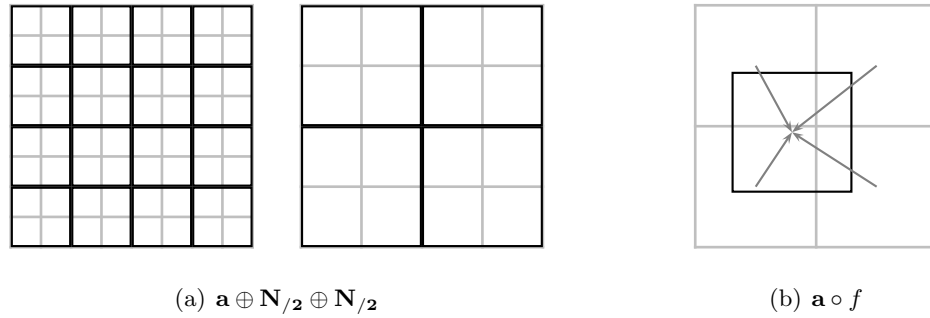


Figure 4.12: Image averaging and interpolation

Although there are other methods for averaging pixels, this successive averaging is used as a simple and computationally effective method. As is indicated from the figure, bi-linear interpolation also is a reasonable method for point sets with a resolution with sizes up to twice the grid resolution, as is possible with the method described for neighborhood operations. Once the entire neighborhood operation is decomposed into a series of halving operations and a final transform, they can be treated as individual operations of a query plan. Also, if the query contained another spatial transform, these can be combined to a single operation, as described in Section 3.3.3. If single query optimization were re-investigated with this neighborhood operation implementation, the image halving operations would float to the top of the QEG, while the transform would descend, perhaps joining with another spatial transform as the last operation.

For multi-query optimization, this implementation lends itself particularly well for sharing intermediate data within the DSMS. For example, in the queries **A**, **B**, **C**, and **D**. After the first optimization of merging restrictions shown in Figure 4.11, most queries move next into neighborhood operations. Some of these cover very similar ROIs, like queries **B** and **C** that have similar point sets over North America. Some queries are contained within other queries, like North America being contained in the query on the hemisphere. Other queries share ROIs, but have resolutions that are coarser.

Figure 4.13 shows a modification to the previous QEG where intermediate data is shared between queries using the described neighborhood operation. The restriction operators are modified as well. Point sets are merged in the first set of restrictions, so that overlapping ROIs are combined into a single inclusive point set. Next, as in the single query optimization, streams follow with one or more image halving operations, approaching the resolutions required by the queries. Induced operations follow this step. Queries require different resolutions, and the induced operations fork from the halving operations at various stages. Whenever forks appear later in an image stream, a new restriction operator is used to shuttle the required point set to the following operators.

This optimization procedure maximizes the amount of sharing that can occur between queries. In fact no data shared among queries is replicated anywhere in the stream. As discussed in Section 4.3.3, this is not necessarily optimal but produces good results with queries whose ROIs tend to overlap.

4.3.2 Building a Multi-query Optimized Query Execution Plan

The example above does not demonstrate all potential paths for sharing and optimization that can occur between multiple queries. The actual methodology to create the multi-query QEG is now described. The execution plan in the multi-query environment is built with the goal of using a single operator path for all queries that potentially share intermediate results. This includes all operators related to resolution changes and derived operations. The following rules are used to optimize all queries. Assume there is a single execution plan that is being maintained.

is always added last into the execution plan, as it is not shared between multiple queries.

Because WMS queries are allowed to define arbitrary final point sets, the last interpolation or projection step cannot generally be shared among queries. However, if queries were limited to specific point sets, for example limiting resolutions to integer numbers of hectares at standard postings, then an additional level of intermediate images sharing among queries would be possible. This would not be a standard WMS query, however.

The order that the queries are considered does not affect the final QEG, since the operators for each individual query are arranged the same and as a result add into the multi-query plan in the same order regardless of how the queries themselves are added.

Figures 4.14(a), 4.14(b), 4.14(c) and 4.15 progressively show the construction of the query plan, using the same set of example queries. These figures demonstrate the operators, especially the restrictions, more graphically as an attempt to demonstrate the sharing between queries.

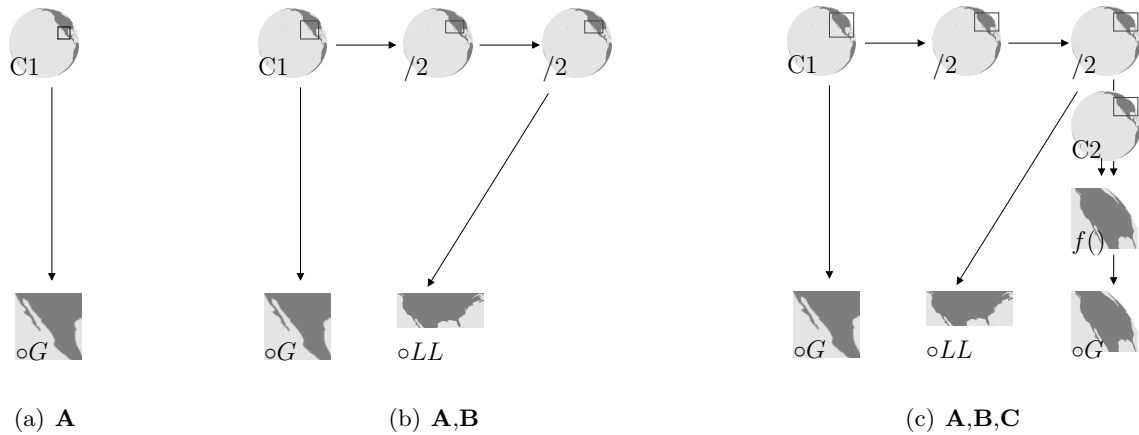


Figure 4.14: Execution plan for queries **A**, **B** and **C**

In Figure 4.14(a), the first query, A, is for the Channel 1, GOES data over Mexico. The query resolution is approximately the same as the image stream, and no averaging is required. The query operations consist of a restriction on the Channel 1 image, and an interpolation to the query point set.

In Figure 4.14(b), query B is added, also for Channel 1. The ROI is North America, and here a coarser resolution is required. The Channel 1 image is reduced in resolution through two halving operators. The restriction on the original image is expanded to include both query ROIs, but the averaging is restricted to North America only. This query also projects the final product to a new coordinate system.

Figure 4.14(c), shows another query, C, over North America being added, this one being a function of two input channels. The processing nodes for averaging the channel 1 image have their restrictions expanded to include this new ROI. In reality, for GOES data, different channels have different resolutions and Channel 2 is already at the required resolution. The restriction on Channel 2 only needs to encompass this last query. A new processing node is added to perform the derived operation on the two images.

Figure 4.15 shows query D requesting a nearly global coverage for the same image function as query C. This requires the restrictions for all previous nodes to be expanded for this final query. Since two queries are interested in the same function, these results are shared between the queries.

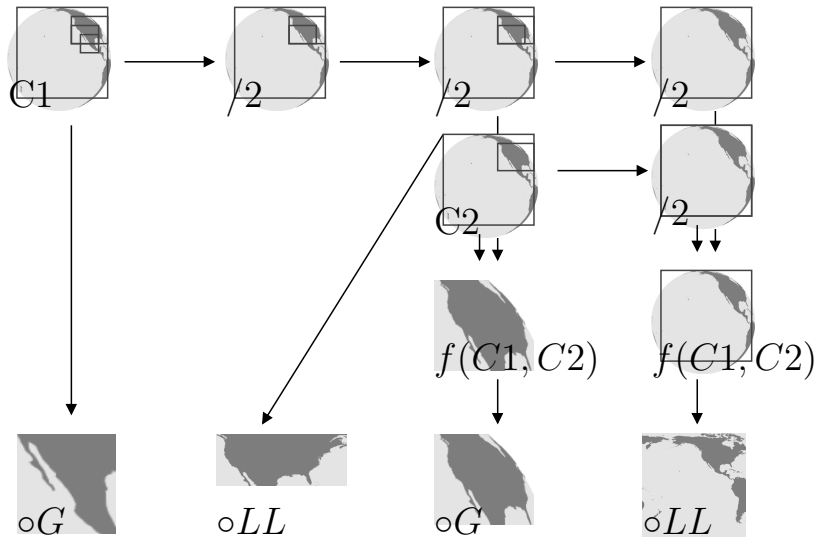


Figure 4.15: Execution plan for queries A, B, C, and D

The final query processing workflow combines shows all four queries using some of the Channel 1 node, all but query A, using a common set of averaging functions as well. Queries C and D, also share all of the nodes of Channel 2, as well as the output of the

derived image function.

This example also shows that the inclusion of queries with large ROIs, like Query D, can lead to large spatial extents on many intermediate nodes. While driving up the total processing time, it does allow for sharing among many queries.

4.3.3 Optimization Problems

There are some instances where the multi-query optimizations described do not lead to the most efficient solution.

In a few cases, the single query optimizations are not the best starting point in optimizing multiple queries. For example, imagine a user launching queries for multiple products but with the same point set. In this case, transforming the data stream earlier results in some savings. Figure 4.16 shows an example of three queries with three different products. The standard optimization strategy has an additional transform operator. Although there is some savings for this strategy, as was discussed in Section 4.2.3, this formulation would change the results slightly due to the reordering of the transform and the neighborhood operations. Leaving the order of the single optimization QEG in the multi-query plan ensures the WMS query will respond with the same results regardless of the other queries in the system.

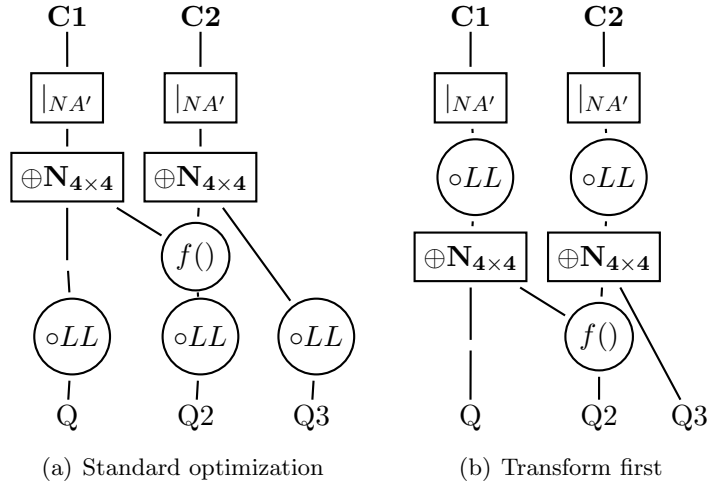


Figure 4.16: Rearrangement of single query QEG

Another example of optimization problems is shown in the formulation of Fig-

ure 4.13. Here, two halving operations are in series with no other stream using the intermediate result. This could more efficiently be performed with a single $N/4$ operator, and save the creation costs of those intermediate rows. If at a later time another query required that intermediate resolution, the operator could be split again into two $N/2$ operators in series.

The final and more troublesome problem with the multi-query optimizations described above concerns the decision to always share existing operators. Since this sharing requires growing restriction point sets to cover all queries, large numbers of the inclusive point set might end up in no final query result. In these cases, a better strategy would be to not share the intermediate operation and instead create a new operator performing the same function on a different processing path. Figure 4.17 shows a simple example of this problem with two small query ROIs, with a wide separation between them.

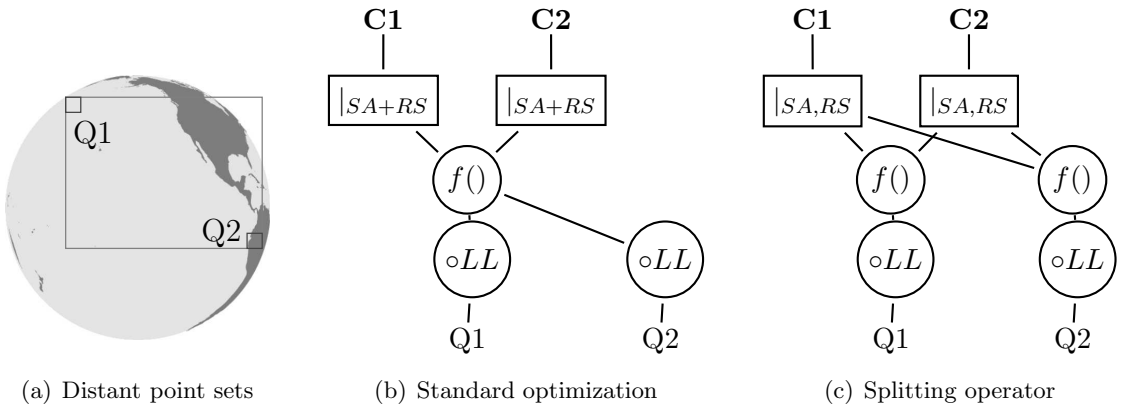


Figure 4.17: Inefficient sharing of intermediate results

The optimization methodology described does not look at splitting operators in such a manner. There are two reasons for this. First, it is not easy to decide where to split operators in the presence of many queries. More importantly, alternative solutions exist for solving this problem. The problem is basically caused by the fact that only rectangular point sets are defined in the restriction operators. Instead of splitting the operator, another solution would involve a restriction operator that allows other point sets beyond rectangular lattices. If, for example, unions of rectangular regions could be defined as a point set, then those intermediate results would not be created and the existing optimizations would continue to offer the best QEG solution. In some cases, like the experiments in Chapter 5,

the underlying applications limit such methods. In other cases, like the on-line system described in Chapter ??, these modifications to the allowed point set definitions could be easily included.

4.4 Related Work

The amount of research into query optimizations for relational databases is extensive. Because access to secondary storage is orders of magnitudes more expensive than access to main memory, many optimization strategies dealing only with the number of pages accessed as a cost model are used. Streaming databases typically work on a different paradigm, maintaining the data and indexes in main memory, and require a different cost model, usually based on minimizing the CPU time. In this context, the QEP for a DSMS is developed from analyzing a number of different organizations of operators that flow together.

Optimizing streaming databases has many similarities with methods for optimizing for multi-queries in traditional databases, Sellis [?] offering some of the earliest examples for finding common sub-expressions. Other studies for multi-query optimizations through common sub-expressions include [?, ?, ?].

While multi-query optimizations have traditionally been based on finding commonality among queries, DSMS have concentrated effort into systems with multiple continuous queries. In DSMS research the notion of organizing multiple queries to speed up processing has been studied under multiple conceptual definitions, including grouped filters [?] and query indexing [?].

Andrade et al. [?] described a method for multi-query optimization of image analysis by decomposing complex functions into more primitive operations that admit better reuse properties, with an emphasis on aggregation functions.

All GIS related operations have a long history. The averaging feature of progressively rescaling imagery by halving the spatial resolution has been used in many applications, including quadtrees [?], Haar wavelets [?], and image pyramids [?].

Chapter 5

Multi-Query Optimization with Existing GIS Applications

This chapter investigates the multi-query optimization techniques of Chapter 4 by developing a system to answer multiple user queries on the Geostationary Operational Environmental Satellite (GOES) image stream, using an existing GIS application as a processing framework.

The system divides the query processing into two major components. A query optimizer maintains the current set of active continuous queries. On the acquisition of a new frame of satellite images, the optimizer produces a dynamic execution plan that is specific to the active queries in the system. The queries are organized into a single standard processing plan designed to share operators and intermediate results. The query executor rewrites this plan into a set of geospatial processing steps and executes the plan.

The system is implemented using data from NOAA's GOES. Individual queries are defined with the WMS query specification. The query optimizer produces a plan that is a directed acyclic graph with specific spatial parameters defined for each processing node. The query executor is implemented using the Geographic Resources Analysis Support System (GRASS) GIS application [?].

Experimental results using predicted query patterns over the visible hemisphere of the weather satellite indicate that developing multi-query optimized plans can improve

performance significantly when compared to queries executed separately.

5.1 Objectives

Chapter 4 predicts savings in processing by developing a multi-query QEP. How much savings is dependent on both the implementation of the individual operators and on the make-up of the current queries within the system. The objective of this chapter is to begin to quantify the savings of a multi-query QEP, using an existing application framework, a real RSI image stream, and real world query parameters.

The GIS application framework chosen is GRASS. GRASS has no notion of image streams, and like most GIS applications, works on discrete images in secondary storage.

