Computational Flood Modeling and Visual Analysis

Donald W Johnson

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation
https://scholarsjunction.msstate.edu/td/1135

This Dissertation is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.
Computational flood modeling and visual analysis

By

Donald W. Johnson

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2016
Copyright by

Donald W. Johnson

2016
Computational flood modeling and visual analysis

By

Donald W. Johnson

Approved:

_________________________
T. J. Jankun-Kelly
(Major Professor / Graduate Coordinator)

_________________________
J. Edward Swan, II
(Committee Member)

_________________________
Song Zhang
(Committee Member)

_________________________
Derek T. Anderson
(Committee Member)

_________________________
Jason M. Keith
Dean
Bagley College of Engineering
This dissertation introduces FESM (Flood Event Simulation Model), a Geographic Information System (GIS) tool designed for use on gaged river systems that can be used to guide logistic support during disaster events. FESM rapidly generates flood predictions using elevation data from real-world sensors or generated by other models. Verification and validation data for FESM are provided. In order to construct a visualization system for interacting with FESM outputs, single buffer and dual buffer techniques for moving massive datasets to the GPU for processing using OpenCL were rigorously tested and timed, and an analysis of the costs/benefits of using buffers or images was conducted. Finally, DRO (Dynamic Raster Overlay), a visualization system for analysis of datasets composed of multiple overlapping flood maps is introduced, and expert feedback is provided on the effectiveness of DRO with selected case studies.

Key words: Flood Inundation, GIS, Flood Mapping, Parallel Processing, Bitmap Operations, Framebuffer Operations, Parallel I/O, OpenCL, Visual Analytics, Multi Surface Display
ACKNOWLEDGEMENTS

I would like to thank Ron Goldman and Robert Simeral for their support, the Vicksburg district MMC team for their help with the testing and validation of the FESM model, my adviser T. J. Jankun-Kelly and other committee members for their advice and help in the creation of this dissertation.
TABLE OF CONTENTS

ACKNOWLEDGEMENTS .................................................. ii

LIST OF TABLES ......................................................... v

LIST OF FIGURES ......................................................... vi

CHAPTER

1. INTRODUCTION ...................................................... 1
  1.1 Motivation ....................................................... 1

2. FLOOD MODELING .................................................. 4
  2.1 Introduction ..................................................... 4
  2.2 Related Work .................................................... 7
  2.3 FESM ............................................................. 15
    2.3.1 Background ............................................... 16
    2.3.2 FESM Inputs .............................................. 16
    2.3.3 FESM outputs .............................................. 17
    2.3.4 Model Overview .......................................... 17
    2.3.5 Slope Rules .............................................. 19
  2.4 Modifications .................................................. 20
    2.4.1 Large Surface Support .................................. 20
    2.4.2 Multiprocessing ......................................... 21
    2.4.3 Integration with Other Models ....................... 21
  2.5 Model Validation and Verification ......................... 21
    2.5.1 Yazoo BackWater Delineation Project (2006) .......... 22
    2.5.2 Kansas City Flood (2011) .............................. 24
  2.6 Conclusions .................................................... 29

3. DATA TRANSFER OPTIMIZATION ................................. 35
  3.1 Introduction .................................................... 35
  3.2 Motivation ....................................................... 36
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Early FESM flood predictions compared to historical flood satellite imagery</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>AMD Algorithm Run Times</td>
<td>63</td>
</tr>
<tr>
<td>3.2</td>
<td>4k Image Transfer Rate and Efficiency</td>
<td>64</td>
</tr>
<tr>
<td>3.3</td>
<td>AMD Transfer Rate &amp; Efficiency</td>
<td>67</td>
</tr>
<tr>
<td>3.4</td>
<td>AMD Buffer Kernel vs Image Kernel Runtime Tests Results</td>
<td>73</td>
</tr>
<tr>
<td>3.5</td>
<td>Algorithm Run Times (NVIDIA)</td>
<td>73</td>
</tr>
<tr>
<td>3.6</td>
<td>NVIDIA Transfer Rate and Efficiency with ~4k Image Dataset</td>
<td>74</td>
</tr>
<tr>
<td>3.7</td>
<td>NVIDIA Buffer vs Image Kernel Runtimes</td>
<td>75</td>
</tr>
<tr>
<td>4.1</td>
<td>Processing time with synthetic data</td>
<td>93</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2.1</strong></td>
<td>FESM comparison to satellite for 10 March 89 [20]</td>
<td>25</td>
</tr>
<tr>
<td><strong>2.2</strong></td>
<td>FESM comparison to satellite for 21 March 87 [20]</td>
<td>26</td>
</tr>
<tr>
<td><strong>2.3</strong></td>
<td>FESM comparison to satellite for 13 Jan 83 [20]</td>
<td>27</td>
</tr>
<tr>
<td><strong>2.4</strong></td>
<td>January 1983 Comparison [20]</td>
<td>28</td>
</tr>
<tr>
<td><strong>2.5</strong></td>
<td>FESM predictions compared to GPS survey photos.</td>
<td>29</td>
</tr>
<tr>
<td><strong>2.6</strong></td>
<td>FESM predictions compared to GPS survey photos.</td>
<td>30</td>
</tr>
<tr>
<td><strong>2.7</strong></td>
<td>FESM predictions compared to GPS survey photos.</td>
<td>30</td>
</tr>
<tr>
<td><strong>2.8</strong></td>
<td>FESM predictions compared to GPS survey photos.</td>
<td>31</td>
</tr>
<tr>
<td><strong>2.9</strong></td>
<td>FESM predictions compared to GPS survey photos.</td>
<td>31</td>
</tr>
<tr>
<td><strong>2.10</strong></td>
<td>FESM predictions compared to GPS survey photos.</td>
<td>32</td>
</tr>
<tr>
<td><strong>2.11</strong></td>
<td>FESM predictions compared to GPS survey photos.</td>
<td>32</td>
</tr>
<tr>
<td><strong>3.1</strong></td>
<td>Flood extent analysis program.</td>
<td>60</td>
</tr>
<tr>
<td><strong>3.2</strong></td>
<td>Naive Approach – 1 Buffer</td>
<td>61</td>
</tr>
<tr>
<td><strong>3.3</strong></td>
<td>Dual Buffer Approach</td>
<td>62</td>
</tr>
<tr>
<td><strong>3.4</strong></td>
<td>Dual Buffer Algorithm Flow</td>
<td>62</td>
</tr>
<tr>
<td><strong>3.5</strong></td>
<td>AMD Image Processing Times for 4k Images</td>
<td>65</td>
</tr>
<tr>
<td><strong>3.6</strong></td>
<td>AMD Efficiency for 4k Images</td>
<td>66</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.7</td>
<td>AMD Buffer Transfer Speed</td>
<td>66</td>
</tr>
<tr>
<td>3.8</td>
<td>AMD Image Processing Times for ~2k Images</td>
<td>68</td>
</tr>
<tr>
<td>3.9</td>
<td>AMD Efficiency for ~2k Images</td>
<td>69</td>
</tr>
<tr>
<td>3.10</td>
<td>AMD Image Processing Times for ~8k Image</td>
<td>70</td>
</tr>
<tr>
<td>3.11</td>
<td>AMD Efficiency for ~8k Images</td>
<td>71</td>
</tr>
<tr>
<td>3.12</td>
<td>AMD Buffer to Image Transform Transfer Rates</td>
<td>72</td>
</tr>
<tr>
<td>3.13</td>
<td>NVIDIA Image Processing Times for 4k Images</td>
<td>76</td>
</tr>
<tr>
<td>3.14</td>
<td>NVIDIA Efficiency for 4k Images</td>
<td>77</td>
</tr>
<tr>
<td>3.15</td>
<td>NVIDIA Buffer to Image Transform Transfer Rates</td>
<td>78</td>
</tr>
<tr>
<td>4.1</td>
<td>An example of multivariate visualization using Oriented Slivers</td>
<td>84</td>
</tr>
<tr>
<td>4.2</td>
<td>Multivariate data display using DDS (Data Driven Spots)</td>
<td>98</td>
</tr>
<tr>
<td>4.3</td>
<td>Multivariate data display utilizing 3D glyphs</td>
<td>99</td>
</tr>
<tr>
<td>4.4</td>
<td>Multivariate data display utilizing glyph blocks</td>
<td>100</td>
</tr>
<tr>
<td>4.5</td>
<td>Multivariate data display utilizing dual-layer visualization</td>
<td>101</td>
</tr>
<tr>
<td>4.6</td>
<td>Multivariate data display utilizing multi-layer visualization</td>
<td>102</td>
</tr>
<tr>
<td>4.7</td>
<td>Multi-surface visual comparison through Ensemble Surface Slicing (ESS)</td>
<td>103</td>
</tr>
<tr>
<td>4.8</td>
<td>Visualization system in Overview Mode</td>
<td>104</td>
</tr>
<tr>
<td>4.9</td>
<td>Visualization system in Focus Mode</td>
<td>105</td>
</tr>
<tr>
<td>4.10</td>
<td>Ordering of OpenCL operations for maximum throughput</td>
<td>106</td>
</tr>
<tr>
<td>4.11</td>
<td>The surfaces that make up the Yazoo Backwater dataset.</td>
<td>107</td>
</tr>
<tr>
<td>4.12</td>
<td>Exploring the Yazoo Backwater datatset</td>
<td>108</td>
</tr>
<tr>
<td>4.13</td>
<td>Notable subregions in the Yazoo Backwater dataset</td>
<td>109</td>
</tr>
</tbody>
</table>
4.14 Effects of a water control structure ........................................ 110
4.15 The surfaces that make up the Bayou Meto dataset. .................. 111
4.16 Exploring the Bayou Meto dataset ........................................... 112
4.17 The surfaces that make up the Amite dataset. ......................... 113
4.18 Exploring the Amite dataset .................................................. 114
CHAPTER 1

INTRODUCTION

1.1 Motivation

Flooding is one of the most frequently occurring and most costly natural disasters. Although the amount varies depending on the severity of events in a given year, total damages are usually in the range of hundreds of millions dollars. Flood modeling can be used to mitigate damages and prevent loss of life by identifying areas at risk before an event takes place; high speed modeling can also be used to provide emergency support during an event.

This dissertation focuses on three topics related to flood modeling and analysis. The first is the development of a flood model designed for use on gaged river systems that can be used to provide logistic support during a disaster event. The second two areas relate to the use of modeling data in the analysis of multiple flood events in a single region. Such data can come from examining historic events, or multiple synthetic events meant to show varying degrees of disaster, or multiple modeling scenarios meant to show the effects of different proposed flood control projects. Working with such data has two major problems. First, the size of any given data set can become very large (multiple gigabytes per output if the modeling was done with high resolution topography.) Second, when attempting analysis of multiple datasets, the number of combinations to be considered grows exponentially as the number of input datasets increases.
In order to allow rapid analysis of these potentially large datasets, the second area of focus in this dissertation, is queuing optimizations to allow a GPU to quickly process datasets that can not be loaded into graphics memory in their entirety. By using data-streaming techniques, the maximum size of a potential datasets is expanded to the capacity of the main memory of the computer system used for processing. Modern desktops can accommodate 32 to 64 GB of ram which would be sufficient for analysis of over 10 simultaneous events even with very high resolution modeling outputs.

The final area of focus is, given a set of flood event output, how can patterns that hold true in all events be found and how can areas that differ be brought to attention. This problem has many of the same traits as ensemble analysis as used in weather forecasting. One key difference is that areas of agreement are not the primary trait that needs to be identified. Being able to see how and where the model results differ is of equal importance. To this end, a method for visually guided analysis of such data sets is proposed and tested.

The contributions of this dissertation are:

- Development of a model for conversion of remote water gage elevation data into flood inundation surfaces;
- Presentation of verification and validation data for the introduced model, demonstrating its efficacy when managing disaster scenar
- Testing and timing of single buffer and dual buffer techniques for moving massive datasets to the GPU for processing using OpenCL;
- Analysis of the costs/benefits of using buffers or images when processing massive datasets;
- Creation of a visualization system for analysis of datasets composed of multiple overlapping flood inundation maps;
- Obtaining expert feedback on the effectiveness of the visualization system in selected case studies.
The remainder of this document is structured as follows: Chapter 2 addresses the development of a hydraulic model (FESM) able to rapidly create inundation prediction maps from sensor-derived water elevation data. Chapter 3 deals with the technical details necessary to transfer large numbers of inundation maps to the GPU for realtime analysis, ultimately resulting in the creation and testing of an algorithm (DBA) allowing for optimized data-transfer between the CPU and GPU. Chapter 4 details a visual analysis system (DRO) designed to allow the dynamic display and analysis of multiple flood maps simultaneously. Chapter 5 presents overall conclusions derived from the discussed projects and research.

In summary, this work first considers the problem of converting sensor-derived water elevation data to maps suitable for visual analysis, and secondly, the problem of providing an optimal interface for meaningful and timely analysis of many such maps simultaneously.

These problems are significant, as flood-mapping GIS tools such as FESM are utilized by the US government to plan and orchestrate responses to floods, which remain the most frequent and costly natural disasters in the country. Since 2005, post hurricane Katrina, FESM has been used by the US Army Corps of Engineers to manage intervention and relief efforts for the majority of major flood events in the Southern United States. As of 2013, the Mapping, Modeling and Consequences branch of the US Army Corps of Engineers has continued to train engineers in branch offices around the country in setting up and utilizing FESM, which is currently considered to be a cutting-edge GIS tool for flood-mapping and response management.
CHAPTER 2
FLOOD MODELING

2.1 Introduction

This chapter introduces FESM (Flood Event Simulation Model), a Geographic Information System (GIS) tool which rapidly generates flood water surfaces and inundation extents by using elevation data, either gathered from real-world sensors, or generated by other models run at lower resolutions. Since its development in 2005, FESM has been utilized by the United States Army Corps of Engineers (USACE) to coordinate emergency response to numerous major flood events occurring in the United States.

Flooding causes significant damage and multiple deaths each year, both in the United States and worldwide. For example, in 2008, floods in six Midwestern states caused over 15 billion dollars in damages and lead to thirteen deaths [17]. Flood models, which can predict the extents and severity of flooding, can be used to reduce damages and prevent deaths, both by influencing the implementation of flood control projects and structures, and by providing critical information for emergency response forces during flood events.

Computer-aided GIS tools have been developed and refined since the 1950s, and heavily utilized by the USACE, the United States Geological Survey (USGS), and the US National Center for Atmospheric Research (NCAR). Because of the continuous need for better prediction and modeling of flood events, computer-based flood simulation tools, a
specific type of GIS, have been in development over the last four decades, with flood models in use as early as the 1970s [8]. Currently, there are many existing flood models, each using one of two approaches to topography representation: either Triangulated Irregular Networks (TINs) or rasterized Digital Elevation Models (DEMs). For flood extant modeling, most flood models perform hydrodynamic process simulation using one of many possible approaches (e.g., Navier-Stokes equations, de Saint-Venant equations, diffusion wave/zero-inertia models). Much research has been conducted to verify the effectiveness of different models, or to accelerate models, either by changing representation equations or dynamic time modeling implementation. Further research has examined the impact of different topographic representation methods and resolutions on output accuracy and computational time (These are detailed in Section 2.2).

Unfortunately, existing GIS flood models have numerous problems, including difficult and extensive setup requirements, complex and cumbersome user interfaces, and performance times inadequate for real-time simulation. Full simulation of hydrodynamic processes requires information that is not easily available. For example, when setting up flood simulation models, it is usually necessary to estimate friction coefficients for the entire simulated domain, and then to adjust coefficient values until the model behaves rationally and output is acceptably accurate. The need to estimate friction and other parameters makes calibration of flood inundation models a difficult task.

When using a flood model to coordinate response to an ongoing emergency, time is critical. Because existing GIS flood model tools were cumbersome and inadequate for emergency response, the USACE required a new tool capable of rapidly simulating flood
events. The new tool would also have to be easy to setup and deploy in previously unmodeled regions. This chapter, therefore, introduces the Flood Event Simulation Model (FESM), a new tool for the rapid prediction of flood inundation surfaces. Rather than simulating input flow into a modeled area to create a changing map of the flood extent over time, FESM utilizes water elevation measurements to quickly generate flood surfaces and inundation maps. A key benefit of this approach is that FESM generates water surface or flood inundation predictions utilizing a single time-step, as opposed to the multiple time-steps required by more conventional GIS flood models, which must rely on hydrodynamic process simulation. This approach allows FESM to produce accurate results more quickly than conventional models by several orders of magnitude, making FESM appropriate for real-time emergency response.

