Analysis of Memory Usage in a LaserJet Printer
by

Novice M.J. Ezell

Submitted to the Department of Electrical Engineering an Computer Science in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Master of Engineering in Computer Science at the Massachusetts Institute of Technology

May 21, 1999
[June 1999]

Author _____
Department of Electrical Engineering and Computer Science
May 21,1999

Certified by _____
M. Frans Kaashoek
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Analysis of Memory Usage in a LaserJet Printer
by
Novice M.J. Ezell

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 1999

In Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Master of Engineering in Computer Science

ABSTRACT

MemUse is a memory usage analysis tool. It was developed to examine how the memory manager subsystem of a LaserJet printer uses memory to print a page. MemUse's main actions include: accept user input to what objects(s) to analyze, create properly formatted strings containing the commands to make the firmware simulator do what is being requested, receive data from the firmware simulator, tally the data, create a data chart and an analysis report.

MemUse was used to analyze the strip size used by the printer. It showed that a different strip size managed memory better that the current strip size. It also uncovered memory issues that were not previously visible to the developer.

Thesis Supervisor: M. Frans Kaashoek
Title: Associate Professor

Acknowledgement

I would like to first give all honor and glory to my Lord and Savior Jesus Christ. The Lord had been my strength and guide throughout this thesis and my years at MIT. Without Him none off this would've been possible. (Philippines 4:13)

I would also like to recognize and thank all those people who supported and helped me through my thesis.

M. Frans Kaashoek, for agreeing to advise me even though I was across the country.

Alan Oyama, for helping me find a thesis topic and taking the time out of his busy Hewlett Packard schedule to guiding me though my thesis work.

Perry Lea, Dan Wilcox, Kevin Hoffman, Honnee Mesa, Paul Rollin, Doug Mellor, Rick Dow and Marion Porter, for taking time out of their busy schedules at Hewlett-Packard to patiently and diligently answer all my questions.

Sharon Whaley, for being a wonderful manager and always providing the resources I needed to get through my work.

I would also like to thank my parents and my husband for their love and support not only throughout my thesis work, but also through my years at MIT.

1 Introduction

In the printer industry, print performance is the driving factor in development. Print performance is measured by print quality, time to print, and physical memory requirements. The memory manager is one of several subsystems in a Hewlett-Packard (HP) LaserJet printer. This subsystem has a significant contribution to the print performance of the printer; it effects time to print and the physical memory requirements. The parameters of the memory manager dictate how memory is allocated and used, which effects print time. The memory manager is also responsible for allocating the physical memory to operate the printer. If there is insufficient physical memory, there may not be enough memory to allocate for a specific print request. If the memory is poorly managed, there may appear to be insufficient physical memory.

This thesis focuses on the development and usage of a memory analysis tool that aids in the evaluation of the memory manager's performance. This tool is called MemUse. MemUse produces an analysis of the memory manager's memory usage for a particular printer under certain conditions. This analysis provides developers with the ability to analyze current parameter setting for this subsystem. With this ability developers are able to examine the effectiveness of current memory manager parameters in current systems and analyze different settings in new products being developed.

The remainder of this section explains the memory manager and how HP is currently measuring its performance. It continues with an explanation of the necessity for a memory analysis tool along with a discussion of a current memory enhancement procedure. It closes with an outline of the rest of the thesis.

1.1 Hewlett Packard Current Approach

The Hewlett-Packard (HP) LaserJet printer is a complex system made up of several subsystems. The performance of the system is reliant on the internal performance of each subsystem. There are few ways available to analyze the internal effect caused by changes made in the individual subsystems during the development of each new product. The

most commonly used procedure is test suite performance. This type of testing consists of printing groups of files. One group may be all text, another may be text and graphic, and still another may be all graphic. These test suites all have a goal of testing the performance of the printer using different aspects of the inner subsystems. Performance is then measured by three factors: time to print, ability to print without running out of memory, and print quality. If adjustments are made within subsystems, the results of suite testing will determine if the changes have had an ill or beneficial effect on the system as a whole.

1.2 The Memory Manager

A LaserJet printer is shipped with a standard amount of external memory. For example the LaserJet 4000 is shipped with 8M bytes of DRAM. The memory manager is responsible for the internal usage of this memory. This external memory needs to be divided between the needs of imaging data for printing and the needs of the rest of the system. Internal memory use needs to be efficient in order to use the external memory to its fullest potential.

A standard letter size page, 8.5" X 11", being printed with 1200 dpi[1] has 10,200 bits per horizontal line. There are 13,200 horizontal lines on the page. If this page is represented in a dot per bit image, it will require 16.83M bytes of memory. In order to keep a LaserJet 4000 printer with a duplexer printing at the specification speed, there must be two duplexed pages internally represented in the system at all times. This requires 64M bytes of memory. This results in approximately a eight to one ration of memory the memory manager has versus the memory it needs to keep the printer printing at specification speed.

Memory is very limited in the system and must be managed efficiently to provide a desired performance. The memory manager must operate using memory strategies to allow the system as a whole to perform competitively and cost efficiently. It would not be

[1] dpi means dots per inch

competitive to have the printer print one page every five minutes, and it would not be cost efficient to sell a LaserJet 4000 standard with 64M bytes of memory.

1.3 The Need For A Memory Analysis Tool

Test suite performance testing allows developers to compare new developments with past products and competitor products. However, this type of testing only exposes the surface of the memory manager's complex nature. Analyzing only how fast the memory manager can process a print request with the memory available to it is not beneficial for long term improvements. There also needs to be an analysis of how the memory manager is managing its memory. If the memory manager is efficiently using the memory provided to it, then the printer will be able to benefit and perform effectively. In fear that modifications will cause internal turmoil, many of this subsystem's parameters are not set to provide the most beneficial memory strategies; the parameters are not necessarily optimal for each product developed. If parameters are changed, suite testing only shows the benefits or hinders it causes to the overall system. It does not investigate the benefits or hinders it caused internally; therefore this inner turmoil could go unnoticed until some time later.

Internal memory information is available through several manual tools. These tools provide memory allocation and de-allocation information for an isolated memory state. This information only explains recent activities; it does not provide a history for overall memory allocations and de-allocations. These manual tools produce pages and pages of output, which the developer must shift through to look for a specific problem. This may require many iterations to find the particular memory state needed for examination. Because of the lack of an easier method or automated tools, the actual internal processes of the memory manager are not examined unless there is a serious problem. Developers are working retroactively instead of proactively, preventing the optimization of parameters for each product; parameters are allowed to remain at an assumed stagnate optimal state. To enable the analysis of the parameters, there needs to be a way to directly analyze the memory manager's memory usage in addition to the printer's print

performance. There needs to be a way to monitor memory usage during the print process.

Memory usage reports produced during the printing process would provide the ability to do an internal analysis of the memory manager's memory usage. When overall print performance is analyzed a defined test suite is printed to see how the printer will respond. Likewise, a specific suite of files sent to the printer would produce the memory usage data needed to analyze the efficiency of memory usage given a certain parameter state. When this suite is determined and the data can be retrieved, the memory manager parameters can be tested for each development. This will allow developers to chose the most beneficial parameter settings for each product being developed.

1.4 Current Memory Enhancement Procedure

Memory Enhancement Technology (MEt) is an internal adjustment developed to enhance the performance of the printer. It provides information to make the print job print faster. MEt examines the actual time needed by the image processor (IP) to process an object for printing. The technology divides any print request into a set of known objects. The objects are those image-processing primitives defined internally in the system that the system uses to construct what is wanted on a page. The idea is that for each type of object there is a finite number of that object that can be used within a given time. Each object is characterized base on the object type, height, width, state, and any object specific parameters. To find the time needed to process a specific object a formula was derived that is a function of these properties. MEt is dived into two parts: IP characterization tool and a system include file.

An IP characterization tool is used to automate the process of obtaining the time required to process each type of object by the image processor. The tool calculates all the necessary data for use in the MEt formulas. The formulas are then evaluated to produce the information needed by MEt during the actual print process. IP characterization tool's main actions are to "accept user input to what object(s) to characterize, create a properly

formatted string containing the commands to make the printer do what is being requested, receive from the printer response to the commands sent, tally the data, calculate the best fit curves, and create an include file for later compilation into the printer firmware". [1]

The include file produced by the IP characterization tool is incorporated into the printer's firmware. This include file provides information for a memory strategy of the memory manager. This strategy allows the memory manager to reduce the memory requirements initially presented by a print job.

1.5 Thesis Outline

This thesis is divided into three additional sections. Section 2 discusses the development of MemUse. Section 3 presents results from the actual use of MemUse. Section 4 gives an overall discussion of the usefulness of MemUse and possible enhancements for the future.

2 Development

MemUse's development goal was to provide an automated method for analyzing memory usage in the memory management subsystem. It is needed to provide developers with information that would allow them to examine the optimality of the memory manager parameter settings.