The well ordered arrangement of streaming images like GOES weather satellite data allows natural divisions of the stream into separate images for different channels and times. Using these divisions, the stream can be separated into files corresponding to images. GIS data processing workflows can be applied to these images with a number of advantages. First, it relates to common current practices and lends itself to simple implementations using existing applications. It results in processing methodologies that are strongly connected to the current GIS practice, including models to develop workflows within these systems. ESRI's model builder [?] is an example of such a system. It is also easier to consider retrospective queries in such a system, as the mechanism to process continuous queries is no different than retrospective queries against previously acquired data. Finally, delivery mechanisms to clients launching the queries can be simplified in this environment as well.

There are also disadvantages. Using traditional GIS applications implies creation of many intermediate data layers. This creates a large increase in the overall size to satisfy individual query as these intermediate images are created and used in the workflow. Because these intermediate images rely on secondary storage, the system can be significantly slower than a complete in-memory streaming based system.

The image stream used is data from NOAA's GOES satellite. The queries are on either the spectral image channels or derived products from these channels. The individual queries are relatively simple and limited in expressiveness, but are designed to satisfy the

most majority of RSI query needs. They defined as in the WMS query specification.

The query optimization procedures described in Chapter 4 attempt to limit the processing time for all queries in the system. For the RSI queries, optimization is primarily concerned with two goals: query writing rules to limit the amount of work done for each query, and exploiting common subsets among individual queries active in the system.

System performance is tested in a number of experimental designs. All experiments are based on query patterns meant to represent typical access to weather satellite data. The experiments are designed to test some of the key areas relating to the optimization of multiple queries, including tests on restrictions, derived products and ensembles of queries.

5.2 Prototype and Experiments

The system described above has been implemented using a combination of the GRASS [?] GIS application and supporting Unix-like tools. GRASS is a procedural GIS application that is especially well suited to raster data manipulation. GRASS is developed as a series of stand-alone programs that can be linked together for more complex operations. GRASS includes the notion of an active region. This applies an implicit restriction for most operations in GRASS.

Because GRASS is implemented as group of standalone applications, the default processing environment is the command shell and standard tools are available for programming the GIS. In particular, this application uses a sophisticated `makefile` as the query plan executor. `Make` is designed to execute Directed Acyclic Graphs (DAGs), and lends itself well to organizing the GRASS processes. The query optimizer is a separate program that organizes the multiple queries into a single processing workflow.

The experiments were developed to replicate sets of queries made on GOES Imager input data and realistic ROIs for the continuous queries. Rather than randomly locate these ROIs throughout the image domain, they were preferentially located around a small number of “hot spots” in the hemisphere. This more closely relates to real world scenarios where specific parts of the RSI data is requested by a large numbers of users. Although the general area of the queries was fixed to a small number of locations, the individual ROI centers,

total sizes and aspect ratios were varied for each ROI. This corresponds to many users requesting slightly different particulars for a general ROI. Some random ROIs were also included in the experimental setup. Table 5.1 describes the parameters for the query ROIs. The query region parameters table lists the various general regions used in the setup. Each general ROI was given equal weight and the queries were randomly distributed among them. The ROI variation table shows the range of variation on the image size, location, and aspect with respect to the selected “hot spot”

Table 5.1: Query ROI parameters

Query Regions		ROI Variations	
Global	GL	Area	0.8-1.2
Northern Hemisphere	NH	Aspect	$\pm 10\%$
United States	US	Center	$\pm 10\%$
Western Coastal	WC		
Mexico	MX		
South America	SA		
Random	RN		

Data from the GOES satellite is received directly from a receiving station co-located with the computer running the query application. GOES scans various sections of the Earth’s surface about once every 15-30 minutes in specific frames. Frames range in size from 100 MB to 440 MB.

Query optimizations described in Chapter 4 were tested against a system running queries that were individually optimized only. Between 5 and 100 queries were posed against the system, and performance measured as the total processing time for each MB of data delivered to queries. Various combinations of restrictions and derived operations were performed on either single channels or all channels in the system. For simplicity, all queries were executed on the default GOES coordinate system.

Most experiments were run against a single set of GOES Imager data corresponding to a full disk input set taken in mid-day. Channel 1 was about 440 MB in size, with 10828

rows and 20792 columns in the input image. The other channels have coarser resolutions and are about a quarter of the size.

Figure 5.1 shows the distribution of the input query ROIs over the hemisphere. Lighter areas on the map indicate more queries for that region. Areas along the western seaboard are most queried. For 100 queries, individual pixels participate in an average of about 28 queries, with participation in up to 66 queries for some pixels. The experiments were run using 5, 20, 50, and 100 individual queries.



Figure 5.1: Query density

As the experiments show, the processing time is related also to the final resolution of images requested by the queries. Two sets of query image resolutions were used in the experiments. In both cases, the query resolution ranged from resolutions on the scale of the finest satellite image—channel 1, approximately 1 km resolution at nadir—to a resolution corresponding to a query image width of 256 pixels, a reasonable lower bound on resolution. The distribution of resolutions were varied. Both distributions weighted the finer resolutions more than the coarser ones, but with different scales. Figure 5.2 shows the histogram distributions of the two example query sets.

The experiments were run on a quad Intel(R) Xeon(TM) CPU 2.80GHz processor with 4GB of memory and a 3.1 TB RAID 5 filesystem using 3ware 9500S-12 SATA-RAID

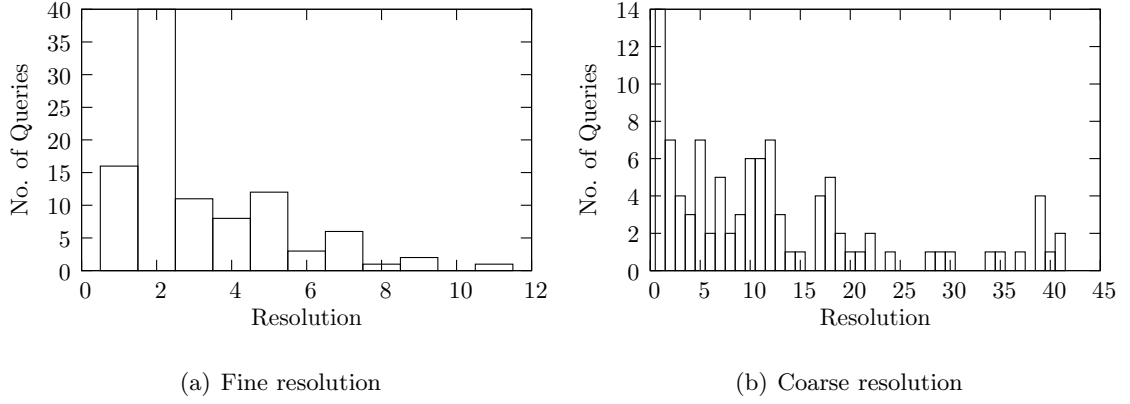


Figure 5.2: Query resolutions

5.2.1 Restrictions

The first tests investigated only the effects of sharing intermediate data for restrictions only. In this example, queries were limited to simple restrictions on image channel 1. For the multi-query optimization plan, this includes sharing of intermediate datasets at the various coarser resolutions of the channel 1 data. Figures 5.3(a) and 5.3(b) show the processing time normalized to the total size of all query output. In both instances sharing intermediate data sets decreases the overall processing time of the system. For the finer resolution case, as the number of queries increases, the mix of ROIs in the set becomes fairly uniform, and the average processing time becomes fairly constant.

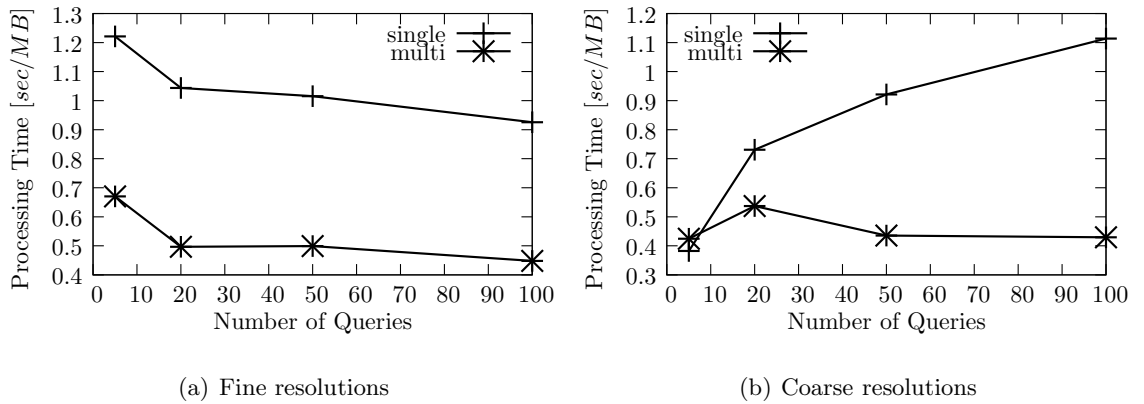


Figure 5.3: Restriction only experiment

The savings are more dramatic for the coarser scales, primarily because each indi-

vidual query is more expensive since more averaging needs to be done. This is not true in the optimized case, as all the averaging is performed one time for all queries. It is unclear why the processing time for the non-optimized case did not stabilize to a more constant processing time in this example.

Overall, there is about a two fold decrease in the processing time using multi-query optimizations.

5.2.2 Derived Products

If simple restriction queries show an improvement in processing time due to multi-query optimization, it would be expected that derived products would show even more improvement. This is because the multi-query optimization saves on two levels, by avoiding image averaging on multiple channels for each query, and by sharing the derived product among all queries.

Two example derivative products were examined. The first was a normalized difference ratio, a product in Queries **C** and **D**, discussed in Chapter 4. This is a moderate computation. Figure 5.4(a) shows this experiment being run on the finer resolution test. As expected, the multi-query optimization runs considerably faster then the single query optimization, by about a factor of 3.

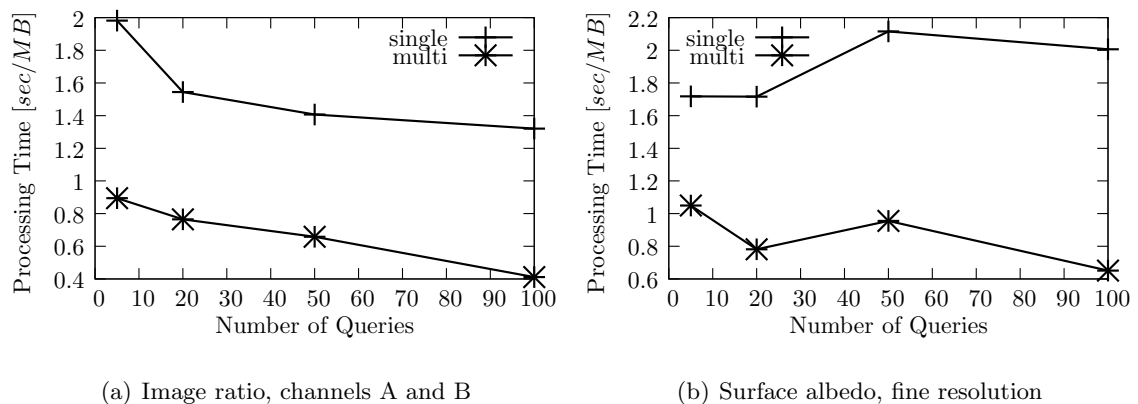


Figure 5.4: Derived products experiment

For a different type of derived product, surface albedo was calculated. Surface albedo is an important parameter for many applications. Albedo is calculated as the mini-

mum pixel value over some preceding days, here up to the previous 14. This assumes that at some point in that time, every pixel in the image had at least one cloud free acquisition and that clouds are brighter than the surface. This is a simple calculation, but expensive in that many raster images need to be accessed to generate the result. Figure 5.4(b) shows the comparison between the single and multi-query optimizations for albedo. As with the other derived product, sharing intermediate results leads to increased processing speed.

5.2.3 Multiple Data Products

The above examples are illustrative, but are not necessarily representative of a normal set of queries acting on such a system, since the above queries all request the same data product. A more common scenario is where queries are spread among a larger number of possible data products. In this case it would be expected that there is less benefit from the multi-query optimizations, as less sharing occurs among the queries.

Figures 5.5(a) and 5.5(b) show the results of a set of queries that access 10 different data products. The same query ROIs were used, but the data products requested were uniformly distributed among the 5 Imager channels, and 5 other derived indexes, all normalized ratios.

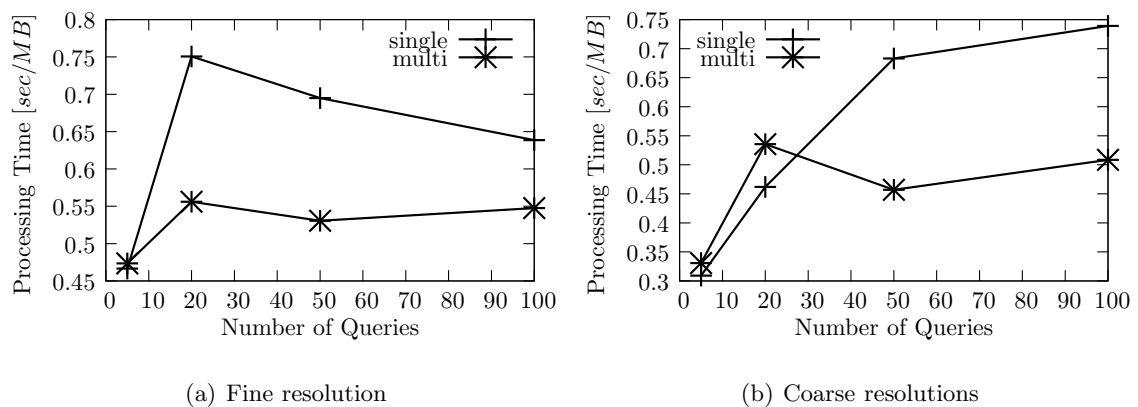


Figure 5.5: Queries on a 10 data products

In this experiment the non-optimized solution performs much closer to the optimized version. Besides the fact that there is less chance for sharing among queries, another fact somewhat idiosyncratic to the GOES data is contributing. The non-visible channels of

the GOES Imager have about a 4 times coarser resolution as compared to the visible channel. The previous experiments used the visible channel as it is the most popular channel, but this example spread most queries to the other channels, less averaging was required by each query. This further reduces the opportunities for queries to share intermediate data.

5.2.4 Reducing Secondary Storage Costs

Some of the tradeoffs between using existing GIS applications versus developing specific applications for streaming RSI data were discussed previously. Chief among considerations was the added cost of secondary I/O usage. One method of minimizing I/O costs while allowing traditional applications like GRASS would be to use a large RAM filesystem for processing images. The current system does not include enough free RAM to perform the experiments described above. However, some preliminary experiments were undertaken. Specifically, the GRASS database of GOES data was copied to a RAM filesystem. A 5 query, multiple data product experiment was run with all intermediate and result data products produced on the RAM filesystem. Comparison of these times to the previously reported show only a moderate, 10% decrease in the overall processing time for the RAM filesystem. While this is not necessarily an indicator of expected speeds for a specially crafted DSMS, it does indicate that other aspects of the workflow processing besides I/O affect system performance.

5.3 Discussion

For large numbers of continuous queries against RSI, optimization over multiple queries has been shown to be an effective method of increasing the overall system performance. How much savings can be expected depends primarily on the relationships among the queries in the system.

The optimization strategy optimizes individual queries first and then combines these using the methodology for building a QEG described in Section 4.3.2. Operators are reused for multiple queries with spatial restrictions modified to encompass all appropriate query ROIs.

The implementation uses existing GRASS GIS applications, to execute the processing steps. Remotely sensed imagery clearly provides a great opportunity to study concepts and paradigms for the management and processing of streaming data, given the existence of various satellites that are used constantly for numerous data products. Even if new processing methods are developed for processing these images operations on a finer scale—for example processing rows of data at a time as described in Chapter ??—the experiments presented here inform the developer on predicted levels of data sharing that will exist in such a system.

The ideas presented here are described in terms of continuous queries over RSI; however similar opportunities exist in other applications and incoming data streams. Another good candidates include model outputs. For example, the Weather Research and Forecasting (WRF) model, can be run in modes that periodically output weather predictions.

5.4 Related Work

The results showing performance increases with optimizations in an existing GIS application were first described by Hart [?]. Previous studies have recognized the critical role played by many data producers manipulating and making available RSI products to support environmental applications. One example, the Earth Science Workbench [?], describes a system for developing standardized workflows on locally received satellite image data that is very similar in context to individual user queries here. Similarly, the Goddard Earth Sciences Data and Information Services Center is developing a service of virtual data products that reproduce the data from original sources dynamically based on the queries to the system [?].