The Contributions of this chapter are the creation of a flood model (FESM) with the following properties:

- Minimize information required to make predictions— during emergencies, abundant information may not be available.
- Simplify model setup— personnel and resources will be scarce during emergencies.
- Enable rapid data processing to generate inundation maps and flood surfaces— time is critical for emergency response.

This chapter remainder of this chapter structured as follows: first the related work in the development of flood modeling is discussed. Following this, the model structure of FESM is covered. Finally validation and verification testing that has been done with one FESM’s results are investigated. Finally conclusion about the model and areas of further research are discussed.
2.2 Related Work

Flood models can be roughly categorized based on the complexity of the mathematical basis used for water propagation. The simplest type of models use create a planar approximation for a flood water surfaces and then compare values of the recorded land elevation values to determine inundation at any given location. More complex and generally more accurate approaches use either Manning equations, the de Saint Venant [39] equations, or the Navier Stocks equations either in full or approximate form [8]. The required information required for simulation will vary depending on the mathematical basis used: A planar solution can be attempted with just water and land elevation data. Alternatively using either the Navier Stocks of de Saint Venant equations will require information on input flow volumes into the simulated area, land surface friction coefficients, the initial state of the water in the simulation area, including elevation and velocity.

Early work in flood models tested several different methods of flood prediction with differences in how water transport was simulated and how the flood plan in question was represented. Lamberti and Pilati [39] described both the complete form of the de Saint Venant Equations that could be used for flood modeling and forecasting. Stewart et al. [64] tested a model that used 1D de Saint Venant Equations with flood plain topography being represented with a finite element mesh to test flood predictions on a part of the Severn River in England. Several models that used more simplified approaches to modeling where tested with the conclusion that simple approaches where sufficient for predicting the rivers hydro-graph but not for mapping inundation extents. Additionally the need for verification and validation of model predictions was stressed. Hardy et al [29] found that
mesh resolution had strong influence on the extents estimated using the Navier Stocks based model TELEMAC-2D.

Bates and De Roo [8] described the LISFLOOD-FP model, one of the first flood models to use a raster based DEM (Digital Elevation Model) instead of the finite element meshes or storage cell based approaches used in earlier models. This model was compared with several earlier models with the conclusion that raster based modeling yielded the most accurate results. Bates et al [9] attempted to test if the representation of the topology had more influence on the model results than the hydraulic processes simulated. This was done by comparing the results of one finite element model and two raster based models with satellite imagery. The results where inconclusive as the uncertainty in classification of the validation imagery was greater than differences observed between any of the tested models; the raster biased models where found to be easier to calibrate. Horritt and Bates [30] tested how the LISFLOOD-FP model responded to changing the resolution of its raster DEM. Hunter et al [37] modified LISFLOOD-FP to use an adaptive timestep, this allowed the raster model to better respond to changes in the flood plane friction parameter by removing reliance on the cell to cell water volume per time step limiter. It also simplified model configuration as the user no longer had to manual select time steps until a stable time step was located. Successful calibration of earlier models was attributed to limitations of available calibration data not being able to display the underling weaknesses of earlier models.

Horret et al [31] evaluated three different flood inundation models (LISFLOOD-FP, TELEMAC-2D, HecRAS) to see how they preformed when calibrated against both hydro-
logical data (discharge of the modeled stream at known locations), and flood extent data (remotely sensed satellite data). TELMAC2D and HecRAS were found to accurately inundation extents using calibration data of either type. LISFLOOD-FP could accurately reproduce discharge data when calibrated with discharge data and extents when calibrated against extents but could not be cross calibrated. This was attributed to LISFLOOD-FP being more strongly influenced by the friction parameter than the other tested models. Later in 2006, Horret et al. [32] tested the effects of different grid resolutions on a triangle mesh based flood model. The main findings were that higher resolution models could get improved areas when those improvements allowed for more accurate modeling of water movement, and modeling at resolutions below 10m should include bathometric data to better model the river bed. Hunter et al. [35] performed testing and comparison of results for 6 different flood models looking for differences in predictions when used in an urban scene with LIDAR data. All models were found to create similar predictions for inundation area and arrival times however the testing revealed that uncertainty in models parameters, most notably friction, becomes more influential to predicted results as the resolution of the tested topography increases. Neal et al. [49] preformed an evaluation of LISFLOOD-FP using the dynamic time step introduced by Hunter [37] with the conclusion the accuracy of model was unchanged and the adaptive time step was 67x faster than the previous formulation.

One of the more active areas of research in flood inundation modeling was how to correctly calibrate models for use. Pappenberger et al. [53] studied how model calibration is subjective and showed that global model evaluation metrics were insufficient for deter-
mining flood threat when all areas in the modeled domain where not equally important. Particularity residential, commercial or transportation related structures need prioritization as the correctness of a model in these areas is more important for the assessment of threat than the overall accuracy in agricultural or undeveloped land. Hunter et al. [36] studied the benefits and problems associated with using various forms of simplified physics in flood models in areas where physical accuracy was judged unimportant.

A common finding in calibration studies was that optimal calibration parameters are dependent on the calibration data used, even changing the scale of the data could affect the optimal calibration. Baldassarre et al. [19] proposed a method of calibrating flood models while admitting that the data input into them was uncertain as a solution to this problem. Multiple satellite scenes at multiple resolutions where used to create possibility of inundation maps model results where then used to create model predictions for inundation probability and calibration was carried out on the probability image instead of the raw inundation maps. Mason et al. [45] determined calibration could be more accurately carried out by comparing the predicted water elevation with observed elevation instead of only focusing on the areal wet dry pattern predicted and observed. Cook and Merwade [18] studied how changes in topographic data effected 1D and 2D flood models. For both types of modeling, inundation area was found to decrease with increased horizontal and vertical precision of the topography. In addition, the effects of placement and numbers of model elements was tested. The number of cross-sections included in the 1D model was found to strongly influence the predicted results.
Sampson et al. [56] tested how the use of terrestrial LIDAR data with a resolution of 10 cm would effect predictions of flood models (aerial based LIDAR normal has resolutions between 1 - 5 meters). They found that when modeled at this resolutions urban structures such as street embankments clearly influenced modeled flows and that use of terrestrial LIDAR data would have a strong influence on flood risk assessment. Stephens et al. [63] tested a method of model calibration where water surfaces where estimated by combining a flood outline extracted from Synthetic Apature Radar (SAR) with an assumption that water elevations change gradually to estimate water elevations for a flood scene from the combination of terrain and radar reflectivity. Model calibration was then attempted against the resulting height field instead of the two dimensional outline of the flood scene. This technique was found to give more accurate calibration and reduce topographic dependencies resulting from the calibration process. Dottori [23] tested flood models in an urban region where elevation values from the modeled event had been recorded. The testing indicated that urban area would be best modeled with fully dynamic 2D models if the flood scenario in question would cause transitions between subcritical and supercritical flows. In less complicated scenarios, simpler models are usable.

One of the difficulties with inundation models particularly as higher resolution models or more complex physics are used, is the time required to obtain model results. Cook and Merwade [18] studied how well multi-processing could be applied to accelerate the LISMIN Flood Model which is a derivation of LISFLOOD-FP. The study tested parallelization using OpenMP, MPI, and dedicated processing hardware. The OpenMP implementation was found to be easier to create where as the MPI implementation preformed more effi-
ciently and scaled to larger numbers of processors. The tested dedicated processing hardware was found to be faster than an single available processor but did not outperform the parallel codes when given sufficient codes.

One of the reasons that the computational cost of flood models increases quickly is that the stable time step for the water transport equations used in many early models scales with $(1/\Delta x)^2$ (Hunter et al. [37]). Thus when the precision of modeled topography is doubled, the computation time required increases by a factor of 16. Similarly, changing a model from 30M satellite based topography to 1M LIDAR topography would require 810,000 times more processing time. Bates et al [10] presented new physics equations for flood inundation where the stable time step scaled with $1/\Delta x$. The new formulations where tested and speed gains of over 1100x where reported for some test cases. The final speed gains where dependent on model resolution and the slopes of the generated water surfaces. Wang et al [68] introduced new models for Diffusion Wave Models (DWM) also called Zero-Inertia Models which are common simplification of Navier Stocks equations in which the inertial components are discarded. The new formulations did not require a flow limiter which was previously used to prevent checker boarding issues. The new formulation was tested with two different adaptive time step formulas: the first is the one proposed by Hunter [37] the second biased of the Courant-Friedrichs-Lewi (CFL) condition. The CFL biased time step provided speedups of up too 44x in tested cases. Other studies have found this techniques to result in massive gains to computational speed. Dottori and Todini [22] test the application of inertial physics formulation instead of the older diffusion based formulas in model based on cellular automata. The reported reduction in run-time was 97% with and
additional 1.2x to 4x speedup coming from the application of a dynamic time step. Leandro et al. [40] developed parallel diffusion wave flood model and tested the speed increases that could be achieved with up to 12 processors. The model was implemented in a tested using both using Mathlab’s parallel computation toolbox and OpenMP with the Fortran language. The reported speedups ranged from 1.7 to 5.2 for the Mathlab implementation and 1.2 to 1.7 for the OpenMP implementation.

Most studies of the effects of LIDAR data on flood modeling have found that, in general, increasing the resolution of the model increases the accuracy of model results. Dottori et al. [21] have cautioned that increased accuracy should not be assumed to follow increasing the resolution of a modeled region. They advise that the potential sources of uncertainty for input data should always be identified and evaluated. Furthermore, the primary goal of flood mapping is to provide useful predictions for inundation and or flood extents which can be hindered by too much detail. Because of these concerns it is recommended that both modeling complexity and the resolution that modeling is attempted on should be considered on a case by case basis and assumptions about either greater physical accuracy or model resolution leading to better results be put aside.

Almost all discussed modeling and validation efforts have been for gaged rivers where there is historical data about flow amount and water elevations from various measuring stations along a river. The input flows into a modeled region is one of the basic required parameters for any physically based river centric inundation model; models predicting flooding directly from precipitation need rainfall maps as inputs as well. Ali et al. [57] proposed a method to allow flood modeling for areas where no historic data exists. The
proposed method works in two phases: first, regional flood data is predicted and then the predicted data is used for modeling instead of historical data.

Zang et al. [71] introduced a flood model that works by growing a triangular network outward from a modeled river channel. This model works directly with water elevations and as such it very similar to FESM, however the use of a triangle network for calculating the flood surface makes it a TIN based elevation model where as FESM is raster based elevation model. Despite this difference how flood surfaces are extended the two model share several traits most importantly limited inputs required for modeling and the ability to rapidly predict inundation maps with limited hardware.

A related area of research for two dimensional models is how GIS information can be used by two dimensional models to obtained the required friction information for their use [70] [60] [43], as well as how they are influenced by data scale [25].

This chapter introduces a new flood inundation model named FESM (Flood Event Simulation Model) that was designed to be usable in response to an ongoing emergency event. FESM’s design has the following goals

- Minimizing the amount of information necessary to make a prediction as abundant information is not necessary available in an emergency situation.
- Simplifying requirements for model setup
- Rapid processing of data to generate an inundation map.

FESM differs from most other models previously discussed in that its basic input is water elevation at measured points along river channels instead of flow amounts. The only model that appears to work in a similar manner is [71]. The flood surface is made by
extending a sloped surface from the river channels that are being modeled. In the remainder of this section describes the background, working, and testing of the FESM model.

2.3 FESM

FESM (Flood Event Simulation Model) is a flood inundation model designed to replace the FEAT Model used by the USACE (US Army Corp of Engineers). The required inputs to the model are the topography in which the simulation will take place in the form of a georeferenced DEM, the path information of river channels, optional path information of sub channels connecting to the main simulation channels, and water elevation information for known points along the simulation channels. FESM differs from most flood inundation models in that it does not consider either flow or friction, and as a result does not need information about these conditions. Another key difference is that FESM does not directly implement either the Naiver Stocks equations, the de Saint Venant equations (Shallow Water equations), or any obvious approximation, of these equations (any attempt to create a water surface is at some level a approximation of the de Saint Venant equations). Water elevation in channel are determined by the input data and linear interpolation along channels paths if the resolution of the simulation grid is smaller than the spacing between known water elevation points. Lateral propagation of water elevation is done by selecting grid locations adjacent to the expanding flood surface, and determining which adjacent locations are potential sources of inundation. The water resulting water level and a grid location depends on the water levels of such sources modified by slope rules.
2.3.1 Background

In 1987 the USACE (US Army Corp of Engineers) published the Wetland Manuel (Technical Report Y-87-1). In this report the Corp suggested that wet lands be determined be based on if an area was inundated continuously for 5% of the growing season [38]. 1992 Energy and Water Development Act, Johnson Amendments required the use of this 5% rule for wetland delineation [62]. As a result of this law the flood model FEAT was developed in 1998. FEAT predicted flood delineation by creation of a water surface through IDW (inverse distance weighted) interpolation of river gage elevation data [44]. An initial flood surface was created with by comparing the interpolated water elevation values with the land elevation values. A cleaning sweep then removed flood location that lacked a path back to a source gage. FEAT suffered from several flaws, the most important of which was curves in the interpolated water surface would under certain circumstances projected flooding to cross levees, even when the water elevation in the channel beside the levee was below the levee height. Work on solving this as well as other problems with the FEAT model lead to the development of a new model called FESM.

2.3.2 FESM Inputs

A simulation with the FESM model requires three input files and supports two optional input files the required and optional files are as follows.

- A DEM containing the terrain data for the simulation
- A ESRI Shapefile containing the digitalized paths of the simulation channels as a series of poly lines. The coordinates used for the line segments must be in the same geospatial projection as the terrain DEM. The database portion of the channels shapefile must contain the id of the upstream and downstream gage associated with each digitized segment of each channel.
• A ESRI Shapefile containing the digitized paths of any sub channels to be used in the simulation. {optional}

• A ESRI Shapefile containing the location of known water elevations points. This file must also be in the same projection as the terrain DEM. The database portion of this file contains water elevations for each gage and each event recorded in the file.

• A ESRI Database file with fields matching the fields of the gage files database portion. This file when provided sets the initial lateral slope at each position recorded in the gages file.

2.3.3 FESM outputs

A run of the FESM model produces 4 output files. They are

• A flood elevation surface, which is a DEM that encodes the water elevation for each point on the input surface DEM or NODATA if inundation did not occur at that location.

• A flood image, which is a georeferenced image recording 1 in locations where inundation occurred and 0 in all other locations.

• A cost surface, which is a georeferenced image that records the number of grid locations water traversed to inundate each location.

• A slope surface, which is a georeferenced image that records what the current lateral slope of the flood surfaces was at each position. The slope surface is only generated if FESM is running with either relative channel slopes or gage defined slopes.

2.3.4 Model Overview

FESM operation works in two primary phases. The first phase takes river elevation data from measuring stations (river gages) and interpolates this data to generate three dimensional poly lines that represents the water elevation along each modeled river channel. These poly lines are then processed to determine the downstream slope at each segment. Once generation of elevation and slope are completed, a spreading algorithm flood cells that are adjacent to currently wet locations by setting the new locations water elevation to
the average of all adjacent wet locations reduced the average slope of adjacent wet locations. A more technical description of the events and processes involved in a model run can be found below.

1. Event Selection; One of the events listed in gage file is selected for evaluation.

2. Gage Evaluation: The 2D geo-spatial coordinate for each gage is combined with stored elevation to create a set of 3D points representing the known water elevation points.

3. Channel Evaluation:
   (a) The poly line segments representing the paths of channel are paired with the indicated source and destination gages.
   (b) Water elevation at each point along each poly line is linearly interpolated at each recorded point based on summed distance of this line segment and all preceding line segments from the source gage point.
   (c) Additional points are inserted into the polylines to insure that each line segment does not cross more than one grid location.

4. Sub Channel Evaluation:
   (a) The nearest 2D point of major channel to the final location of a sub channel is located
   (b) The elevation from that point is applied to every point of the sub channels poly line segment digitization.
   (c) Additional points are inserted to insure that any line segment does not cross more than one grid location.

5. Main Loop Initialization:
   (a) For each point in a 3D point set created my by merging the points that create the line segments for the channels and sub channels.
      i. Calculate the indices (i,j) of the grid location in the land elevation grid with the same geo-spatial location.
      ii. Set the water elevation for (i,j) to water elevation of the source point.
      iii. Set the flood image value at (i,j) to 1.
      iv. Set the cost value at (i,j) to 0.
      v. Set the status value at (i,j) to 1 (processed).
      vi. Calculate the indices of the four neighboring grid locations and place those with a status value of 0 onto the processing queue
(b) Initialize the slope surface according to the current slope rules.

