IMPLEMENTING DYNAMIC VISUALIZATION OF INTERACTIVE TEXT STREAMS ON MOBILE DEVICES

A thesis submitted
to Kent State University in partial
fulfillment of the requirements for the
degree of Master's of Computer Science

by
Ryan O'Neill
MAY 2014
Thesis written by
Ryan O'Neill
M.S., Kent State University, 2014
B.S., Cleveland State University, 2006
B.A., Cleveland State University, 2006

Approved by

Dr. Ye Zhao, Chair, Master's Thesis Committee
Dr. Arden Ruttan, Member, Master's Thesis Committee
Dr. Paul Farrell, Member, Master's Thesis Committee

Accepted by

Dr. Javed Khan, Chair, Department of Computer Science
Dr. James L. Blank, Dean, College of Arts and Sciences
TABLE OF CONTENTS

LIST OF FIGURES ......................................................................................................................... VI
LIST OF TABLES ............................................................................................................................. VII
DEDICATION ....................................................................................................................................... VIII
ACKNOWLEDGEMENTS .................................................................................................................... IX

CHAPTER 1 INTRODUCTION ........................................................................................................... 1
  1.1 Motivation .................................................................................................................................. 1
  1.2 Problem Statement .................................................................................................................. 3
  1.3 Objectives .................................................................................................................................. 4
  1.4 Approach and Contribution ...................................................................................................... 4

CHAPTER 2 STREAMIT BACKGROUND AND RELATED WORK ........................................... 6
  2.1 STREAMIT Feature Overview .................................................................................................. 7
    2.1.1 Similarity Based Projection ................................................................................................. 7
    2.1.2 Temporal Evolution ........................................................................................................... 8
    2.1.3 Force Directed Graph Placement ...................................................................................... 9
    2.1.4 Dynamic Keyword Importance ....................................................................................... 10
    2.1.5 Dynamic Clustering .......................................................................................................... 11
  2.2 Visualization Work on Mobile Platforms .................................................................................. 12

CHAPTER 3 PLATFORM AND DEVELOPMENT DETAILS ........................................ 14
  3.1 Android Overview ................................................................................................................... 14
    3.1.1 Development Details ........................................................................................................ 15
  3.2 Targeted Android Platform ...................................................................................................... 17
3.2.1 Device Hardware................................................................................................. 17

3.3 Development Tools Used...................................................................................... 18
  3.3.1 Development Kits................................................................................................ 18
  3.3.2 Cygwin .............................................................................................................. 18
  3.3.3 Android Debug Bridge ...................................................................................... 19
  3.3.4 AVD Manager and Android Emulator ............................................................... 20
  3.3.5 Necessitas Qt Suite .......................................................................................... 20
  3.3.6 Ministro ........................................................................................................... 22

CHAPTER 4 INTERFACE IMPLEMENTATION ......................................................... 23
  4.1 Qt Framework ........................................................................................................ 23
    4.1.1 Background .................................................................................................... 23
    4.1.2 Development Considerations ........................................................................ 25
  4.2 Qt on Mobile Platforms ......................................................................................... 25
    4.2.1 Preliminary Mobile Support ......................................................................... 26
    4.2.2 Necessitas in Qt 4 ........................................................................................ 26
    4.2.3 Official Android Support ............................................................................... 27
  4.3 STREAMIT-Mobile Interface Modifications ...................................................... 28

CHAPTER 5 RESULTS AND PERFORMANCE ...................................................... 30
  5.1 Mobile Application Results .................................................................................. 30
    5.1.1 What Works ................................................................................................... 30
    5.1.2 What Doesn’t ................................................................................................. 31
  5.2 Comparison .......................................................................................................... 31
5.2.1 Results ............................................................................................................. 33

CHAPTER 6 CONCLUSIONS AND FUTURE RESEARCH .......................... 37

6.1 Conclusions ........................................................................................................ 37

6.2 Future Work ........................................................................................................ 37

REFERENCES ........................................................................................................... 40
LIST OF FIGURES

Figure 1.1. Tablet Market Splitting. Image courtesy of ben-evans.com [5] ................... 2
Figure 2.1. System Overview [13] ............................................................................... 8
Figure 2.2. Cluster Generation [13] .............................................................................. 11
Figure 3.1. Accessing Native Code via JNI. Image courtesy of Intel [37] ...................... 15
Figure 3.2. NDK vs SDK App Structure. Image courtesy of Intel [37] ......................... 19
Figure 3.3. AVD Manager Configuration ......................................................................... 20
Figure 3.4. Necessitas Qt Creator .................................................................................. 21
Figure 4.1. STREAMIT Desktop Interface ................................................................... 28
Figure 4.2. STREAMIT-Mobile Interface .................................................................... 28
Figure 5.1. Desktop Simulation Output Example .......................................................... 32
Figure 5.2. Mobile Simulation Inputs ........................................................................... 34
Figure 5.3. Mobile Simulation in Progress (Full Screen Zoom) .................................... 35
Figure 5.4. Mobile Simulation Output ......................................................................... 35
Figure 5.5. Mobile Simulation Cluster Display ............................................................ 36
Figure 5.6. Mobile Simulation Triangulation Display .................................................... 36
LIST OF TABLES

Table 3.1. Nexus 7 Specifications................................................................. 17
Table 4.1. Qt Language Bindings .................................................................. 24
Table 5.1. Simulation Time Comparison.......................................................... 34
DEDICATION

I dedicate my thesis work to my always supportive family; thank you for continuously asking me how my work was progressing. I might have forgotten I was supposed to write my paper without the good-natured (and consistent) reminders.