None of the above systems looked at combining multiple image processing workflows into a more efficient computation strategies. This is in part because the focus of these efforts centered on traditional historical queries.

Chapter 6

The Dynamic Cascade Tree

The multi-query optimization techniques developed previously require an operator that is able to provide restrictions for multiple query ROI. RSI row tuples enter the operator at high data rates, and there can be many individual ROIs handled by the system. Later, in Chapter ?? operators for an on-line DSMS will be discussed. The implementation of the spatial transformation operator will require a similar restriction operator with very many ROIs, a individual ROI for each row in the output point lattice coordinate system.

In this chapter, query processing for image restrictions on streaming RSI data is investigated. The approach uses a Dynamic Cascade Tree (DCT) to index spatio-temporal query ROIs and to efficiently determine what incoming RSI data is relevant to what queries. The DCT exploits spatial trends in incoming RSI data to efficiently filter the data that is of interest to the individual query ROI. Experimental results using random input and Geostationary Operational Environmental Satellite (GOES) data give a good insight into processing streaming RSI and verify the efficiency and utility of the DCT.

Most continuous queries against an RSI data stream include operations to restrict the spatio-temporal data to specified ROIs. Such *Continuous Query (CQ) ROI* are part of more complex queries users issue against a DSMS. Clearly, query ROIs specified by different users may overlap. This is typical for RSI streams which have geographic *hot spots* or regions that are of interest to many users. An RSI stream management system needs to (1) efficiently intersect incoming image data with a possibly large number of query ROIs,

and (2) it should provide a means that allows queries to share the incoming data for further processing. These aspects are illustrated in Figure 6.1 where one ROI R_i is associated with each of the user queries Q_i . In the figure, incoming RSI data intersects with two query ROIs R_1 and R_2 (left). Instead of filtering incoming data for each individual query Q_1 , Q_2 , and Q_3 (middle), a mechanism is needed to determine what incoming image data is relevant to what queries and pipelines the relevant data to subsequent query operators. The DCT is such a mechanism, which filters and streams relevant data to subsequent operators of individual queries, here only Q'_1 and Q'_2 (right). In efficiently processing these spatial restrictions, the organization of the incoming data stream needs to be considered.

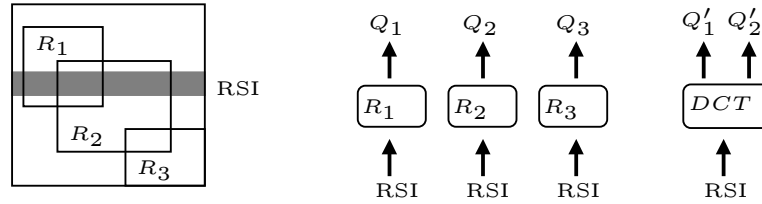


Figure 6.1: Restriction operation on multiple queries

Section 3.2 describes how RSI data is transmitted from the instrument and shows an important characteristic exploited in the following approach. Consecutive packets in a stream of RSI data have close spatial and temporal proximity, though there are some exceptions, for example, where the last pixel of a line in an image is followed by the first pixel of a new image. In the DCT, this spatial trend is used to influence on how multiple queries against a stream of RSI data are processed.

The Dynamic Cascade Tree (DCT) is a space efficient structure to index query ROIs that are part of more complex queries against RSI data streams. The concepts underlying the DCT are introduced using an incoming stream of points. The problem generalizes to solving many *stabbing point* queries. The DCT is then extended to consider rectangular extents from streaming RSI data. The DCT supports the efficient processing of a *moving data stream*. A data stream query is one that, for a window describing the spatial extent of incoming image data (pixel, row, or image), will identify all CQ ROIs that spatially and temporally overlap the data window. This not only allows the pipelining of image data to

those queries to which the data is of interest, but it also facilitates the sharing of image data among queries in the case of non-disjoint query regions. The design of the data structures and algorithms underlying the DCT are guided by the some important requirements, which are typical for RSI data streams:

- The DCT indexes CQ ROIs and is sufficiently small to be kept in main memory. It also has to support efficient insertion and deletion of CQ ROIs.
- The geospatial data stream comes from a single source corresponding to the real-time incoming streaming satellite data. Sequential stream data is usually in close proximity and have a regular trajectory through space. The DCT has to account for both the size and the spatio-temporal trends of the incoming data stream.
- Because of the size of the CQ ROIs and the size and shape of the data in the input stream, selectivity is high. For example, incoming RSI data stream packets intersect about 20% of CQ ROIs for typical GOES applications.

Section 6.1 introduces a simplified version of the DCT for incoming point queries. Section 6.2 describes the DCT for window queries and shows how it filters incoming image data streams for multiple CQ ROIs. Section 6.3 discusses the performance of the DCT in general terms and discusses the parameters affecting performance. In Section 6.4, several experimental results are presented. These include experiments on random data to study the performance under changing input parameters, and experiments closer to real world scenarios using GOES input data as an example. Section ?? explores adding a temporal dimension to the DCT. Section ?? describes related research for similar problem domains.

6.1 DCT for Point Queries

The problem of quickly answering multiple queries on a stream of RSI data is basically solving a normal *stabbing query* [?] for a point. That is, as query result, a stabbing query determines all query ROIs that contain the current point delivered by the RSI data stream. For RSI data, the stabbing points are special in that the next stabbing point

is typically very close to the previous stabbing point. The goal is to take advantage of the trendiness of stabbing points and to develop index structures that improve the search performance for subsequent stabbing queries.

The structure proposed in the following builds an index that is dynamically tuned to the current location of RSI data. For a given point, the DCT maintains the regions around that point where the query result will change. Stabbing queries can be answered in constant time if the new stabbing point has the same result as the previous query and will incrementally update a new result set based on the previous set when the result is different. The structure is designed to be small and quickly allow for insertions and deletions of new query ROIs. It assumes some particular characteristics of the input stream, notably that the stream changes in such a way that many subsequent incoming RSI data will contribute to the same result set(s) to region queries as the current point. Therefore, the cost of maintaining a dynamic structure can be amortized over a large set of queries.

6.1.1 DCT Components

Figure 6.2 gives an overview of the DCT data structure. The figure shows a set of query ROIs a, b, \dots, f , the node, cn , corresponding to the most recent stabbing point from the data stream, and the associated structures for the DCT. The figure describes a DCT that indexes two dimensions. There is no required order in how the dimensions are referenced, and the example shows the vertical (y) dimension being the first dimension indexed in the DCT.

The components of DCT are pleasantly simple extensions to a binary tree. In the example and following pseudo-code, there are two simple search structures, *List* and *2KeyList*. *List* supports INSERT(key,value), DELETE(key), and ENUMERATE(). Keys in *List* are unique for each value. One implementation could use a simple skip list [?] to implement *List*.

The *2KeyList* is incrementally more complex. It supports INSERT(key_1 , key_2 , $value$), DELETE(key_1 , key_2) and ENUMERATE(key_1) using two keys. The combination of two keys is unique for each value. ENUMERATE(key_1) enumerates all the values in the *2KeyList*, entered with key_1 . An implementation of *2KeyList* could be a skip list using

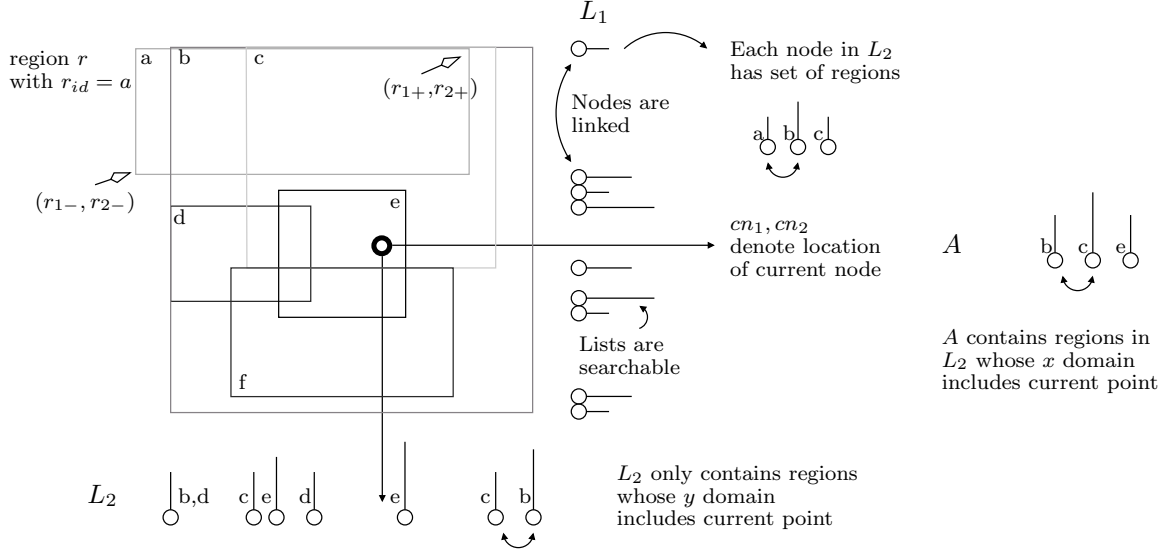


Figure 6.2: Dynamic Cascade Tree (DCT) with query ROIs a, b, \dots, f ; A query ROI, r , is described by minimum (lower left) corner (r_{1-}, r_{2-}) and maximum (upper right) corner (r_{1+}, r_{2+})

key_1 , where each node has an associated skip list using key_2 . With this implementation, for $2KeyList \rightarrow DELETE(key_1, key_2)$, if the deletion causes an empty set in the associated key_1 node, then that entire node is deleted.

List leaf nodes contain ROI ids, r_{id} as keys with a pointer to the query ROI, r as the value. For the $2KeyList$, key_1 is the value of the endpoint, and key_2 is the identifier of the query ROI, denoted r_{id} . The leaf nodes of a $2KeyList$ correspond to the half-open line segments. Leaf nodes of a $2KeyList$ also have pointers to the next and previous nodes in sorted order, allowing for linked list traversal to the leaf nodes. One reason for choosing a skip list implementation is that the forward pointers already exist, and only an additional back pointer is added to a normal skip list.

The DCT maintains a separate $2KeyList$ for each dimension of the individual query ROIs. In the example of Figure 6.2, these are L_1 and L_2 . In addition, the DCT maintains a *List*, A , of all query ROIs, which overlap the current stabbing point. A second structure, cn , maintains pointers to nodes within each $2KeyList$, corresponding to the most current stabbing point.

The $2KeyList$ for the first dimension, L_1 contains the minimum and maximum endpoints in the 1st (y) dimension, (r_{1-}, r_{1+}) , for every query ROI r .

The next *2KeyList*, L_2 contains keys on the endpoints in the 2nd (x) dimension and r_{id} . L_2 does not contain the endpoints of all the ROIs in DCT, but only the ROIs whose 1st dimension (y) overlap with the current node, cn_1 .

If the query ROIs contained more dimensions, additional *2KeyList* structures would be added to the DCT, where each subsequent *2KeyList* only indexes those ROIs that overlap the current point up to that dimension.

In Figure 6.2, cn contains two pointers, cn_1 and cn_2 to nodes within both L_1 and L_2 corresponding to the location of the most recent stabbing point.

A is the final *List* that contains all the currently selected query ROIs that correspond to the current stabbing point query. Just as L_2 contains only a subset of the ROIs of L_1 that contain the cn_1 node, A contains the subset of L_2 where the cn_2 node is contained by the 2nd dimension of each ROI. The L_1 , L_2 , and A structures make up a cascade of indexes, each a subset of the previous index.

6.1.2 Updating Query ROIs

The DCT is initialized by creating the *2KeyList* and *List* structures, adding a starting node for L_2 and L_1 outside their valid range, and assigning cn_2 and cn_1 to those nodes.

Algorithm 6.1.1 shows the pseudo-code for inserting query ROIs into DCT. Insertion and deletion are simple routines. For insertion, a ROI is first inserted into L_1 , and then successively into L_2 and A if the ROI overlaps the current node, cn in the other dimensions. DELETE-REGION() is similar to the insertion, taking the ROI r as input.

Algorithm 6.1.1: INSERT-REGION(DCT, cn, r)

```

procedure INSERT-ITH(DCT,  $cn, r, i$ )
     $I \leftarrow L_i$            comment:  $i$ th  $2KeyList$ 

    if ( $i > \text{dimensions of } r$ )
        then  $A \rightarrow \text{INSERT}(r_{id}, r)$ 
    else  $\left\{ \begin{array}{l} I \rightarrow \text{INSERT}(r_{i-}, r_{id}, r) \\ I \rightarrow \text{INSERT}(r_{i+}, r_{id}, r) \\ \text{if } (r_{i-} \leq cn_i.key \text{ and } r_{i+} > cn_i.key) \\ \text{then INSERT-ITH(DCT, } cn, r, i + 1) \end{array} \right.$ 

    main

    INSERT-ITH(DCT,  $cn, r, 1$ )

```

The structures L_2 and A need to be maintained when new ROIs are inserted and deleted, and for each new stabbing point. Since L_2 contains ROIs overlapping the current node, cn , when a new stabbing point arrives where a y boundary for any ROI in the DCT is crossed, then the L_2 structure needs to be modified to account for the ROIs to be included or deleted from consideration. A similar method needs to be associated with boundary crossings in the x dimension while traversing L_2 and modifying A .

6.1.3 Querying

The algorithm for reporting selected (active) query ROIs for a new stabbing point $np = (np_1, np_2)$, begins by traversing the $2KeyList$ L_1 in the y direction from the current node $cn = (cn_1, cn_2)$ to the node containing np_1 going through every intermediate node using the linked list access on the leaf nodes of L_1 . At each boundary crossing, as ROIs are entered or exited, those ROIs need to be added to or deleted from the L_2 $2KeyList$. When the point has traversed to the node containing np_1 , then traversal begins in the x direction, moving from cn_2 to the node containing np_2 . As with L_1 , when the traversal hits x boundary points, then the entered query ROIs are added to A and the exited ROIs

are deleted from A . When the traversal reaches np , then cn contains pointers to the nodes containing the np , L_2 contains x endpoints to all the ROIs with y domains that contain np_1 , and A lists all ROIs that contain np . A is then enumerated to report all the query ROIs that are affected by the new stabbing point np .

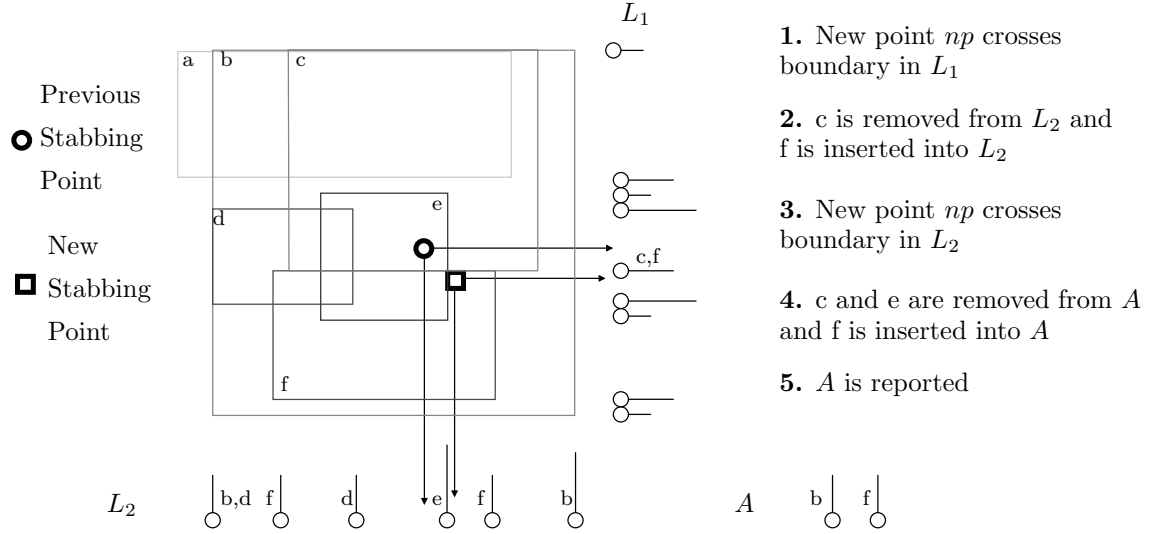


Figure 6.3: Stabbing point moving with new query

Figure 6.3 shows an example of an update of the structures within DCT on reporting ROIs for a new stabbing point. This extends the example of Figure 6.2. In this example, the new point has crossed a y boundary that contains two ROI endpoints, c and f . As the current point traverses in the y direction to this new point, the x endpoints of ROI c are removed from L_2 , and the endpoints of f are added to L_2 . When the endpoints of these ROIs are deleted, the ROIs themselves are also deleted from A . In the example, c is deleted and f inserted into A . After reaching np_1 , L_2 is traversed in the x direction. In the example, this results in e being deleted from A . Finally, A is enumerated, completing the procedure.