6. Main Loop:

(a) Retrieve a location index pair \((i,j)\) from the processing queue.
(b) Get a list of neighbors which are flooded.
(c) Remove neighbors whose water elevation is less than the current positions land elevation
(d) Calculate a water elevation from the average of remaining neighbors slope adjusted water elevations
(e) Calculate the resulting water depth and compare it to a minimum propagation depth (this is done to prevent the propagation of extremely shallow planes of water when the water surface slope and terrain slope are close to each other).
(f) if the calculate water elevation is greater than the current surface height and the calculate water depth is greater than the minimum propagation depth
   • Set the water elevation at \((i,j)\) to the calculated water value
   • Set the slope at \((i,j)\) to the average of contributing neighbors slopes.
   • Set the status at \((i,j)\) to 1 (processed).
   • Set the flood image value at \((i,j)\) to 1
   • Calculate the non flooded neighbors of \((i,j)\)
   • Add non flooded neighbors with a status value of 0 (new) to the processing queue
   • Set the status of non flooded neighbors to 2 (pending).
(g) repeat until the output points set of the previous loop is empty

2.3.5 Slope Rules

The FESM model was originally developed for simulation of backwater floods, as such it initially assumed that the slope of the propagated water surface was zero. This assumption became invalid when FESM was adapted to be used in more general circumstances. To allow simulation of other types of flooding FESM support three slope rules. They are

• Global Slope: In this mode each time water moves from one grid location in the simulation the water elevation is reduced by a user specified amount. This is simplest available model and performs poorly if the simulation contains multiple bodies of water.
• Channel Relative Slope: In this mode after water elevation as been interpolated for a digitization of a channel. The down stream slope of the channel is calculated and that slope multiplied by a user specified scaling factor is stored into a slope surface. The slope surface is the propagated along with the water surface in the main loop.

• Gage Relative Slope: In this mode a separate input file supplies lateral slope information that is paired with each recorded water elevation for the gages. The slope is interpolated along the channels in the same way as water elevation and then stored in the slope surface for use in the main loop.

2.4 Modifications

Several important modifications have been made to the FESM model since its initial creation. These include support for arbitrarily large data inputs, multiprocessing support and integrations with the flood models Hec-RAS and FLO-2D.

2.4.1 Large Surface Support

FESM supports arbitrarily sized inputs by using an internal virtual memory system. Instead of trying to load inputs files into memory at the beginning of execution, input data is loaded in blocks in "lazy" manner. When the amount of memory loaded for any surfaces passes user definable bounds the least recently accessed block is written to disk and the newly required data is written into the now free memory block. This system allows computer with limited memory to successfully process arbitrarily large input files provided sufficient disk space is available for the backing files associated with each computational surface.
2.4.2 Multiprocessing

In order to efficiently process large inputs FESM, will use all available CPUs on the host machine during the flood computation loop. This is done by splitting the cells to be check for each cost into n different lists (one list for each available processor). Each list is then processed a separate CPU. The computational surfaces are shared between all CPUs. Locks with data block level granularity are used to prevent different processors from simultaneously modifying the same locations while allowing simultaneous processing of locations in different data blocks. After all CPUs have completed their current processing, the resulting lists of new locations to be checked are merged, and then redistributed to the CPUs. The process keeps the work load of each cpu balanced.

2.4.3 Integration with Other Models

FESM can start its computation using model outputs from either Hec-RAS or FLO-2D to generate the initial flood location and slope values. When used this way FESM operates as a mapping tool generating a flood surface using water elevation information from the source model.

2.5 Model Validation and Verification

If a model used for emergency response it is important that it is both validated, its results are checked to insured it behaves in a logical manner, and validated, the predicted results are compared to occurring results. FESM was validated as part of the Yazoo Backwater Delineation Project during which multiple FESM flood predictions of historic events
where compared to recorded flood images. Later during the 2011 flood FESMs results were verified with helicopter based surveys.

2.5.1 Yazoo BackWater Delineation Project (2006)

The FESM model was validated as part of the Yazoo Backwater Delineation Project [20]. During the validation process, flood inundation maps output by FESM where compared to previously captured satellite imagery of historic flood scenes. Table 2.1 shows the FESM output results, with 3 relative slope settings, compared to imagery of 6 historic flood scenes. Two observations can be made based on this data.

- Increasing the relative slope of lateral propagation always reduces the coverage generated by FESM
- Increasing the slope tends to increase the amount of locations missed by the FESM. In particular the rate of increase in false negatives grew faster than the rate of false positive in each scene, for sloop values greater than 0.3 relative slope. This is why the percentage accuracy in coverage dropped.

Based on these observations, insufficient slope is not a likely cause of coverage failure. What then is the cause? Visual inspection the results for 3 flood scenes can be seen in figures 2.1,2.2 and 2.3. In each figure, areas in red are areas flood by both the flood scene and FESM, areas in blue are flooded only by the flood scene, and areas in green are flooded only by the FESM model. In each case FESM tends to correctly predict all flooding near the river channels and usually slightly over-predicts this amount. The areas missed by the model are mostly small scatter and unconnected regions that may be attributed to the accumulation of precipitation, natural or artificial lakes, and deliberate inundation, but not out of bank inundation. A particularly good example of this can be seen in Figure 2.2
The above table shows Chanel Relative Slope predictions (for values of 0.3, 1.0, and 1.3) for six major flood events. Scene Acres is the number of flooded acres in the satellite image of the event; Predicted Acres is the number of Acres flooded by the model simulation; % Correct is the % of predicted acres that are part of the observed flooding; % False is the % of predicted acres that are not part of observed flooding, i.e. false positives. In general increasing slope lowers both accuracy and false positives. The results shown in this numerical analysis were found to be vulnerable to systematic error because FESM was not utilizing an integrated rain model. This effect was more pronounced with smaller floods. To provide a more useful and accurate analysis of FESM’s accuracy, a visual analysis was performed (see Figure 2.1 - Figure 2.3).

Another source of error is temporal effects. The model assumes that sufficient time is available for propagation to complete. However there is no guarantee that the flood scenes where captured at the maximum extent of a flood. The flood waters in any given scene could be in a process of either rising or falling. For example, consider the inset of figure
2.3. In the main image, the magnified area is shown as the model predicts with state data from January 13. The magnified area shows the model's prediction using stage data from 4 days prior, which gives a much better fit. In this case, water was likely draining back into the river when the satellite image was taken.

Another example of this can be seen in Figure 2.4 which shows the January 13, 1983 flood modeled with a slight negative slope. Based on these studies and other similar studies for different regions, FESMs were determined to accurately model out of bank inundation with a tendency to slightly overestimate flood extent.

2.5.2 Kansas City Flood (2011)

This section contains a series of comparison images between flood imagery captured by a helicopter and predicted flood results. These images were captured during survey flight around Kansas City during the 2011 flood. The comparison images were used to verify the accuracy of flood predictions using the FESM model.

In April and May of 2011, a series of four storm systems crossing the Mississippi Valley, combined with springtime snowmelt from the upper Midwest, produced one of the most severe flood events in the history of the United States. During the flooding, as part of the Missouri Flood Event mapping and monitoring effort, helicopters equipped with GPS-aware cameras were sent to key areas of interest within the Kansas City District to perform flood damage assessments. With access to GPS-keyed aerial photographs of flooded areas, together with the flood predictions for the same areas, generated by FESM, it was possible to cross compare the flood model with real-world survey images. To accomplish this, the
Figure 2.1: FESM comparison to satellite for 10 March 89 [20]
Figure 2.2: FESM comparison to satellite for 21 March 87 [20]
Figure 2.3: FESM comparison to satellite for 13 Jan 83 [20]
Figure 2.4: January 1983 Comparison [20]

Negative slope being used to better match Jan 83 flood.
FESM flood model was superimposed over true-color satellite-image maps provided by Google Earth. Visually distinct landmarks were used as points of comparison [1].

![Image a](image_a.png) ![Image b](image_b.png)

Figure 2.5: FESM predictions compared to GPS survey photos.

During the major flood events of 2011, the Kansas City District of the USACE tested the real-world accuracy of FESM [1] by projecting flood model predictions onto true-color satellite-image maps provided by Google Earth (see image a, above). Image (b) shows an aerial photograph of the marked area in image (a). In image (b), it can be seen that several buildings (white rectangles) are flooded, while the road to their left remains dry, matching the FESM prediction.

### 2.6 Conclusions

Given stage data and channel path information, FESM is able to rapidly predict flood extents. Validation and calibration performed during preparations for the 2006 Yazoo Backwater Project Report revealed that FESM slightly overestimated flood extents using two traditional slope propagation methods, Global Slope and Chanel Relative Slope (see Table 2.1, p. 23); because of this, a third slope propagation method, Gage Relative Slope, was devised (see Section 2.3.5, p. 19). In later aerial survey comparison tests, during the 2011 Missouri-Mississippi Flood, FESM (again utilizing the Channel Slope propagation...
Figure 2.6: FESM predictions compared to GPS survey photos.

Image (a) shows flood model predictions superimposed over a true-color satellite-image map provided by Google Earth [1]. Image (b) shows an aerial photograph of the marked area in image (a). In image (b), it can be seen that the wooded area immediately to the right of the road is flooded, while the road itself remains dry, matching the inundation surface predicted by FESM.

Figure 2.7: FESM predictions compared to GPS survey photos.

Image (a) shows flood model predictions superimposed over a true-color satellite-image map provided by Google Earth [1]. Image (b) shows an aerial photograph of the marked area in image (a). In image (b), it can be seen that the area of flooding encompasses the road and buildings at the base of the hill, matching the inundation surface predicted by FESM.
Figure 2.8: FESM predictions compared to GPS survey photos.

Image (a) shows flood model predictions superimposed over a true-color satellite-image map provided by Google Earth [1]. Image (b) shows an aerial photograph of the marked area in image (a). In image (b), it can be seen that the area of flooding encompasses all of the forest in the foreground, and extends into the fields and woods across the river, matching the inundation surface predicted by FESM.

Figure 2.9: FESM predictions compared to GPS survey photos.

Image (a) shows flood model predictions superimposed over a true-color satellite-image map provided by Google Earth [1]. Image (b) shows an aerial photograph of the marked area in image (a). In image (b), it can be seen that the raised road beyond the bridge remains dry, while the surrounding area is flooded, matching the inundation surface predicted by FESM. Note, the flood simulation shows the bridge itself as being flooded, because the ground below the bridge is flooded; the model ignores all objects, such as bridges, that are not part of the input DEM (Digital Elevation Model).
Figure 2.10: FESM predictions compared to GPS survey photos.

Image (a) shows flood model predictions superimposed over a true-color satellite-image map provided by Google Earth [1]. Image (b) shows an aerial photograph of the marked area in image (a). In image (b), it can be seen that the wooded area at the base of the bridge is flooded, while the raised road beyond the bridge (upper right corner) remains dry, matching the inundation surface predicted by FESM. Note, the flood simulation shows the bridge itself as being flooded, because the ground below the bridge is flooded; the model ignores all objects, such as bridges, that are not part of the input DEM (Digital Elevation Model).

Figure 2.11: FESM predictions compared to GPS survey photos.

Image (a) shows flood model predictions superimposed over a true-color satellite-image map provided by Google Earth [1]. Image (b) shows an aerial photograph of the marked area in image (a). In image (b), it can be seen that the raised road (bottom) remains dry, while the flood extends over the fields beyond, ending at the wooded slope edging the area; this matches the inundation surface predicted by FESM. Note, the flood simulation shows the bridge itself as being flooded, because the ground below the bridge is flooded; the model ignores all objects, such as bridges, that are not part of the input DEM (Digital Elevation Model).
method) output predictions which were found to very accurately match observed flood extents when compared to photographic records (see Table 2.5 - Table 2.11).

Additional work in the following new areas will likely improve the quality of FESM’s predictions

- Testing of gage-based slopes when calibrating FESM for a given region. Gage-based slopes should allow better matching of model predictions to observed results, particularly in flood events covering large river reaches (large domains).

- Testing of methods to utilize historical (or incoming) hydrographs to predict the correct gage-slope for each sensor location. Such methods would allow automated use of gage-slope based lateral water propagation and avoid error-prone manual configuration of gage slope values. Because manual configuration is time-consuming and tedious, it is often skipped in favor of using less-accurate channel-based slope propagation.

- Unlike previous flood models, FESM is fast enough for real-time flood simulation. Therefore, further research should test FESM’s accuracy and performance when being fed real-time water elevation data from a sensor net.

This chapter has outlined the flood model FESM, which, since its creation, has been used by the Mapping, Modeling and Consequences branch of the US Army Corps of Engineers to manage intervention and relief efforts in the majority of major flood events in the Southern United States (since 2005, post hurricane Katrina). It excels in flood prediction for emergency management since, compared to other commonly used flood models, it requires orders of magnitude less time to generate results. This speed is possible because FESM is not required to directly solve the differential equations governing water movement, in any form, which would otherwise require thousands, or millions, of time steps; FESM instead directly manipulates the flood surface elevation field. The ability to rapidly model and predict flood extents is of particular importance because floods are currently both the most frequently occurring and the most costly natural disasters in the United
States, accounting for millions to tens of billions of dollars of damage and much loss of life each year. FESM is significantly simpler to set up and calibrate than other previous flood models, meaning more and more districts are able to easily utilize FESM for faster and more accurate flood response, and will more easily be able to run simulations to plan for future contingencies.
CHAPTER 3
DATA TRANSFER OPTIMIZATION

3.1 Introduction

The first chapter of this dissertation introduced FESM, a GIS tool which rapidly generates flood water surfaces and inundation extents using elevation data. After working extensively with the USACE, the author noted a need for a tool to allow interactive analysis of multiple flood scenes in the same geographic region. Software able to accomplish this would have to be capable of simultaneously rendering numerous extremely large data sets, at high speed, for real-time interactive display. To address this need, the author created a visual analytics program, Dynamic Raster Overlay (DRO) (see Chapter 4).

Because of the extreme size of the datasets (flood maps) being considered, it was necessary that DRO be able to process datasets that were too large to fit in GPU memory. To overcome this problem, and to ensure optimal render speed for real-time interactivity, a new algorithm was designed to facilitate highly efficient data streaming and visualization with OpenCL image objects. The Dual Buffer Algorithm (DBA) allows data transfer to occur on one set of buffers while data-mapping and processing occurs simultaneously on another set of buffers. Essentially, when programs utilize the DBA, the graphics card is always mapping and/or processing old data while simultaneously downloading new data. By
using the DBA, transfer rates should approach the throughput cap of the PCIE bus between
the graphics card and the CPU.

This chapter introduces the Dual Buffer Algorithm, and details several tests of the effi-
ciency of the DBA and component processes in comparison to other possible approaches.
This work extends the available body of knowledge through an investigation of the ef-
fectiveness of using images (as opposed to buffers) in the use of visualization OpenCL
kernels. This work also identifies and explores the non-linear costs of moving data to im-
age buffers as image size increases. A comparison is also conducted regarding the relative
run times of kernels that differ only in their data-storage structures (images versus buffers).
In culmination, the results of these tests provide necessary information to choose memory
transfer structures for specific rendering and analysis tasks.

3.2 Motivation

Modern GPUs are powerful tools utilized for volume rendering, scientific modeling,
medical imaging and other tasks involving parallel computation. For real-time analysis,
rapid performance is desired so that the required frame-rates for human interaction can be
achieved (ie., $\geq 10$ fps). As data transfer between the CPU and GPU is slow, the use of
GPU memory, or buffers, is vital for performance in visualization applications.

Currently, three factors limit GPU performance in scientific visualization tools. First,
processing of data on the GPU is limited by the throughput of the connecting bus between
the GPU (graphics memory) and the system memory (ie., main memory / CPU memory).
Secondly, graphics memory is limited, meaning large data sets must be segmented or sub-
sampled by some method in order to be processed. Finally, although *image buffers* can be processed more efficiently on GPUs than *data buffers*, image buffers are used almost exclusively in entertainment applications, while most scientific applications rely on less optimized data buffers.

Though data buffers are more commonly used in visualization, this work uses OpenCL image buffers primarily for the processing and depiction of data, in this case, with a GIS tool named DRO (Dynamic Raster Overlay), a flood visual analytics system. Image buffers offer cached access to GPU memory at the cost of imaging coding on upload; this work outlines approaches to diminish this cost while demonstrating the benefits of image buffers.