I also dedicate my work to my incredibly supportive partner, Briana. Thank you for putting up with my constant preoccupation while I was finishing. I’m sure it was a trying experience dealing with me, but most of the time I could barely even tell how annoyed you were.

Also, thanks for proofreading, Devin and Eric. I won’t say which one of you did the better job; you know who you are.
ACKNOWLEDGEMENTS

I’d like to acknowledge the creators of the STREAMIT visualization system, especially my advisor Dr. Ye Zhao for his valuable input and assistance on this project, and for continuing to work with me after the extended hiatus from completing my thesis. I’d also like to acknowledge the support of my employer, Georgia Tech Research Institute, and specifically my supervisor Dr. Greg Rohling, who made it very clear that I would be finishing my degree.

Ryan O'Neill

April 02, 2014, Kent, Ohio
CHAPTER 1

Introduction

1.1 Motivation

The mobile computing market has experienced explosive growth in recent years. Global smartphone sales nearly doubled between 2011 and 2013, going from an estimated 472 million devices sold in 2011 to 968 million sold in 2013 [1] [2]. During the same period, tablet sales nearly tripled, from an estimated 60 million in 2011 to 195 million sold in 2013 [3] [4], with Android tablets recently leading the charge in what was once an iPad dominated market [5]. Annual tablet sales are anticipated to exceed traditional PC sales by the end of 2015 [6]. While these devices are becoming ubiquitous, they are also quickly becoming more powerful, with current generation tablets having hardware that rivals laptop hardware. Current generation consumer tablets are equipped with high pixel density displays, multi-core processors, dedicated GPUs, and extended battery life [7].

While the most obvious contributors to this massive increase in demand are casual users, professional and scientific markets are showing interest in incorporating tablets as well [8]. Visualization applications on mobile hardware has also become an expanding area of research, demonstrated in [9], [10], and [11]. While some initial implementations commonly incorporated a mobile device solely for a display, employing various types of distributed calculations, it is becoming increasingly more common to perform computations natively on the device itself.
Following the increased consumer popularity of these devices, online content generation is occurring at a faster pace than ever before, and a large amount of this content, be it news columns, blog posts, or content from Facebook or Twitter, can be represented by text streams. Viewing and understanding these rapidly updating streams of text requires an effective, interactive method to explore and analyze their huge amounts of data on the fly.

One such approach for visualizing text streams in real time is the STREAMIT visualization system, implemented by Alsakran et al [12] [13]. Their approach uses a force directed simulation into which text documents are continuously inserted. Users can explore
incoming documents and document clusters which are presented in a dynamic 2D display. Documents are clustered by keywords or topics, and emerging patterns are visualized in real-time [13]. The user interface and output of the STREAMIT desktop application primarily uses mouse interaction inside its visualization sub-window and simulation controls, which adapts well to the touch screen displays of modern tablet hardware and makes STREAMIT an ideal candidate for a tablet implementation.

1.2 Problem Statement

While the rapidly increasing number of mobile devices in use has dramatically increased the opportunities and necessity for efficient text stream analysis, on-the-fly analysis on mobile devices is a topic that is only recently receiving much attention. Characteristics that diminished the usefulness of mobile platforms only a few years ago, such as limited display resolution and general lack of processing power, have made enormous strides. Mobile hardware has reached a point where commodity off-the-shelf devices are capable enough to be incorporated in visualization and analysis applications, and given an appropriately designed interface, the touch screens present on modern smartphones and tablets have the potential to be more intuitively usable than their desktop counterparts.

A requirement of a visualization system on a mobile platform would be the necessity for a clean interface with easily differentiable elements to account for limited screen size and pixel density. It would also need to contain touch friendly controls and not rely on keyboard or mouse interaction. At the same time, it must be responsive to user input and able to react in real time on the targeted hardware.
1.3 Objectives

I intend to present an important preliminary contribution to implementing real-time text stream content visualization on an off-the-shelf mobile device; specifically, a tablet running the Android operating system. This will be accomplished by implementing the STREAMIT visualization system on an Android tablet. The user interface will be adapted to be touch-friendly and properly accommodating of a portable display size. The main intention is to demonstrate the capabilities of modern, readily available mobile hardware and touch screen interfaces as a viable platform for the task of dynamic, real-time text stream visualization in comparison to desktop computing hardware.

1.4 Approach and Contribution

The approach presented in this thesis implements the STREAMIT visualization system, which is programmed as an x86 desktop application running in Windows, onto a common, low-cost tablet running Android 4.3.1. My implementation involved transitioning build environments from Microsoft Visual C++ compilation to Linux GNU g++ compatible code, and from an x86 desktop development process to a remotely deployed Android application built for an ARM processor.