Algorithm 6.1.2: REPORT-REGIONS(DCT, cn, np)

```

procedure UPDATE-ITH(DCT,  $cn, np, i$ )
  if ( $i > \text{max dimension of } r$ )
    then return

   $I \leftarrow L_i$            comment:  $i$ -th 2KeyList of DCT

  while ( $np_i < cn_i.key$ )
    for each  $r \in I \rightarrow \text{ENUMERATE}(cn_i)$ 
      do {
        do {
          if ( $r_{i-} = cn_i.key$ )
            then DELETE-ITH(DCT,  $cn, r, i + 1$ )
          else INSERT-ITH(DCT,  $cn, r, i + 1$ )
           $cn_i \leftarrow cn_i \rightarrow \text{PREV}()$ 
        }
      }

  while ( $np_i > cn_i \rightarrow \text{NEXT}().key$ )
     $cn_i \leftarrow cn_i \rightarrow \text{NEXT}()$ 
    for each  $r \in I \rightarrow \text{ENUMERATE}(cn_i)$ 
      do {
        do {
          if ( $r_{i+} = cn_i.key$ )
            then DELETE-ITH(DCT,  $cn, r, i + 1$ )
          else INSERT-ITH(DCT,  $cn, r, i + 1$ )
        }
      }

  UPDATE-ITH(DCT,  $cn, np, i + 1$ )

main
  UPDATE-ITH(DCT,  $cn, np, 1$ )

  return ( $A \rightarrow \text{ENUMERATE}()$ )

```

Algorithm 6.1.2 describes the REPORT-REGIONS() procedure, which reports query ROIs for a new stabbing point, while updating the dynamic structures of the DCT.

REPORT-REGIONS() simply calls UPDATE-ITH() on the first dimension, and then reports all ROIs in the *A List*. The procedure UPDATE-ITH() recursively visits each dimension in the DCT and adds and deletes ROIs into the associated *2KeyList* for that dimension. In UPDATE-ITH(), The first **while** loop traverses the dimension backwards. At

each endpoint, the corresponding ROI is either added or removed from *2KeyList* in the next dimension. The second loop executes a similar traversal in the forward direction, also adding and removing ROIs from the next *2KeyList*. Only one of the **while** loops is executed at each invocation. After traversing to np_i , `UPDATE-ITH()` is called again for the next indexed dimension, $i + 1$. This continues through all dimensions of the DCT. Note that `INSERT-ITH()` will add ROIs into the *A List*, when traversing the final dimension of the DCT. When all dimensions have been traversed, *A* contains all the ROIs that overlap np in all dimensions.

6.2 DCT on Window Queries

The original DCT data structure used a single point as an input *stabbing query* on a number of CQ ROIs [?]. This is typical of pixel-by-pixel input RSI data. Here, the DCT is extended to use rectangular input extents to solve more general stream types. The problem of quickly answering multiple queries on a stream of RSI data corresponds to solving a window query. That is, for a given input data extent, the DCT returns all CQ ROIs that overlap this input. For streaming RSI data, input extents correspond to individual packets of contiguous data that come from the RSI stream. The ROIs correspond to the spatial extent of the individual continuous queries registered in the DSMS.

The most important aspect for RSI data in terms of designing the DCT is that the incoming data stream comes with contiguous data packets that are typically in close proximity spatially and temporally to the previous data. The goal of the DCT is to take advantage of this proximity and develop an index structure that improves the search performance for subsequent parts of the stream.

The structure proposed in the following builds an index that is dynamically tuned to the current location of RSI data. For a given input data extent, the DCT maintains the CQ regions around that window where the result set will change. New RSI input can be processed very quickly if the new data has the same result set of CQ ROIs as the previous data and will incrementally update a new result set based the previous set when the result is different. The structure is designed to be small, and quickly allow for insertions and

deletions of new ROIs registered by the DSMS. It assumes some particular characteristics of the input stream, notably that the stream changes in such a way that many successive data extents from the RSI stream will share the same result set of CQ ROIs. Therefore, costs of maintaining a dynamic structure can be amortized over the incoming data stream. Section 6.3 and 6.4 examine in more detail the actual performance with different CQ ROIs and data stream parameters.

6.2.1 Revised DCT Components

Figure 6.4 gives an overview of the DCT. The figure shows a set of CQ ROIs a, b, \dots, f , and a window corresponding to the most recent data in the RSI stream. Also shown are the associated structures that make up and maintain the DCT. This figure describes a DCT that indexes two dimensions, although the DCT can be extended to more, as shown in Section ???. In this example, the figure shows the dimensions being cascaded in the x then y dimensions.

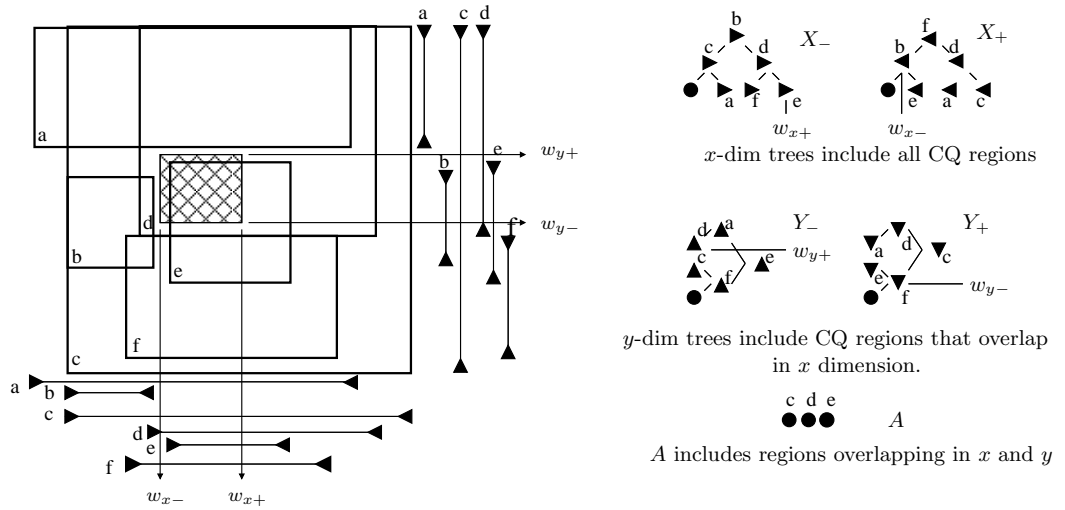


Figure 6.4: Dynamic Cascade Tree (DCT) with six CQ ROIs, a, b, \dots, f . Shown are the indexed ROIs, the current window, and the cascading trees, X_- , X_+ , Y_- , Y_+ , A , and the window node pointers, w_{x-} , w_{x+} , w_{y-} , w_{y+} that make up the data structure.

The DCT maintains separate trees for both the minimum and maximum endpoints of each of the CQ ROIs for each dimension. In Figure 6.4, these are denoted as X_- and X_+ for the x -direction, and Y_- and Y_+ for the y -direction. These are termed *endpoint trees* in the DCT. A final tree, A , maintains an index of all the ROIs that overlap with the

current data in the input stream. The figure shows these trees notionally, emphasizing their ordering and structure; the endpoint trees using one endpoint and the r_{id} , and the A tree only using r_{id} . The values of the nodes contain pointers back to the ROIs, or the associated CQ query in the DSMS.

Which of the CQ ROIs are included in the trees varies with the dimension. The trees for the first dimension, x , contain the minimum (in X_-) and maximum (in X_+) endpoints for *every* CQ ROI. Y_- and Y_+ do not contain the endpoints of all the ROIs in DCT, but only *the ROIs with x extents that overlap the current data stream's x extents*.

This is easily expanded to more dimensions, where each additional dimension adds another set of trees to the DCT that hold the minimum/maximum endpoints of the CQ ROIs in this new dimension. Again, each of these new trees only index those ROIs that overlap the current window up to that dimension.

A is the final tree and contains all the ROIs that overlap with the current window in all dimensions. Just as the trees in the y -dimension contain only a subset of the ROIs that are indexed in the x -dimension, A contains the subset of the ROIs where the y -dimension of the window and the CQ ROIs overlap. A contains ROIs that overlap with the window query in every dimension, and therefore these ROIs intersect the current query window. These tree structures in each dimension make a cascade of indexes, each a subset of the previous index.

In addition, pointers identifying where the current window is located are maintained. This is accomplished by pointing to nodes within each of the endpoint trees for each of the dimensions in the DCT. The pointers match the closest endpoint with a value less than or equal to current window location. These are pointers to existing nodes and not the actual values of the window endpoints themselves. Since nodes correspond to line segments, these identify regions where the current result set is still valid.

Figure 6.4 shows these pointers in each dimension. They are designated as w_{x-} and w_{x+} for the x -dimension, and w_{y-} and w_{y+} for the y -dimension. Note that the minimum window endpoints, for example w_{x-} , point into the tree of maximum endpoints for the CQ ROIs. Similarly, the maximum window pointers are in the trees of minimum endpoints. This is more fully explained in Section 6.2.4 describing how the DCT returns CQ ROIs for

new data extents, but in short imagine as a spatial extent grows in size, new CQ ROIs would become added to the result set as the maximum edge of the extent crosses the minimum edge of new ROIs. Similarly, ROIs would be added as the data extent minimum crosses new region maximum edges. The same idea applies for shrinking edges. Tracking the movement of the spatial extents of the incoming data stream from query to query is how the DCT incrementally maintains a result set of intersecting CQ ROIs.

The individual components of the DCT can be implemented using simple binary tree structures. The structures need to support insertion, deletion, and iteration of nodes in both forward and reverse directions. These can be implemented by Standard Template Library (STL) [?] objects like *set* or *map*, or by similar structures like a slightly modified skip list [?]. The endpoint trees use keys for each node are made up from two values, one corresponding to an endpoint value for each CQ ROI in one dimension, and another corresponding to a unique identifier for each ROI, r_{id} . The two values are combined so that spatial order is maintained. Different CQ ROIs that share an endpoint value are differentiated with the r_{id} 's. Individual nodes correspond to the half-open line segments between two endpoints in a single dimension. The trees all have an additional node with a minimum endpoint that is less than any possible region, shown as a dot in the trees of Figure 6.4. These allow half-open line segments to completely cover a potential location of the incoming data stream.

6.2.2 Initialization

The DCT is initialized by creating the minimum/maximum endpoint trees for each dimension of the DCT. Initial nodes for each endpoint tree are created by adding a node with a value that is less than any possible value for a region in each dimension. These are shown as dots in the trees of Figure 6.4.

The current window location is then created by assigning the pointers w_{x-} , w_{x+} , w_{y-} , and w_{y+} to the appropriate minimum node for each of the endpoint trees in each dimension. The A structure is initially empty.

6.2.3 Inserting and Deleting ROIs

Insertion and deletion of CQ ROIs are simple routines. For insertion of a new ROI r , first the x -dimension endpoints are inserted into X_- and X_+ . The new ROI is checked for overlap with current stream data extents, that is both $w_{x-} < r_{x+}$ and $w_{x+} \geq r_{x-}$ hold, where r_{x+} and r_{x-} are the maximum and minimum values of the new ROI in the x dimension. If the new ROI does overlap the current window, then ROI is also added into trees Y_- and Y_+ . If the new ROI overlaps in this dimension, $w_{y-} < r_{y+}$ and $w_{y+} \geq r_{y-}$, then the ROI is added into A using r_{id} as the index. Insertion maintains the validity of the result set, A with respect to the current data stream window.

Deletion is similar, following the cascade of the DCT, with one modification. The current window pointers need to be checked to verify that they are not pointing to a node that is being removed. If they are then they need to be modified. For example, to delete ROI r , if w_{x+} points to this ROI, then decrement w_{x+} to the previous node in X_- . If w_{x-} points to the maximum endpoint of r , then decrement w_{x-} to the previous point in X_+ . These changes maintain the validity for Y_- , Y_+ , and A . The endpoints of r are then deleted from X_- and X_+ . If the ROI intersects the current window, it needs to be deleted in a similar manner from Y_- and Y_+ , and potentially A as well.

6.2.4 Queries

Figure 6.5 describes the changes to the DCT for new stream data with new extents. As X_- and X_+ are not changed, only the values of Y_- , Y_+ , and A are shown. Figure 6.5(a) shows the query change due to the movement in the x dimension and 6.5(b) shows the change due to the movement in the y dimension. Just as the trees for each dimension and A need to be maintained when new ROIs are inserted and deleted, each new window query requires maintenance of these trees as the window changes its location. These changes are made incrementally on the existing structure of the DCT.

Since Y_- and Y_+ contain ROIs overlapping the current window's x extent, when new data arrives with different x -extents, Y_- , Y_+ , and A need to be modified. These modifications occur when the window region end points cross a boundary of CQ ROI. For each

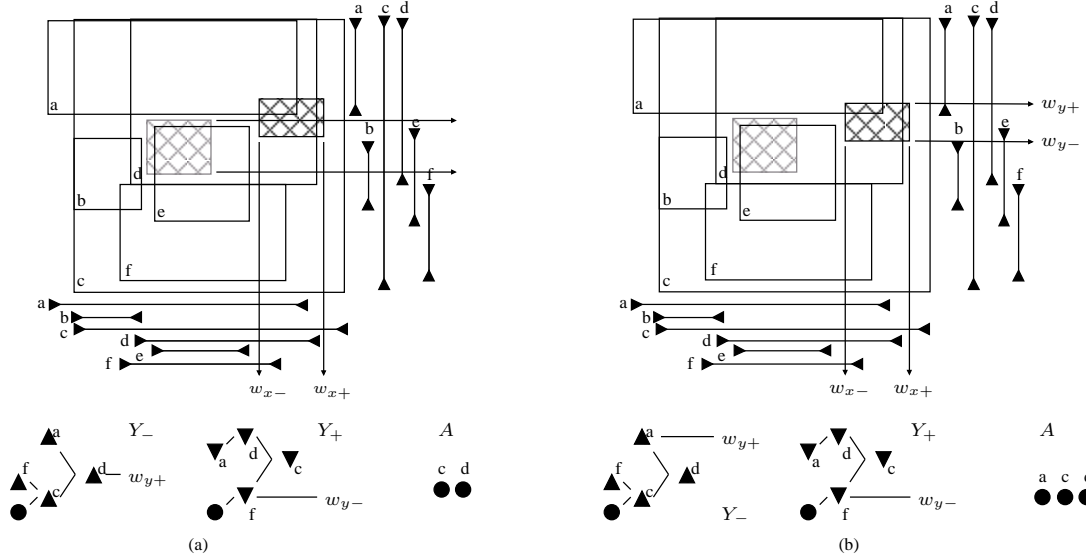


Figure 6.5: Query on a new region. The original window position (as in Figure 6.4) is shown in grey, and the new region in black. (a) shows the updated Y_- , Y_+ , and A structures after moving the window in x . (b) shows the final structures after moving in y .

x ROI boundary in the DCT that is crossed, the y -dimensional trees need to be modified to account for inclusion or deletion of this new ROI. A similar method needs to be associated with boundary crossings in the y -dimension, requiring modification of A .

The algorithm for reporting overlapping CQ ROIs for window, $w = ((w_{x-}, w_{y-}), (w_{x+}, w_{y+}))$ begins by traversing the endpoint trees in the x direction. Figure 6.5(a) shows an example where the new query region has increasing minimum and maximum x edges. An increasing minimum edge implies that the movement can result in ROIs being deleted from the result set. The movement of w_{x-} is tracked by incrementing along the nodes of X_+ . Moving w_{x-} to a new node corresponds to a boundary crossing of the window minimum with a region maximum, and in this case corresponds to one ROI, e , being removed from Y_- and Y_+ . This deletion is cascaded to A as well. The increasing maximum, w_{x+} , corresponds to a movement that would add ROIs. However, in this case, although the window maximum is greater, no minimum nodes are crossed, and w_{x+} remains pointing to the minimum node, of ROI e , as in Figure 6.4. The net result of moving the window in the x direction is that ROI e is removed from Y_- , Y_+ , and A .

Next, the movement of the window in the y direction is accounted for as shown in Figure 6.5(b). In this case, although the y minimum of the window has changed, no

maximum boundaries are crossed, and w_{y-} remains pointing to the maximum value of ROI f . Moving the window maximum in y , however, results in the minimum edge of ROI a to be crossed. w_{y+} now points to the minimum edge of a , and a is added to the result set A .