The memory usage information provided by MemUse is for an internal analysis. By just looking at the amount of external memory provided for printing, a developer does not know how the internal subsystem is performing. If the developer could look at how the memory manager uses memory to prints specific objects, he could start to understand how the internal subsystem was performing.

Because MemUse needs to look at particular objects instead of random information, the IP characterization tool of MEt was seen as a good starting point for developing a tool for per object analysis. The IP characterization tool analyzes how many primitives can be placed on a page in a certain amount of time, and MemUse needs to analyze the different memory consumption levels produced by the number of primitives on a page. Therefore, MemUse's main actions closely follow that of the IP characterization tool: accept user input to what objects(s) to analyze, create properly formatted strings containing the commands to make the firmware simulator do what is being requested, receive data from the firmware simulator, tally the data, create a data chart and an analysis report.

In the remainder of this section I will describe the design and implementation of MemUse and explain the workings of the finished tool.

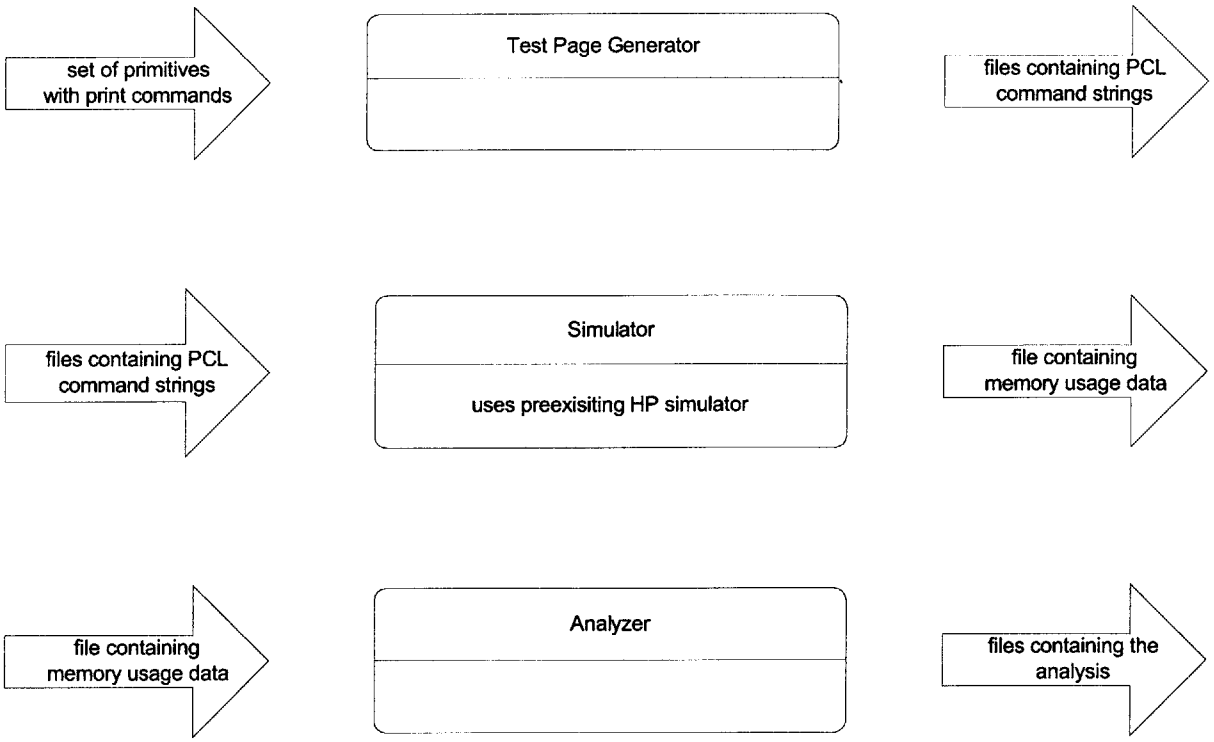


Figure 2-1 Design Overview of MemUse

2.1 Design

MemUse is divided into three major modules (not including the interface): test page generator, simulator, and analyzer. These modules work with the user's input from the user interface to perform the necessary operations. The test page generator generates printable files. Each file contains PCL[2] command strings to print a page. The simulator reads the PCL files and simulates the printing of each file. During the printing simulation, memory usage data is produced from the MemUse code placed in the firmware. This memory usage data is collected by the simulator. The analyzer organizes the memory usage data and computes the analysis report. Figure 2.1 shows a design overview diagram.

MemUse provides a text and graphic user interface.

[2] Printer Control Language (PCL) is a printer language developed by HP.

2.1.1 Test Page Generator

To print a page, the printer must receive the request in specific printer language commands. Current HP printers accept several languages. Two examples are PCL and Postscript. MemUse was designed with PCL as the default. MemUse's design is flexible to handle any language accepted by a HP printer.

The test page generator's job is to produce the language command files representing the selected suite of print requests. The files generated differ based on primitive characteristics: type, size, location, and multiplicity. One file may contain the commands to have one copy of a primitive of size six placed in the lower right hand corner of the page. Another file may contain the commands to have two copies of a primitive of size seven with one in the right hand corner and the other in the center of the page.

With every group of files generated by the test page generator, an additional file is added. This file contains the PCL commands to print a very small period. This file is added as a reference point. Since the period is very small, most of the memory used to print this page is reflective of the memory used for page setup. This file provides a way to give an estimate of the page setup cost that will be inherent in printing all other files created in this group.

Name	Area	Description
CharA	Plain text	A
PoundRaster	Raster object	#
LabelABlack	Image using black fill	A
LabelAPat	Image using pattern fill	A
RuleBlack	Object with black fill	black filled rectangle
RulePat	Object with pattern fill	Pattern filled rectangle
PolygonBlack	Polygon with black fill	Black filled hexagon
PolygonPat	Polygon with pattern fill	Pattern filled hexagon

Figure 2-2 Default Primitive Set

2.1.1.1 MemUse Primitives

MemUse uses language primitives instead of the image-processor primitives used by MEt. The image-processor primitives used by MEt do not traverse the firmware as is needed for the memory usage analysis. Image-processor primitives also operate only in one language. To get the breadth of information needed and to allow the tool to do analysis for different languages, MemUse could not benefit from using the image-processor primitives.

Language primitives are primitives of the printing language being used (e.g. PCL). These primitives are represented by distinct command(s) in the language. MemUse has the default primitive set shown in figure 2.2. The primitives chosen as part of the default set encompass several types of print paths in the firmware. Having a diverse set of primitives will help show the different memory requirements each path demands.

The primitive set used by MemUse is easily modified. A developer can add or delete from the default primitive set when development requirements change. To handle another language, the developer will have to supply his own set of primitives.

2.1.2 Simulator

The main function of this module is to simulate the files created and produce memory usage data during the simulation. This module is a wrapper; it is used to operate an external firmware simulator that actually does the simulation and outputs the memory

usage data. The module has the ability to start, send files to, and exit the firmware simulator. The tool was tested and initially used with the LaserJet 5si simulator; however, the tool is not limited to this type of simulator. The module's ability to control the simulator by system scripts and the simulator's ability to output memory usage data determines the type of simulator that can be used.

This module uses the list of file names created by the test page generator. It uses these names to send each created file to the firmware simulator. To ensure that every file is being printed with the same initial amount of memory, this module causes the firmware simulator to simulate a powercycle before sending each file.

2.1.2.1 Obtaining Memory Usage Data

For the simulator to produce useful memory usage data, it was important to identify important characterizing factors of memory consumption. This investigation included identifying the important areas in the firmware where memory usage information could be retrieved.

Initially ten functions were identified to produce memory information. Sample outputs from these functions were produced. Two of the ten functions produced the same information as two other functions in the set. These two functions were removed and the set was reduced to just eight:

1. **Sum Free Blocks:** This is the sum of free memory that the memory manager calculates being in the system before any memory enhancement are performed.
2. **After Sum Free:** This is the sum of free memory that the memory manager calculates being in the system after memory enhancements are performed.
3. **Number Free Blocks:** This is the actual number of free memory blocks that the memory manager calculates being in the system before memory enhancements are performed.

4. **After Number Free:** This is the actual number of free memory blocks that the memory manager calculates being in the system after memory enhancements are performed.
5. **Max Free Space:** This is the amount of memory in the largest free memory block that the memory manager calculates being in the system before memory enhancements are performed.
6. **After Max Free:** This is the amount of memory in the largest free memory block that the memory manager calculates being in the system after memory enhancements are performed.
7. **Total Page Memory:** This is the amount of memory needed to compose the page as calculated by the memory manager before memory enhancements are performed.
8. **After Page Mem:** This is the amount of memory needed to compose the page as calculated by the memory manager after memory enhancements are performed.

MemUse is not concerned with the memory requirements initially presented to the memory manager, because it is analyzing how the memory manager is using its memory strategies to efficiently use memory. Since the memory enhancements are a part of memory strategies used by the memory manager, it is important for MemUse to use data gather after the enhancement are performed. It is also important for MemUse to look at overall memory affects and not particular memory sums. Particular memory sums, like 'After Page Mem', can be misleading when analyzing memory consumption. 'After Page Mem' leaves out memory consumed by processes that may not be directly attached to the information being printed on the page.