Basic interaction with a GPU has three main steps: uploading data to the device, processing or display of uploaded data, and retrieval of results. Because both upload and download are significantly slower than the potential processing speed of a GPU, performance is best when those steps are performed only once. This can only be done if the dataset to be processed can fit entirely into graphics memory. When this is not the case, or when the size of data can not be known beforehand, two strategies are commonly employed. Data can be sampled to provide lower resolution information that will fit in available graphics memory; alternatively, when the goal of processing is computation instead of display, it may be necessary to process the input data at its original density. In this case, data must be subdivided into pieces that will fit into available graphics memory. There are at least two ways to attempt this subdivision. Data can be spatially subdivided, or broken into segments, that can be processed discreetly. However, if steps are not taken to synchronize the processing of boundaries, this approach can cause artifacts at the edges.
of segments. Alternatively, to avoid subdividing data sets, processing may be done iteratively, building a working composite which is continuously updated as layers of data are processed. If the scope of the data is sufficiently vast, it may be necessary to apply both spatial and iterative subdivision to process the data set. Thus, intelligent management of the CPU–GPU transfer of data is paramount.

OpenCL is the only cross platform, hardware-independent, high-level graphics hardware programming interface widely available. When using OpenCL to move data to or from the GPU, two main formats are available, image buffers (henceforth referred to as images) and data buffers (henceforth referred to as buffers). While buffers are simple linear memory structures, images have an internal format dependent on the GPU driver, normally a block-based structure. For data processing, images have several advantages over buffers. The primary advantage is that image reads are cached [2], allowing an order of magnitude faster access than data that resides in GPU main memory. Images are also easily shared with the graphics environment for display, and support accelerated packing and unpacking of data. However, despite these advantages, current literature focuses primarily on data transfer between data buffers, and seldom discusses methods of efficiently streaming data into image objects. This work explores the application of image buffers to data processing and visualization.

This work was motivated as part of a solution designed to perform ensemble processing and visualization of flood coverage images (See Chapter 4). Given a collection of simulated and observed flood data, the system allow users to rapidly find regions of high and low flood overlap. A second goal was to identify image clusters, which, within the scope
of the project, were defined as subsets of input data where all members had a significant
degree of overlap over the entire data. The analysis program was required to handle ex-
tremely large input images, particularly if LIDAR (LIght Detection and Ranging) datasets
were used. In addition, there was no hard limit to the number of images that might be
considered simultaneously. The need to process an unknown number of potentially large
images required a solution capable of handling input sets so vast that all data would be
unable to fit in GPU memory. However, CPU-based analysis would not be fast enough to
provide real time interaction with the large datasets. Coupled with a requirement that users
be able to frequently and quickly change the working set of images, a streaming OpenCL
approach was utilized.

The DRO visualization system consists of several OpenCL kernels that perform the
outlier and clustering calculations whose results are displayed in OpenGL. While the ker-
nels are specific to this particular domain, the approach can be applied in other systems
with coupled GPU-powered analysis and visualization. Initial results using a single re-
ceiving image were slower than desired, eventually leading to the development of the dual
buffer algorithm herein discussed.

3.3 Related Work

In scientific applications, early research on hardware accelerated rendering focused
on volume rendering, specifically for displaying the data obtained from medical scanning
equipment. One early example is Cabral et al. [14] where graphics hardware was used
to increase the performance of back-projection based volume rendering; hardware accel-
erated results were more than 100 times faster than the CPU. Lum et al. [41] illustrated how graphics hardware and parallel rendering could be combined to allow visualization of large time varying datasets. Hadwiger et al. [27] later illustrated another method whereby graphics hardware could be used to accelerate volume rendering through the inclusion of a segment volume used to isolate individual objects in the volume. Eventually frameworks that simplified access to hardware acceleration and parallel rendering began to appear (e.g., Bhaniramka and Demange [12]). Slightly more recent work includes a summary of techniques usable in real time volume rendering [24] and advanced illumination methods for volume rendering [28]. Recent work in ray casting techniques (most of the early volume rendering was back projected ray casting) includes Lux and Fröhlich [42] and Zhu et al. [72]. Early works use texture GPU memory in a manner that presages the algorithm described here, though our work uses general purpose image buffers; our data is also streamed.

As the aforementioned methods were being developed, graphics hardware continued to advance. The fixed functionality pipeline of early graphics hardware was replaced by programmable units. These units first had to be controlled with assembly code; for example NVIDIA’s language cg [51]. From assembly languages, the programming of graphics hardware progressed to high-level graphics programming languages with syntax modeled on C, namely Microsoft’s High Level Shader Language (HLSL) [46], and OpenGL’s OpenGL Shader Language (GLSL) [5]. For general computation purposes, both of these languages would be overtaken by new languages specifically for this purpose (all previous languages were designed for graphics but could be forced to do general computation). The primary
languages in this category are NVIDIA’s Cuda [52] and the Khronos Groups’ OpenCL (Open Compute Language) [4].

In the area of optimizing data streaming, Vo et al. [67] introduces a framework supporting multi-core systems. Unfortunately, the system described does not interact with GPUs, largely due to limitations of the connected graphics framework (VTK).

Basic information on recommended usage for OpenCL and CUDA can be found in [3] for AMD GPUs and [2] for NVIDIA GPUs. There are several notable studies on streamed data processing with CUDA. In their work on the Dax toolkit [47], Moreland et al. present a high level framework capable of reorder task to minimize, or in some cases eliminate, overhead from I/O. In "CudaDMA" [11], Bauer et al. describe a framework that optimizes stream performance in CUDA by using warp specialization techniques and support for different types of buffering models. Another framework (ISP) and study is presented by Ha et al. [59]; this study included analysis of different streaming modes, and the effects of reordering upload, execution, download, and optionally compression of data. Another framework with capabilities for streaming data between different types of processing units is presented by Vo et al. in [66]. A recent work of Sewell et al. [58] presented a framework supporting use of CUDA capable graphics hardware for parallel visualization. Another recent work in this area is Rosen [55], which describes a system for visualizing memory conflicts generated when running CUDA kernels. Such conflicts, caused by hardware dependent variables, greatly slow the performance of computation, and must be checked for and solved on a per-device basis. Studies using OpenCL are more limited. One important
study using OpenCL was done Spafford et al [61], which studied the effects of buffering techniques, work group size, and data transfer size, when using OpenCL buffers.

Our work extends available work by evaluating the effectiveness of using images (as opposed to buffers) in the use of visualization OpenCL kernels. We also study the effect of the non-linear cost of moving data from buffers to images, especially has image size increases. Finally, a comparison of the relative run times of kernels that differ only in data structure (images or buffers) is conducted. These tests together provide the necessary information to choose appropriate memory transfer structures.

### 3.4 Algorithm

A naive approach to streaming data involves the use of a single pair of buffers to transfer data between the client and device. Pseudo code illustrating this method is shown in Figure 3.2. The algorithm herein presented exploits the ability of the GPU to simultaneously transfer and process data. Pseudo code illustrating this method is shown in Figure 3.3. When a single pair of buffers or images is utilized, the GPU will, at best, cycle between uploading and processing data. However, by utilizing two pairs of either images or buffers to send and receive data, processing may take place in one pair while I/O takes place in the other. The flow of operations and their dependencies for the dual buffer method are illustrated in Figure 3.4.

For either the naive single buffer approach or the proposed dual buffer approach, multiple implementation methods are possible. In the course of testing the efficiency of the dual buffer approach, a total of three implementation methods were tested.
The first method, which seemed to be the obvious approach, utilized `clWriteImage()` to transfer data directly from client memory to permanent device-resident image structures. The control algorithm (1b initial) employed a single device-resident image, while the experimental algorithm (2b initial) utilized two device-resident image structures. Unfortunately, the function `clWriteImage()` first creates a hidden duplicate linear-format copy of the data in client memory, and secondly a linear-format copy on the device side, before finally transforming the data into image-format on the device side. This resulted in a transfer rate on the test system that was only about 40% efficient.

A second implementation method that eliminated the unneeded data copy in client memory by using data buffers initially before converting to images was also tested. The limitation of this approach is that data transfer and transformation (into an image format) was coupled together, preventing other memory transfers from being initiated until after the final transformation of the data had completed. Due to this bottleneck, we focus our study on the naive approach and the optimized one discussed next.

The third and final implementation method decoupled the transfer and transformation of data. This was accomplished by creating permanent dedicated receiving buffers on the device, which were paired with permanent dedicated image structures. The control algorithm (1b final) utilized one buffer-image pair, while the experimental algorithm (2b final) utilized two buffer-image pairs. Just as with the second implementation method, data was initially stored in OpenCL buffers allocated in client memory. Data was then transferred from client memory to the receiving buffers as one operation using `clCopyBuffer()`. A second operation, `clCopyBufferToImage()`, was utilized to handle the transformation of
data into image format once the transfer completed. The advantage of this system is that, the entire time data in one device buffer is being transformed into an image, as well as the time taken for the resulting image to be processed, becomes a window in which data can be transferred from the client into the second buffer-image pair.

All described methods were designed for use with an asynchronous OpenCL queue. While it is possible to utilize these methods in synchronous mode, efficiency will be poor. OpenCL event references were used to coordinate tasks within each variant algorithm. It is important to ensure that a kernel does not execute utilizing as input either an image that was currently being updated, or an image that had already been processed and not yet been updated. Likewise, event references were utilized to ensure that new data was not loaded into an image or buffer that was currently being employed by either a kernel or a buffer-to-image copy.

3.5 Testing

Tests were performed in three different areas. The first tests measured the impact, on the overall runtime of an accumulation kernel, of using either single or dual buffer algorithms for data transfer. The second tests determined how buffer-to-image copying performance changed depending on target image dimensions. The third set of tests examined the runtime effects of using either images or buffers in computation kernels which performed the same calculations.

All tests were performed on a machine running Windows 7 Sp1 with 8 GB of installed RAM and a single 1080p monitor. The system utilized a single graphics card, an AMD
Radeon HD 7950. Tests were performed with default settings and using the Catalyst 13.4 driver.

3.5.1 Dual Buffer Tests

Timing data was gathered by recording the total time necessary to transmit each of N images from client to device memory, and process the received images with an accumulation kernel. Tests were conducted for N equals 10, 100, 1,000, and 10,000. In all cases, timing began immediately after the accumulation image-structures used by the kernel were cleared and ended as soon as the final call to the accumulation kernel was reported complete by the OpenCL runtime. Actual time values were provided by the function gettimeofday() implemented in Windows using GetSystemTimeAsFileTime(). All recorded times were the result of averaging 100 repeated tests of the same number of iterations.

Timing tests were initially performed only on images with dimensions of 4k (3712x4416). After these tests had been concluded, the optimized algorithms (1b final, 2b final) were tested again with images measuring 2k (1856x2208) and 8k (7424x8832). This was done to see if image dimensions influenced processing time. Original plans also called for an additional test image set measuring 16k (14848x17664), however the graphics hardware utilized did not support images of this size.

3.5.2 Buffer to Image Transformation Tests

Testing of the single and dual buffer algorithms revealed that the time required to transform data from buffers to images in OpenCL was not linearly associated with image size. Ideally, in order to determine the buffer-to-image performance for each possible image
dimension, the entire range of possible image dimensions would be tested. However, the domain of possible image dimensions is too large for exhaustive testing, with most modern graphics hardware allowing for over 268 million possible image dimensions. Therefore, two sampling grids were utilized to test buffer-to-image transfer rate for different image dimensions. The first sampling pattern began with a 128x128 pixel image and increased image dimensions in steps of 128 pixels, culminating in an 16,384x16,384 image and sampling 16,384 possible images dimensions. The second sampling pattern started with a 100x100 image and increased image dimensions in steps of 100 to a maximum image size of 16,300x16,300, resulting in 26,569 samples. Two sampling patterns were utilized to increase confidence that any resulting pattern was not simply a result of a given sampling pattern. For each image dimension, time required to transform data from an appropriately sized buffer to the image was measured. Each measurement was repeated 1,000 times, and the average transformation time was recorded.

3.5.3 Buffer vs Image Kernel Runtime Tests

Kernel runtimes were measured for 4 different computational kernels. The first tested kernel (image-kernel 1) was the kernel from the previous single and dual buffer tests which utilized an image for input and a pair of images for storage of intermediate results. The second tested kernel (buffer-kernel 1) was an accumulation kernel which utilized a buffer for input and another buffer for storage of intermediate results. It was possible to utilize a single buffer because one may both read-from and write-to the same buffer within a kernel, as opposed to images, which must be either read-only or write-only within a kernel (in current
versions of OpenCL). The second pair of kernels were designed to eliminate differences
between image-kernel 1 and buffer-kernel 1. The third tested kernel (image-kernel 2) was
an image-based accumulation kernel where input image dimensions are passed as kernel
arguments rather than obtained by querying the input image. The purpose of this modifica-
tion was to allow image-based kernel behavior to more closely match buffer-based kernel
behavior (since there is no query support for buffers). The final tested kernel (buffer-kernel
2) was a buffer-based accumulation kernel that used two buffers to handle intermediate
results, in order to create a buffer-based kernel that generated the same number of memory
read and write operations as the image-based kernels. Each of the four tested computation
kernels was run 10,000 times. Performance measurements were taken using the profiling
tools available through AMD’s CodeXL program.

3.6 Results and Discussion
3.6.1 AMD Dual Buffer Results

Timing data was collected for the described algorithms when run with 4k image inputs;
for simplicity, this paper only reports timing data for the initial naive algorithms (1b initial,
2b initial) and the final optimized algorithms (1b final, 2b final). In addition, timing data
was only recorded for the optimized algorithms (1b final, 2b final) when run with 2k and
8k datasets. Raw timing data for the 4k data set can be seen in Fig 3.1 and Fig 3.5.

Testing of the first two algorithms (1b initial, 2b initial) showed that the two buffer
approach was significantly faster. However, when transfer rates were calculated (see Fig
3.2) it was clear that neither technique was near the theoretical transfer rate cap (8.0 GB/s
for a PCIE 2.0 bus). The outlook for the dual buffer technique was somewhat improved
by the discovery that available bandwidth to the device on the test system, as reported by AMD’s bandwidth test program (BufferBandwidth.exe), was actually only 5.07 GB/s. Even so, neither technique appeared to be effective.

For accurate calculation of efficiency, for each algorithm, with each input set, transfer speed tests were done for all buffer sizes between 64 kiB and 64 MiB in intervals of 64 kiB. The results of these test can be seen in Fig 3.7. For all image sizes, efficiency was calculated by dividing the time to transfer and processes data with a given algorithm and dataset, by the buffer-to-buffer transfer time for a matching amount data with an identically sized buffer.

Both one buffer and two buffer techniques showed significant improvement when optimized. The efficiency of the one buffer technique showed dramatic improvement, increasing by 68% to 91%, dependent on iterations (N). The efficiency of the two buffer technique also improved dramatically, increasing by 94% to 108%, dependent on iterations (N).

For both optimized algorithms (1b final, 2b final) the slower per image processing time for the $N = 10$ tests can be attributed to GPU warm up time (see Figure 3.6). The unoptimized algorithms (1b initial, 2b initial) do not exhibit this behavior, but their low efficiency overall makes it likely that the GPU never even reached full transfer speed while running. For all tests, running time appears to grow linearly with increasing iterations (N), excepting $N = 10$ where speed is slightly slower than a linear pattern would predict.

Timing data for ~2k and ~8k tests can be seen in Fig 3.1; transfer rate and efficiency numbers are shown in Fig 3.3. In the case of ~2k images, transfer rates are universally worse than for the ~4k images. Some of the change in data transfer rate is the result of
less efficient buffer copying when using smaller buffers. In terms of efficiency, the dual buffer algorithm (2b final) performs almost as well at ~2k resolution as at ~4k; however, surprisingly, the one buffer algorithm performed significantly worse at ~2k then at ~4k.

Tests performed with the ~8k data set present a very different picture. At first glance, the speed numbers for processing ~8k images appear to be identical for the optimized one buffer (1b final) and dual buffer algorithms (2b final). A closer examination reveals that 1b final is actually faster than 2b final with ~8k images, for all values of \(N\) except \(N = 10\) (see Figure 3.10). This result is in opposition to all previous tests with datasets of smaller images (see Figure 3.1), but the differences are significant based upon a \(t\)-test \((p \leq 2.2e^{-16})\). With ~8k image sets, the 1 buffer algorithm in fact demonstrated an efficiency ~5% higher than the 2 buffer algorithm, for all values of \(N\) except \(N = 10\) (see Figure 3.11).