This example shows one type of movement of the window region, but the same algorithm works for all changes of the window region. The window can grow and contract on all sides, or move in any direction. The algorithm works the same way; each edge of the window query is handled separately either adding to or deleting from the subsequent trees and the final result, based on the direction of movement of the edge.

There is one caveat to this algorithm for updating the DCT based on the movement of edges individually. Without modification of the above algorithm, the expansive movements of the new window query need to be performed before the shrinking movements. Large movements of the window can produce movements across both edges of an CQ ROI. Expanding first prevents a ROI from being deleted first on a shrinking movement, and then erroneously inserted on expansion. Executing deletions first however decreases the size of the trees in the DCT. To allow shrinking movements first, during the expansive movements, range checks are done to verify the ROI indeed belongs in the overlapping set before inserting it into subsequent trees.

6.3 DCT Performance

The window query performance of the DCT is highly dependent on the location of the ROIs, the properties of the incoming window queries, and the interaction of these parameters. For queries over n ROIs, with k being the result count, the execution time for window query can range from $O(k)$ in the best case, when the result set is the same as the previous set to $O(n \lg n + k)$ in the worst case, when every ROI is entered in a single movement of the incoming spatial extents. Before looking at some experimental results, there are some simple guidelines to consider for the application of the DCT.

6.3.1 ROI Insertion and Deletion Cost

The DCT data structure is small and robust to many insertions and deletions of CQ ROIs. Insertions and deletions take $O(\lg n)$ time as the query ROI is potentially added to the endpoint trees in some constant dimension and A . These trees are simple to maintain dynamically in $O(n)$ space.

6.3.2 Cost Versus the Number of Boundaries Crossed

The DCT is designed for trending data, which can be measured by the number of ROI boundary crossings from one window query to the next. The structure works best when the number of ROI boundaries crossed by successive queries is not large. When no boundaries are crossed, then no internal lists are modified, and reporting ROIs runs in $O(k)$ time. When a boundary is crossed in movement of the window query, then each ROI with a crossed boundary needs to be inserted into or deleted from subsequent endpoint trees. This is true for at least one set of endpoint trees or A , even for CQ ROIs whose domains in other dimensions do not overlap the new window and thus do not contribute to the final result. The cost of a window query in this case can be as high as $O(n \lg n + k)$, since all ROIs could potentially be inserted into dimensional trees during one window query.

The DCT data structure does indexing somewhat lazily in the sense that for insertions of new ROIs that do not overlap the current window, only the first dimension is indexed, and not the other dimensions. The indexing costs on window queries can be thought of as finally incurring those indexing costs. However, the problem with the DCT is that these costs can occur many times in the motion of the input stream. Rather than indexing ROI values once, the DCT re-indexes a subset of points multiple times as boundaries are crossed. The hope is that many successive window queries will be in the same ROI with the same result, and that the low cost for those queries will make up for the extra cost of maintaining a dynamic index.

6.3.3 Trajectory of the Trending Windows

Another aspect affecting performance is the movement trajectory of the window queries. For example, consider a DCT in two dimensions like that shown in Figure 6.4 with trajectories that are monotonic in the x and y dimensions. In this case, ROIs are inserted into the y endpoint trees and A at most one time, and once more potentially for deletion. The total time for maintaining the DCT then is at most $O(n \lg n)$, fixing a bound on the dynamic maintenance costs. For m window queries over that trajectory, the cost of all queries would be $O(n \lg n + mk)$ where k is the average number of results per query. In comparison, for two-dimensional segment tree implementation, the total cost would be $O(n \lg^2 n + m \lg^2 n + mk)$, where $n \lg^2 n$ is the cost of building a static segment tree. Similar savings exist for trajectories that are generally monotonic, such as most RSI data streams.

On the other hand, more erratic window trajectories can result in poor performance. Consider a window movement that repeatedly crosses all n ROI boundaries. Each query iteration would require $O(n \lg n)$ time, as the y dimensional trees are repeatedly made up and torn down.

Also, the DCT as described in Figures 6.4 and 6.5, which indexes on x and then y , favor window that trend in the y direction over windows that trend in the x direction. This is because boundary crossings in the x dimension have to modify more trees, and the trees tend to be bigger, so they take more time. Movements in the y dimension only modify A , a single tree that is the smallest tree in the DCT. Also, movements in the x direction result in insertions into Y_- and Y_+ which can be somewhat wasted in the sense that these ROIs may never contribute to a final result set, whereas boundary crossings in the y direction will always affect the result set. Although, the time it takes to update the DCT due to a boundary crossing is $O(\lg n)$, for either x or y dimension, y modifications will always be faster.

This shows that order in the cascade is very important, and dimensions that see more boundary crossings for successive window queries into the DCT should be pushed deeper into the structure. Boundary crossings are of course dependent on the trajectory of window queries and the organization of the ROIs in the DCT. Section 6.4 reports on

indexing tests on the DCT and quantifies some of the performance features identified in this section.

6.3.4 Skipping Worthless Insertions

When the next window of the input stream trends a long way with respect to the number of query boundaries traversed, then the time for reporting queries goes up to at least the number of ROIs contained in all the boundaries crossed. ROIs that are both entered and exited in the course of a single traversal to a new window are even worse. Their endpoints are needlessly added, then deleted from the DCT, at a cost of up to $O(\lg n)$, and never queried. This can easily be remedied, but for clarity was left out of the initial UPDATE-ITH() algorithm. When encountering a ROI at a boundary crossing, check that the ROI will remain a valid ROI in the first dimension when the window has finished its traversal before inserting into the next structure. This prevents wasted index modifications, but does not help with the basic problem of long traverses or input windows that cross back and forth across expensive boundaries.

6.4 Experiments

To test the performance of the DCT, two experimental setups were used. The first tests are on randomly moving data stream and random CQ ROIs, with a number of variations on specific input parameters. The second experiment was developed to more closely replicate the queries made on a DSMS with GOES RSI data as input, and more realistic ROI parameters.

Comparisons were made to an existing in memory R*-tree implementation from the *Spatial Index Library* [?]. For better comparison, the DCT implementation included some components from this same library. All trees in this implementation of the DCT were simple set objects from the Standard Template Library (STL) [?]. The total size of this experimental DCT implementation was about 600 lines of C++ code. All experiments were run on a single 1.6 GHz Pentium M CPU with a 1MB L2 cache and 512MB of main memory.

6.4.1 Random Continuous Queries with a Random Data Stream

The more extensive tests were run using a set of CQ ROIs that were randomly located throughout a two dimensional space, with some variation of parameters as shown in Figure 6.6(a). For most tests, the input data stream moved randomly though the region, with the default parameters also shown in Figure 6.6(a). The CQ ROIs were distributed uniformly throughout a square region. Some region parameters are modified for certain tests, these tables show the default values. Aspect is the ratio of lengths $\frac{x}{y}$ of the CQ ROIs or the spatial extent of data from the input stream. For most tests, the window query moved in a random direction from one query to the next.

Sometimes, the extents of the input stream would trend off test area. When this occurred, a new starting point within the region was chosen to reset the window location.

These experiments do not correspond closely to any real world application, but are instructive in testing the performance of the DCT with respect to various parameters. All experiments plot the average response time for a window query as a function of a single parameter. All experiments were run using 4K and 16K CQ ROIs.

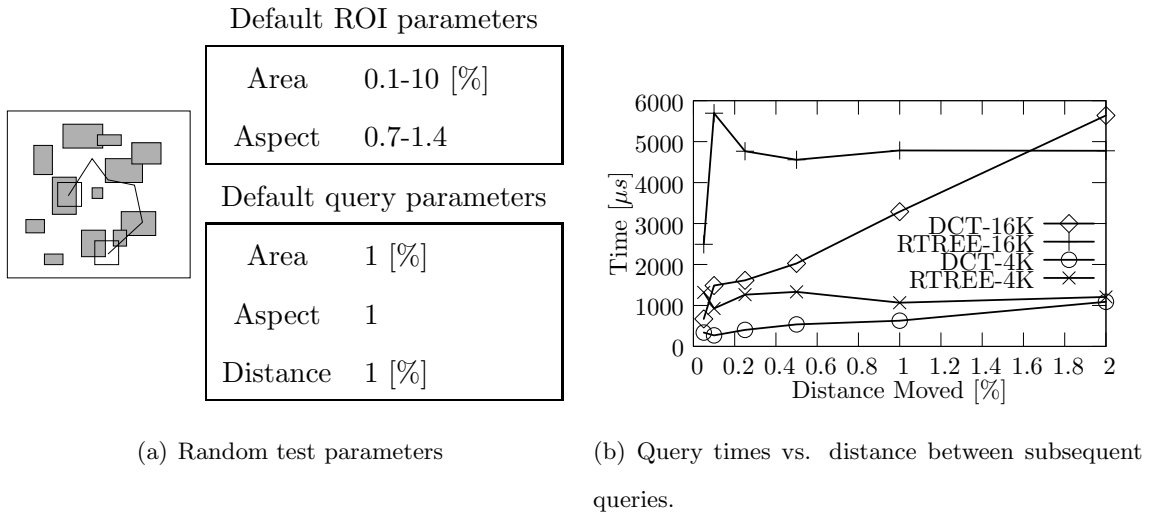


Figure 6.6: Test parameters and distance test

Figure 6.6(b) shows the average window query time while varying the average distance moved from on window query to another. The direction of the move was randomly determined. This is one of the most important parameters to consider when using the DCT.

The window queries must come close to one another for the index to work effectively. As expected, while the R*-tree is insensitive to this parameter, the DCT performance is very dependent on the distance moved for a window query. As the distance between window queries increases, the amount of information that can be shared in the trees of the DCT become less and less. For data in the input stream at a size of 1% of the total area, a move over a distance of 2% virtually eliminates all sharing of information between queries. At what point the distance between queries becomes limiting is dependent on other application parameters. For example, as more ROIs are added into the DCT, more boundaries are likely to be crossed over the same distance moved.

The size of the stream data extents also affects the performance of the DCT. In general, larger extents in the data from the incoming stream would be expected to share more results from query to query and therefore improve the performance of the DCT. Figure ?? compares the DCT to the R*-tree for different size extents in the incoming stream. The average response time increases for both implementations, but less so for the DCT. For this experiment, the result sets for the window queries increases with window query size as well, so it is expected that the response time increases with stream data extents. The DCT results are somewhat artificially high for another reason. As was described, the query window is relocated to a new random location when the window trends off the experiment's region. Since this is more likely to occur with larger stream extents, more of these relocations occur in those experiments and since each relocation corresponds to larger query distance movements in the DCT, this increases the average distance between data in the stream.

As described in the introduction, the intent of the DCT is for use in streaming RSI, and these usually entail incoming data packets of a row, or small number of rows of data. Since these correspond to the extents of the individual data in the incoming geospatial data stream, DCT queries can have a high aspect ratios, that is, $\frac{x}{y} \gg 1$.

Figure ?? shows the change in response time as a function of aspect ratio. The R*-tree performance is slightly degraded as the aspect ratio increases, while the DCT performance remains insensitive to this parameter.

Section 6.3 described how the DCT can be affected by the average trajectory of the successive data from the stream. Performance is expected to improve as window trajectory

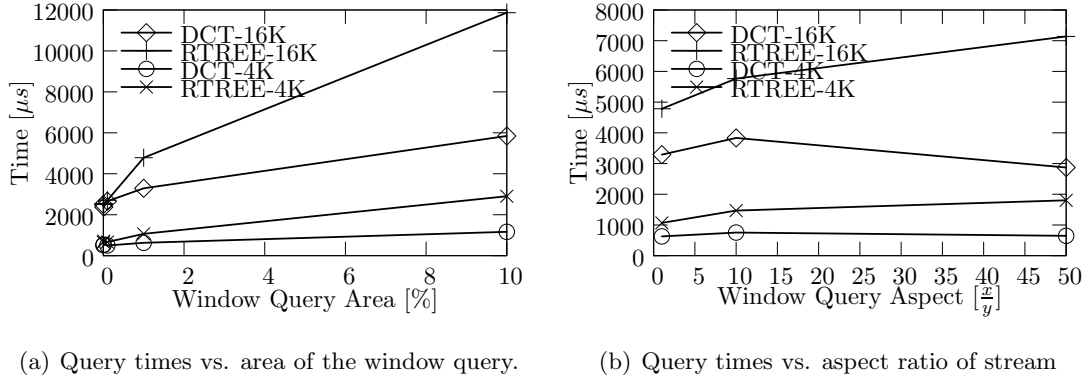


Figure 6.7: Area and aspect tests

aligns with the dimensions farther down in the DCT cascade. Figure ?? illustrates this gain. As expected, the R*-tree is insensitive to this parameter, whereas the DCT's performance increases by a factor of two based on the trajectory of the query window.

Figure ?? shows the affect of the average size of the CQ ROIs on the DCT performance. As the size of the ROIs increases, the average response time increases for both search structures, due in part to the fact that more ROIs overlap with each individual window query.

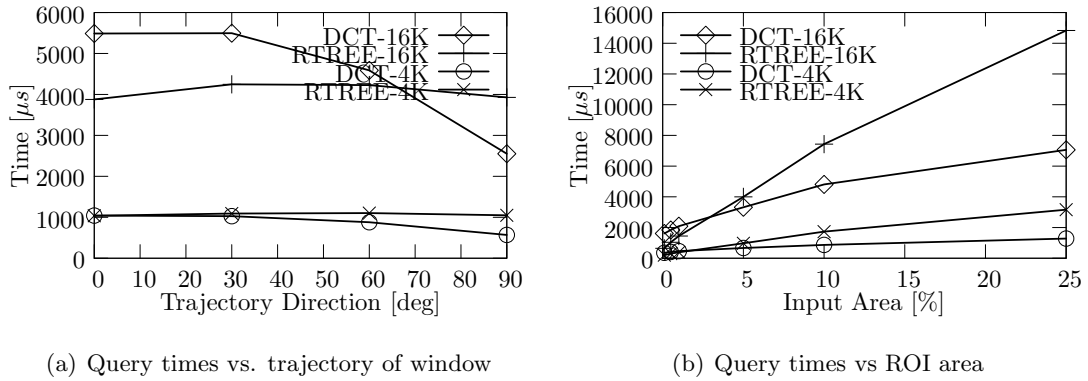


Figure 6.8: Trajectory and ROI experiments

The R*-tree slows down more than the DCT, since more of the results from successive window queries can be shared from query to query. This is an important consideration for applications where the CQ ROIs are large.

In summary, the DCT performance is affected by the number of ROI boundaries

crossed for successive window queries, and the trajectory of the window queries. Also the size of the CQ ROIs and input data extents affect performance. Often the performance changes are different than seen in the more typical R*-tree. How the DCT performs under more realistic situations is described next.

6.4.2 GOES Experiment

From the discussion of the DCT performance and the experimental results for random windows and ROIs, the parameters associated with higher performance of the DCT can be anticipated. These include: (1) relatively large extents of data in the stream, possibly with a high aspect ratio, (2) small distances from data to data in the stream, and (3) a regular trajectory of the data in the stream, with the DCT tuned to that direction. The DCT also works well with large CQ ROIs, having a relatively high selectivity. These are all aspects of a typical satellite RSI data stream.

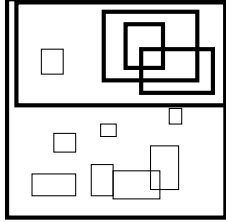
For a more realistic scenario for the DCT, an experiment using queries for weather satellite data, from NOAA GOES was developed. Figure 2.1 (page 18), included an overview of the GOES sensors. The experiment uses Imager data as input. GOES offers a continuous stream of data for ROIs ranging from the continental United States to a hemisphere centered near Hawaii. Query regions also tend to be approximately square regions covering relatively large extents.

GOES streaming RSI data is well suited to the DCT. In each frame, the data trends only in the downward direction and incrementally. Therefore, endpoints are only added into the X_- , X_+ structures one time, limiting the maintenance time to $O(n \lg n)$ for each frame. For normal GOES data, the starting column of each row does not change within a frame and the Y_- , Y_+ and A structures are the only structures that change in determining which ROIs overlap any given set of incoming streaming rows of data.