As a result of these requirement for MemUse, 'After Sum Free' is the only information used for the analysis report. 'After Sum Free' captures memory usage after all memory enhancements and it captures the overall memory level effected by printing a particular job.

```
288:PS:**Memory Output**
Sum Free Blocks: 10642034
After Sum Free: 10643723
Number Free Blocks: 11
After Number Free: 16
Max Free Space: 7585744
After Max Free: 7585767
Total Page Memory: 36700
After Page Mem: 32600
288:PS:**Memory Output**
Sum Free Blocks: 10642096
After Sum Free: 10643744
Number Free Blocks: 17
After Number Free: 16
Max Free Space: 7585776
After Max Free: 7585776
Total Page Memory: 36720
After Page Mem: 32624
```

Figure 2-3 Example of data output for two files

All eight types of information will be present in the data chart created by MemUse. Even though MemUse is not using the additional information for its analysis, the additional information is important for memory investigations in general. Since MemUse is a memory analysis tool, it should provide any information that could help the developer analyze memory even if it will not directly analyze the data.

Figure 2.3 shows an example of the data output produced by the simulator.

2.1.3 Analyzer

After simulation the analyzer retrieves all the data outputted by the simulator, organizes it into a data chart, and creates an analysis report. The production of the chart is simple: the data is converted into a format for a spreadsheet program, which can then graph the results. The data chart contains all the memory usage data produced, unlike the analysis report, which only focuses on one aspect of this data (as disclosed next).

2.1.3.1 Analysis Report

It was decided to only use 'After Sum Free' for the analysis report. This information captures the point of memory usage at its final stage in the print process. All memory enhancements have been performed and this memory state reflects how much memory is being used to print the job. Using this data for each primitive, MemUse produces the analysis report.

The analyzer examines how memory consumption compares to the number of primitives being printed on the page. If any print job is broken into primitives, it is important to understand what type of memory demands each primitive type will have based on the number of each primitive type present. After analyzing the primitives separately, the analyzer looks at all the primitives together. This computation of all the primitives gives an average usage for the primitive set. This can be helpful for comparative analysis done across different copy number ranges.

Given that:

M = initial amount of memory in the system
F_p = number of files simulated for primitive p
A_p = average memory used per copy of primitive p
A = the average memory used per copy over all primitives
T_p = percent of total memory A_p represents
T = percent of the total memory A represents
m_i = memory used by file i
c_i = the number of primitives file i will print on a page

The analyzer uses the following memory usage analysis for each primitive p. The memory used by each copy of primitive p on a page can be calculated by dividing the memory used to print a file containing p by the number of p's in the file. The average memory used overall to print one primitive p on a page can be calculated by summing up the memory used per copy for each file containing p and diving by the number of file:

$$A_p = \frac{\sum_i \frac{m_i}{c_i}}{F_p}$$

The average memory used for primitive p, A_p, represents some fraction of the total memory initially available in the system. The fraction can be calculated by dividing A_p by the initial total memory available. This fraction can then be represented as the percent of the total memory a copy of the primitive consumes:

$$T_p = \frac{A_p}{M} \times 100$$

The analyzer uses the following memory usage analysis for a combination of all primitives. The average memory used to print any primitive in the primitive set can be calculated by summing all A_p's for all the primitives in the primitive set and dividing this sum by the number of files generated for the entire primitive set:

$$A = \sum_p \frac{A_p}{F_p}$$

The average memory used to print any primitive in the primitive set, A, represents some fraction of the total memory initially available in the system. The fraction can be calculated by dividing A by the initial total memory available. This fraction can then be represented as the percent of the total memory a copy of any primitives consume:

$$T = \frac{A}{M} \times 100$$

Figure 2.4 shows an example of an analysis report. In addition to the information described above, the analysis report also shows the maximum and minimum memory consumed for each primitive. This is provided to show when the maximum and minimum memory usage levels are reached. As shown in figure 2.4 the maximum and minimum memory usage levels do not always happen with the maximum and minimum number of copies. The range of copies used to produce the report is displayed at the top of the report.

```
ANALYSIS REPORT
Copy range(0-3000 by 100)


Average memory used to print a copy of any primitive: 319,840.31
Average % of total memory used to print a copy of any primitive: 1.99%


**Information per primitive**
init:
Size---> 100
Maximum memory used---------------------------->   2,250,144  (for 1 copies)
Minimum memory used---------------------------->   2,250,144  (for 1 copies)
Average (memory used/copy)--------------------->   2,250,144
Average % of total memory (memory used/copy)--->   14.06%

charA:
Size---> 8
Maximum memory used---------------------------->   2,358,432  (for 3000 copies)
Minimum memory used---------------------------->   2,282,848  (for 100 copies)
Average (memory used/copy)--------------------->   78,390.63
Average % of total memory (memory used/copy)--->   0.48%

ruleBlack:
Size---> 52,252
Maximum memory used---------------------------->   2,274,656  (for 2700 copies)
Minimum memory used---------------------------->   2,214,464  (for 100 copies)
Average (memory used/copy)--------------------->   76,044.13
Average % of total memory (memory used/copy)--->   0.47%
...
```

Figure 2-4 Sample Analysis Report

2.1.4 Using the Tool

Before using the tool the developer has to set the tool options located in the options menu of the tool. After the options are set the tool can be used in one of four modes:

1. **Auto:** The tool performs an analysis of the entire primitive set using the tool options.
2. **Primitive Default:** The developer is prompted to choose one primitive to use from the primitive set. The analysis is done on the chosen primitive using the tool options.
3. **Primitive With User Information:** The developer is prompted to choose one primitive to use from the primitive set. The user is also asked to provide specific copy numbers and specific locations for all the copies. The analysis is done on the chosen primitive with the copy numbers and locations given.

4. **File Input:** The developer is prompted to enter the names of the file(s) that are to be analyzed. In this case the tool will generate no files. The analysis produced will be on each file as a whole. Primitive analysis will not be performed.

For 'auto' and 'primitive default' modes, locations of the primitives are chosen randomly and the copy number range is determined by the tool options. 'File Input' mode allows a high level view of memory consumption for a particular job. Total page memory consumption information is produce; a primitive analysis is not performed in this mode.

When a mode is selected and necessary user input is entered, the developer can operate the tool in 7 different way. While the tool is running, information will be displayed in the shell that executed MemUse. This information allows the user to monitor the progress of MemUse while it is working.

The seven modes of operation:

1. **Run complete:** this mode will run the complete analysis. The data chart and the analysis report will be produced. *(This mode will display information signaling the start and finish of generation, simulation, and analysis. It will also display information during simulation. During simulation information will be displayed when the tool is turned on or off, when a powercycles occur and when files are being sent.)*
2. **Generate Filenames:** this mode will create the names of the files that would be created under the current conditions. The files will not be created. This is used if the specified group of files already exist. Filename generation is needed when simulation or analysis is done without test page generation. *(This mode will display information signaling the start and finish of filename generation.)*
3. **Generate test pages:** this mode will create the files. *(This mode will display information signaling the start and finish of test page generation.)*
4. **Simulate test pages:** this mode will simulate the files. It will receive the names of the files to simulate either from test page generation or filename generation; therefore, to use this the user must first either generate the files or the filenames. *(This will display information signaling the start and finish of simulation. It will also display information during*

simulation. During simulation information will be displayed when the tool is turned on or off, when a powercycles occur and when files are being sent.)

5. **Create analysis report:** this mode will analyze the data in the memory data file, producing a data chart and an analysis report. It will receive the names of the files to use in the analysis either from test page generation or filename generation; therefore to use this mode the user must first either generate the files or the filenames. Simulation before analysis is not required. If simulation is not done before analysis, it is assumed that the simulation output memory data file exists and has the data for the filenames it is using. *(This will display information signaling the start and finish of analysis.)*
6. **Show memory data chart:** this mode will display the data chart.
7. **Show analysis report:** this mode will display the analysis report.

Figure 2.5 gives an example of output produced from running MemUse with 'Run Complete'.

2.2 Implementation

MemUse was developed using Java. The implementation of MemUse's design resulted in each module becoming its own class: TestPageGenerator, Simulator, and Analyzer. The user interface was developed in graphical and text form. The text interface is also its own class: MemUse. The graphical interface is a combination of several classes with the main class being XMemUse.

To help the information follow between the interfaces and the modules, several smaller classes where developed. Options is one of these helper classes. Options is used to hold the options the user sets in the tool's option menu[3]. This helper class is used a lot by all three modules. Other helper classes help keep the primitive information organized, provide language specific information, and provides useful functionality for internal operations. Overall there are 7130 lines of Java code for the MemUse tool.