The development was accomplished with the Android 4.3.1 (API Level 18) SDK as well as including native code compilation via the Android NDK r8b revision. The user interface has been ported from desktop Qt 4.7.4 to Qt Necessitas 4.8.2. Interface elements have been modified for the touch screen tablet interface, and performance issues and limitations inherent to the platform are addressed. The mobile application I present, which
I refer to as STREAMIT-Mobile, is a capable alternative implementation of the STREAMIT visualization system, running on low-cost, low-power mobile hardware.
CHAPTER 2

STREAMIT Background and Related Work

This chapter contains an overview of the STREAMIT text stream visualization system created by Alsakran et al in [12] [13], which my work was based on directly. Examples of two streaming data visualization systems running on mobile Android hardware are presented as well.

I have included a general synopsis of various aspects of the text stream analysis and visualization techniques employed by the STREAMIT system, with enough detail included to give one a general understanding of the overall behavior of the program. With the focus of my presented paper and mobile application being on the porting of STREAMIT’s existing methodology to a mobile platform, I have opted not to linger overly long on the desktop application’s implementation details. Further details of the STREAMIT visualization system can be found in the cited STREAMIT resources [12] and [13], and the most current information on STREAMIT can be found at:

http://www.cs.kent.edu/~zhao/streamit
2.1 STREAMIT Feature Overview

STREAMIT is an interactive, real-time visualization system used for display and analysis of continuous text stream input. Documents are fed into the software, and are analyzed and visualized in a force-directed simulation on the fly [13].

2.1.1 Similarity Based Projection

As documents inputs are fed into the system, similarity between them is computed based on a dynamically adjustable keyword importance. The documents are visualized as “document particles”. The similarity between the documents defines the potential energy between their corresponding document particles, and as they are inserted, the particles are automatically placed and arranged to minimize the potential energy of the system [13].
2.1.2 Temporal Evolution

As the number of documents in the stream grows, document particle clusters form based on document similarity. As new documents are continuously inserted, they automatically join appropriate clusters. For any given moment in the temporal evolution of the text stream, the system will settle to an equilibrium state representative of the clusters of similar document particles. Documents that display erratic particle motion, such as constantly jumping from cluster to cluster over time, are potentially representative of outliers or developing trends in the incoming text stream [13].
2.1.3 Force Directed Graph Placement

As Alsakran et al. outline, the potential energy of the system is calculated with a global summation of the pairwise energy between all currently simulated particles [13]. With each pair of particles having a potential energy $\Phi_{ij}$ defined by:

$$\Phi_{ij} = \alpha * (|l_i - l_j| - l_{ij})^2$$

Where document particles $p_i$ and $p_j$ have positions $l_i$ and $l_j$, $|l_i - l_j|$ represents their current Euclidean distance, and $l_{ij}$ is their ideal distance based on similarity. The keyword similarity $l_{ij}$ is computed by:

$$l_{ij} = 1 - \delta(p_i, p_j)$$

where $\delta(p_i, p_j)$ is the cosine similarity between particles $p_i$ and $p_j$. Documents with a large similarity will have a small ideal distance, and will cluster together in the visualization.

A numerical simulation optimizes the particle positions toward an equilibrium state by minimizing the global potential. The global potential is the sum of pairwise energy of all particles in the system:

$$V(l_1, ..., l_N) = \sum_{i} \sum_{j > i} \Phi_{ij}$$

Where $N$ represents the particle number, and $l_1 ... l_N$ are the current particle locations. At each time step until each particle reaches equilibrium position, forces act on each particle to attract or repel them from one another:

$$F_i = -\nabla_i V(l_1, ..., l_N)$$

From Newton’s second law, $F_i = m_i a_i$ where $m_i$ is the mass of a given particle, the
acceleration of each particle at a particular time step can be obtained by:

\[ a_i = \frac{\sum_j \alpha (|l_i - l_j| - l_{ij})}{m_i}, \]

which is then used to update the particle location. When each particle’s individual movement drops below a threshold value, position updates stop and the system is considered at an equilibrium. An empirically derived constant parameter is applied to the force calculations to stabilize the simulation [13].

### 2.1.4 Dynamic Keyword Importance

Document similarity is computed based on keywords. During text stream visualization, the user is able to interactively modify the significance of keywords on the fly. Frequently encountered keywords are presented in the user interface, through which the user can dynamically modify their importance. Modifying keyword importance will effectively change the keyword’s “weight” in the simulation, effecting the clustering behavior of documents related to the given topic [13].

The advantage of dynamically modifiable keyword importance is that the end user can alter their topics of interest over the course of analyzing a text stream. The contents of a text stream could potentially be continuously processed over a very long period of time, and the document contents could change significantly. The ability to dynamically adjust keyword importance, coupled with allowing the user to adjust both the maximum document age and the maximum number of documents in the visualization, aids in long term text stream analysis [12].
2.1.5 Dynamic Clustering

By itself, the act of grouping document particles based on similarity would give a user a means to visually identify trends in the stream based on cluster contents, but they would still be left with a disjoint set of documents to analyze. To support cluster-level operations and analysis, STREAMIT discovers clusters in the simulation based on their geometric proximities and provides cluster specific operations to the user. The user may even narrow the focus of their simulation to a specific cluster that they are interested in, hiding unrelated documents [13].