For the experiment, five thousand (5,000) ROIs were indexed. Rather than randomly locate these ROIs throughout the image domain, they were preferentially located around a small number of “hot spots” in the hemisphere. This more closely relates to real world scenarios, where specific parts of the RSI data is requested by a large numbers of users. Although the general area of the queries was fixed to a small number of locations,

the individual ROI centers, total sizes and aspect ratios were varied for each ROI. This corresponds to many users requesting slightly different particulars for a general ROI. A small fraction of random ROIs was also included in the experimental setup. Table ?? describes the parameters for the CQ ROIs. The contributions table shows the various general regions and their contributions to the total ROIs. The other table shows the variation of the individual ROIs. The figure graphically depicts the experimental setup.

Table 6.1: Query ROI statistics

Contributions		ROI Variations	
30%	North America	Area	0.8-1.2
15%	Western Seaboard	Aspect	$\pm 10\%$
15%	Northern Hemisphere	Center	$\pm 10\%$
15%	Hemisphere		
9%	CA		
9%	Mexico		
2%	Random (1000x1000)		
5%	Random (6000x4000)		

Ten thousand (10,000) window queries were performed on the CQ ROIs. The RSI data blocks corresponding to the input data stream were taken from a sample of the GOES West Imager instrument, each query corresponding to 8 rows of data. Depending on some assumptions about other aspects of a DSMS indexing GOES data, this corresponds to somewhere between 7.5 to 60 minutes of streaming time.

Again, the DCT was compared to the in-memory R*-tree implementation. The R*-tree implementation required 48 seconds to perform all the queries, for an average query time of 4800 $[\mu s]$. The DCT performed the queries in 9 seconds, 900 $[\mu s]$ per query. It is also interesting to note that of those nine seconds for the DCT, only 0.88 seconds were used in the maintenance of the DCT and its structures. Nearly all the other 8.22 seconds involved copying the final result *A* structure into new lists for output. This was done to match the output of the R*-tree, but scenarios can be envisioned where applications access

the A structure directly after a query, reducing this overhead cost.

With respect to GOES data, the reason for extending the structure to account for more general trending data has to do with projection systems. Like most remotely sensed imagery, the original data is in its own projection system, while users would like to have data streamed in a projection system of their choice, UTM for example. Re-projection is an expensive operation that should be avoided for data not answering a specific query. Allowing users to specify queries in their projection system requires have a number of DCT structures, one for each input projection system. All query ROIs will be projected into the GOES system, and roughly described with a bounding box. For each projection included with a ROI identified in this first DCT structure, the bounding box of the incoming rows of GOES data will be re-projected into that system, and represented with a new bounding box. This box will be used to identify the final intersections of the incoming GOES stream with the ROIs. In those other projection schemes, the incoming rows will still trend in a smooth line, but in both x and y coordinates.

6.5 Temporal Dimension

The focus for both the discussion and the experiments has so far been on the spatial dimensions. This is primarily because the DCT takes most advantage of the spatial organization of the incoming RSI data stream. As has been discussed, adding additional dimensions is a simple extension of another intermediate set of endpoint trees added to the cascade in the DCT. The most obvious choice for another dimension is the temporal domain, since most restrictions in user queries include a temporal restriction as well.

Figure ?? shows an extension of the DCT in the temporal domain. Here, time is the first dimension of the cascade. As discussed in Section 6.3, it is best to order the dimensions so the most varying is on the deeper levels of the DCT. For relatively low bandwidth pixel-by-pixel RSI streams, the monotonic increase of time would make it a candidate for the level directly before the A tree. However, for most RSI data, putting time as the first dimension makes most sense, because most of the variation comes in the spatial domains.

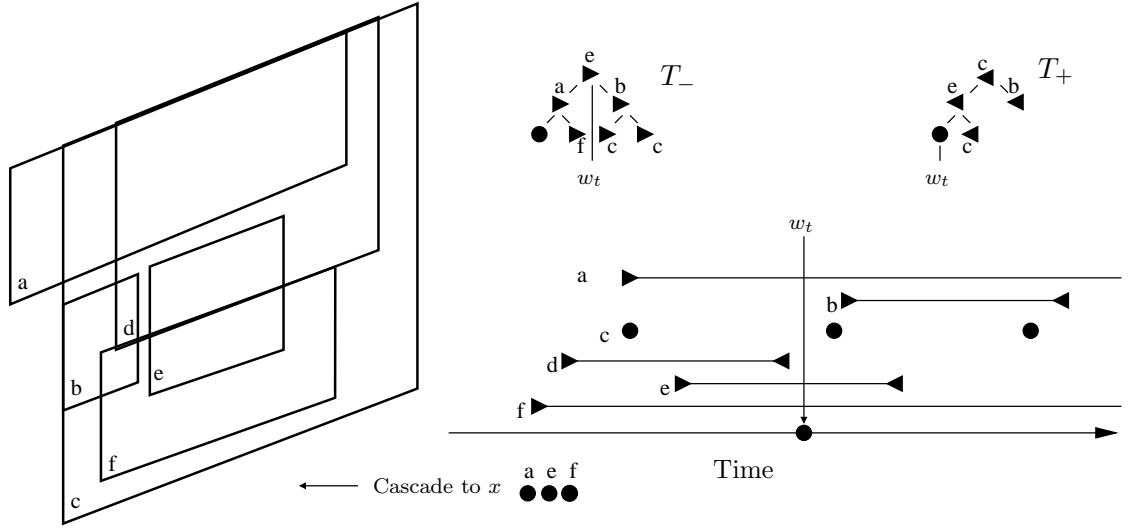


Figure 6.9: Temporal dimension added to the DCT. Shown are region's temporal extents, the endpoint trees T_- and T_+ , and the temporal node pointer, w_t .

There are some features more specific to the temporal domain. One possibility is for unbounded regions, two of which, a, f , are shown in the figure. These correspond to CQs without a specified ending time. Unbounded regions can be modeled in the DCT by simply not including endpoints in the T_+ endpoint tree. The temporal domain is more likely to contain a discontinuous region, for example, ROI c in the figure. This corresponds to a periodic query, for example, a request for the noon satellite image for a period of days. Although more problematic for CQ extents, discontinuities can generally be handled with multiple entries in the DCT endpoint trees.

One nice feature with making time the first structure of the cascade is that it can also serve as providing a structure to prune ROIs that have expired. When w_t crosses over the maximum time endpoint for any of the CQ ROIs, the DCT will cascade a deletion through the remaining trees. These ROIs can then also be removed from the temporal endpoint trees as well, removing those ROIs completely from the DCT. An interesting point to this is that w_t will always be pointing to the minimum node of the T_+ endpoint tree, as finished ROIs are deleted. Because of the monotonicity of time for the incoming data stream, besides using the standard trees for the temporal dimension, more efficient structures like heaps could be investigated for this dimension.

6.6 Non-spatial Multi-dimensional Data

The focus of the DCT has been on spatial data. However, these techniques could be applied to a general multi-dimensional data space. The obvious modifications could be made and extended to n dimensions as outlined above. One important issue to address is the order of the cascade of *2KeyList* structures. As described in Section 6.3, if ROIs are dispersed equally, it is generally best to move the most varying parameter to the end of the cascade, and move the least varying to the top. This structure is also appropriate for range queries over a single dimension over trending data by maintaining only the top *2KeyList*. Since the index sizes are relatively small, it is conceivable that a system could consistently maintain one dimensional DCT structures, and then dynamically begins to build 2 or n dimensional structures when queries requesting such ROIs are instantiated. The advantage of this method is that the structures are no longer maintained as the queries requesting those ROIs are deleted.

6.7 Related Work

The structure and performance characteristics of the DCT have previously been described [?, ?]. The primary goal of the DCT is to allow for many spatial ROIs to be simultaneously evaluated as the stream of input spatial data arrives in the DSMS. The obvious application of the DCT is in multi-query evaluation, where a single operator uses the DCT structure to perform the spatial restrictions for a large number of queries.

Similar multi-query optimizations have been discussed in streaming databases where they have been described as group filters [?, ?] and in spatio-temporal databases where this optimization has been described as query indexing [?].

Many data structures have been developed for one and two dimensional stabbing and window queries including among others, interval trees, priority search trees, and segment trees [?]. Space partitioning methods of answering window queries include quadrees, hashes, and numerous variants of R-trees.

The common methods for solving window queries in two dimensions are multi-level segment trees [?] and R-trees [?]. It is difficult to modify either method to take advantage

of trending data. Using multi-level segment trees, one dimension of the ROI is stored in a segment tree, while the second dimension is indexed with an associated interval structure for each node in the first segment tree. Storage for these structures can be $O(n \lg n)$, with query times of $O(\lg^2 n)$. Dynamic maintenance of such a structure is more complicated and requires larger storage costs [?]. It is difficult to modify the multi-level segment tree to improve results for trending data. If the input window moves a small distance, which doesn't change the query results, it still would take $O(\lg n)$ time to respond. That is because even if every node in the multi-level segment tree maintains knowledge of the previous point, it would still take $\lg(n)$ time to traverse the primary segment tree would still need to be traversed to discover that no changes to the query occurred.

R-trees solve window queries by traversing through minimum bounding rectangles that include the extent of all regions in the sub-tree, generally with good performance. Since these rectangle regions generally overlap, there can be no savings from knowing the previous window, as there is no way to know if an entirely new path through the segment tree needs to be traversed. R^+ -trees [?] style trees may be modified for trending windows, since the minimum bounding rectangles are not allowed to overlap and maintaining the previous window can help verify a query hasn't left a particular region. R^+ -trees, however, have problems with redundant storage, dynamic updates, and potential deadlocks [?].

Another approach similar to the DCT described in this chapter is that of the query index [?, ?]. The query index builds a space partitioning index on a set of static query ROIs and at each time interval, allows a number of moving objects to probe the index to determine overlapping queries. Experiments in main memory implementations show that grid-based hashing of query ROIs generally outperform R-tree or quad-tree based methods. The query index is most effective for small ROIs and query points, and experiments show performance degradation with larger ROIs or high selectivity query windows [?].

All of these indexes described above anticipate a large number of moving objects to be indexed against the query ROIs. The RSI application described above is different in the sense that the DCT index is designed for a single or small number of moving objects, where the input rate of data for that moving object is very large. In this application, the desire is for a small index that can efficiently route that high volume data stream to the

query regions, rather than indexes that are interested in the location of the moving objects.

The DCT is also an incremental approach to answering queries, where updates are made to a current query result. SINA [?], describes an incremental method to solving the problem intersecting moving objects, though the approach focuses on efficient integration of incremental query changes with disk-based static queries, and a complete main-memory implementation would be more similar to the query index approach.

In another sense, the DCT is a method of dynamically maintaining boundaries around a current window where the current result set is valid and identifying where this result set will change. Another method of dynamically describing a neighborhood of validity for a query using R-trees was proposed in [?], which builds an explicit region of validity around a current query point. This method is meant to minimize transmission costs to a client, and the technique makes many additional queries to an R-tree style index to build these regions of validity.

Chapter 7

Multi-Query Execution Using an Online System

Chapter 5 described a multi-query optimization system using an existing GIS application. While this system is promising in its approach, a DSMS system designed specifically for RSI data would behave differently, operating on smaller discrete parts of an image directly as RSI data arrive. This chapter will provide more detailed information on the various components of an online DSMS. It is meant as a high level implementation design. The focus of the discussion will be on the requirements particular to the processing of the GOES GVAR data.

The data from the stream will be a manipulated a single row at a time. This is most consistent with the data stream arrival pattern and will allow for the smallest in-memory footprint of the system as a whole. In addition, as was discussed in Section 3.2.1 (page 32), row-scan ordering allows for processing benefits especially in the presence of image restrictions. Figure 1.1 (page 2) described an overview of the *GeoStreams* architecture. Figure ?? revisits this overview in more detail. The *Query Manager (QM)* is the predominant subsystem in the architecture. It is supported by a *Row Memory* subsystem (Section ??), which provides an interface to create, subset, and destroy rows of data in the system. Another subsystem, *GVAR Input* (Section ??), handles the interface to the GVAR data stream and creates rows for input into the system. The third supporting subsystem,

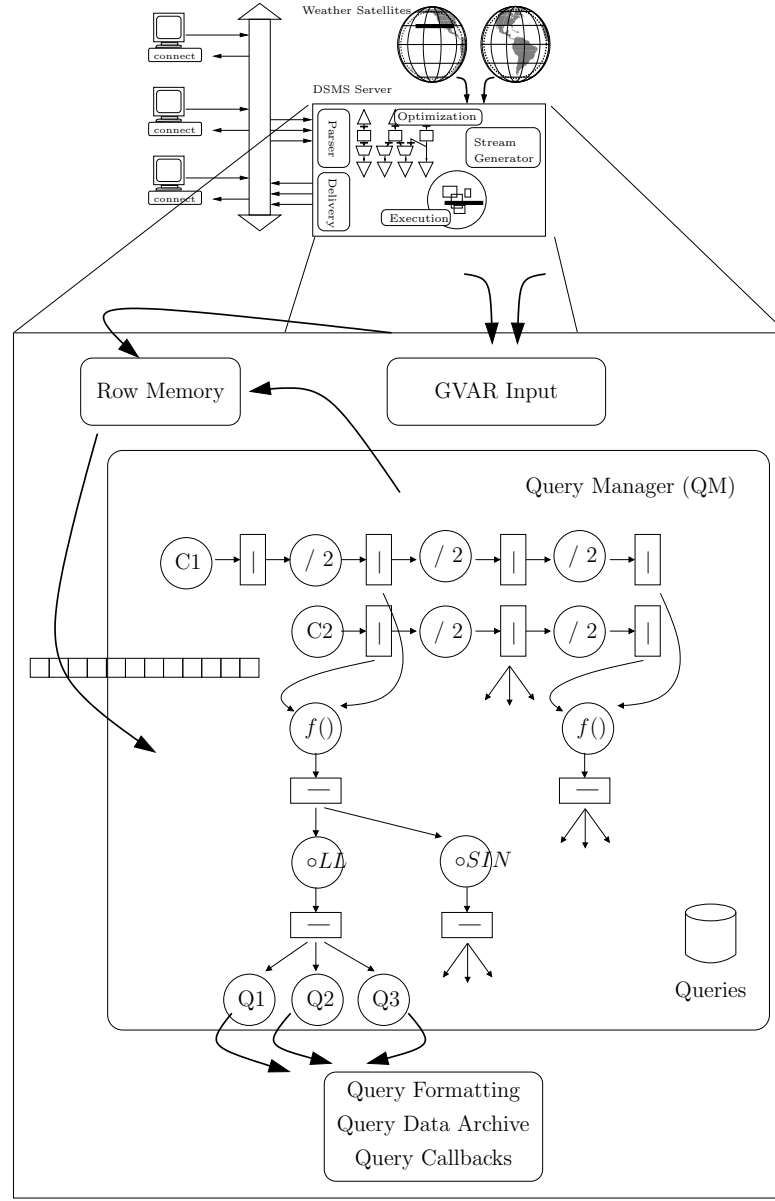


Figure 7.1: *GeoStreams* system overview

Query Data Management (Section ??), provides support for making the data available to the users. This includes formatting the data and providing archive and callback support to users and their queries.

The QM creates the QEP for the system, creates needed operators, and executes the plan. This chapter introduces the individual operators and briefly describes some of their requirements and behaviors. Section ?? describes the restriction operation, based on

the DCT, designed to allow multiple restrictions to be satisfied simultaneously. Section ?? includes a general class of induced operators. Section ?? describes the halving operator, and Section ?? provides details on the spatial transformation operator. Many of these topics have been discussed in more detail in previous chapters.

The general description of the various components of the *GeoStreams* architecture are agnostic to the choice of programming language. However, for some specific implementation issues, the system described assumes a rapid-prototype implementation based on the Perl programming language [?]. Perl might not initially be considered the most appropriate language, particularly because of the overhead on untyped variables for manipulations. However, all operations on the RSI data will be performed using the Perl Data Language (PDL) [?], which is a strongly typed addition to Perl. PDL is an array and mathematics package designed for number manipulation.

One source of computation time is due to the DCT index (Section 6). The DCT will be used in restriction operations (Section ??) and also as an important component of the geometric transformation (Section ??). The code for the implementation of the DCT is written in C++ and linked into the Perl application through an Application Program Interface (API).

The queries themselves are maintained in a PostgreSQL database [?]. In reality, the query database is small enough to allow an in memory implementation, but externalizing the database allows for more diagnostic testing on the system.

7.1 Rows and Row Memory

The basic datum in this system is the row or row tuple. Row objects need methods to allow access to the individual pixel values as well as describing the geographic point set that they define. In addition, there are a number of memory related processes related to the rows. They need to be created, copied, and slices of rows need to be defined.