[3] An example of MemUse's options tool menu is in appendix D.

```
START GENERATION
GENERATION COMPLETE
START SIMULATION
<Last Ready Number>0
...
<Last Line>59:DISPLAY: READY
New READY line number:59
**MEMUSE_ON**
...
<Last Line>63:DISPLAY: READY
New READY line number:63
**FIRST PART POWERCYCLE**
<Last Line>64:DISPLAY: PROCESSING JOB
<Last Ready Number>63
<Last Line>65:DISPLAY: READY
New READY line number:65
**SECOND PART POWERCYCLE**
<Last Line>69:DISPLAY: PROCESSING JOB
<Last Ready Number>65
<Last Line>70:DISPLAY: READY
New READY line number:70
**SENDING FILE**: sh ../SIM/mobux.input ../TMP/init_100_1_1.pcl
<Last Line>78:DISPLAY: PROCESSING JOB
<Last Ready Number>70
<Last Line>79:DISPLAY: READY
New READY line number:79
**FINISHED FILE**:sh ../SIM/mobux.input ../TMP/init_100_1_1.pcl
**FIRST PART POWERCYCLE**
<Last Line>80:DISPLAY: PROCESSING JOB
<Last Ready Number>79
<Last Line>81:DISPLAY: READY
New READY line number:81
**SECOND PART POWERCYCLE**
...
FINISHED SIMULATION
START ANALYSIS
FINISHED ANALYSIS
```

Figure 2-5 Sample shell output produced by MemUse as it is running

When implementing MemUse's design it was important to make it language, primitive set, and firmware simulator independent. This resulted in a class called UserDefined, two data files, three shell scripts and three control files. The UserDefined class has all static public methods that are maintained by the user. These methods provide a way for the tool to get accurate language dependent information. For example, this file contains a method that returns the language specific command to position the printing cursor in the desired location on a page.

The two data files are used by the TestPageGenerator class. These two files contain the primitive set and the sizes to use with each primitive. These files are briefly discussed in section 2.2.1 and the specific format is briefly discussed in appendix C.

The three shell scripts and three control files are used to control the firmware simulator. The information in these scripts should be able to turn the simulator on and off, to turn MemUse abilities on and off in the simulator, to send a file to the simulator, and be able to cause a powercycle in the simulator. These scripts are briefly discussed in appendix C.

2.2.1 TestPageGenerator

The TestPageGenerator class is the heart of the system. It must produce files in the correct format for the firmware simulator to work properly. It also must produce the correct sequence of files requested for analysis.

To allow MemUse to be independent of the primitive set and printer language, TestPageGenerator is dependent on several files and a helper class. The file names are set in the tool options menu and the helper class must be updated when different languages are used.

1. *Primitive file* is needed to retrieve primitive information. The primitive file has the primitive type and the language specific commands need to print the primitive. This file contains all the primitives that the tool will be able to use; therefore the tool is dependent on the user putting the proper information in this file
2. *Size file* provides the different sizes to use when printing the primitives. All primitives in the primitive file must be represented in the size file.
3. The *class UserDefined* provides language dependent information. The user must update this file when he uses other languages besides PCL.

TestPageGenerator class has four constructors which dictate how the public methods work. These four possible constructions result in the tools four operating modes.[4]

1. Given *only an Options object*: files for each primitive in the primitive file will be produced using the sizes in the size file.
2. Given an *Options object and one primitive type*: files for the one primitive will be produced using each size that corresponds to it in the size file.

[4] The four modes where discussed in section 2.1.4: 'Auto', 'Primitive Default', 'Primitive With User Information', and 'File Input'.

3. Given an *Options object, a primitive type, a location list and a copy list*: files for the one primitive will be produced using the location and copy lists given. The files will be produced for each size that corresponds to the primitive in the size file.
4. Given *a list of filenames*: the list of filenames given will be returned.

This class has two public methods: generatePages() and generateFilenames(). GenerateFilenames() uses the same type of algorithm as generatePages() excepted actual files are not created; it just returns the names of the files that would've been created. The TestPageGenerator creates the file in three parts. To do this it uses three private methods: printFileBegin(), printFileEnd(), and printFileBody(). PrintFileBegin() and printFileEnd() are hard coded to the appropriate HP specified codes needed to start and end a page. PrintFileBody() uses the primitive information, user input, and tool options to print the body of the file.

Algorithm 2.1 is a general version of the algorithm used in generatePages() assuming construtor 1 above was used. When constructor 1 is used an array of Primitive objects is created from the primitive file. Each Primitive objects knows its type and what commands to use to print itself on a page. A copy number list is also created from information given in the Options object . GeneratePages() uses this primitive list to iterate over all the primitives in the primitive set.

First generatePages() constructs a sizelist for each primitive. It combines these lists into a list. Then generatePages() creates the files requested. For every primitive, files are produced with the primitive being each size in the size list for that primitive. The files produced depend on the numbers in the copy number list. The number of files produced for each copy number is controlled by the repeat number. If the repeat number is two, then two files would be produced for each copy number in the copy list. GeneratePages() creates the filename for the file, creates the file in the file system, and then outputs the necessary information into the file. After the information is placed in the file, the file is closed and the name is added to a filename list, which is later used by the Simulator and

Analyzer class. The filenames used for the file have the format: 'primitive type'_'size used'_'copy number used'_'the current repeat number'

2.2.2 Simulator

This class is used to operate an external firmware simulator. With the use of shell scripts it has the ability to start, send files to, and exit the external simulator. These scripts are discussed in appendix D. For the external simulator to produce the memory output needed, code must be added to the external simulator code. The following pieces of code were added to the default external simulator, LaserJet 5si:

1. Control code: Code was added to dynamically enable or disable MemUse's code in the simulator.
2. Output restriction code: It is recommended that MemUse output be the only output being produced by the simulator while MemUse is in use. Therefore, all other output code in the simulator is disabled when MemUse control code is active.
3. Information output code: The functions needed for outputting the memory data used be by MemUse was added. This code is only active when the MemUse control code is active.

```
class TestPageGenerator
  TestPageGenerator(o:options)

  generatePages()
    for p=0; p<primSet.length; p++
      list = new list[]
      primSizeList = 0
      foreach s:size in size file that is classified as a size for primitive primSet[p]
        list.addElement(s)
      sizeList[p][primSizeList++]=list
    for p=0; p<primSet.length; p++
      foreach s:size in list sizeList[p] do
        foreach c:copy number in copyList do
          foreach r=0; r<repeat; r++
            filename= primSet[p].type()+"_"+s.toString()+"_"+c.toString()+"_"+r.toString()+".pcl"
            output = createFile(filename)
            printFileBegin(output)
            printFileBody(output,primSet[p],s,c)
            printFileEnd(output)
            output.close()
            filesCreated.addElement(filename)
    return filesCreated
```

- **options** is an object that contains all the options set by the user of the tool. This includes the size file, primitive file, and the information to construct the copy list.
- **primSet** is the set of primitives being used. Each primitive contains the PCL command to print the primitive on the page. The primitive set was constructed using a primitive file specified by the user.
- **sizeList** is a list of lists. This list contains the list of sizes for each primitive in the primitive set. The number of lists in sizeList should be equal to the number of primitives in primSet.
- **primSizeList** is a counter used as an index to specify which size list is being added.
- **copyList** is a list of numbers representing how many copies of a primitive are to appear on a page.
- **repeat** is the number of files the user wants with the same size and copy characteristics. Locations are different.
- **printFileBegin** puts the PJL[5] set up commands at the beginning of the file being created.
- **printFileBody** puts the PCL commands in the file being created. The PCL commands will cause the specified primitive *primSet[p]* to be printed with the specified size *s*. The commands will also cause *c* copies of this primitive to appear on the page. This function randomly generates all locations used to place the primitive on the page.
- **printFileEnd** puts the PCL commands needed to end a print job at the end of the file being created.
- **filesCreated** keeps track of all the names of the files created by generatePages()

Algorithm 2-1 Pseudo code for generatePages()

[5] Printer Job language(PJL) is for job level printer control.

3 Analyzing A Memory Manager Parameter

I chose to examine one memory manager parameter: strip size. The strip size is one of the less complicated parameters (but important) in the subsystem. Examining this parameter did not require me to learn about multiple dependencies in the system and I did not need extensive training to understand how to modify the parameter in the system. I did have to learn certain restrictions and constraints about this parameter, but the information was easy to understand and independent of most other parameters in the system.

The strip size determines how the printer will divide a page up to be printed. How the strips are sized effects how memory strategies will perform in the subsystem. The current strip size has been the same for many years. Developers do not know why the size was chosen and they are not clear whether another size would be better. Using test suite performance tests, a developer can see how the size can effect the speed and ability of the printer to print particular jobs. The current strip size has been doing well in these suite tests.

What if the strip size currently being used is not the best? What if the memory manager could work more efficiently with another size and other optimization would work better? My goal was to use MemUse to look at the strip size possibilities. MemUse would provide the information that may suggest another size that may be more effective for the memory manager. With this analysis, developers could then take advantage of the parameter's possibilities.