One possible explanation for the degradation of performance with the dual buffer algorithm (2b final) with larger images is that it has higher resource demands, and as a result could strain GPU resources. Essentially, because the two buffer algorithm uses slightly more GPU resources than the single buffer algorithm, it would be the first to experience slow downs at large image sizes. Specifically, the dual buffer algorithm, compared to the one buffer algorithm, requires one additional permanent buffer-image pair. However, dependent on the device manufacturer and model, parts of GPU main memory may not be connected to texture/constant cache structures, making portions of GPU memory unusable for image storage; because available GPU image memory could be less than total GPU
memory, the dual buffer algorithm could run into resource contention issues before the one buffer algorithm.

### 3.6.2 AMD Buffer to Image Transformation Results

The result of the buffer to image transformation tests can be seen in Figure 3.12. The displayed images show the transfer rate recorded when transfer data into increasingly larger images. For example for the 128x128 sampling pattern’s image the bottom left pixel shows the results for a 128x128 image, the pixel to the immediately to the right shows the results for a 128x256 image, the pixel immediately above shows the results with a 256x128 image, and so forth. The image showing the results of the 100x100 block sampling works in the same way, the only exception being that image dimensions increase in steps sizes of 100 rather than 128. The images are color coded showing data transfer rate on a grey scale color map. The color white corresponds to transfer rates between 0 and 1 GB/s. The pixel color become dark with each increase in transfer rate in steps of 1 GB/s. The color black corresponds to the transfer rate range of 31 to 32 Gb/s. All transfer values above 32 GB/s are colored blue. Both images display clear bands where the transform rate is much higher than average. Additional bands with less drastic changes in transfer rates can be seen in the larger test image sizes.

Both images show a banding pattern with curves of alternating high and low performance. The difference between bands becomes less pronounced as overall image size increases. This pattern shows that image dimensions clearly affects the transformation time of images. If mapping had only depended on the amount of data being transferred it...
should have followed the pattern seen in the buffer transfer tests Fig. 3.7, and the resulting image should have been retaliative constant, with the exception of very small images. A major implication of this pattern is that use of images can be significantly accelerated by processing data in segments that match one of the higher performance dimensions for the buffer to image transform operation.

There was no noticeable effect on changing how image size information was passed to image based computation kernels.

### 3.6.3 AMD Buffer vs Image Kernel Runtime Results

The results of the comparison of kernel runtimes when using images and buffers can be seen in Figure 3.4. In these tests the fastest image kernel was 15.89% faster than the fastest buffer based kernel. There was no noticeable effect from changing how image dimensions were passed to the image based kernels. The buffer based kernel that used two buffers to handle the accumulation of results noticeably out performed the kernel that used only a single buffer with read and write access. This occurred despite the higher number of memory access operations executed by the two buffer kernel.

### 3.6.4 NVIDIA Results

There are two major platforms in high performance GPU computation, AMD and NVIDIA. In Sections 3.5 (p. 44) the performance of the DBA was tested on an AMD system. However, prior to July 2015, testing the performance of the DBA on NVIDIA hardware was impossible because there was no NVIDIA support for a version of OpenCL capable of running the DBA. However, current NVIDIA drivers support OpenCL 1.2, mak-
ing it possible to repeat key DBA performance tests on NVIDIA hardware. The NVIDIA test system used 7 Tesla C2705 Graphics Cards. The PCIE transfer rate from memory to GPU was measured at 6.2 GB/s. Each of the significant timing tests previously done on the AMD system with 4k image buffer transfers (see Sections 3.5.1 - 3.5.3, p. 45 - 46) were repeated on the NVIDIA system.

### 3.6.4.1 NVIDIA Dual Buffer Tests

Table 3.6 (p. 74) shows processing time and efficiency for ~4k images recorded on the NVIDIA system. Comparing the results of the timing tests on the AMD system (Table 3.2, p. 64) and the NVIDIA system (Table 3.6, p. 74), two observations are immediately apparent.

Firstly, the use of properly defined transfer buffers (the primary differentiating factor between the initial and final algorithms) has a notable effect on stream processing time. In other words, on both AMD and NVIDIA systems, use of properly defined transfer buffers will greatly reduce total processing time. Therefore, both the optimized algorithms (1b-final, and 2b-final) will always be faster than the non-optimized algorithms (1b-initial and 2b initial).

A second observation is that performance across the two systems, in terms of transfer speed, appears comparable, while performance, in terms of transfer efficiency, appears lower on the NVIDIA system. Essentially, both the AMD and the NVIDIA systems obtained approximately equal transfer rates. However, the NVIDIA system had higher avail-
able bandwidth (6.2 GB/s) compared to the AMD system (5.07 GB/s), which it did not fully utilize, meaning the efficiency of transfer ended up being lower.

Because the dual-buffer algorithm simultaneously processes and transfers data, it should always have a higher efficiency than a similarly optimized single-buffer algorithm, which must transfer and process data sequentially. This was the case for the results recorded on the AMD system (Figure 3.6 p. 66). In comparison, the efficiency of the both the optimized and initial dual-buffer algorithms (2b-final and 2b-initial) was worse than the efficiency of the comparable single-buffer algorithms (1b-final and 1b-initial) when tested on the NVIDIA system see Figure 3.14 (p. 77). The fact that the optimized dual-buffer algorithm performed with equal-to-less efficiency than the optimized single-buffer algorithm on the NVIDIA system suggests that IO and processing were being interleaved rather than being performed simultaneously on the NVIDIA system. This may be due to the test NVIDIA system utilizing seven GPUs, which may have stressed the PCIE Express bus that connected the GPUs with the main memory. The AMD test system, in contrast, utilized only one GPU. Additionally, the OpenCL 1.2 runtime provided for NVIDIA systems, may have been poorly optimized, and simply executed tasks in order, rather than simultaneously. Regarding the fact that the initial single-buffer algorithm performed with greater efficiency than the initial dual-buffer algorithm on the NVIDIA system 3.14 (p. 77), may indicate that, since the NVIDIA system was not performing tasks simultaneously, the dual-buffer algorithm was simply more complicated.
3.6.4.2 NVIDIA Buffer to Image Transformation Results

Following tests comparing the relative performance of the initial and optimized single-buffer and dual-buffer algorithms on the NVIDIA system, additional tests were performed to determine the impact of image dimension on the time necessary to transform data from buffers to images. On the AMD system, results of these tests revealed that images with certain dimensions could be transformed much faster than images with other dimensions, with ideal image dimensions revealing distinctive curves (Figure 3.12 p. 72). A similar but much weaker pattern of optimal image dimensions was observed on the NVIDIA system (Figure 3.15 p. 78). Essentially, image dimensions have a much smaller impact on buffer-to-image transformation time on the NVIDIA system.

3.6.4.3 NVIDIA Buffer vs Image Kernel Runtime Results

After examining the impact of image dimensions on transform time, additional tests were performed to compare relative performance with the optimized single-buffer and dual-buffer algorithms when utilizing either images or buffers as the primary format for data processing on the GPU. These tests revealed that images had superior processing time on the AMD system, even when used with a very simple kernel (figure 3.4, p. 73). However, when these tests were repeated on the NVIDIA system, performance (time for a computation kernel to complete) with images was found to be ~100% slower than performance with buffers (Table 3.7, p. 75). While this does not mean that images should never be used on NVIDIA hardware, it may be better to use images only when a kernel must make multiple reads to its data-sources in order to complete. These findings are in direct
opposition to the behavior shown in the AMD system, which indicated that, when possible, images should always be used on AMD GPUs.

The results of the comparison of kernel runtimes when using images and buffers on NVIDIA hardware can be seen in Figure 3.7. In these tests, the fastest image kernel was over 100% slower than the fastest buffer based kernel. There was no noticeable effect from changing how image dimensions were passed to the image based kernels. The buffer kernel that did accumulation in a single buffer had better performance. This is the intuitive result but was contrary to observations seen on AMD hardware.

Because of the differences between AMD Test and NVIDIA Test NVIDIA test were repeated on a second machine however as the results did not reveal a different pattern the second set of test results is not discussed in detail.

3.7 Conclusions

As datasets grow in size, it becomes increasingly important to efficiently process data using GPU memory, especially for applications requiring interactivity. To that end, to appropriately utilize hardware resources, it is necessary to understand the trade-offs inherent in choosing current buffer techniques versus the dual-image approach herein suggested. In this work, we have analyzed the performance of image buffers for scientific visualization kernels and performed a range of tests evaluating their performance standalone and vs. traditional buffer approaches. To conclude, we summarize these results and provide some guidelines for the use of image and data buffers in visualization. Future work is also discussed.
3.7.1 Dual Buffer Algorithm

Based on the observed transfer rates for the hardware tested, 480 MB of data can be processed per frame using the dual buffer algorithm, while maintaining a frame rate of 10 FPS when using 4k images. 2k and 8k images allow 410 MB and 360 MB per frame respectively. These values could be increased by applying compression to the input data and decompressing on the GPU as part of the transformation from buffer to image. Alternatively, increases in the raw transfer rate of the underlying hardware will also increase these values.

An important finding of this research are the scaling factors for the proposed algorithm. The algorithm scaled linearly with increasing iterations for all image sizes tested. For the ~2k and ~4k images, the optimized algorithm had better efficiency, whereas the optimized algorithm performed slightly worse with the ~8k image set. The maximum observed efficiencies with the dual buffer algorithm using the ~2k, ~4k, and ~8k image sets were 89%, 92%, and 66%. Image dimensions were found to have a strong impact on the run time of the algorithm when using AMD hardware and a weaker effect on NVIDIA hardware.

It is also important to note that the performance advantages demonstrated by the dual buffer algorithm do not depend on the specific computational kernel used in these tests. This is because the observed gains in performance are the result of enabling the system to simultaneously upload data, convert between buffer and image formats, and process images. Whatever processing kernel is utilized has independent run time and will therefore only affect time during that step of the algorithm. This means the dual buffer algorithm could easily be used with an arbitrary computation kernel, for various visualization or
computational tasks, while still improving overall efficiency. For example, the dual buffer algorithm could be used to process tiles from a large image where the processing step was the execution of some type of filter; this would not affect the time required to transfer tiles to the GPU, nor the time to convert the tile data from a buffer to an image. The efficiency of the dual buffer algorithm for an arbitrary processing kernel is

\[ e = \frac{1 + m_{1-n} + p_{1-n}, c_{1-n} + m_n + p_n}{c_{1-n}} \]

where \( c_i \) is the time need to copy the \( i^{th} \) buffer to the device, \( m_i \) is the time to transform the data of the \( i^{th} \) image from a buffer to image format, and \( p_i \) is the time required to process the \( i^{th} \) image.

One additional finding of this research is that a major cause of performance loss when using external devices is unnecessary synchronization with the CPU. When controlling data processing with the GPU, it is clear that performance can be greatly improved by avoiding synchronization whenever possible. In many cases, the necessary ordering of GPU operations can be achieved by explicitly specifying dependency relationships between events, using event references to control when given events can be executed.

### 3.7.2 Buffer to Image Transformation

On the AMD tested hardware, a pattern was observed of alternating bands of high and low performance in image dimension space. Knowledge of this pattern can be used to choose image dimensions where performance is high; this allows one of the overheads of using images—the time required to transform the data—to be reduced. The NVIDIA test system also displayed the same pattern but with much less variation between high low per-
formance. Further testing is necessary to see what, if any, pattern exists for other GPUs. In addition, further testing is necessary to see if the observed pattern is maintained when data is being written to part of an image. If this is the case, knowledge of the high performance dimensions could be used to accelerate data transformation into any size image.

3.7.3 Buffer vs Kernel Runtimes

The two different hardware architectures tested gave very different pictures in this test. On the AMD system using images significantly reduced the runtime of the tested kernel. Based on the simplicity of the tested kernel, it can safely be predicted that use of images will improve the performance of any non-compute-bound kernel running on AMD hardware. The NVIDIA hardware performed better when using buffers and significantly so. As using images requires extra overhead and yielded worse performance testing on individual kernels would be necessary to determine which kernels could be accelerated by the caching enabled by the use of images. For more precise claims, additional testing with different kernels will be necessary. This test, and the previous test on image transform speed, allow both the cost and the benefit, in terms of run time, for using images as the primary form of I/O to a kernel to be estimated. This allows the correct form of I/O, for optimal performance, to be selected without extensive testing.

The contributions of this chapter are the design and testing of a Dual Buffer transfer Algorithm (the DBA) for moving data between the CPU and GPU. On AMD systems the DBA was shown to approach the transfer speed limitations of the PCIE express bus. On NVIDIA systems, it was discovered that the DBA’s use of two buffers is not currently use-
ful, likely due to limitations in current NVIDIA OpenCL drivers. In addition, the relative benefits of using OpenCL images on AMD and NVIDIA hardware were recorded, and the conversion rate for the transformation from buffer-data to image-data was measured for a large sampling of image dimensions for both GPU types.

Our ongoing work will explore how kernel complexity influences the effect of images vs buffers in computational kernels. One goal of this testing would be to enable accurate prediction of the effects of changing I/O methods, so that, given a kernel and computational hardware to execute it, an optimal configuration could be identified without need for extensive testing.
The flood analytics program highlights region of multiple flood overlap using a blue saturation map (high overlap corresponds to high saturation); these extents (and others) are calculated from the $N$ input surfaces into a single composite image using the discussed technique. The list in the right panel allowed users to add or remove surfaces from consideration in real time. A composite image of the currently selected images is shown in the upper left panel. The lower left panel displays a histogram showing the frequency of overlap classes, indicating distribution of flood occurrences.
1. Clear accumulation Buffers

2. set the values of $\textbf{pos1}$, $\textbf{pos2}$, and $i$ to 0, 1, and 0

3. Load data into $\textbf{input}_i$ from client memory

4. Run accumulation kernel where

   • $\textbf{input}_i$ is an input
   • $\textbf{accum}_{[\textbf{pos1}]}$ is an input
   • $\textbf{accum}_{[\textbf{pos2}]}$ is an output

5. swap the values $\textbf{pos1}$ and $\textbf{pos2}$

6. increment the value of $i$

7. if $i \geq N$ stop, otherwise go to 3

Where $i$, $\textbf{pos1}$, and $\textbf{pos2}$ are integers

$\textbf{input}_i$, $\textbf{accum}_{[0]}$, and $\textbf{accum}_{[1]}$ are OpenCL images

$N$ is the number of images to processes

Figure 3.2: Naive Approach – 1 Buffer
1. Clear accumulation Buffers

2. Set the values of \texttt{pos1}, \texttt{pos2}, and \texttt{i} to 0, 1, and 0

3. Load data into \texttt{input\_i[pos1]} from client memory

4. Run accumulation kernel where
   
   \begin{itemize}
   \item \texttt{input\_i[pos1]} is an input
   \item \texttt{accum\_[pos1]} is an input
   \item \texttt{accum\_[pos2]} is an output
   \end{itemize}

5. Swap the values \texttt{pos1} and \texttt{pos2}

6. Increment the value of \texttt{i}

7. If \texttt{i} ≥ \texttt{N}, stop; otherwise, go to 3

Figure 3.3: Dual Buffer Approach

Figure 3.4: Dual Buffer Algorithm Flow

Order of operations for the final dual buffer algorithm (2b final) with two input buffer/image pairs. Arrows indicate dependency: an event will not start until all events connecting to it have completed.
Table 3.1: AMD Algorithm Run Times

<table>
<thead>
<tr>
<th>Data Set</th>
<th>N</th>
<th>1b initial</th>
<th>2b initial</th>
<th>1b final</th>
<th>2b final</th>
</tr>
</thead>
<tbody>
<tr>
<td>~2k</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>16,901.01</td>
<td>10,060.57</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>154,798.84</td>
<td>97,795.6</td>
</tr>
<tr>
<td></td>
<td>1,000</td>
<td>-</td>
<td>-</td>
<td>1,544,108.34</td>
<td>967,545.27</td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>-</td>
<td>-</td>
<td>15,253,992.48</td>
<td>9,641,951.45</td>
</tr>
<tr>
<td>~4k</td>
<td>10</td>
<td>79,224.52</td>
<td>69,423.91</td>
<td>47,112.72</td>
<td>35,602.03</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>779,224.59</td>
<td>702,200.18</td>
<td>414,403.68</td>
<td>338,179.28</td>
</tr>
<tr>
<td></td>
<td>1,000</td>
<td>7,913,592.64</td>
<td>6,936,346.74</td>
<td>4,160,507.91</td>
<td>3,354,251.81</td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>78,868,351.01</td>
<td>70,314,931.82</td>
<td>41,207,446.97</td>
<td>33,782,002.28</td>
</tr>
<tr>
<td>~8k</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>185,410.59</td>
<td>179,550.34</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>1,649,694.40</td>
<td>1,778,611.77</td>
</tr>
<tr>
<td></td>
<td>1,000</td>
<td>-</td>
<td>-</td>
<td>16,571,037.83</td>
<td>17,769,066.44</td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>-</td>
<td>-</td>
<td>165,437,132.54</td>
<td>178,148,199.43</td>
</tr>
</tbody>
</table>