In the process of executing a query plan for a complex set of queries, many intermediate data rows need to be created. Some row data is newly created, while data from restriction operations are created from an existing row of data. Since the same query plan

will be in operation for many rows of input GVAR data, overall, the total amount of memory used will be fairly constant, but the individual operators will constantly be creating and destroying rows of data.

There are three main allocation requests; requesting a new row of data, requesting a copy of the row and requesting a subset of an existing row. Operators needing row objects specify the type of data that they are requesting. They also define the geographic point set of any new data row. This allows operators to keep track of what specific row and in what coordinate system a row belongs.

When requesting a data subset the row memory system should not create new data, but only a pointer to the existing data. Operators can also copy an existing row of data. When copying rows of data, or creating rows of data from subsets, datatype and geographic point set information is automatically derived from the source row.

The row is implemented as a subclass of the PDL variable. All data creation methods and data slicing that are available in PDL are accessible for the row. In particular, data slices are implemented virtually, pointing into the data structure of the original row.

Information relating to the geographic point set of the row is stored in the PDL header and automatically copied to slices of data. Specifically, three items are stored in the header; the Coordinate Reference System (CRS) identification [?], the location of the upper left pixel, and the north and south resolution between pixels.

In terms of managing the row memory, there are some advantages to this strategy. Since Perl and PDL use a reference counting scheme for memory management, operators do not have to worry about explicitly allocating and de-allocating variables. This significantly simplifies operator scheduling and management of queues of row data within operators.

However, a problem is that no consideration is made to the fact that many of the rows created and destroyed are of the same size and type and could effectively be reused rather than deleted and recreated. However, the implementation does not preclude a more efficient memory implementation. Specifically, the row class can redefine both the creation and deletion of row objects. In order to speed up the running time of the application, a specialized memory manager can be developed that does not burden the system memory allocation with all these requests, but instead reuses previously allocated memory.

A simple implementation of this would be that when the PDL memory manager attempts to destroy a row, the object is instead cached. At a later time, on the creation of a new row, if a cached variable of the same type and size exists, it is reused rather than being created again. The row cache could be cleaned out when new queries are formulated or when new frames enter the system. Since many input rows would have similar intermediate and output rows, the amount of sharing would be considerable.

7.2 GOES GVAR Subsystem

The GOES GVAR subsystem receives the raw input GVAR data stream, converts the data into an internal representation of rows, then submits the data to the QEP. Section 2.1.2 (page 16) contains further information describing the GOES satellite and the GVAR stream characteristics. There are only a few simple interactions of the GVAR input operator.

Although basically a simple operation, there are a number of special caveats for this operator. First, the operator must decompose the GVAR stream into a number of different row types based on the channels in the Imager. Also, this operator must decode the GVAR metadata and encode the frame and geographic point set information that is used in creation of rows. Finally, individual detectors from the Imager, corresponding to different sets of rows in the image, have different radiometric characteristics and need to be calibrated prior to being input into the system.

The GVAR operator collects satellite information asynchronously by monitoring an open socket connected to the system that receives and decodes the data from the antenna. Entire blocks of data do not arrive at one time, so the operator accumulates socket packets until an entire block of data is retrieved. Header information in the GVAR stream allow the operator to determine the block size. On a complete block, the data is read into a special sub-class of the row object. Additional header information allows further sub-classing of the GVAR block based on its type. One particular block, the GVAR doc block, carries metadata information regarding the current frame and scan within the stream. These doc blocks are submitted to the QM, which can use the information to calculate new query plans

between GVAR frames of data, predetermine the overall extent of an individual frame, and determine if any errors in reception occurred during the frame download.

The other blocks of interest are those that carry image data. Again, these GVAR blocks are sub-classed based on the channel in question. The GVAR stream divides the image into Scans, which have eight visible rows (channel 1), four rows of data in the infrared and temperature channels 3-5, two rows of data in the infrared channel, and one row of shortwave data, channel 2. These correspond to the various sizes of the images, so that a single scan covers the same region for all channels, despite the different sizes. This means that the order of rows that are sent to the QEP is somewhat non-intuitive. Within each channel, the images are sent in row-scan order, but the rows are interspersed among one another.

From GVAR blocks, new rows of data are created. This is for two reasons. First, the data as it arrives from the server has 10 bit data words, and second, the GVAR operator performs radiometric calibrations on each row of data separately, although these effects are small. Figure ?? shows a comparison between the detector pixel value and its radiance value for each of the 8 visible detectors in the GOES satellite.

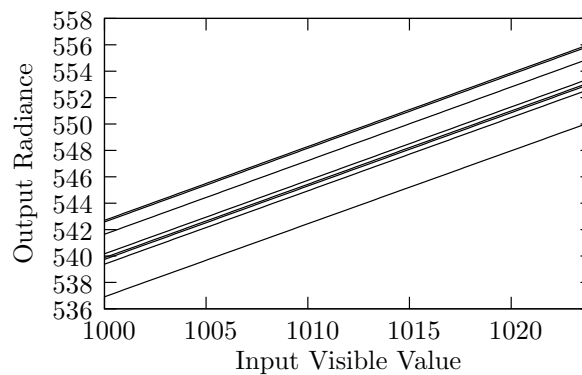


Figure 7.2: Radiometric calibration of GOES visible detectors

Once the created rows are sent to the QM, the GVAR block data is released and the GVAR operator continues with the next block of incoming data.

7.3 Query Manager (QM) Subsystem

The Query Manager (QM) is tasked with managing the current queries in the system, building an appropriate Query Execution Plan (QEP) for the system, and executing the plan as new data arrives. The QEP includes both internal operators that are required to satisfy the queries and those that interface with the query data subsystem to schedule data transfer with individual user queries.

Queries can be added or deleted from the system at any time. However, these modifications only affect the QEP at the next new frame, and so these operations are queued to be performed at the next image frame start.

A new QEP is required for each new GOES frame. Therefore, when the GVAR operator alerts the QM of a new frame, the QM builds a new QEP based on the queries currently in the system. How the QM optimizes the queries and builds the plan was described in Chapter 4.

Once the QEP has been created, new rows of input received from the GVAR operator are processed through the system. Any new rows of data created by the individual operators are continued through the QEP when they arrive. No operator is allowed to block. For example, the operator for induced algebraic operations, Section ??, may require inputs for multiple image channels, but cannot wait for all channel inputs before continuing. This ensures that the QEP can be completed for each input row. This means that any operator can return zero or a number of rows on each invocation, based on whether the operation can complete.

In order to keep track of the current queries in the system from frame to frame, an external database is used. Queries are stored in a PostgreSQL database [?]. External query storage is good for a number of reasons. First, since the queries are only accessed on each new frame start, access to the queries does not need to be done particularly quickly. Storing the queries in an external database is a simple method of saving state if the *GeoStream* system becomes unstable. Equally important, additional reporting and diagnostic tools can be implemented outside of the *GeoStream* environment. For example, current queries can be accessed by a separate connection to the PostgreSQL database.

Because of the decision of using the Perl memory management for keeping track of the row, the QM does not need to explicitly monitor when rows are destroyed. This offers some simplifications in the system. As was previously described, some operators like the induced algebraic operator require inputs from multiple channels. The operator can be implemented by saving pointers to as yet unused row data while waiting for all channels to arrive. Because Perl's memory manager monitors this extra pointer, the data is not destroyed at the end of the execution of the QEP for that row but instead when that row is finally used and released in the algebraic operator.

7.3.1 Restrictions

Restrictions are the only operators that share information among multiple queries. From a single input stream, multiple output streams corresponding each restriction in the operator are output. The goal of the restriction operator is to make this process quick in time to satisfy the multiple restrictions, but also small in terms of sharing as much information between queries as possible. As described in Section 4.3, there are many restriction operations in the QEP.

The inner workings of the DCT, an index developed to handle multiple restrictions was described in detail in Chapter 6. Figure ?? gives an overview of the input and output row tuples from the restriction operation.

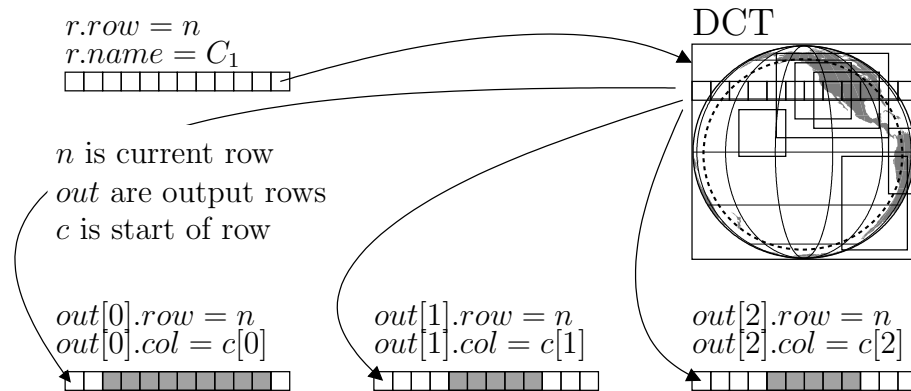


Figure 7.3: Restriction implementation

For each row that enters the system, the operator uses a DCT index to determine

which following operations require part of that row data. Each operator in the QEP attached to the restriction includes a ROI and the DCT returns individual rows for each overlapping region. In the example, three new rows are returned. Although each returned value is a separate row object, the data from the original row data is shared between all of the objects.

7.3.2 Induced Algebraic Operators

Induced operators perform a function on the value sets from a number of input images where the operation is performed for each point in the lattice of the input images. As described in Section 3.2.4 (page 40), the behavior of functional operations on images with different point sets is to operate on the intersection of the point sets. Since point sets are ordered in data streams, this allows for functional operators to be non-blocking in the sense that when inputs are available for all inputs the operator is able to carry out the function. In the case where the point sets do not match, the function can eliminate non-matching rows from an input buffer, while looking for the next potential match. Induced operators must create new images as a result of their operation. The data creation adds considerably to the cost of an operation.

Despite the fact that the WMS interface limits the number of available induced operations to the user, the implementation of the induced operator will be for a generic operation that can be dynamically created for any expression. Figure ?? shows an example instantiation of an induced operation node.

When the QM creates a new induced operator, it sends the algebraic operation as a valid PDL expression, with mappings for each required row used in the expression. The mappings are read to determine what inputs are required. For each input, a buffer is created to hold intermediate row objects. Because of the nature of the input stream, it is possible for a few rows of one type to arrive before any of another required type. For this reason, a queue is required in each operator to store as yet unused rows of data. Operationally, these queues will never get too large, since the number of queued rows is limited by the format of the stream. Also, because the individual rows follow a specific ordering, it is possible for the algebraic operators to determine when rows will not be used.

The passed string is also used to create a small routine to evaluate the function

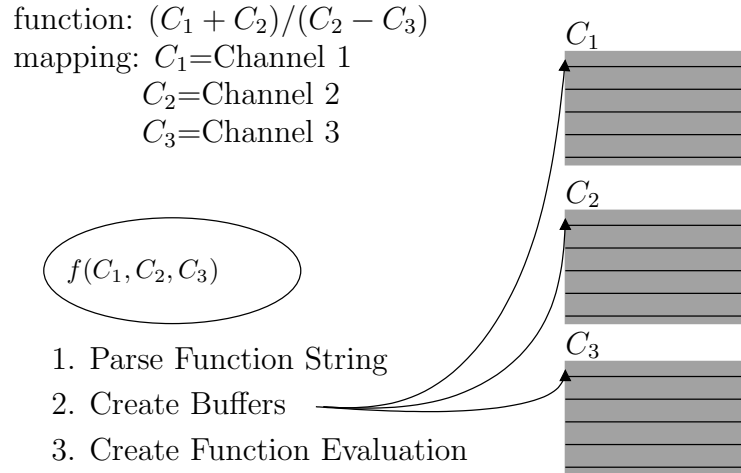


Figure 7.4: New induced operation

when enough row objects are received. In a Perl implementation, the mapping can be applied to the string and passed to a PDL interpreter. Incoming rows have an identifier as part of the header information of the row.

Once the operation node has been initialized, execution takes place as rows are passed to the node. Figure ?? describes the induced operation execution.

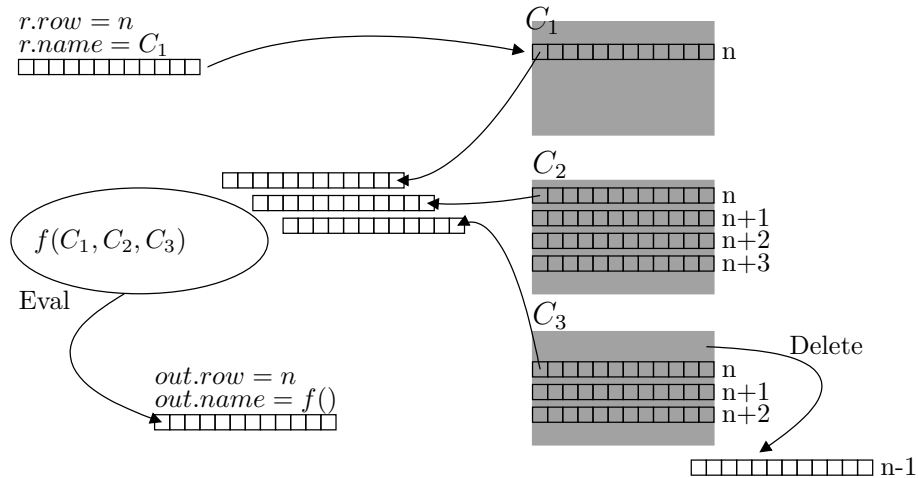


Figure 7.5: Induced operation execution

Execution of the operator occurs for every row that is input into the node. When a new row is input and a row exists for each required input, then the expression is evaluated and the result is returned. The operator does not always return rows on input, only when

enough data is in the operator to allow the execution of the function.

Figure ?? also shows a row in the C_3 queue being dropped, since it is clear from the ordering on the image, C_1 , that row $n - 1$ does not exist on the C_1 image stream.

7.3.3 Halving Operator

The query optimization strategy for the averaging function of Section 4.3.1 describes how queries of different resolutions are created by successively taking 2×2 averages of input images, which are half the width and length of the input image. Images are halved until they are approximately the size of the requested query. Halving operators work on a single image and output a new row of data for every two that arrive.

Halving operators are not induced operators, but their behavior is similar to a uni-variable algebraic operator. The main difference being that the output point set is different than the input point set. Figure ?? shows a notional implementation of the halving operation.

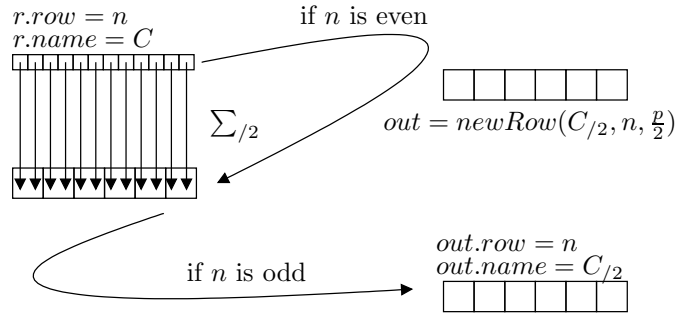


Figure 7.6: Halving operation

When an even line enters the system, a new row object is created. This object has half the number of points, and with a point set lattice that has twice the resolution of the incoming image. A loop is then performed on the input row, and adjacent pixel values are averaged into a single value and input into the new row. When an odd line enters the system a new row object will already exist. In this case the same loop is run over the input row but the output values are averaged with the existing values in the new row object. This row is then output from the operator.

Corner cases are not discussed in this example and the implementation could

handle these in a number of ways. The simplest method would be simply to require that the input to the operation requires an input point lattice that has an even number of rows and columns. A more robust method would include checks on the input stream and output appropriate new row data even in the absence of a complete incoming point lattice.

7.3.4 Geometric Transformations

As described in Section 3.1 (page 28), spatial transformations map image values from one point set to another. The function, f , is a mapping from $\mathbf{Y} \rightarrow \mathbf{X}$, and $\mathbf{a} \circ f = \{(\mathbf{y}, a(f(\mathbf{y}))) : \mathbf{y} \in \mathbf{Y}\}$. In geospatial applications in general, and in the case of queries to satellite data in particular, transformations can involve changing the coordinate system of an image's point lattice from that of the satellite to some ground based reference system used by GIS applications. A coordinate transform is the typical final step for most queries in the *GeoStreams* system.

In the typical formulation of a transformation, the transformed image is built using the assumption of random access into the input image, which implies that the entire image must be saved to memory before a transformation operation begins. In the DSMS additional structures are built to speed processing and prevent this blocking.