3.1 Running MemUse

MemUse was run using the LaserJet 5si simulator, and the default PCL primitive set. All runs were done using auto mode; therefore the entire primitive set was examined. After running MemUse for the first time, the files generated where saved and used for all

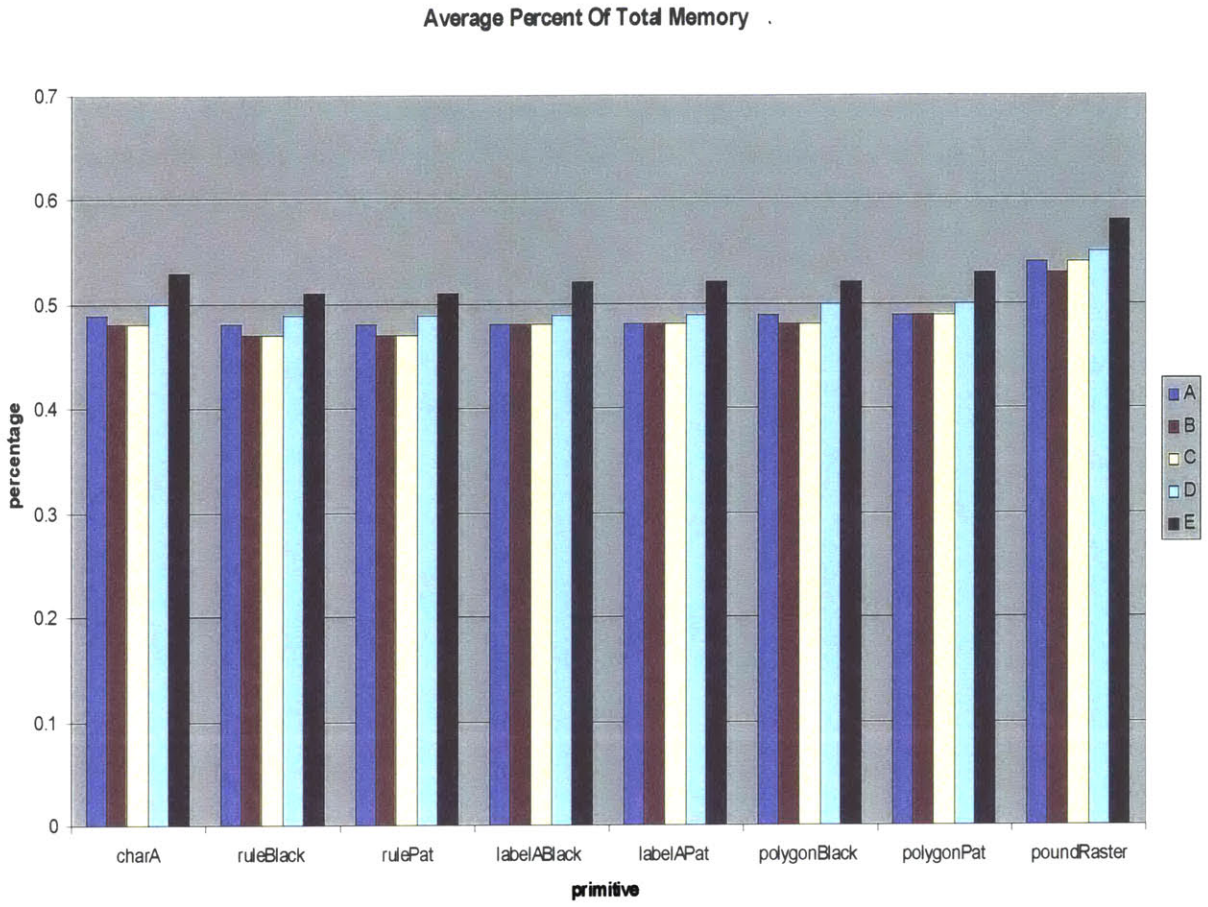


Figure 3-1 Results from running MemUse with different strip sizes: A, B, C, and D

subsequent runs[6]. This allowed all runs to be performed with the primitives in the same locations.

Each run used a copy number range of 0-3000 copies using an increment of 100. Four runs where done with four different strip sizes: A, B, C, and D. Strip size C is the current default strip size. The other strip sizes are increments or decrements of the default with E being the largest size and A being the smallest.

[6] The first run was done using 'Run Complete'. Subsequent runs were done by using the sequence: 'Generate Filenames', Simulate test pages', and 'Create analysis report'.

3.2 Results Produced

Appendix A contains the analysis reports produced by the four runs. Figure 3.1 shows a graph of one aspect of the analysis report. This graph plots the percentage of the total memory a primitive consumes. (Tp was discussed in section 2.1.3.1). Each grouping on the graph represents a primitive, and the four bars represent the percentages calculated when using each of the strip size.

Looking at the graph in figure 3.1, you can see that strip size C and B are very close in memory usage. Strip size B uses less memory or the same memory as strip size C in all cases. In the cases where strip size B uses less memory, the difference in the memory usage is small.

To get a better understanding of this difference the 'After Sum Free' information in the data charts were graphed for each primitive. These graphs show how much memory is left as the current file containing x copies of a primitive is about to be printed. Each graph looks at one primitive containing a line for each different strip size run. The graphs for all the primitives are in appendix B. Graphs for charA, polygonBlack and poundRaster are also in figures 3.2-3.4.

Overall graphs in figures 3.2-3.4 show that strip size B is generally better than strip size C. In figure 3.2 and 3.3, strip size B always uses less memory than strip size C. However, in figure 3.4 there is one point in which strip size C does better than strip size B. This point occurs at 600 primitives. This type of change can also be seen in the labelAPat graph in appendix B. In this case, strip size C is better at 900 primitives.