This table shows the processing time for all recorded tests. The initial naive algorithms (1b initial, 2b initial) were only tested with the ~4k image set, whereas the optimized algorithms (1b final, 2b final) were also tested with the ~2k and ~8k datasets. For each combination of dataset and algorithm, time values increase in a linear pattern, with the value for N=10 being slightly higher than the trend indicated by the other points. Additionally, note that for all tests the dual buffer algorithm performed faster, except when using ~8k images. Because the dual buffer algorithm uses slightly more resources, this degradation of performance may be caused by resource contention.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data Set</th>
<th>Iterations (N)</th>
<th>Transfer Rate (GB/s)</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1b initial</td>
<td>~4k</td>
<td>10</td>
<td>2.069</td>
<td>39.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>2.104</td>
<td>39.81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>2.071</td>
<td>39.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>2.078</td>
<td>39.33</td>
</tr>
<tr>
<td>2b initial</td>
<td>~4k</td>
<td>10</td>
<td>2.361</td>
<td>44.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>2.334</td>
<td>44.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>2.363</td>
<td>44.72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>2.331</td>
<td>44.11</td>
</tr>
<tr>
<td>1b final</td>
<td>~4k</td>
<td>10</td>
<td>3.479</td>
<td>65.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>3.956</td>
<td>74.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>3.940</td>
<td>74.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>3.978</td>
<td>75.27</td>
</tr>
<tr>
<td>2b final</td>
<td>~4k</td>
<td>10</td>
<td>4.604</td>
<td>87.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>4.847</td>
<td>91.72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>4.887</td>
<td>92.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>4.852</td>
<td>91.82</td>
</tr>
</tbody>
</table>

This table shows the transfer rates and efficiency that can be derived from the timing results shown in Table 3.1 (p. 63) when using ~4k images. Efficiency was calculated by comparing the time to transfer and process data, with the indicated algorithms, to the time required for a simple buffer-to-buffer transfer of the same amount of data.
Figure 3.5: AMD Image Processing Times for 4k Images

Although the lines in this figure appear parallel, they are not. Lines with different slopes appear parallel in logarithmic space because space itself is distorted. The difference between the lines at bottom of the graph is orders or magnitude less than the difference between the lines at the top.
Figure 3.6: AMD Efficiency for 4k Images

The small loss of efficiency when $N = 10$ is likely due to warm up time for the GPU.

Figure 3.7: AMD Buffer Transfer Speed

This figure shows transfer rate of a copy from CPU resident buffer to a GPU resident buffer as function of buffer size. Buffer transfer rate is very poor with small buffer sizes, but improves rapidly and settles on a rough average of 5.1 GB/s. The values from this test were used as the baseline values when calculating the efficiency in other tests.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data Set</th>
<th>Iterations (GB/s)</th>
<th>Transfer Rate</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1b final</td>
<td>~2k</td>
<td>10</td>
<td>2.425</td>
<td>51.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>2.647</td>
<td>55.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>2.654</td>
<td>55.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>2.687</td>
<td>56.56</td>
</tr>
<tr>
<td>2b final</td>
<td>~2k</td>
<td>10</td>
<td>4.073</td>
<td>85.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>4.190</td>
<td>88.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>4.236</td>
<td>89.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>4.250</td>
<td>89.56</td>
</tr>
<tr>
<td>1b final</td>
<td>~8k</td>
<td>10</td>
<td>3.536</td>
<td>63.29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>3.975</td>
<td>71.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>3.957</td>
<td>70.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>3.963</td>
<td>70.93</td>
</tr>
<tr>
<td>2b final</td>
<td>~8k</td>
<td>10</td>
<td>3.652</td>
<td>65.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>3.687</td>
<td>65.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>3.690</td>
<td>66.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>3.681</td>
<td>65.87</td>
</tr>
</tbody>
</table>

This table shows the transfer rates and efficiency that can be derived from the timing results shown in Table 3.1 (p. 63) when using ~2k and ~8k images. Efficiency was calculated by comparing the time to transfer and process data, with the indicated algorithms, to the time required for a simple buffer-to-buffer transfer of the same amount of data. The ~2k image sets has noticeably lower efficiency than the ~4k which was seen in Figure 3.2. In the ~8k the one buffer algorithm actually out performs the dual buffer algorithm.
Figure 3.8: AMD Image Processing Times for ~2k Images

Parallel lines seen logarithmic space are not actually parallel; in actuality, the timing curves depicted diverge at a near constant rate.
Figure 3.9: AMD Efficiency for ~2k Images

Efficiency for ~2k images is noticeably lower than for ~4k images. The slight decrease in efficiency at N=10 is most likely due to warm up time for the GPU.
Figure 3.10: AMD Image Processing Times for ~8k Image

This graph shows processing times for both the dual buffer (2b final) and single buffer algorithm (1b final) when processing ~8k images. Although the speed curves appear very close, the dual buffer algorithm is, in fact, slower. Because of the surprising nature of this result, a t-Test was performed, revealing that the difference is statistically significant ($p < 2.2e^{-16}$).
Figure 3.11: AMD Efficiency for ~8k Images

This graph shows efficiency for the optimized dual buffer (2b final) and single buffer algorithm (1b final) when processing ~8k images.
Figure 3.12: AMD Buffer to Image Transform Transfer Rates

The images above show the transfer rates for all tested image sizes with each pixel representing results for a specific image size. In the left-hand image, the bottom left pixel indicates the transfer rate achieved with 128x128 images. Each step right or up indicates an increase by 128 pixels in the X or Y dimensions, respectively; with the upper right pixel representing results for 16,384x16,384 images. The right-hand image follows the same pattern, with the bottom left pixel indicating the results for 100x100 images, and with an X and Y dimension step size of 100 pixels; and with the upper right pixel indicating results for 16,300x16,300 images. The majority of results are displayed with a grey-scale color ramp of 32 steps (with a step size of 1 GB/s) showing transfer rates ranging from white (0-1 GB/s) to black (31-32 GB/s). High outlying results (above 32 GB/s) are displayed in blue. Implicit curves of alternating high and low performance are easily noted.
As described in Secton 3.5.3 (p. 46), four separate kernels were tested for runtime performance. Image-kernel 1 (the optimized dual-buffer algorithm described in Section 3.4, p. 42) was designed to query the image structure to determine image dimensions. Buffer-kernel 1 was created to compare image access time to buffer access time (to test the efficacy of images as the primary data processing container on the GPU). Buffer-kernel 1 accumulated data using a single buffer by using both read and write access (which is not possible with image kernels). After testing, it was noted that the two initial kernels were not symmetric in terms of frequency of memory access. Therefore, image-kernel 2 and buffer-kernel 2 were designed such that each kernel had symmetric memory access patterns. In both cases, image-kernels outperformed buffer-kernels on the AMD test system. Unexpectedly, buffer-kernel 2 performed better than buffer-kernel 1, even though it required more memory operations. This implies that either, (a) an optimization occurs when buffers are used exclusively for reading/writing, or (b) performance degradation occurs when a buffer may be used for reading and writing within a single kernel.
Table 3.6: NVIDIA Transfer Rate and Efficiency with ~4k Image Dataset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data Set</th>
<th>Iterations (N)</th>
<th>Transfer Rate (GB/s)</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1b initial</td>
<td>~4k</td>
<td>10</td>
<td>2.539</td>
<td>40.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>2.502</td>
<td>40.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>2.156</td>
<td>34.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>2.374</td>
<td>40.64</td>
</tr>
<tr>
<td>2b initial</td>
<td>~4k</td>
<td>10</td>
<td>2.520</td>
<td>38.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>2.380</td>
<td>30.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>1.894</td>
<td>31.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>1.981</td>
<td>38.10</td>
</tr>
<tr>
<td>1b final</td>
<td>~4k</td>
<td>10</td>
<td>4.657</td>
<td>75.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>4.238</td>
<td>68.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>4.201</td>
<td>67.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>4.194</td>
<td>67.64</td>
</tr>
<tr>
<td>2b final</td>
<td>~4k</td>
<td>10</td>
<td>4.120</td>
<td>67.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>4.120</td>
<td>67.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,000</td>
<td>4.202</td>
<td>67.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,000</td>
<td>4.200</td>
<td>67.73</td>
</tr>
</tbody>
</table>

Efficiency was calculated by comparing the time to transfer and process data, with the indicated algorithms, to the time required for a simple buffer-to-buffer transfer of the same amount of data. The NVIDIA system obtains maximum transfer times similar to the original AMD system, however efficiency is lower because the NVIDIA system has higher available throughput.
To test the efficacy of images as the primary data processing container on the GPU, four separate kernels were tested for runtime performance (see Section 3.5.3, p. 46). After testing on an AMD system (Table 3.4 p. 73), tests were repeated on an NVIDIA system. Compared to the AMD system, NVIDIA system tests showed an entirely different pattern of behavior for all four kernels: (a) buffer-kernels outperformed image-kernels; (b) buffer-kernel 1 outperformed buffer-kernel; and (c) image-kernel 2 outperformed image-kernel 1.

These results demonstrate that, on an NVIDIA system, barring a compute-bound kernel, it is better to avoid using images. Finally, image-kernel 2 outperforming image-kernel 1 implies that, on the NVIDIA system, it was faster to read image dimensions as kernel arguments than to read image dimensions directly from the data structure.

Table 3.7: NVIDIA Buffer vs Image Kernel Runtimes

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Average Time (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>image-kernel 1</td>
<td>2085</td>
<td>20,845,189</td>
</tr>
<tr>
<td>buffer-kernel 1</td>
<td>839</td>
<td>8,386,488</td>
</tr>
<tr>
<td>image-kernel 2</td>
<td>1858</td>
<td>18,580,059</td>
</tr>
<tr>
<td>buffer-kernel 2</td>
<td>1064</td>
<td>10,637,612</td>
</tr>
</tbody>
</table>
Figure 3.13: NVIDIA Image Processing Times for 4k Images

The reported times for both optimized and unoptimized variants of the 1 and 2 buffer algorithms are almost identical. The processing time was lower for the first tested buffer kernel which shows there is not a penalty for using a single buffer for reading and writing data, as was the case in the AMD tests. The second image kernel had the faster processing time which indicates it is faster to pass information about images as kernel parameters than to read attributes of the images inside the kernel.
Figure 3.14: NVIDIA Efficiency for 4k Images

Efficiency is identical for the optimized algorithms except at 10 iterations which is the most easily influenced test by noise. The lower efficiency of two buffer algorithm can be attributed to its greater complexity, if data transfer and processing where occurring concurrently (as is intended) this increase in complexity would overshadowed by the gains from concurrent data transfer and processing.
Figure 3.15: NVIDIA Buffer to Image Transform Transfer Rates

The images above show the transfer rates for all tested image sizes with each pixel representing results for a specific image size. In the left-hand image, the bottom left pixel indicates the transfer rate achieved with 128x128 images. Each step right or up indicates an increase by 128 pixels in the X or Y dimensions, respectively; with the upper right pixel representing results for 16,384x16,384 images. The right-hand image follows the same pattern, with the bottom left pixel indicating the results for 100x100 images, and with an X and Y dimension step size of 100 pixels; and with the upper right pixel indicating results for 16,300x16,300 images. The majority of results are displayed with a grey-scale color ramp of 32 steps (with a step size of 1 GB/s) showing transfer rates ranging from white (0-1 GB/s) to black (31-32 GB/s). High outlying results (above 32 GB/s) are displayed in blue. There is the suggestion of the implicit curves seen on the NVIDIA test but the effect is greatly diminished.
CHAPTER 4

VISUALLY GUIDED COMPARISON

4.1 Introduction

Flooding is one of the most frequently occurring natural disasters, causing many deaths worldwide and billions of dollars in damage each year [17]. GIS flood models, which are designed to predict the extents and severity of flooding, can be used to reduce damage and prevent deaths, both by influencing the implementation of flood control projects and structures, and by providing critical information for emergency response forces during flood events.

As tools to mitigate the effects of floods, flood models can be used to predict how flood control projects will affect future floods, either by drawing on historical flood data, or by simulating potential future scenarios. To determine optimal mitigation approaches, multiple past or potential scenarios must be considered simultaneously. Unfortunately, as more scenarios are considered at once, the number of modeled outputs to describe permutations of potential actions increases exponentially. Achieving a useful analysis of such large masses of complex output can be extremely difficult.

Current approaches to analyzing such datasets are costly in time and money. One factor contributing to this cost is that GIS tools display surfaces via ordered painting, in which
one surface is drawn, then the next, until all surfaces have been displayed. In this manner later surfaces obscure surfaces earlier in the rendering sequence.

In order to view obscured information, users must either reorder surfaces and wait for the render process to complete, or use a GIS tool with transparency options. Though users can reorder surfaces to determine information about obscured surfaces, doing so can obscure information that was previously visible. As the number of surfaces being considered increases, it becomes difficult to recall details about previously viewed orderings. Transparency is also not a complete solution because results depend on the order of rendering. Additionally, blending too many surfaces prevents users from determining which surfaces are interacting at any given point [7].

When using either reordering or transparency, an additional problem is that the size of any given output can easily reach multiple gigabytes, and rendering is not instantaneous, since existing GIS tools do not utilize hardware-accelerated APIs. This is an issue when trying to analyze multiple data sets describing different historical or statistical events.

Part I of this work (see Chapter 2) introduces a new GIS tool, Flood Event Simulation Model (FESM), which utilizes direct gauge water elevation measurements to rapidly predict flood extents. Part II of this work (see Chapter 3) introduces an algorithm, the Dual Buffer Algorithm (DBA), which was designed to facilitate rapid transfer of data between the CPU and the GPU. This section of the proposal introduces a new scaleable visual analytics GIS tool, Dynamic Raster Overlay (DRO), which utilizes the DBA to allow hardware accelerated viewing and analysis of multiple expansive grid-based data sets (i.e., multiple
flood surfaces) simultaneously. These flood surfaces can be supplied by FESM, or by other flood models, or can come from other sources, such as satellite imagery.

DRO is responsive enough for interactive usage, letting users display multiple rasters, combine them, and dynamically change participating rasters while seeing results in real-time. DRO allows users to easily identify when multiple displayed surfaces are interacting at particular spatial locations. This is accomplished using an overview + details on demand methodology. Users can customize output to highlight specific degrees of overlap, thereby identifying regions in which the specified level of overlap exists, and determining the specific datasets overlapping to form each region.

Prior to DRO, this type of analysis, which has often been undertaken by the USACE, was both slow and difficult to achieve, as no tools or optimized methodology existed specifically designed to accomplish these tasks. The USACE was required to utilize software which required significant time and multiple user steps in order to visualize each proposed query. DRO is easy to setup and accomplishes the same tasks dynamically, in real-time, thus allowing interactive usage and improving efficiency by many orders of magnitude. DRO will allow the USACE, or other organizations, to simultaneously analyze multiple flood scenes, from historical data or simulations, in order to determine areas of overlap variability and thus the real-world significance of actual or potential changes, such as the occurrence of disasters or the implementation of water control structures.
Core DRO Contributions:

- Visual output is stable and accurate, regardless of input surface ordering.
- DRO clearly indicates areas where multiple surfaces interact.
- Specific surfaces that compose any region of interaction are quickly and easily identified.
- Allows for visual analysis of surface interactions
- Required data prepossessing is minimal.
- Using GPU-accelerated OpenCL kernels, DRO easily handles massive data-sets and/or numerous simultaneous inputs.
- DRO output is easily imported into existing geo-spatial GIS analytic tools.

Essentially, DRO possesses numerous features not previously available in GIS analytics tools, and overcomes or mitigates many of the problems faced when analyzing multiple flood scenes with other available tools. DRO’s features combine to allow a visual analytics approach that is better integrated into experts’ workflows, allowing more iterations of exploration. While DRO is currently limited by graphics memory in regards to maximum size surfaces (rasters / data sets) that can be analyzed, this limitation can easily be overcome by allowing input segmentation.

The remainder of this chapter overviews DRO’s implementation, case studies, and evaluation by GIS professionals.

