

George Wright

**Optimising RFB using intelligent  
caches**

Computer Science Tripos

Corpus Christi College

April 20, 2009



# Proforma

Name: **George Wright**  
College: **Corpus Christi College**  
Project Title: **Optimising RFB using intelligent caches**  
Examination: **Computer Science Tripos, May 2009**  
Word Count: **TODO<sup>1</sup>**  
Project Originator: Mr G. Wright  
Supervisor: Mr S. Hay

## Original Aims of the Project

To implement a caching system in the remote framebuffer protocol (RFB) in such a way that areas of the screen could be intelligently determined as suitable for caching, thus leading to less data being transferred over a network connection. The completed project was then benchmarked against the unmodified code without the caching system in place, and the amount of data transferred for set desktop sessions was measured and analysed.

## Work Completed

The RFB protocol was updated to allow for cached regions, and a caching mechanism was implemented in the **TightVNC** source code. Three cache replacement algorithms (First-in, First-out; Least Recently Used and Clock) were implemented and two algorithms for determining the area of the framebuffer suitable for caching were implemented. The work was tested using automated benchmarking tools written specifically for the purpose and the amount of data transferred during the session was measured and analysed.

---

<sup>1</sup>This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

## Special Difficulties

None.

## Declaration

I, George Wright of Corpus Christi College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date



# Contents

1	Introduction	1
2	Preparation	3
3	Implementation	5
4	Evaluation	7
5	Conclusion	9
	Bibliography	11
A	Project Proposal	11

# List of Figures

# Chapter 1

## Introduction

The Remote Framebuffer Protocol (RFB) is a simple client-initiated protocol to allow thin clients to request and receive the necessary data from a server in order to render a remote desktop. It works on the basis of a “framebuffer”; that is, a memory area which stores the state of every pixel in the viewport, and rectangles contained within the bounds of this framebuffer can be requested through well-defined messages in the RFB protocol specification.

The full protocol specification deals with many aspects of remote working, such as input devices, displays and encoding parameters for the actual graphical data. For the purposes of this project, only the display protocol will be modified.

Currently a typical RFB display update process is as follows:

- Client sends `FramebufferUpdateRequest` to the server.
- Server receives and sends `FramebufferUpdate` to the client.

The `FramebufferUpdateRequest` message has five parameters; these define the area of the framebuffer that the client is interested in, specified by its x-position, y-position, width and height. The fifth parameter deals with “incremental” updates; if this is set to `false` then the server will submit all the framebuffer data, not just the sub-rectangles which have changed since the previous update.

Whilst this protocol is careful to only transfer data across the wire which is necessary due to a change (that is, it will not transfer any area of the screen which has not changed since the previous update unless explicitly asked to), there is still the possibility of redundant data being transferred which the client does not need to receive.

The most obvious example of this is in the case of a window being moved around the desktop. In this case, the server will keep resending the contents of the window as the x and y coordinates of the pixel data contained within the window has changed.

This can be avoided with the use of an intelligent client-side cache. In this case, the window's contents could be cached at the client side and when the window is moved the client can simply retrieve the required pixel data from the cache and render it at a different location within the viewport, thus significantly reducing the amount of data transferred.

The second major example of such a cache being useful is to reduce the amount of data transferred due to areas of the screen becoming visible again after a time delay; for example, the desktop background will become visible whenever there are no windows overlapping the desktop, or the "Start menu" will be brought up again when the user clicks on the button.

In this case, the pixel data for the desktop background or the menu would be cached when it first appears in the framebuffer, and on subsequent appearances the data would be retrieved from the cache.

In order to implement such a cache, an algorithm (to be called the *cache policy algorithm*) would be required to determine which areas of the desktop would be suitable for caching. There are two generic ways to implement such an algorithm; the first would be to infer areas of the screen based only on the framebuffer data, here called the *raster method*. This is advantageous as it would not be specific to any particular windowing system. The second is to hook into the windowing system and retrieve window coordinates that way, here called the *vector method*.

The purpose of this project is to determine whether the presence of such a cache would result in a measurable decrease in the amount of data that is transferred over the wire in a typical RFB session. Further objectives are to determine which cache replacement policy result in the fewest cache misses and whether the raster or vector approach to the cache policy algorithm results in the highest measurable decrease in data transferred.

# Chapter 2

# Preparation



# Chapter 3

## Implementation



# Chapter 4

# Evaluation



# Chapter 5

# Conclusion



# Appendix A

## Project Proposal