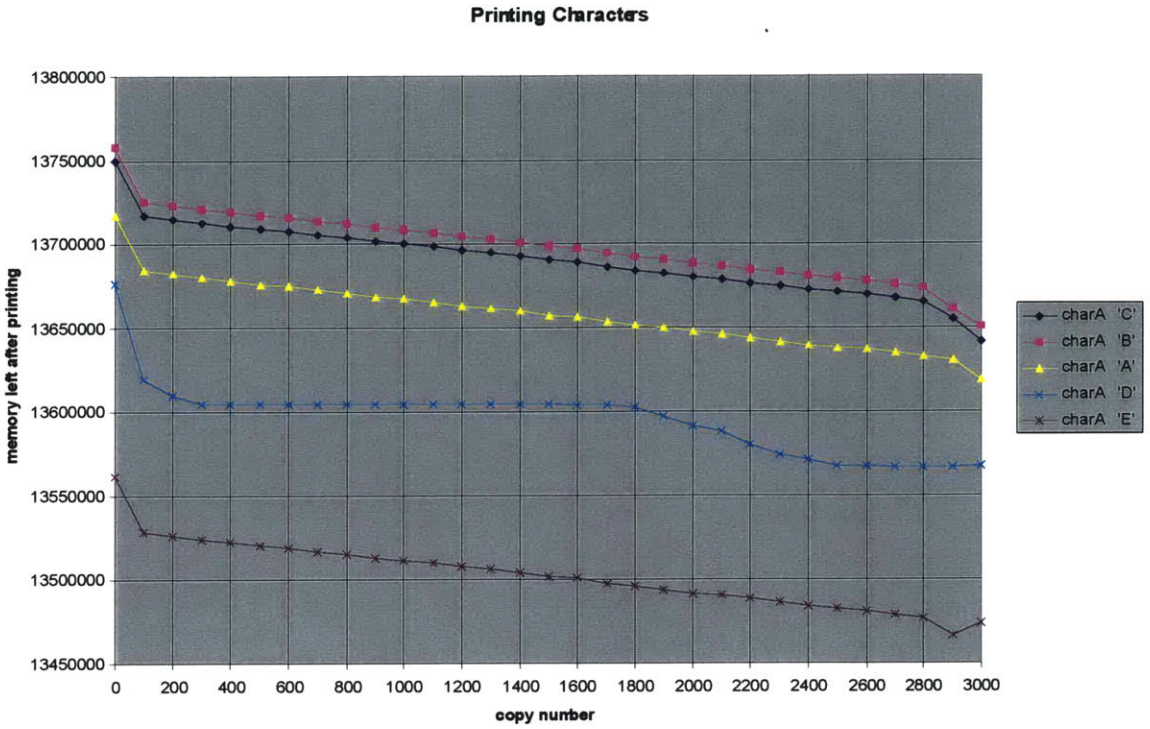


Figure 3-2 Results for print a character with different strip sizes

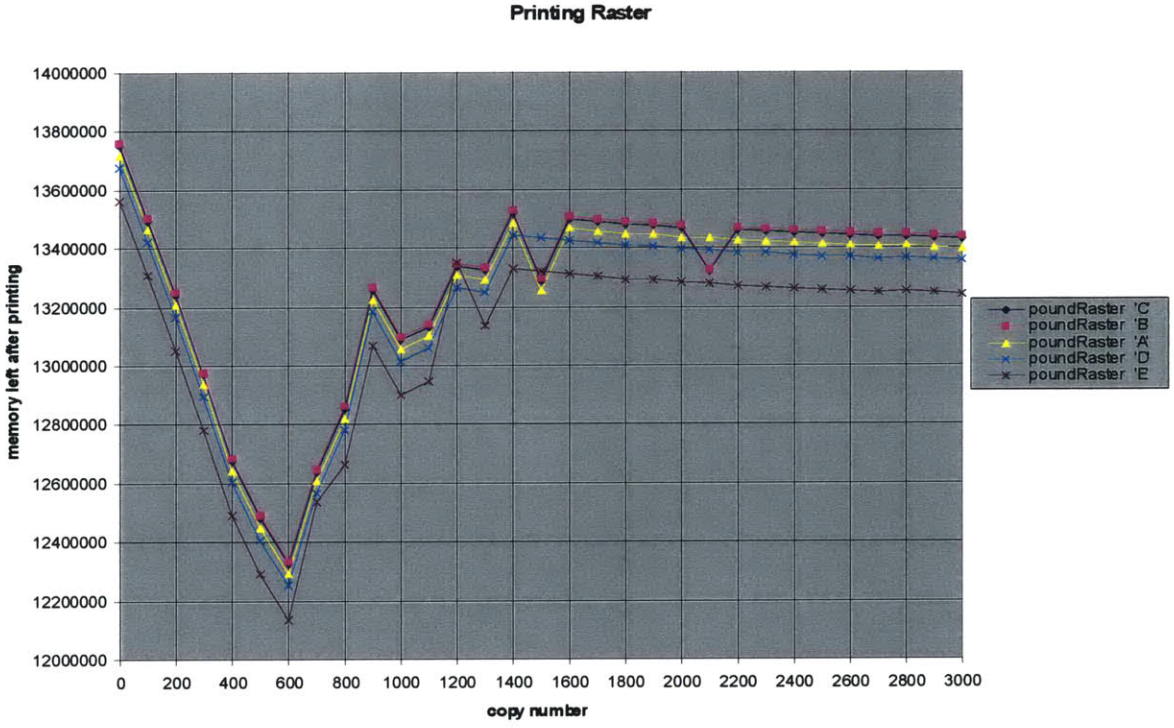


Figure 3-2 Results for printing raster with different strip sizes

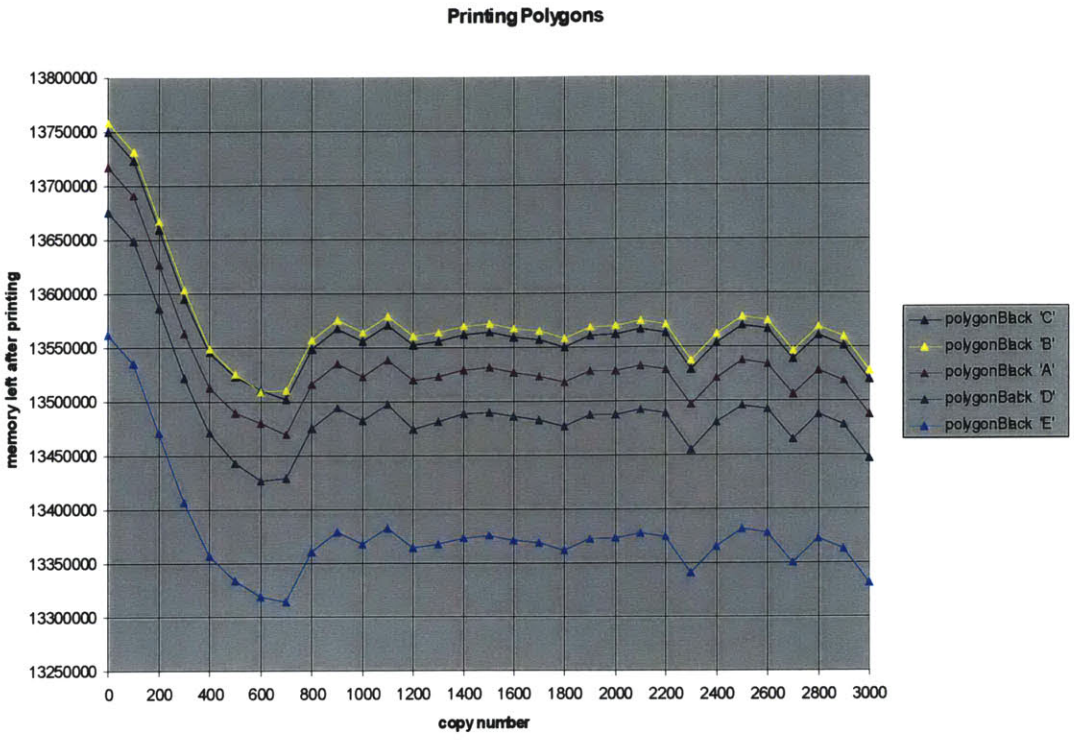


Figure 3-4 Results for black filed polygon with different strip sizes

3.3 Analysis

Overall, strip size B is better than strip size C. There are two occasion where strip size C does beat strip size B as discusses in section 3.2 above. Since the overall gain in memory is small, these two occurrences may be a valid reason to keep the original strip size. The deciding factor would be test suite performance testing. If changing the strip size has an adverse effect on the system's print performance, it may be a positive trade off to give up the small memory gain. If this trade off is made, developers can revisit this tradeoff in future development to see if a larger memory gain has been accomplished by a result of other optimizations.

Besides revealing that the current strip size is not best, this analysis uncovered other interesting details in the memory system that developers will need to investigate. Figure 3.3 shows the results as it pertains to the raster primitive. In this graph the lines for strip sizes C and B both take an unexpected dip at 1500 and 2300 primitives. One dip brings

the memory usage level higher than that of all other strip sizes. This difference in memory usage is larger than the differences found between strip size C and B since C and B where overall noticeably better than the other strip sizes. Why is this dip occurring? All other dips in the graph happen with all strip sizes. Why are these two points any different? These questions will lead to an investigation of what the raster path may be doing to cause memory usage to increase at these points.

Figure 3.3 also shows that the raster path reaches its maximum at 600 primitives. This parameter could be examined to see if increasing or decreasing the number of primitives at which this maximum occurs would be beneficial to the system as a whole.

Overall MemUse has demonstrated its ability to help in the analysis of memory management. This particular analysis of strip size illustrated how MemUse uncovers internal memory issues that were not previously visible to the developer. A good example of this uncovering was the dips found in the raster graph.

4 MemUse's Future

In general, several enhances can be made to MemUse's current abilities. MemUse should have a standard set of primitives for each language compatible with the HP LaserJet printers. It would also be beneficial if MemUse could produce its own graphs given any category selected from the data chart.

Ideally it would also be good for MemUse to actually be able to break up a given page of information in to primitives and produce a primitive analysis. This would be a very useful in conjunction with test suite performance testing. When a job runs out of memory, MemUse could be used to locate the problem areas.

Overall MemUse should be a complete memory testing tool. Currently MemUse can be used to examine parameter changes or primitive performance. In the future it should be able to analyze other aspects of the memory manager as well. There are a variety of analyses that developers can use to examine the internal memory management performance. For example, an analysis report for memory fragmentation and memory enhancement performance would be very useful.

5 Appendix A

This appendix contains the analysis reports produced from four runs. Each run was performed on the entire primitive set. Each run differs based on the strip size being used by the printer. The reports appear in increasing order of strip size: A, B, C, and then D.

Strip Size A

```
                                ANALYSIS REPORT
Copy range(0-3000 by 100)


Average memory used to print a copy of any primtive: 324,480.87
Average % of total memory used to print a copy of any primitive: 2.02%


**Information per primitive**
init:
Size---> 100
Maximum memory used---------------------------->   2,282,912  (for 1
copies)
Minimum memory used---------------------------->   2,282,912  (for 1
copies)
Average (memory used/copy)--------------------->   2,282,912
Average % of total memory (memory used/copy)--->   14.26%


charA:
Size---> 8
Maximum memory used---------------------------->   2,381,120  (for 3000
copies)
Minimum memory used---------------------------->   2,315,616  (for 100
copies)
Average (memory used/copy)--------------------->   79,515.36
Average % of total memory (memory used/copy)--->   0.49%


ruleBlack:
Size---> 52,252
Maximum memory used---------------------------->   2,307,424  (for 3000
copies)
Minimum memory used---------------------------->   2,247,232  (for 100
copies)
Average (memory used/copy)--------------------->   77,169.26
Average % of total memory (memory used/copy)--->   0.48%


rulePat:
Size---> 52,252
Maximum memory used---------------------------->   2,330,272  (for 3000
copies)
Minimum memory used---------------------------->   2,248,432  (for 100
copies)
Average (memory used/copy)--------------------->   77,216.6
Average % of total memory (memory used/copy)--->   0.48%


labelABlack:
Size---> 3
Maximum memory used---------------------------->   2,636,304  (for 2700
copies)
Minimum memory used---------------------------->   2,274,672  (for 100
copies)
```

```
Average (memory used/copy)---------------------->   78,277.83
Average % of total memory (memory used/copy)--->   0.48%


labelAPat:
Size---> 3
Maximum memory used----------------------------->   2,520,944  (for 2900
copies)
Minimum memory used----------------------------->   2,277,264  (for 100
copies)
Average (memory used/copy)---------------------->   78,321.89
Average % of total memory (memory used/copy)--->   0.48%


polygonBlack:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used----------------------------->   2,530,016  (for 700
copies)
Minimum memory used----------------------------->   2,309,792  (for 100
copies)
Average (memory used/copy)---------------------->   79,451.06
Average % of total memory (memory used/copy)--->   0.49%


polygonPat:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used----------------------------->   3,308,320  (for 3000
copies)
Minimum memory used----------------------------->   2,315,744  (for 100
copies)
Average (memory used/copy)---------------------->   79,921.2
Average % of total memory (memory used/copy)--->   0.49%


poundRaster:
Size---> 0
Maximum memory used----------------------------->   3,707,760  (for 600
copies)
Minimum memory used----------------------------->   2,514,896  (for 1400
copies)
Average (memory used/copy)---------------------->   87,542.6
Average % of total memory (memory used/copy)--->   0.54%
```