4.2 Related Work

Multivariate visualization is an active field of research in which many potential solutions have been proposed. In Taylor [65], several methods for visualizing multiple fields on a single surface are summarized including texture based techniques, a spot based blending
technique (Data Driven Spots or DDS) [13] and a line segment representation technique
(Oriented Slivers) [69]. An example of the oriented slivers technique can be seen in Fig
4.1. The DDS technique works by combining spots of distinct color and varying alpha
values, with the alpha values based on the scaled value of the field being visualized. The
color and alpha value resulting from the spot and data combination is then blended onto
a background that is originally neutral gray. The spots used for each field would have
differing sizes and animation could be used to move the spots across the surface to better
sample the data. The Oriented Slivers used slivers with different orientations to represent
each variable. The size of sliver could be used to indicate the intensity at that position on
the surface.

In Forsell et al. [26], a technique that used three dimensional glyphs as well as color
to visualize multiple variables on a single surface was tested. Soon after, Cai et al. [15]
presented a technique that used two layers to display information from multiple remotely
sensed images. The first layer is an overview resulting from conversion of the multiple
greyscale inputs into color, the second layer is composed of pie chart glyphs, where the
filling of the pie chart is based on the relative strength of the input for each layer. The
two views are then blended together with the relative strength of each layer depending on
the size of the currently displayed area. Hsiao’s dissertation [34] reviews several ways of
displaying multiple fields of data on a single surface, including DDS and glyph based tech-
niques, with examples given for both techniques. Cai et al. [16] demonstrated the use of
dual layer visualization, DDS, and Oriented Slivers for displaying remotely sensed hyper-
spectral imagery. In addition, visualizations were made that combined all three techniques.
Figure 4.1: An example of multivariate visualization using Oriented Slivers

Image from Wiegle et al., 2000) [69]. Images (a)-(h) show individual data layers, with each layer depicting concentrations of a different mineral within a surveyed area. For each displayed layer hash-marks are generated with a unique orientation. Additionally, the opacity of each hash-mark indicates the strength of the represented data at the sampled location. Image (i) is a composite showing layers (a)-(h) overlayed. Image (j) is the same composite with layers (f) and (h) assigned to be oriented at 90° and 180°; in this manner areas of overlap between layers (f) and (h) are discernible as “plus signs” (i.e., “+” symbols). The Oriented Slivers technique allows for the simultaneous visual analysis of multiple surfaces. However, the resulting cross-hatch composite images become increasingly more difficult to interpret as the number of surfaces increase. Additionally, detail at data boundaries can be obfuscated by the nature of the hash-mark layer textures. Finally, the use of sampling lowers the resolution of all displayed data.

Another area of relevant research is ensemble visualization, which deals with visualization of multiple model outputs where the input parameters for each model run are adjusted slightly. This area of research is important because some of targeted data groups in this
paper utilize multiple flood scenes showing the effects of different flood control features, and such data could be considered an ensemble data-set. Phadke et al. [54] showed how a sphere-based technique, a 3D extension of DDS, and a glyph based technique similar to the one shown in Hsiao [34] could be adapted to ensemble visualization. Specifically, ensemble visualizations were made by allowing agreement to modify, the size, the opacity, or color of existing elements.

Another approach to ensemble visualization ESS (Ensemble Surface Slicing) is shown in Alabi et al. [6], where ensemble members are compared by slicing the modeled space of the simulations. The first slice is rendered with results from the first model, the second from the second model and so forth. This causes clear visual discontinuity where ensemble outputs differ.

House et al. [7] introduced a technique for viewing simultaneous three dimensional surfaces using textures to allow differentiation of the viewed surfaces. Guidelines for the generation of textures for this purpose were given in [33]. In both works the scope was limited to two simultaneous surfaces, an upper surface and lower surface.

The primary difference between our work and the previous is that previous attempts focused on the simultaneous visualization of the multiple data surfaces and the values of the depicted data on the surfaces. The data inputs into our system are boolean in nature rather than scalar, therefore it is not necessary to display their value — only their presence. This allows more effort to be placed on analysis of interactions, of surfaces, which is not directly addressed in previous work. The number of possible interactions is restricted to a manageable number by limiting analysis to a single level of overlap at a time.
4.3 Technique Overview

The proposed visualization system uses the overview + details-on-demand ideology. The first level of visualization, the overview, is created by summing the number of surfaces that define a given pixel for all locations in all surfaces. This is then visualized using a color ramp where color intensity increases as the number of participating surfaces for a pixel increases. Due to the context of flood visualization, a white to blue ramp was chosen for this phase. In addition, a histogram showing the counts for each level of overlap is shown below the color coded image, and a list of participating surfaces is shown to the side of the image. The list allows particular surfaces to be added or removed from the visualization; reordering is not supported because the final image is not dependent on the order that surfaces are processed. The histogram allows easy access to statistical information about the degrees of overlap currently being visualized. In addition it can be used to select one level of overlap for further details. The overview display can be seen in Figure 4.8.

When a level of overlap is selected, the visualization system changes to its’ focus mode. In this mode, the color coding of all non-selected levels is desaturated, and the color coding for the selected level of overlap is replaced using a categorical color map. Each unique group of surfaces that is present in the selected level is redrawn with a unique color. To aid in identification, the list of surfaces is marked with squared indicators color matched to the categorical color for each surface. These markers can be used to determine which surfaces are participating in the creation of the selected sub-region. Additionally, the histogram bar for the selected level of overlap is redrawn with subregions sized according to percent contribution of each participating group of surfaces. Moving the mouse over any sub-
group on the selected histogram bar will display the percent contribution of that group and highlight the surfaces that create that sub-group in the surface list. Figure 4.9 demonstrates the focus mode, showing all the regions where three surface overlap in the data found in Figure 4.8.

Both the overview and focus modes allow the view point to be panned and or zoomed, to better visualize interactions at any particular area. The surfaces for both modes can be exported as GeoTIFF files for analysis in other tools.

### 4.4 Implementation

One of the major challenges encountered when processing multiple flood inundation maps, is the size of each input is potentially large. This makes direct processing on a CPU a poor choice when the system designed to be used in an interactive manner. Processing with multiple CPUs is an option, particularly with the current abundance of multi-core processors. However, barring the use of supercomputing clusters, which are not universally available, the most potent processing solution available is GPU programing. Modern GPUs have hundreds to thousands of stream processors or CUDA cores and allow multiprocessing to a much higher degree than using multiple CPU cores and/or vector instruction sets. To allow the system to work on the widest possible range of devices, OpenCL was chosen as the interface used to communicate with the GPU.

There are several challenges to efficiently processing vast datasets on a GPU.

1. GPU memory is much more limited than CPU memory.
2. Data transfer to and from the GPU is limited by the bandwidth of the PCIE bus.
3. The linear format of CPU image data does not interact ideally with a GPU processing data in rectangular block.
To address these problems, the OpenCL kernels were designed to be iterative and spatially partitionable. The iterative nature of the kernels means that only data about one surface, and the intermediary buffers that hold the processing state, need to be resident at a time on the GPU. Multiple surfaces are processed by streaming surface data from main memory to the GPU. The kernels being spatially partitionable means that the results at any location depend only on the values defined at that location for all processed surfaces. As a result, when a single input surface, the necessary support buffers, and images can not be loaded into GPU memory due to the size of the input surface, processing can still ensue by segmenting the input into parts that are then processed sequentially.

4.4.1 Algorithms

This visualization system uses two different algorithms for processing on input imagery. The first algorithm produces the overview images, while the second produces the focus images for selected levels of detail. The first algorithm is executed whenever the currently active surfaces change. This can occur when a surface is added, deselected, or selected. The steps in the overview algorithm consist of:

1. Clear the accumulation buffers.
2. Combine the previous accumulation buffer with the current data image to generate a new accumulation buffer. Repeat this step for each surface that is active for the visualization.
3. Create a color image by applying a color map to the final accumulation buffer.
4. Count values of the final accumulation buffer to retrieve the statistical data to display the overview histogram.
Two accumulation buffers are necessary because a single image can not be both read and written by an OpenCL kernel. The second algorithm, used when a level of overlap is chosen, consists of:

1. Clear the accumulation buffers, except the buffer with the current results from the previous algorithm. If the number of input surfaces is odd the active buffer will be the 2nd buffer, likewise if the number of inputs is odd the active buffer will be the first buffer.

2. For each input surface, write identity flag values into the accumulation buffer. The flag values are increasing powers of 2, where the first surface has a flag value of 1, the second a value of 2 and so forth. These values are masked using a combination of the selected level of overlap and the final accumulation surface from the first algorithm. The results is that identity values are only accumulated at locations with the correct overlap value.

3. Count the values in the final accumulation buffer

4. Analyze the non-zero values from the previous counting operation and determine which surface groups the correspond too.

5. A color image is generated by combining the image generated in algorithm 1 and desaturating it at all locations except the locations that pass the previously described masking test. For locations that pass the masking test, categorical colors are displayed depending the identity value recorded at that location.

This algorithms where implemented with multiprocessing OpenCL kernels, which are described in the next section.

4.4.2 Kernels

Processing data with OpenCL requires the creations of kernels (GPU Programs). To accomplish the steps described in the above algorithms, six different kernels where necessary. The first kernel used at the beginning of both algorithms is the initialization kernel. It is responsible for clearing data in the accumulation buffers. Following initialization, multiple calls are made to either the surface accumulation kernel or the category accumulation
kernel, depending on which algorithm is being executed. Both accumulation kernels combine the input surface with a previous state held in one of the accumulation buffers, they differ in what values they are accumulating. The surface accumulation kernel increases location counts by one, while the category accumulation kernel increases counts by a user specified value. The category accumulation kernel also uses the final output accumulation buffer data from algorithm one as a mask. There are two display kernels that are responsible for creating a color image by processing one of the accumulation buffers. They are responsible for creation of the overview and focus images. The final kernel is a counting kernel that determines the frequency of occurrence of all values of the processed. It is used for the creation of the displayed histograms as well as to allow a selected region of overlap to be broken down into its component sub surfaces. The six kernels are further described below:

1. **The initialization kernel.** This kernel zeros the contents of an integer image buffer. It is used at the beginning of computation to clear results from previous computations.

2. **The surface accumulation kernel.** This kernel computes number surfaces that had a defined value at each location in the input surfaces. It works by combining a previous state and the input data. The output is an image that contains the number of overlaps present at each location. This output will be processed by the both the counting kernel and the phase one display kernel.

3. **The category accumulation kernel.** This kernel computes the sums of the category labels. Like the surface accumulation kernel this kernel combines a previously stored state with its input data. In addition this kernel uses the results of the surface accumulation kernel and a user supplied value as a mask. For example if the user supplied value is two values will only be accumulated at positions where the supplied accumulation image contains a value of two. The output is an image containing the summed category labels at each location that passed the masking operation zero at other locations. Category label are powers of two which allows a summed value to be analyzed to determine the participating surfaces that created it. The output of this kernel is processed by the phase 2 display kernel and the counting kernel.
4. **The counting kernel.** Returns the number of times each value was present in the input surfaces. The output is a buffer which holds the number of times the index value of a location was seen. For example position zero holds the count of zeros, and position one the count ones and so forth. The statistical information retrieved by this kernel is necessary for the rendering of all histograms in the visualization system and is used to determine what surfaces discovered surface id’s encode.

5. **The phase one display kernel.** This kernel converts the accumulated surface counts from the surfaces accumulation kernel into a RGBA image for display. The output is rendered by OpenGL.

6. **The phase two display kernel.** This kernel converts the results of the surface accumulation kernel and the category accumulation kernel to create a RGBA image where pixels that contained a user selected number of overlapping surfaces have been recolored with categorical colors. The output of this kernel is also passed to OpenGL for display.

All calls to OpenCL are handled through an event controlled command queue. This allows data to be transferred into one transfer buffer and that buffer to be copied into a data image in parallel with either the surface accumulation kernel or the category accumulation kernel processing the contents of the other data image. The specific ordering of data transfer and processing can be seen in Figure 4.10.

### 4.4.3 Resources

The OpenCL resources required for this system consist of

- Two data transfer buffers located in pinned memory.
- Two data images in GPU memory.
- Three accumulation images in GPU memory of type unsigned int 16.
- Two display images which are acquired from OpenGL as shared resources
- One counting buffer.

The data transfer buffers are sized according to input data dimensions. The size of counting buffer is determined by the largest possible value which could be encountered;
this value will be less than $2^{16} - 1$. For the data used in our first test case, which has dimensions of 3712*4316, the required GPU memory is approximately 625 MB. This value does not change as the number of surfaces used in the visualization increases.

4.4.4 Scalability & Technical Limits

The use of OpenCL places certain limits on processing.

- The largest image that can be processed as a single unit is determined by the maximum image size of the OpenCL implementation.

- The use of 16-bit integers limits the maximum number of surfaces processed in the overview phase to $2^{16} - 1$ and by the focus phase to 15. These numbers could be increased by using 32-bit accumulation surfaces, but doing so would double the memory required to hold the accumulation buffers and increase the potential size of the counting buffer exponentially.

One of the goals of this visualization system was scalability, to allow efficient processing of LIDAR datasets. However, sufficiently large datasets were not available thoroughly to test the system's scalability. To address this, the first algorithm was synthetically tested by passing repeating surfaces as inputs. This allowed the first algorithm to be tested with a very high number of surfaces and consequently processing a large amount of data. This is similar to how multiple LIDAR datasets would stress the system. The resulting times showed that the processing speed grows linearly with the number of surfaces processed (Figure 4.1).

4.5 Case Studies

To demonstrate the utility of our approach, we present three case studies from different scenarios. They demonstrate the efficacy of the work on model and remotely sensed data in a variety of use cases.
Table 4.1: Processing time with synthetic data

<table>
<thead>
<tr>
<th>N</th>
<th>Processing Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>35,602.03</td>
</tr>
<tr>
<td>100</td>
<td>338,179.28</td>
</tr>
<tr>
<td>1,000</td>
<td>3,354,251.81</td>
</tr>
<tr>
<td>10,000</td>
<td>33,782,002.28</td>
</tr>
</tbody>
</table>

The overview algorithm was tested for scalability by processing surfaces repeatedly to allow timing information to be collected for very large values of N. The resulting number show that performance is linear as the number of surfaces increases. Timing data is accurate to ± 0.01 s

4.5.1 Yazoo Backwater

The Yazoo Backwater an approximately 1,550 square mile region between the eastern main line levee of the Mississippi River and the western Will Whittington Canal levee, bounded on the north US highway 82. This area has been the focus of many hydrological studies performed by the USACE (Army Corps of Engineers). The inputs for this study came from three groups: historic median annual 14-day durations floods from different time periods, a series of frequency flood simulations showing events with decreasing frequency of occurrence, and simulation data recreating the effects of the 2008 Mississippi River Flood on the Yazoo Backwater Area. The individual flood maps can be seen in Figure 4.11; all the data considered in this test came from hydrologic models. A quick examination of the data reveals that the 50 year flood is noticeably larger than all other events and can be excluded from further consideration. After removing the 50 year flood, selecting the first few levels of overlap shows that all flooding in the northern area of region occurs only in frequency events; this is revealed by only surface interaction made up of frequency...
flood events covering the entire area. This means that none of the indicated areas are ever flooded for a period of two weeks, in the events considered (Figure 4.12).

The four overlap region also shows two additional areas with significant contributions. The pink ring around the central region reveals that the 5 year frequency flood has matching extents with the 1901 to 1931 14-day duration events in the central part of the basin, see Figure 4.13. The orange area on the left hand side shows a local correspondence between the 2008 flood and the 5-year frequency flood, showing the localize severity of that event.

The selection of the all but one level of overlap reveals three distinct surfaces. The first is defined for all surfaces except the surface that represents the years from 1958 to 1978. The second surface is defined for all surfaces except the years 1932 to 1957. The third surface is the result of channel being added which did not have date recorded in earlier years. The changes between the first and second surface show the effects of a water control structure completed in 1958 (Figure 4.14), specifly the areas that this structure caused to start and stop flooding.