The cluster detection is handled with a Delaunay triangulation, building a triangle mesh with vertices at each particle position, followed by a graph cut operation to remove all edges with length larger than a predefined threshold. During cluster generation, each cluster is assigned an ID and a color to help differentiate it from unrelated clusters [13].

(a) The original particles. (b) Delaunay triangulation results (c) Graph cut results.

Figure 2.2. Cluster Generation [13]
2.2 Visualization Work on Mobile Platforms

Visualization on mobile devices is a relatively recent area of research, but there have already been several significant contributions. Although streaming text visualization is as-of-yet a relatively untapped portion of the topic, there some approaches that employ data stream visualization techniques similar to the approach of the STREAMIT-Mobile visualization software presented here.

An approach is presented by Gaber et al. for effective visualization of data streams on mobile phones in [9]. Their presented example connects a mobile device with a data stream of GPS locations, which are analyzed by a clustering algorithm and overlaid on local area maps obtained from Google Maps. Their focus is on clutter reduction in the visualized results, to aid readability on handheld devices with limited screen real estate.

Gaber et al. further expand upon their Adaptive Clutter Reduction approach, as well as presenting further enhancements to their clustering visualizer, adding dynamic control over clustering visualization variables such as cluster growth controls, selective focusing, and screen fencing to narrow the area of visualization and conserve mobile device resources [10].

Correia et al. present an interesting approach using a mobile application to visualize patterns in the routines of everyday behavior. Their software runs on an Android smartphone, collecting the GPS location data of the user’s movements. The software then applies a pattern recognition algorithm to identify the user’s routines and predict their future behavior [11].
The application continuously updates your location history by connecting to a data stream of GPS coordinate information. As it captures coordinate data, the software transforms the geographic vs. temporal information into a series of simplified daily routine graphs. Each day’s routine is represented by a vertical bar indicating the temporal dimension, and color coding based on approximate location. The visualization also makes predictions of the next few days’ activity based on your history [11].
CHAPTER 3

Platform and Development Details

This chapter examines details of the targeted mobile platform, including an overview of the Android operating system structure and characteristics and how they relate to application development. Details on the targeted tablet’s hardware and software configuration are presented, and how the chosen device relates to the application requirements are discussed.

3.1 Android Overview

Android is an open source operating system based on the Linux kernel, maintained and distributed by Google Inc., and designed for mobile devices such as smartphones and tablets [14]. Android is a very popular target platform for developers, evidenced by the size of the Google Play store, which as of December 2013 has over 1 million apps available for download [15]. Android also accounts for the majority of the global smartphone market, comprising 79 percent of the global market share in 2013, compared to iOS’s 15 percent [16]. When we shift the focus to tablet market share, Apple’s iOS, which held an advantage over Android for several years, was surpassed in 2013 when Google’s Android OS grew to 62 percent of the global market [17].

The abundance of devices available and wide range of mobile hardware supported by the Android operating system, coupled with its arguably more open development
structure [18], makes it an ideal target development platform for demonstrating visualization capabilities of mobile devices.

3.1.1 Development Details

Internally, the Android software stack is built on top of the Linux kernel and runs Java applications using Java core libraries. Each Java application instance is constrained to its own VM (virtual machine) called Dalvik [14]. Dalvik uses its own Java Byte Code format and is designed around optimizing the usage of mobile devices’ low power CPUs and limited memory. Below the application level, Android relies on the Linux kernel to handle system level functionality, such as power management, threading, and hardware interactions [14].

Developers can build Android Java applications via the software development kit (SDK) provided by Google. The Android SDK supports both Windows, OSX, and Linux desktop operating systems and includes an Android emulator and AVD (Android Virtual Device) Manager capable of generating virtual devices representative of multiple hardware

![Diagram of SDK API, Java App, JNI, Native Libs, NDK API, and C Framework.](image)

*Figure 3.1. Accessing Native Code via JNI. Image courtesy of Intel [37]*
and software version configurations. The Android SDK includes support for the Eclipse IDE (integrated development environment), but there are projects incorporating support for other IDEs such as Microsoft Visual Studio and Qt Creator [14].

Starting with Android 1.5, Google supports C and C++ development through the Android NDK (Native Development Kit), which is used in conjunction with the Android SDK. While coding applications in C++ with the NDK is not a guarantee of a performance improvement, tasks that are computation-limited can see a benefit [19]. Unlike building Java applications with the SDK, building applications with the NDK is handled by directly calling command-line programs to build, deploy, and debug applications. There are third-party projects that incorporate the NDK build process into IDEs such as Eclipse, Visual Studio, and Qt Creator.