Figure ?? shows the instantiation of a new geometric transformation. In the figure, the small black squares represent the points in the new coordinate system and the gray squares represent the original coordinate system. The points in the new system were specified by the query. The QEP described in Chapter 4 and implemented by the QM will have determined the appropriate number of halving operations so that the incoming data is of the appropriate resolution. The boundary points of the new coordinate system are projected to the GOES coordinate system, determining the extents of the image in that system. The entire boundary is used, since the image corners do not always catch the full extent. Once the extent of the new point set is known, the resolution of the point lattice is determined. The transformation strives for an approximate one-to-one correspondence between pixels in both point sets and chooses the number of halving operations accordingly.

Then, a Geographic Lookup Table (GLT) is constructed for each row of data in the final coordinate system. Each point in each row is projected to the GOES coordinate system

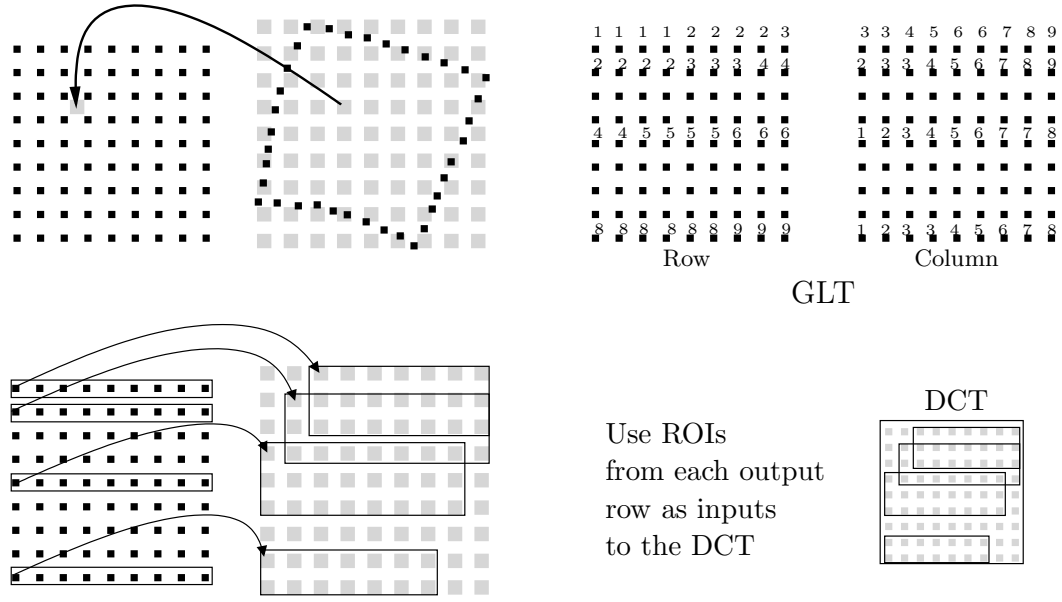


Figure 7.7: Geometric transformation instantiation

and the row and column are saved in supporting GLT images. These will be used in the execution of the transform. Each row in the final projection also has a ROI corresponding to what input pixels are required to complete the output for each row. These ROIs are added to a DCT index used for this transformation.

Creating a new image transformation operation is time consuming in that many projection operations need to be performed and supporting images and DCT indexes are required. This happens when data is not coming from the input stream because the QM creates a new QEP between image frames.

Figure ?? shows the coordinate transform operation in execution. As each input row of data enters the operator, it is used with the DCT index to determine which output rows are dependent on that input row. For each of these output rows, a loop is performed over the points in the row. The GLT row image is examined, and if the value matches the input row, then the appropriate point is determined from the GLT column image. This value is then used to calculate the output value for that point. In a nearest neighbor interpolation the closest input value is used. For other interpolation methods, a weight is assigned to the input value based on the location of the output point and the weighted value is added to the output pixel value. For bi-linear interpolation, this means that four input points are

used in the calculation of each output value.

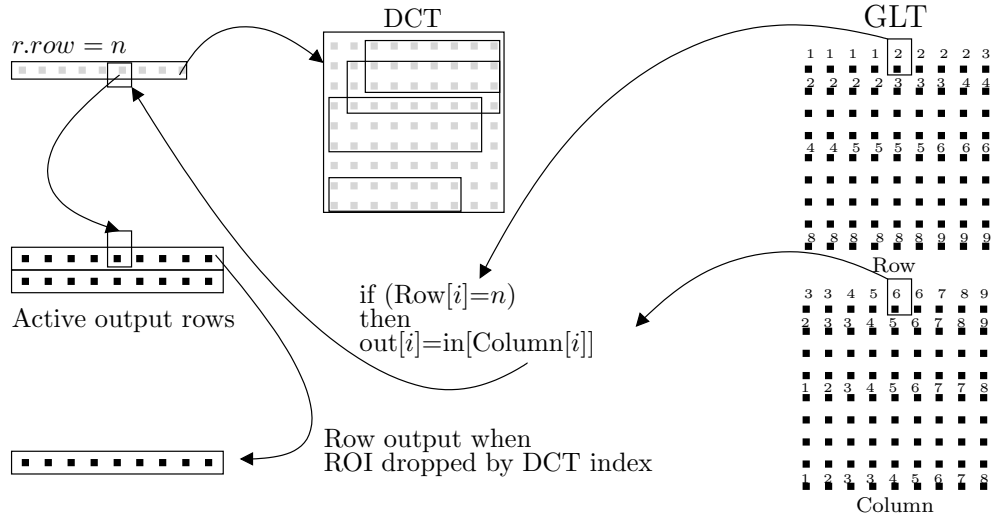


Figure 7.8: Geometric transformation execution

Some extra pointers can speed up the loops on the points of each output row. Extra pointers can identify the starting and ending points in each row that is yet unfilled. This eliminates most of the looping over pixels that already have had their values computed.

As described in Chapter 6, the DCT maintains its output set incrementally and can report on each query to its index which ROIs have left the set on that iteration. This information can be used by the transformation operation to determine when output rows are complete and can be output from the operation. This must be done in a way to preserve the point set order of the output image.

There are some combinations of coordinate systems that could require many output rows to be queued in order to retain this ordering, but in practice the coordinate systems usually match to the extent that few output rows are prevented by the ordering from being output.

7.4 Query Data Management

As the QEP executes for each input row of data, the continuous queries receive output rows of data. It is the responsibility of the query data subsystem to reformat these output rows to a form that is suitable for the user. This includes choosing a suitable image

format and to develop a scheme for the interaction between the server and the clients to deliver this continuous output.

7.4.1 Query Data Formats

The WMS specification allows for a number of image formats to be requested by the client. Most of these are display based formats such as Portable Network Graphics (PNG), and Joint Photographic Experts Group (JPEG) standards. These standards allow for images to be displayed in the largest number of applications, but are not designed for geospatial data. Other formats such as Geotiff [?] and GML in JPEG 2000 (GMLJP2) [?] are designed specifically for remote sensing applications and allow for encoding spatial information and other metadata directly in the image itself. As previously mentioned, one advantage of the WCS query standard over the WMS standard is in its ability to choose from a larger set of image formats for data.

However, there are some additional advantages for encoding output in the PNG format. The major advantage is that PNG uses line oriented compression and encoding. This means that while encoding the output, each individual row could be compressed and encoded separately. There would be no need to store an entire frame of each output query in order to compress the image. The line oriented encoding also allows for the progressive display of an image. If a user was truly interested in real-time display from the system, this could be accomplished with the PNG format. PNG compression is also lossless and the RSI tends to compress fairly well with line-oriented compression.

As stated, PNG has the problem that there is no standard method for embedding spatial reference information in the file itself. This is not a limitation in the PNG format, but rather in the lack of a standard for embedding that information. The GMLJP2 standard could easily be used as a template for a similar GML in PNG format.

There are also some extensions to the PNG format that allow for multiple temporal image frames to be included in a single file. This could also be a useful representation for continuous queries.

For these reasons, the PNG image format is a good implementation to use for output images. Depending on the data delivery method used by the system, additional

formats like GMLJP2 could also be included.

7.4.2 Data Delivery

When a user launches a continuous query, they cannot simply wait for the results, as is expected in the WMS protocol. Instead, the user needs access to intermediate parts of the continuous query while it is still producing new output. There are three potential solutions for this form of delivery: a continuous output format could be utilized; the service could divide the output image stream into separate files and create a query data archive of these files; or the system could support client defined callbacks to interact directly with client applications. Figure ?? shows these mechanisms.

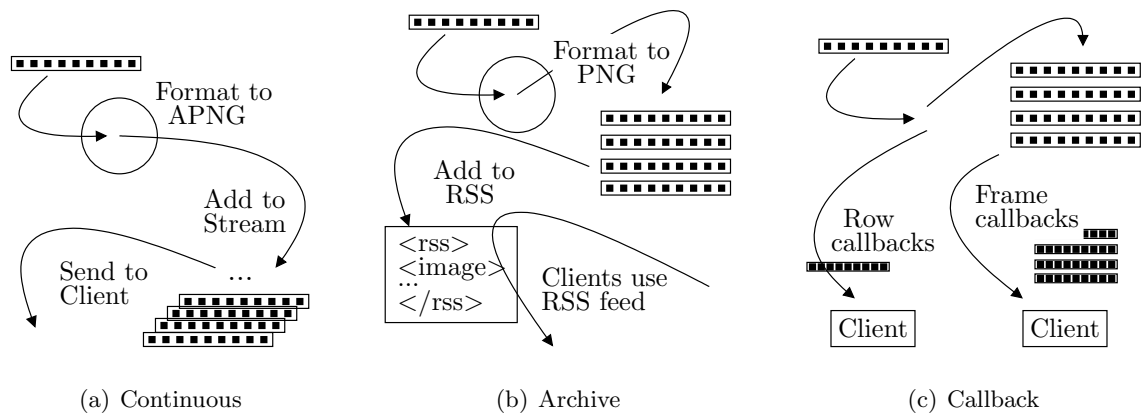


Figure 7.9: Delivery mechanisms

There are two extensions to the PNG format that would allow for a continuous image stream to be represented. Multiple-image Network Graphics (MNG) [?] and Animated Portable Network Graphics (APNG) [?] both add the ability to have multiple frames in a single file. They also can represent non-ending sets of image frames. APNG especially is a simple extension to the PNG image format, and the query data management subsystem could easily create such a stream.

A major problem for continuous format images is that it assumes that there remains an open network connection between the server and the client, which is not always the case. The best use of a continuously streaming format would be where a user launched a continuous query in order to do real-time monitoring of the RSI data, providing a con-

tinuous up-to-date display of the current satellite data. In this case, an interruption of the network connection might signal to the QM that the query is no longer of interest, either because the user is no longer interested, or because the lack of the connection means the user cannot access the real-time stream regardless. A reasonable strategy in implementing queries requesting a continuous format is that if the connection between the client and server ends then the query is deleted from the system.

An alternative solution is for the server to cache to disk the results of the user queries and give the client a window of opportunity to retrieve those results. Advantages of this approach are that no continuous connection is required between client and server and that the entire post-evaluation part of the service could be delegated to another specialized stand-alone application. An example of this scenario would be a service based on Really Simple Syndication (RSS) [?]. When a user launches a new query, the server responds with the Uniform Resource Locator (URL) of an RSS feed. From this point forward, the user downloads query results using standard client RSS applications. Here the *GeoStreams* application would store the query results as a single file for every image frame. On completion of a frame, a formatted image would be added to the query data archive and the appropriate RSS feed would be updated. From that point, the *GeoStreams* application would treat the data as delivered. Any other interaction between client and server would take place outside of the *GeoStreams* application. This query archive method is the simplest and most robust solution for the *GeoStreams* architecture and is the default implementation method.

The final method would be to allow the client to register a callback for the server to perform on some event in the stream evaluation. Callbacks could be registered on various events in the output image stream, new rows arriving in the query results or new complete frames for example.

The callback methodology is advantageous because it can replicate both of the previously defined methods in a common way. For example, real-time queries could be supported with a callback on each new row of data, while the archive method could be supported with callbacks on each new frame. Another advantage is that the callback method could eliminate the need of an intermediate image format completely. Instead, the callback could include functions passing the *GeoStreams* system row objects directly.

The biggest problem with the callback method is that the system is required to develop a callback implementation. In addition, standard practices need to be defined in terms of how the system should respond if callbacks fail for some reason during some part of the query evaluation.

The *Query Manager*, *Row Memory*, *GVAR Input*, and *Query Data Management* subsystems combine to form a complete online application. This chapter has defined an implementation for each subsystem with some considerations for rapid prototyping in the Perl programming language. Next steps would include more specific object definitions, coding, and integration of existing DCT code.

Chapter 8

Future Work

Remotely-Sensed Imagery provides a great opportunity to study new concepts for the management and processing of streaming data, given the many satellites that deliver data products in environmental sciences, land use, disaster management, climatology, and other fields.

The basic concepts underlying the plans for a complete *GeoStreams* DSMS architecture for queries on streaming RSI data were described. The effectiveness of some of the basic components for the system have been demonstrated. For example, using the DCT as a method for indexing multi-query restrictions, was shown to be a very effective index.

Work is started on developing an on-line system using the WMS specification as a basis for web-based access to the DSMS, using the specifications from Chapter ?? . This is the first priority for follow-on research. However, there are many research opportunities for other aspects of the *GeoStreams* project.

Chapters 3 and 4 developed the basic operations for the algebra used in this project and some potential optimizations on these operations. There is a wealth of other information from image algebra that could be incorporated into a more complete system. These include new operations, or the combination of operations to perform important image processing tasks. Each of these new tasks would need investigation into optimizations and implementation details.

As queries become more sophisticated, the types of optimizations may change in

character. In Chapter 4, single queries could all be optimized the same way. As query expressions become more sophisticated this may not be the case. Rather than optimizing each query first and then separately optimizing among queries, the global QEP may need to include a deeper search of potential solutions, where single and multi-query strategies are combined.

Even using the current optimization framework, significant improvements exist. Section 4.3.3 described some problems with the current multi-query optimizations, in particular, using a single restriction operator for widely separated ROIs. One solution is to allow more general point sets other than the square lattices presented here. An interesting research topic would investigate specifying ROIs with constraint-based semantics where regions are described as unions of half-plane intersections [?]. This could have considerable effect on the implementation of the DCT or other, perhaps new, ROI indexes.

Some implementation problems were also demonstrated in Chapter 5. Developing a system based on existing GIS software offers many advantages, but there are implementation issues that need to be addressed. Examination of the times it takes in that system to process only 100 relatively simple continuous queries compared to the speed at which new spatial frames of data arrive from the GOES system imply that processing load will quickly become a problem. Future work could investigate scale issues with load shedding and multi-processor solutions. Load shedding in generic streaming databases often entail skipping the processing of certain input tuples in the data stream. An obvious method of load shedding on an image stream is to simply skip certain frames for all or selected queries.

Besides load shedding, multiple processors could be used to improve query execution. The GRASS library was not developed to be thread safe and multiple processors can't speed up individual operations, but possible strategies still exist. For example, since most of the processing of continuous involve processing on the current frame, one simple multi-processing scenario could simply use multiple processors and their associated local disk storage in a round robin technique over frames. Rather than processing every new frame from the GOES satellite on one processor, if there were n processors in a cluster, then each processor could be assigned to handle every n th frame from the GOES system.

Section 5.3 also discussed that in addition to RSI, another important class of

potential input to such a system could be the outputs of modeling data. The Weather Research and Forecasting (WRF) model was given as an example. One interesting feature of using modeling data is that many queries on modeling data would be related more to data mining activities rather than the more traditional queries discussed in this work. Incorporating data mining techniques in the context of an imagery DSMS would be an important and challenging area of study.

Publishing Abbreviations

ACM!	ACM!
ASA!	ASA!
ASTM!	ASTM!
CIDR!	CIDR!
DAPD!	DAPD!
FGDC!	FGDC!
FMLDO!	FMLDO!
ICDE!	ICDE!
IJGIS!	IJGIS!
IPDPS!	IPDPS!
IEEE!	IEEE!
NASA	National Aeronautics and Space Administration
NEDIS!	NEDIS!
OGC	Open Geospatial Consortium
PODS!	PODS!
POSC!	POSC!
QLQP!	QLQP!
SIGACT!	SIGACT!
SIGART!	SIGART!
SIGMOD!	SIGMOD!
SRMS!	SRMS!
SSTD!	SSTD!
SSDBM!	SSDBM!
TKDE!	TKDE!
VLDB!	VLDB!