5.1 Strip Size B

```
                                ANALYSIS REPORT
Copy range(0-3000 by 100)


Average memory used to print a copy of any primtive: 318,680.33
Average % of total memory used to print a copy of any primitive: 1.99%


**Information per primitive**
init:
Size---> 100
Maximum memory used---------------------------->   2,241,952  (for 1
copies)
Minimum memory used---------------------------->   2,241,952  (for 1
copies)
Average (memory used/copy)--------------------->   2,241,952
Average % of total memory (memory used/copy)--->   14.01%


charA:
Size---> 8
Maximum memory used---------------------------->   2,349,328  (for 3000
copies)
Minimum memory used---------------------------->   2,274,656  (for 100
copies)
Average (memory used/copy)--------------------->   78,109.5
Average % of total memory (memory used/copy)--->   0.48%


ruleBlack:
Size---> 52,252
Maximum memory used---------------------------->   2,266,464  (for 3000
copies)
Minimum memory used---------------------------->   2,206,272  (for 100
copies)
Average (memory used/copy)--------------------->   75,763
Average % of total memory (memory used/copy)--->   0.47%


rulePat:
Size---> 52,252
Maximum memory used---------------------------->   2,288,848  (for 3000
copies)
Minimum memory used---------------------------->   2,207,472  (for 100
copies)
Average (memory used/copy)--------------------->   75,810.39
Average % of total memory (memory used/copy)--->   0.47%


labelABlack:
Size---> 3
Maximum memory used---------------------------->   2,595,344  (for 2700
copies)
```

```
Minimum memory used---------------------------->   2,233,712  (for 100
copies)
Average (memory used/copy)--------------------->   76,872
Average % of total memory (memory used/copy)--->   0.48%


labelAPat:
Size---> 3
Maximum memory used---------------------------->   2,479,984  (for 2900
copies)
Minimum memory used---------------------------->   2,236,304  (for 100
copies)
Average (memory used/copy)--------------------->   76,915.93
Average % of total memory (memory used/copy)--->   0.48%


polygonBlack:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   2,490,864  (for 600
copies)
Minimum memory used---------------------------->   2,268,832  (for 100
copies)
Average (memory used/copy)--------------------->   78,046.23
Average % of total memory (memory used/copy)--->   0.48%


polygonPat:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   3,267,360  (for 3000
copies)
Minimum memory used---------------------------->   2,274,784  (for 100
copies)
Average (memory used/copy)--------------------->   78,515.3
Average % of total memory (memory used/copy)--->   0.49%


poundRaster:
Size---> 0
Maximum memory used---------------------------->   3,666,800  (for 600
copies)
Minimum memory used---------------------------->   2,473,936  (for 1400
copies)
Average (memory used/copy)--------------------->   86,138.66
Average % of total memory (memory used/copy)--->   0.53%
```

5.2 Strip Size C

```
                                ANALYSIS REPORT
Copy range(0-3000 by 100)


Average memory used to print a copy of any primtive: 319,840.31
Average % of total memory used to print a copy of any primitive: 1.99%


**Information per primitive**
init:
Size---> 100
Maximum memory used---------------------------->   2,250,144  (for 1
copies)
Minimum memory used---------------------------->   2,250,144  (for 1
copies)
Average (memory used/copy)--------------------->   2,250,144
Average % of total memory (memory used/copy)--->   14.06%


charA:
Size---> 8
Maximum memory used---------------------------->   2,358,432  (for 3000
copies)
Minimum memory used---------------------------->   2,282,848  (for 100
copies)
Average (memory used/copy)--------------------->   78,390.63
Average % of total memory (memory used/copy)--->   0.48%


ruleBlack:
Size---> 52,252
Maximum memory used---------------------------->   2,274,656  (for 3000
copies)
Minimum memory used---------------------------->   2,214,464  (for 100
copies)
Average (memory used/copy)--------------------->   76,044.13
Average % of total memory (memory used/copy)--->   0.47%


rulePat:
Size---> 52,252
Maximum memory used---------------------------->   2,297,072  (for 3000
copies)
Minimum memory used---------------------------->   2,215,664  (for 100
copies)
Average (memory used/copy)--------------------->   76,091.6
Average % of total memory (memory used/copy)--->   0.47%


labelABlack:
Size---> 3
Maximum memory used---------------------------->   2,603,536  (for 2700
copies)
```

```
Minimum memory used---------------------------->   2,241,904  (for 100
copies)
Average (memory used/copy)--------------------->   77,152.86
Average % of total memory (memory used/copy)--->   0.48%


labelAPat:
Size---> 3
Maximum memory used---------------------------->   2,488,176  (for 2900
copies)
Minimum memory used---------------------------->   2,244,496  (for 100
copies)
Average (memory used/copy)--------------------->   77,196.76
Average % of total memory (memory used/copy)--->   0.48%


polygonBlack:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   2,497,248  (for 700
copies)
Minimum memory used---------------------------->   2,277,024  (for 100
copies)
Average (memory used/copy)--------------------->   78,326.36
Average % of total memory (memory used/copy)--->   0.48%


polygonPat:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   3,275,552  (for 3000
copies)
Minimum memory used---------------------------->   2,282,976  (for 100
copies)
Average (memory used/copy)--------------------->   78,796.53
Average % of total memory (memory used/copy)--->   0.49%


poundRaster:
Size---> 0
Maximum memory used---------------------------->   3,674,992  (for 600
copies)
Minimum memory used---------------------------->   2,482,128  (for 1400
copies)
Average (memory used/copy)--------------------->   86,419.96
Average % of total memory (memory used/copy)--->   0.54%
```

5.3 Strip Size D

```
                                ANALYSIS REPORT
Copy range(0-3000 by 100)


Average memory used to print a copy of any primtive: 330,281.62
Average % of total memory used to print a copy of any primitive: 2.06%


**Information per primitive**
init:
Size---> 100
Maximum memory used---------------------------->   2,323,872  (for 1
copies)
Minimum memory used---------------------------->   2,323,872  (for 1
copies)
Average (memory used/copy)--------------------->   2,323,872
Average % of total memory (memory used/copy)--->   14.52%


charA:
Size---> 8
Maximum memory used---------------------------->   2,411,920  (for 3000
copies)
Minimum memory used---------------------------->   2,356,576  (for 100
copies)
Average (memory used/copy)--------------------->   80,921.56
Average % of total memory (memory used/copy)--->   0.5%


ruleBlack:
Size---> 52,252
Maximum memory used---------------------------->   2,348,384  (for 3000
copies)
Minimum memory used---------------------------->   2,288,192  (for 100
copies)
Average (memory used/copy)--------------------->   78,575.46
Average % of total memory (memory used/copy)--->   0.49%


rulePat:
Size---> 52,252
Maximum memory used---------------------------->   2,370,768  (for 3000
copies)
Minimum memory used---------------------------->   2,289,392  (for 100
copies)
Average (memory used/copy)--------------------->   78,622.8
Average % of total memory (memory used/copy)--->   0.49%


labelABlack:
Size---> 3
Maximum memory used---------------------------->   2,677,264  (for 2700
copies)
```

```
Minimum memory used---------------------------->   2,315,632  (for 100
copies)
Average (memory used/copy)--------------------->   79,684.2
Average % of total memory (memory used/copy)--->   0.49%


labelAPat:
Size---> 3
Maximum memory used---------------------------->   2,561,904  (for 2900
copies)
Minimum memory used---------------------------->   2,318,224  (for 100
copies)
Average (memory used/copy)--------------------->   79,728
Average % of total memory (memory used/copy)--->   0.49%


polygonBlack:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   2,572,784  (for 600
copies)
Minimum memory used---------------------------->   2,350,752  (for 100
copies)
Average (memory used/copy)--------------------->   80,858.5
Average % of total memory (memory used/copy)--->   0.5%


polygonPat:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   3,349,280  (for 3000
copies)
Minimum memory used---------------------------->   2,356,704  (for 100
copies)
Average (memory used/copy)--------------------->   81,328.23
Average % of total memory (memory used/copy)--->   0.5%


poundRaster:
Size---> 0
Maximum memory used---------------------------->   3,748,720  (for 600
copies)
Minimum memory used---------------------------->   2,555,856  (for 1400
copies)
Average (memory used/copy)--------------------->   88,943.83
Average % of total memory (memory used/copy)--->   0.55%
```