The internal structure of an application built with the Android NDK will typically involve some portion of native code connected to a Java application via the JNI (Java Native Interface). This allows the managed code in Java applications to directly interact with precompiled native functions [14]. Using JNI, a developer may encapsulate sections of code that are most likely to benefit from native compilation (such as computationally intensive calculations) while retaining access to the Android SDK API in the managed portion of the application. As of Android 2.3, it is possible to write, compile, and deploy an application without including any Java whatsoever, but the code will still be executed on a Dalvik VM, and some Java calls will be occurring in the background, including some Android platform features accessed via the JNI [19].
3.2 Targeted Android Platform

The development device used in my implementation is the Google Nexus 7 tablet (2012 version), running Android 4.3.1 “Jelly Bean”, API Level 18. The Nexus 7 was released in July of 2012, and is one of the most commonly owned Google tablets in terms of initial sales and market share, having sold around 7 million units worldwide. One year after its release date, the Nexus 7 was estimated to comprise approximately 10% of the total Android tablet market [20].

3.2.1 Device Hardware

The Nexus 7 uses a 7” LCD capacitive touchscreen, in 16:10 widescreen aspect ratio with 1280x800 (216ppi) display resolution.

The tablet’s CPU and GPU are integrated within an Nvidia Tegra 3 SoC (system-on-a-chip), containing a Cortex-A9 quad-core ARM CPU running at 1.3GHz alongside a

<table>
<thead>
<tr>
<th>Table 3.1. Nexus 7 Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OS</strong></td>
</tr>
<tr>
<td><strong>Display Size</strong></td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
</tr>
<tr>
<td><strong>SoC</strong></td>
</tr>
<tr>
<td><strong>CPU</strong></td>
</tr>
<tr>
<td><strong>GPU</strong></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td><strong>Storage</strong></td>
</tr>
</tbody>
</table>
twelve-core Nvidia GeForce ULP (ultra-low power) GPU running at 416MHz. The tablet comes with 1GB DDR3L low voltage RAM, running at 1333MHz [7].

3.3 Development Tools Used

The STREAMIT-Mobile application was developed using the Android SDK and NDK, and employed the Qt 4.8.2 framework via the Necessitas project. All development was completed on Windows 8 64-bit, using a combination of the Necessitas Qt Creator IDE and the Android command line tools.

Deployment of the application was accomplished via Google’s adb command line tool (Android Debug Bridge), and testing of the application was done on both an Android Virtual Device generated with the AVD Manager, and directly on hardware with a developer-unlocked Nexus 7 tablet.

3.3.1 Development Kits

The Android SDK provides the API and developer tools necessary to develop, deploy, and debug Java applications for the Android platform. The Android NDK acts as a companion kit to the SDK, and is required if you need to implement some portion of your application in native code [19].

3.3.2 Cygwin

Cygwin is a Linux-like environment for Windows, with the purpose of providing Linux scripts and software compatibility in a Windows installation. Developing Java applications with the Android SDK from Windows does not require Cygwin, but the Native
Development Kit does, due to several of the NDK specific command line tools having dependencies on Linux utilities [19].

3.3.3 Android Debug Bridge

The Android Debug Bridge is a command line tool (adb) for communicating with Android devices, whether they are virtual devices running in an emulator or a device connected to the development machine by USB. The Debug Bridge acts as a client-server program, communicating with a daemon running on the targeted device. It is necessary for pushing/pulling files and installing/removing packages from the connected device from the command line [14].

Figure 3.2. NDK vs SDK App Structure. Image courtesy of Intel [37]
3.3.4 AVD Manager and Android Emulator

The AVD Manager is packaged with the Android SDK, and provides a user interface for creating and modifying AVDs (Android Virtual Devices). Generated AVDs are then run on your development machine via the Android Emulator. Generating AVDs allows efficient testing of an application across multiple hardware and software configurations without requiring multiple physical devices. Configuration of an AVD mimicking the profile of the Nexus 7 is pictured in Figure 3.3.

3.3.5 Necessitas Qt Suite

STREAMIT-Mobile makes use of the Qt Framework via the Necessitas Qt Suite, a community project with the objective of making the Qt 4 framework available on the Android platform. Necessitas uses the Android NDK to compile a Qt application.
as a shared library. A Java application wrapper is then packaged alongside the library in an APK (Android Package) file, which can then be deployed to an Android device. The Necessitas Qt Suite also includes a modified version of the Qt Creator IDE that supports compilation, deployment, and on-device debugging of Qt Android applications via wrapped calls to the NDK command build tools and the Android Debug Bridge. The modified IDE also incorporates portions of the AVD Manager to generate and interact with virtual devices without leaving the IDE (see Figure 3.4). The most recent beta versions of Necessitas are feature rich and stable enough that there are multiple Qt apps built with Necessitas available for download in Android’s Google Play store [21].

For the development of Qt 5, Digia, the company that holds the Qt trademark, began pursuing official support for the Android platform by way of incorporating the Necessitas community project into their upcoming release. The release of Qt 5.1 in July of 2013 saw

![Figure 3.4. Necessitas Qt Creator](image)
an inclusion of an experimental “technology preview” for both Android and iOS. The release of Qt 5.2 in December 2013 was the first to include official support for the Android platform, announced in [22] [23]. While the Qt 4 support available through the Necessitas project provides significant utility to the platform, the unofficial nature of the system results in a sometimes convoluted and “temperamental” development process. Digia’s decision to officially include Android as a supported platform in their Qt framework should profoundly simplify the process of coding mobile Qt applications.