4.5.2 Bayou Meto

The Bayou Meto Wildlife Management Area is located off the Arkansas river southeast of Little Rock, AR and east of Pine Bluff, AR. It is an important waterfowl management area in the southeastern united states. Seven remotely sensed flood maps from this area were used to test the visualization system with satellite imagery, instead of the model outputs used previously. The individual images in this dataset can be seen in Figure 4.15. The initial visualization for this data set appears to be quite noisy. This can partially be
explained by the fact farmers in the region will flood some of their fields during waterfowl season; however the particular fields artificially flooded in any given year changes. This results in more or less random areas of isolated flooding in each scene. However, some insight can still be retrieved. The maximum level of overlap easily shows the primary channel of the Arkansas river and various cutoff lakes that where difficult to see in the initial image (Figure 4.16 image (d) ). The all but one and all but two levels of overlap reveal the extents of the two wildlife management areas in the study region (Fig 10c), as well as the wetlands region in the north eastern part of the images (Figure 4.16 image(c)).

4.5.3 Amite

The Amite river is located in southern Mississippi and northern Louisiana joining the Mississippi river near Baton Rouge. The river channel in its mid-reach exhibits extreme volatility [48]. Seven flood scenes from this region where analyzed to see how the visualization system dealt with a constantly changing river channel. The initial visualization’s histogram reveals there is almost no overlap between the visualized surfaces. The majority of coverage is one surface class, with most of remaining locations being only two surfaces interacting. This lack of overlap makes analysis of may surfaces simultaneously of little use. However the visualization tool can be used to for change detection between groups of 2 to 3 surfaces that are temporally adjacent (Figure 4.18).

4.6 Expert Feedback

To determine if the proposed system would be useful to experts, hydraulic engineers from the Army Corps of Engineers where interviewed and allowed to interact with the
visualization system and the test datasets. Feedback from the engineers indicated that efficiency with which scenes with multiple interacting surfaces could be analyzed was greatly improved. The features that this system allows users to identify would require multiple hours of work to locate using existing techniques, and because of the difference in required user time, there is no guarantee that all features identifiable with method would have been located using traditional techniques. The ease of use and speed with which analysis can be conducted could determine if studies would be possible given monetary and time constraints. Some suggested areas where our visual analytics system would be of use include:

- Determination of best fit scene from remotely sensed data to calibrate a model against.
- Comparison between output of different models simulating the same event.
- Rapid comparison of flood extents, currently this can require hours using existing software.
- Defining domains to modeled while minimizing potential over or under estimation.

Overall, our system was found to have positive potential impact and suggestions for additional features were provided by our experts.

4.7 Future Work

Currently, a formal user study to test the efficacy of this technique is being designed. The primary difficulties are finding a large enough pool of qualified testers and obtaining suitable test data. The visual analysis system can easily allow a lay users to find potential features. However, in-depth understanding of those features requires expert knowledge about both the geographic region displayed and the modeled/remotely sensed data. Expert
trials thus require novel data that is both correct but presently unseen by knowledgeable users; this makes selection of useful test data difficult. We foresee two possible experiments: Accuracy tests that can be performed by non-experts and insight-based analysis [50] for the experts on novel data.

In addition, two potential enhancements to the technique were requested during the interviews. The first was to extend the technique to work with pre-categorized data, where each category would be treated as a separate input. This can currently be achieved by preprocessing surfaces but one of the goals of this system is to minimize the amount of preprocessing required. The second requested feature is to allow direct visualization of floating point data, this feature like the first can currently be emulated with preprocessing.

4.8 Conclusion

We have presented a new GPU-accelerated visualization system that combines overviews, region selection, and statistical data to allow rapid comparison of multiple flood extents. This system was tested with both model derived data and remote sensed images. Interacting with the visualization system allows features to quickly be located using both types of data. Initial consultations with field experts indicate that the proposed system would greatly improve such datasets are analyzed, and could save significant amounts of both time and money.
Figure 4.2: Multivariate data display using DDS (Data Driven Spots)

Image from Taylor, 2002 [65]. DDS is meant to viewed through an animated display, since each data layer is depicted via a moving sampling texture, with each texture moving in a unique direction (see Bokinsky, 2003) [13]. This image depicts multiple data fields (from the US Census) on a map of the western US. Each layer is represented using textures with unique combinations of color and spot-based sampling patterns. Some textures use three dimensional elements for additional emphasis. Layers are rendered in an arbitrary sequential order over a gray background layer, with each layer blended into previous layers so the final combination preserves previously rendered data. The DDS technique allows for the simultaneous visual analysis of multiple surfaces. However the animated composite images become increasingly more difficult to interpret as the number of surfaces increase. Additionally detail at data boundaries can be obfuscated by the nature of the spot-based sampling, which can easily show false contours. This is somewhat overcome by the use of animated textures. Finally, the use of sampling lowers the resolution of all displayed data.
Figure 4.3: Multivariate data display utilizing 3D glyphs

Image from Forsell, 2005 [26]. The 3D glyphs method is designed to allow for visual analysis of multiple data layers, with two data layers being encoded in the shape of the displayed 3D glyphs. Remaining variables can then be encoded using alternative means such as color or texture. In the above examples, image (a) demonstrates how the relative values of two data layers can be encoded as 3D shapes. Image (b) shows the output of a 3D glyphs visualization using three variables (temperature, precipitation, and wind speed) where two variables determined the shape of the 3D glyphs and the final variable determined color. The output was designed to identify regions where a forest fire was likely to spread. The four red saddle shapes circled in image (b) depict areas in which wind speed and temperature were high and precipitation was low. In tests, users were able to visually identify target areas by examining the output visualization. However, the method does not appear to be easily scalable beyond four variables, and the granularity of the presented information is, by necessity, very low.
Figure 4.4: Multivariate data display utilizing glyph blocks

Image from Hsiao, 2010) [34]. The above image shows environmental data from four different data layers by sampling each layer in a regular grid and displaying a 2x2 glyph block for each sample region. Each variable is assigned a color ramp and the value for the variable in each glyph block is determined by averaging the value for that variable within the sampled region. While the glyph block technique can technically be scaled to include any number of variables, it is particularly important that the colors assigned to variables be isoluminant (as they are not in the above example). Unfortunately, even when using isoluminant colors, as the number of variables increases the output becomes increasingly difficult to visually interpret. Finally, as with any regular grid based sampling method, sampling errors increase as the data resolution increases relative to the display resolution.
Figure 4.5: Multivariate data display utilizing dual-layer visualization

Image from Cai et al., 2006) [15]. The Dual-Layer technique is designed to interpret satellite data which consists of multiple data-layers, each of which depicts the same region viewed through different sensor bands. Dual-Layer visualization allows users to view three types of layers: (1) a false-color hyper-spectral image made by combining multiple sensor band images (ie., data-layers); (2) a pie-chart layer which samples the data in a regular grid and which depicts the relative strength of each data-layer as a pie-chart; and, (3) a combined image made by blending the hyper-spectral image with the pie chart layer. In the above example, images (a), (b), and (c) depict the hyper-spectral, pie chart, and combined layers respectively. Images (d), (e), and (f) depict a zoomed-in view of the same layers centered on a region of anomaly.
Figure 4.6: Multivariate data display utilizing multi-layer visualization

Image from Cai et al., 2010 [16]. The Multi-Layer technique is designed to interpret satellite data which consists of multiple data-layers, each of which depicts the same region viewed through different sensor bands. Multi-Layer visualization allows users to view three primary types of layers: (1) a modified DDS (Data Driven Spots) layer (see Figure 4.2, p.98), in which sampling density for each data layer, rather than opacity, increases with increasing data values; (2) a pie-chart layer which samples the data in a regular grid and which depicts the relative strength of each data-layer as a pie-chart (as first described in Cai et al., 2006) [15]; and, (3) a layer that displays anomalies using shaded spheres and/or Oriented Slivers. The Multi-Layer visualization system presents one of two views based on the zoom level currently being utilized. First, an over-view mode shows the modified DDS and anomaly layers. As users zoom in or select sub-regions for closer viewing, the system switches to detail-view mode, which adds the pie-chart layer, which can be displayed with varying levels of opacity. In the above example, image (a) shows the over-view mode visualization for the entire data domain, while images (b), (c), (d), and (e) show the detail-view mode with the pie-chart layer set to varying degrees of opacity to highlight different aspects of the data. A notable innovation of the Multi-Layer technique is that by changing sampling density, rather than opacity, when rendering the DDS layer makes the different regions within the data far more recognizable.
Figure 4.7: Multi-surface visual comparison through Ensemble Surface Slicing (ESS)

Image from Alabi et al., 2012) [6]. Images (a) - (d) depict four Gaussian blob surfaces. Image (e) shows an ESS rendering combining strips taken from each of the four test surfaces. In this manner differences between the test surfaces are visible as discontinuities between adjacent surface strips. In the above example it can be seen that surfaces (a), (c), and (d) are identical, and that surface (b) differs. The strength of ESS visualization is that luminance discontinuity makes differences between adjacent surface strips easy to detect. However, the technique relies on a sampling methodology which will not necessarily detect all discontinuities; essentially, as the number of surfaces increases, the sampled percentage of each surface decreases.
Figure 4.8: Visualization system in Overview Mode

The visualization system showing the overview display with data from the Yazoo Backwater area. The combination of the color map and the histogram allow the level of overlap at any position to easily be determined.
Figure 4.9: Visualization system in Focus Mode

The visualization system displaying the focus on the third level of overlap with data from the Yazoo Backwater area. Here, we focus on regions where three surfaces overlap. Color blocks group the involved surfaces together. In this cases the sets are: (min_2y, min_5y, min_10y) colored in green, (min_10y, min_5y, may17_08) colored in orange, and (min_10y, min_5y, and h1901_31rc03). Several addition surface sets exist but have minimal contribution to the scene.
Figure 4.10: Ordering of OpenCL operations for maximum throughput

Arrows represent event dependencies, and event can not complete until all event connected to it by arrows have completed. Two image and two data buffers are used in alternating order to store and process the surfaces.
Figure 4.11: The surfaces that make up the Yazoo Backwater dataset.
Flooding in the upper regions of the project area occurs only during frequency events, this indicates short term flooding only. The green surface shown in (a) is made from the 25 year frequency flood. In (b) the surface is the combination of the 25 and 10 year frequency floods. In (c) and (d) the green surface represents the combination of the 25, 10, and 5 year frequencies, and the 25, 10, 5 and 2 year frequencies respectively. Because all of the indicated green locations are covered only by frequency events it means none of them are ever flooded for a period of at least 14 days.
Figure 4.13: Notable subregions in the Yazoo Backwater dataset

The purple ring surrounding the dark blue center area (Figure 7a) shows the degree that the 14-day duration flood has decreased over the period from 1901 to the present. Note how it surrounds the areas shown in (b) and (c).
This selection shows the effects of flood control channel completed in 1958. The green surface shows where all flood surfaces are defined except the 14 day duration for the time period from 1958 to 1978. The orange surface shows where flood surfaces were defined except the 14 day duration surface for 1932 to 1957. In 1958 water transport channel was completed that was intended to move water from the northern part of the basin to the southern part. This structure resulted in the indicated areas in the northern area no longer being flooded for 14 days and thus being removed from the 14 day duration surface. At the same time the orange areas became part of the surface due to being flooded longer.
(a) Surfaces from left to right: Mar 7 1997, Feb 8 93, Mar 1 89, Mar 10 98

(b) Surfaces from left to right: Mar 15 94, Mar 18 95, mar 23 97

Figure 4.15: The surfaces that make up the Bayou Meto dataset.
The initial display for the Beyou Meto dataset is shown in (a). The surface breakdown for none overlapping surfaces is shown in (b). Much of the noise in this dataset is caused by farmland that is deliberately flooded to attract waterfowl; which areas are flooded in this way at any given time is random. In (c) the green is part of a wildlife management area. The orange and pink areas are part of a wetlands area. In (d), the areas of complete agreement are shown.

Figure 4.16: Exploring the Bayou Meto dataset
Figure 4.17: The surfaces that make up the Amite dataset.
The initial display (a), shows an almost total lack of overlapping surfaces. Note how the histogram shows most of the values in 1, 2, and 3 columns. In (b), the overview for two surfaces, July 14, 1991 and Jan 19 1994 is displayed. In (c), differences between the two selected dates are displayed. Green shows locations only defined in Jan 1994 and orange shows locations only defined in July 1991. In (d), locations defined at both dates are highlighted.
CHAPTER 5
CONCLUSIONS

5.1 Contributions

Part I of this dissertation introduces a flood model, Flood Event Simulation Model (FESM), which, when used with direct gauge water elevation measurements, allows for the rapid prediction of flood extents. This model is significant in that most hydraulic models utilize flow rates, rather than elevation data to predict water elevations and inundation extents. By utilizing elevation measurements rather than flow rates, FESM calculates the predicted water surface using only a single time-step, rather than thousands of time-steps, and thus speeds up performance by several orders of magnitude. This increase in speed can be critical when modeling is being done in real-time in response to ongoing real-world emergencies.

FESM requires less input data and configuration than more traditional approaches to flood modeling. In addition, to overcome accuracy issues caused by limited real-world water-elevation sensors, traditional models (Hec-RAS and FLO-2D) can be run at low resolution to supply additional data points for FESM surface prediction. Utilizing Navier-Stokes 1D or 2D flow-models at low resolution to supply additional data points for FESM surface mapping allows for increased accuracy while preserving the reduced computational costs associated with the FESM elevation-based method.
FESM was originally developed for the United States Army Corps of Engineers (US-ACE) as a tool for wetland delineation; however, during the 2005 post Hurricane Katrina flooding of New Orleans, it became clear that the rapid real-time modeling capabilities of FESM made it an ideal tool for emergency response and flood mapping. Since 2005, FESM has been used repeatedly by the USACE to direct and coordinate emergency response to floods across the United States. FESM has been used to model floods in Mississippi, Louisiana, Texas, Oklahoma, Arkansas, and Missouri. FESM was utilized most recently in conjunction with the 2015 floods in upper Texas and Oklahoma.

Part II of this dissertation explores data transfer techniques to allow rapid analysis of large data sets. After introducing the FESM model in Part I, this dissertation then examines data management problems which arise when attempting to process large flood inundation maps. A dual-buffer technique, Dual Buffer Algorithm (DBA), is introduced to handle the volumes of data created by flood models. Part II of this dissertation, therefore, includes an analysis of the efficiency offered by differing techniques of moving datasets between main memory and a GPU during hardware accelerated rendering or analysis.

The DBA data transfer method allows highly efficient CPU to GPU data streaming and is shown to approach the maximum transfer potential of the CPU to GPU bus on the hardware tested. DBA allows multiple surfaces to be processed on the GPU without available GPU memory limiting the number of surfaces considered, i.e., more surfaces can be processed on the GPU than can be placed in GPU memory simultaneously.

An additional finding was that processing time required to transform data from a buffer format to an image format, which can be more efficiently processed by the GPU, was
shown to have a non-linear relationship with image size. This in turn revealed the existence of ideal image dimensions wherein the mapping operation was significantly faster than with other possible image dimensions. Additional testing along these lines could allow further GPU processing acceleration for arbitrary data by using optimally dimensioned image segments. Even in the worst case, processing of uniform data fields would still be accelerated by segmenting data into optimally sized images.

While the DBA data transfer technique allowed the iterative processing of very large two-dimensional data sets for real-time analysis, there was still no interface methodology allowing useful human interpretation of data from multiple flood scenes. Therefore, Part III of this dissertation details the development of Dynamic Raster Overlay (DRO), a visual analytics system allowing for hardware accelerated interactive viewing of expansive grid-based data sets. DRO allows users to display multiple rasters, combine them, and dynamically select which individual rasters are interacting at any given moment. Users can then customize output to highlight specific degrees of overlap, thereby identifying regions in which the specified level of overlap exists, and determining the specific data sets overlapping to form each region.

Prior to DRO, this type of analysis was both slow and difficult to achieve, as no tools or optimized methodology existed specifically designed to accomplish this task, which has often been undertaken by the USACE, utilizing software which required significant time and multiple user steps in order to visualize each proposed query. DRO accomplishes the same tasks dynamically, in real-time, thus allowing interactive usage and improving efficiency by many orders of magnitude. DRO will allow the USACE to simultaneously analyze
multiple flood scenes, from historical data or simulations, in order to determine areas of overlap variability and thus the real-world significance of actual or potential changes, such as the occurrence of disasters or the implementation of water control structures.

Currently, DRO is limited by graphics memory regarding the maximum size surfaces (rasters / data sets) that can be analyzed; however, this limitation can easily be overcome by allowing input segmentation.

In summary the contributions of this work have been

- Introduction of the Flood Model FESM
- Verification and validation of the FESM model
- Introduction of the dual buffer algorithm (DBA) for data transfer from the CPU to GPU
- Testing of the DBA on NVIDIA and AMD hardware
- Introduction of a Visualization analysis system for multiple overlapping images
- Case studies on the usage of this visualization system
REFERENCES