3.3.6 Ministro

Ministro is an Android app required by Qt programs developed with Necessitas. When the application starts, it contacts the Ministro service on the device and sends a list of Qt modules used in the application. The Ministro service automatically detects any missing Qt library dependencies, which are downloaded and installed in the background. If the application cannot find the Ministro service on the device, the user is directed to the Ministro installation page in the Google Play store [24].
CHAPTER 4

Interface Implementation

This chapter examines background on Qt, the selected application framework, and details the user interface implementation and design in STREAMIT-Mobile. The user interface of the STREAMIT desktop application required modifications to both design and implementation to allow interaction via touch screen instead of a mouse and keyboard.

4.1 Qt Framework

Qt is an open source application framework written in C++ and available under GPL v3 and LGPL v2 licenses, as well as commercial licenses for developing proprietary applications. It is described as a “cross-platform application and UI framework”, which includes a widget toolkit for building user interfaces, as well as a large number of non-GUI classes including thread management, SQL queries, XML parsing, and network support [25]. Although written in C++, current versions of Qt have bindings available for multiple languages.

4.1.1 Background

The Qt framework has been publicly available since 1995, initially released by a Norwegian company called Trolltech. Qt was maintained by Trolltech for the first 13 years of its life, including preliminary support for mobile platforms in 2000 when they released Qt for embedded Linux and X11, as well as an initial release Qtopia, which was an application framework designed for PDAs (Personal Digital Assistants) and mobile phones.
In 2008, shortly after the release of Qt 4.4, Nokia acquired Qt from Trolltech and in addition to continuing desktop releases, began focusing on development for their mobile platform, with the announcement detailed in [27]. In 2012, Nokia completed the sale of Qt to the current owner Digia, who developed the major release Qt 5 in December 2012. Qt is currently a very well established application framework, and is used in many very well-known pieces of software, including KDE, Autodesk Maya, Google Earth, and Skype [28].

Qt 4 and 5 both have official language bindings for C++ and QML (Qt Modeling Language), which is a JavaScript based declarative scripting language included with the

<table>
<thead>
<tr>
<th>Language</th>
<th>Qt 4 Support</th>
<th>Qt 5 Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>QtAda 2005 (Qt 4.6+)</td>
<td>Qt5Ada</td>
</tr>
<tr>
<td>C++</td>
<td>Official</td>
<td>Official</td>
</tr>
<tr>
<td>C#</td>
<td>Qyoto</td>
<td>N/A</td>
</tr>
<tr>
<td>Java</td>
<td>Qt Jambi</td>
<td>N/A</td>
</tr>
<tr>
<td>JavaScript</td>
<td>N/A</td>
<td>Official (Qt Quick)</td>
</tr>
<tr>
<td>Lisp</td>
<td>CommonQt</td>
<td>N/A</td>
</tr>
<tr>
<td>Lua</td>
<td>QtLua</td>
<td>N/A</td>
</tr>
<tr>
<td>Pascal</td>
<td>FreePascal Qt4</td>
<td>N/A</td>
</tr>
<tr>
<td>Perl</td>
<td>PerlQt</td>
<td>N/A</td>
</tr>
<tr>
<td>Python</td>
<td>PySide, PyQt</td>
<td>PyQt</td>
</tr>
<tr>
<td>QML</td>
<td>Official (Qt Quick)</td>
<td>Official (Qt Quick)</td>
</tr>
</tbody>
</table>
Qt Quick UI creation kit, a component of the Qt Framework. In addition, both Qt 4 and 5 are available for a variety of bindings for other languages, achieved via third party interfaces, several of which are detailed in [29], [30], and [31]. Examples of available language bindings for Qt 4 and 5 are shown in Table 4.1.

4.1.2 Development Considerations

The STREAMIT visualization system depends on the Qt framework, which was one of the primary considerations when deciding on the direction to take development. Replacing the Qt dependency with another application framework would not only add significant overhead to the port, but would cause some unwanted code divergence between the two projects. Even considering the available support at the time being unofficial and experimental, staying within the same framework offered a significantly simplified development route to port to a new platform.

The availability of the Qt Necessitas framework, combined with the popularity of the Android and readily available development tools, made the choice of target platform straightforward.

4.2 Qt on Mobile Platforms

The most recent versions of Qt 5 contain official support for the Android platform, accomplished through incorporation of the work completed for Qt 4 by the community project Necessitas. The official inclusion has been considered a significant addition to the Qt framework, but Trolltech actually brought Qt support to mobile devices several years earlier [22].
4.2.1 Preliminary Mobile Support

Qt began support for embedded Linux platforms in 2000 [26], and by 2006 it was already in use on millions of mobile devices worldwide running the Qtopia platform (later renamed Qt Extended) [32], including several Linux handhelds and multiple smartphones developed by Openmoko, a project with the goal of creating open source mobile phones [33]. Perhaps somewhat ironically (or perhaps not; “ironically” is an easy word to misuse), a major contributing factor for the decline of Qt’s mobile device support was Google’s 2007 announcement of their Android operating system and Android’s official release the following year; information on the announcement and speculation on its effects are detailed in [34]. Android very quickly overshadowed both platforms, and by 2009 the Qt Extended platform had been cancelled.

An open source fork of Qt Extended named “Qt Extended Improved” was created in 2009, and the Openmoko project still has active community support, including the creation of QtMoko, a Debian-based distribution designed to run on some Openmoko handsets the implements the Qt Extended Improved application platform, but it is a very minor player next to Android’s market saturation. Information on the Openmoko platform can be found at [33].

4.2.2 Necessitas in Qt 4

The Necessitas project is a port of the Qt 4 framework to the Android platform. The initial alpha version became available in early 2011 containing the preliminary port of the Qt framework, a modified version of Qt Creator integrated with the platform, and the Ministro application, which automatically downloads and installs any required Qt libraries
After several more alpha releases, a stable beta version was released in October 2012, announced with [35]. Shortly afterwards in November 2012, it was announced that the Necessitas project would be incorporated into the upcoming Qt 5 release, and what is currently the final version of Necessitas was released in December 2012 [23].

4.2.3 Official Android Support

The official inclusion of Android as a supported platform occurred with the Qt 5.2 release, just over a year after the official announcement. The work was completed with the input of the primary author of the Necessitas project, Bogdan Vatra, who began contributing as the Qt Project maintainer of the Qt 5 android port in late 2012 [36].
4.3STREAMIT-Mobile Interface Modifications

The largest consideration when bringing the STREAMIT desktop application to a tablet was display size. The 7” display does not have enough available resolution to keep every UI element on screen at once.

With the Main Window being the primary area of focus, I opted to allocate it the majority of the display size by default, keeping the keyword table and document table hidden unless requested by the user. The menus are also moved to the native Android menu format, being accessed by the navigation bar at the bottom of the screen. For the purpose of rendering simplicity, the interface is locked to a landscape orientation.

![Figure 4.1. STREAMIT Desktop Interface](image1)

![Figure 4.2. STREAMIT-Mobile Interface](image2)
User interaction inside the simulation window required modification to aid usability with a touch screen interface. The desktop software handled scrolling via the mouse wheel. For STREAMIT-Mobile, the implementation was modified to instead use multi-touch pinch-zooming.

The desktop application displays underlying document data for any rendered document particle by showing the relevant text in a tooltip, which is only displayed when the user hovers the cursor over a particle. Android does not use a cursor, nor does Qt Necessitas support desktop-style tooltips. Instead, in the STREAMIT-Mobile implementation a user requests a document particle’s information by performing a single-tap inside the particle, which draws a label alongside it containing the keyword values and simulation weight. Only a single particle’s label is displayed at a time inside the graph, and remains displayed until the user taps another particle, or dismisses the label by tapping an empty area of the graph.
CHAPTER 5

Results and Performance

This chapter examines the results of the STREAMIT-Mobile Android implementation, and a comparison of application capability and appearance to the desktop version of the visualization system.

5.1 Mobile Application Results

5.1.1 What Works…

The current version of STREAMIT-Mobile is able to connect to simulated text stream datasets, allows user control of simulation variables such as clustering parameters, keyword importance values and colors, and displays the temporally evolving document particles. Clustering, keyword weighting, and input modifications are all handled and displayed appropriately.

The user is able to display triangulation between nodes in the graph, and cluster highlighting, as well as select any rendered node to show its underlying keyword data and weight values. The simulation inputs menu and keywords table are displayed in a tabbed dock that can be shown or hidden at the user’s request. When the dock is hidden the simulation window is switched to a full screen display.
5.1.2 What Doesn’t…

The mobile version of the visualization system cannot handle as large a number of document particles as the desktop version. While the desktop application may begin to stutter slightly in its animations during very large simulations, the mobile application will slow down noticeably if taxed too hard.

The node label paradigm in the desktop software uses tooltips to display node information. This does not translate at all to a touch screen, as there is not cursor. STREAMIT-Mobile implements touch events on rendered nodes to display document information information to the user, but it is definitely not as intuitive to use when compared to the desktop implementation.

A development-specific issue that I have been unable to solve is a problem maintaining a connection with the Android gdbserver running on hardware, making debugging a long and arduous process. I believe a significant portion of the lingering issues are configuration related, and would be alleviated, if not outright solved, by moving the mobile codebase to the Qt 5 Framework. Unfortunately, with the platform support only becoming officially available in December 2013, the migration to Qt 5 was impossible to accomplish during the scope of my thesis.

5.2 Comparison

To compare the output of the STREAMIT-Mobile application to the original STREAMIT visualization system, I ran identical example datasets through each version, along with matching all parameters between the two systems. The simulations were each run to the same time step, and screenshots of both visualizations are included here.
Simulation times between the desktop and mobile applications were measured and compared. The testing procedure used three different simulated input files, containing 50, 94, and 221 input items. The measurements, indicated in Table 5.1 below, are for both versions of the application to process the contents of the simulation file, generate and insert appropriate particles for each input, inject them into the scene, and finally reach an equilibrium where all particle movement is ceased. The mobile version saw varying amounts of increases to processing time based on simulation size. For very small simulations, there was a very marked difference, with the mobile version taking 144% more time than the desktop. A larger simulation graphing 94 nodes only showed an improvement, with only a 33% increase in processing time compared to the desktop. The largest set of simulation inputs test saw the mobile performance drop again, with a 95% comparative increase in processing time. Based on the measurements, there are likely some

![Desktop Simulation Output Example](image-url)
sections of code specific to initialization that take considerably longer on the mobile platform, with their impact being lessened as the amount of time spent inserting nodes increases. However, shortly after initialization the mobile simulation runs into a scalability issue, suffering performance issues managing growing amounts of nodes rendered in the graph simultaneously.

5.2.1 Results

The results indicate that despite the more limited computing power and longer processing times, the visualization output and available user interactions in the mobile application track very favorably to those of the simulation run on the desktop. The desktop is still a more powerful and flexible platform, and has a distinct development advantage due to the maturity of developer tools available. However, despite shortcomings inherent to the platform, the tablet was able to handle many of the same visualization tasks respectably. I believe with further optimization and tuning more specific to the platform, performance could be brought much closer to that of the desktop version.
Table 5.1. Simulation Time Comparison

<table>
<thead>
<tr>
<th>Simulation Size</th>
<th>Desktop Processing Time (s)</th>
<th>Desktop Speed (Nodes/s)</th>
<th>Mobile Processing Time (s)</th>
<th>Mobile Speed (Nodes/s)</th>
<th>Processing Time Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 Inputs</td>
<td>9.26</td>
<td>5.40</td>
<td>22.63</td>
<td>2.21</td>
<td>+144%</td>
</tr>
<tr>
<td>94 Inputs</td>
<td>39.86</td>
<td>2.36</td>
<td>53.07</td>
<td>1.77</td>
<td>+33%</td>
</tr>
<tr>
<td>221 Inputs</td>
<td>247.43</td>
<td>0.89</td>
<td>484.18</td>
<td>0.46</td>
<td>+96%</td>
</tr>
</tbody>
</table>

Figure 5.2. Mobile Simulation Inputs
Figure 5.3. Mobile Simulation in Progress (Full Screen Zoom)

Figure 5.4. Mobile Simulation Output
Figure 5.5. Mobile Simulation Cluster Display

Figure 5.6. Mobile Simulation Triangulation Display
CHAPTER 6

Conclusions and Future Research

This chapter examines the conclusions drawn from my implementation, the potential for related future work on STREAMIT-Mobile, and possible areas of future research.

6.1 Conclusions

I believe the resulting mobile application produced by migrating the STREAMIT codebase to Android demonstrates that commonly available mobile hardware is currently advanced enough to be incorporated into text stream visualization and analysis, as well as other potential visualization topics. During the development, the limitations I experienced were almost entirely related to the experimental state of the developer tools I was using. Many of these issues could be avoided in future attempts to write native Android applications with Qt by taking advantage of the newly implemented official support.

6.2 Future Work

With the recent release of Qt 5, both the STREAMIT desktop application and STREAMIT-Mobile could benefit from updating from Qt 4.8 to the latest Qt 5.2. This would simplify the code maintenance of each project by removing the third-party dependency on Necessitas and minimizing the code differences between the desktop and mobile applications. Since Qt 5 also includes official support for iOS, upgrading to the
newer Qt release would open up the possibility of supporting both Android and iOS with limited project divergence.

There is some room for performance improvements in the STREAMIT force-directed graphing code and improvements to memory management that could substantially benefit the performance of the mobile version. In addition, there have been updated versions of Google’s Nexus tablets released over the past year. The 2013 Nexus 7 in particular has a noticeably higher resolution and pixel density than the 2012 version this application was developed on. Moving development to a higher resolution device would alleviate the problematic screen clutter that can occur on the 1280x800 display.

Social media lends itself particularly well to the mobile platform, and I believe visualization of Facebook or Twitter content could be an especially appropriate application for STREAMIT-Mobile. The main challenge would be content differentiation of visualized document particles. Keyword lists for social media updates would typically be quite a bit shorter than keyword lists related to something like a news article, and would be more likely to result in sets of discrete clusters instead of varying levels of similarity between contents. This would still give insights into temporal evolution of the visualized text stream, but useful comparison of document contents would likely require a more complex means of computing similarity than the current approach.

Moving away from mobile-specific applications towards more general usage, I believe that applying similar force directed graph visualization to the streaming of complex numerical data could prove to be a useful area of research. My own professional field involves military aircraft countermeasure analysis, which commonly involves thousands
of simulations running across computing nodes in a cluster, continuously generating 
collections of output. Our current analysis techniques only act on a snapshot of the data at 
any given time, but having the ability to monitor simulation results on the fly could 
potentially provide benefits to real time countermeasure analysis.
References