5.4 Strip Size E

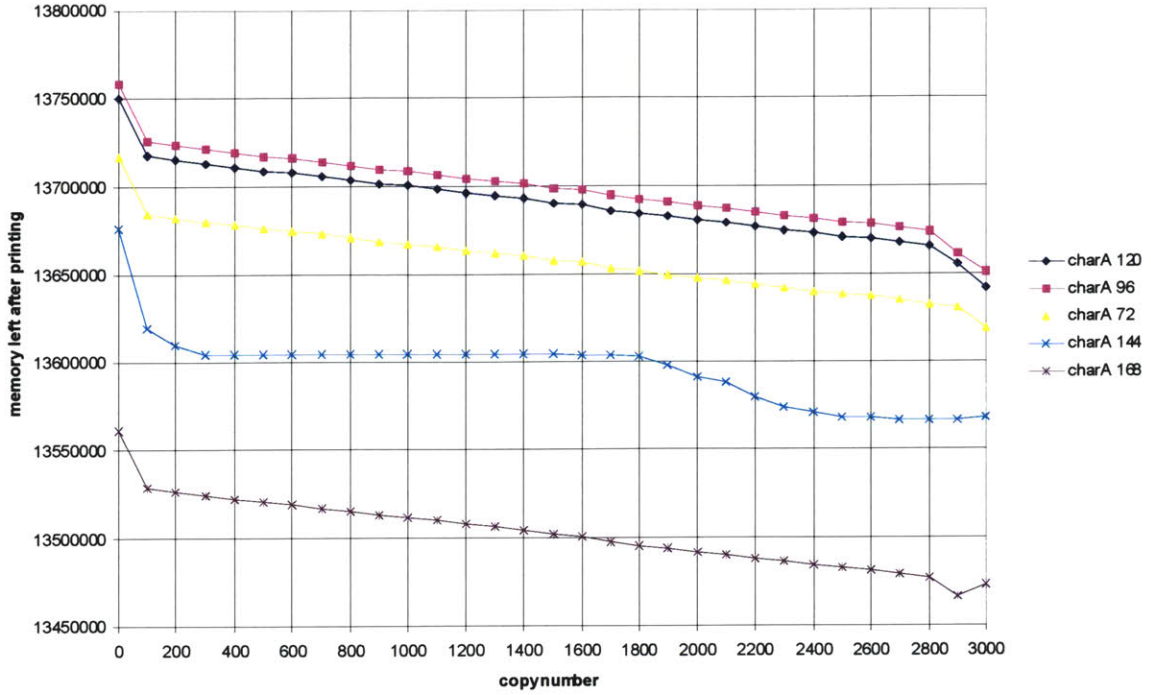
```
                                ANALYSIS REPORT
Copy range(0-3000 by 100)


Average memory used to print a copy of any primtive: 346,523.42
Average % of total memory used to print a copy of any primitive: 2.16%


**Information per primitive**
init:
Size---> 100
Maximum memory used---------------------------->   2,438,560  (for 1
copies)
Minimum memory used---------------------------->   2,438,560  (for 1
copies)
Average (memory used/copy)--------------------->   2,438,560
Average % of total memory (memory used/copy)--->   15.24%


charA:
Size---> 8
Maximum memory used---------------------------->   2,533,392  (for 2900
copies)
Minimum memory used---------------------------->   2,471,264  (for 100
copies)
Average (memory used/copy)--------------------->   84,859.16
Average % of total memory (memory used/copy)--->   0.53%


ruleBlack:
Size---> 52,252
Maximum memory used---------------------------->   2,463,072  (for 3000
copies)
Minimum memory used---------------------------->   2,402,880  (for 100
copies)
Average (memory used/copy)--------------------->   82,512.86
Average % of total memory (memory used/copy)--->   0.51%


rulePat:
Size---> 52,252
Maximum memory used---------------------------->   2,485,952  (for 3000
copies)
Minimum memory used---------------------------->   2,404,080  (for 100
copies)
Average (memory used/copy)--------------------->   82,560.26
Average % of total memory (memory used/copy)--->   0.51%


labelABlack:
Size---> 3
Maximum memory used---------------------------->   2,791,952  (for 2700
copies)
```

```
Minimum memory used---------------------------->   2,430,320  (for 100
copies)
Average (memory used/copy)--------------------->   83,621.46
Average % of total memory (memory used/copy)--->   0.52%


labelAPat:
Size---> 3
Maximum memory used---------------------------->   2,676,592  (for 2900
copies)
Minimum memory used---------------------------->   2,432,912  (for 100
copies)
Average (memory used/copy)--------------------->   83,665.5
Average % of total memory (memory used/copy)--->   0.52%


polygonBlack:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   2,685,664  (for 700
copies)
Minimum memory used---------------------------->   2,465,440  (for 100
copies)
Average (memory used/copy)--------------------->   84,795.2
Average % of total memory (memory used/copy)--->   0.52%


polygonPat:
Size---> 230,0,115,115,-115,115,-230,0,-115,-115,115,-115
Maximum memory used---------------------------->   3,463,968  (for 3000
copies)
Minimum memory used---------------------------->   2,471,392  (for 100
copies)
Average (memory used/copy)--------------------->   85,264.7
Average % of total memory (memory used/copy)--->   0.53%


poundRaster:
Size---> 0
Maximum memory used---------------------------->   3,863,408  (for 600
copies)
Minimum memory used---------------------------->   2,647,840  (for 1200
copies)
Average (memory used/copy)--------------------->   92,871.66
Average % of total memory (memory used/copy)--->   0.58%
```

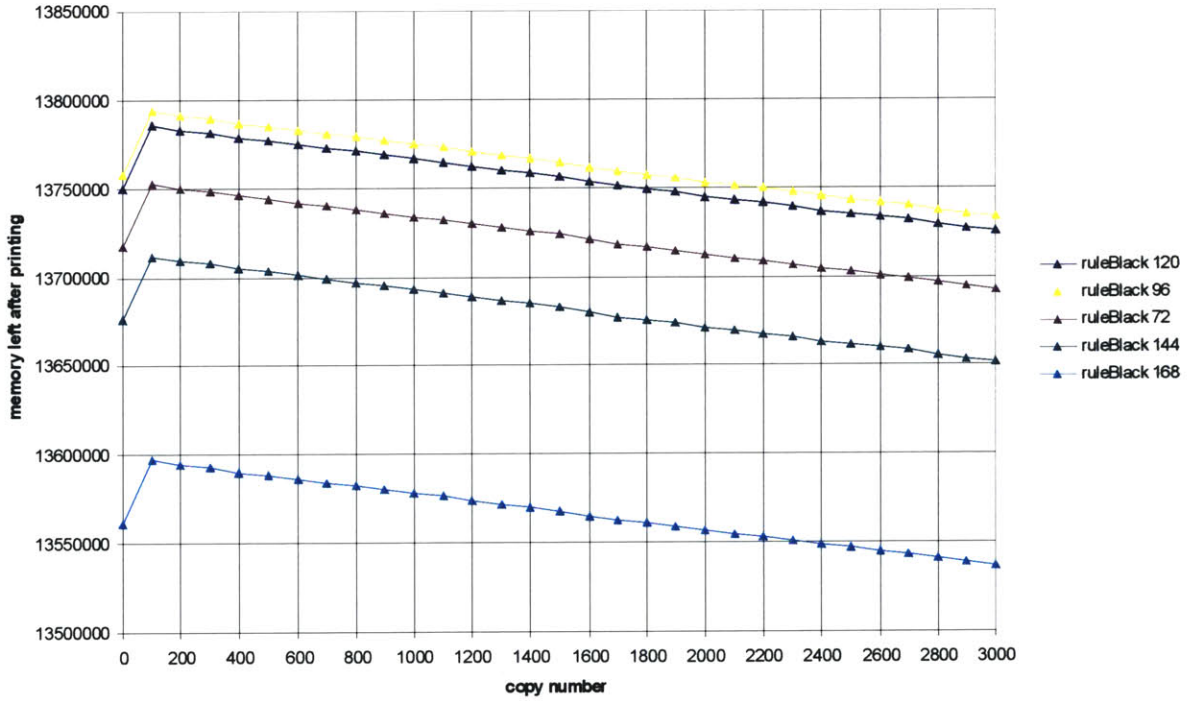
6 Appendix B

This appendix contains graphs of 'After Sum Free' from the data chart. Each graph looks at one primitive's results based on the strip size being used by the printer.

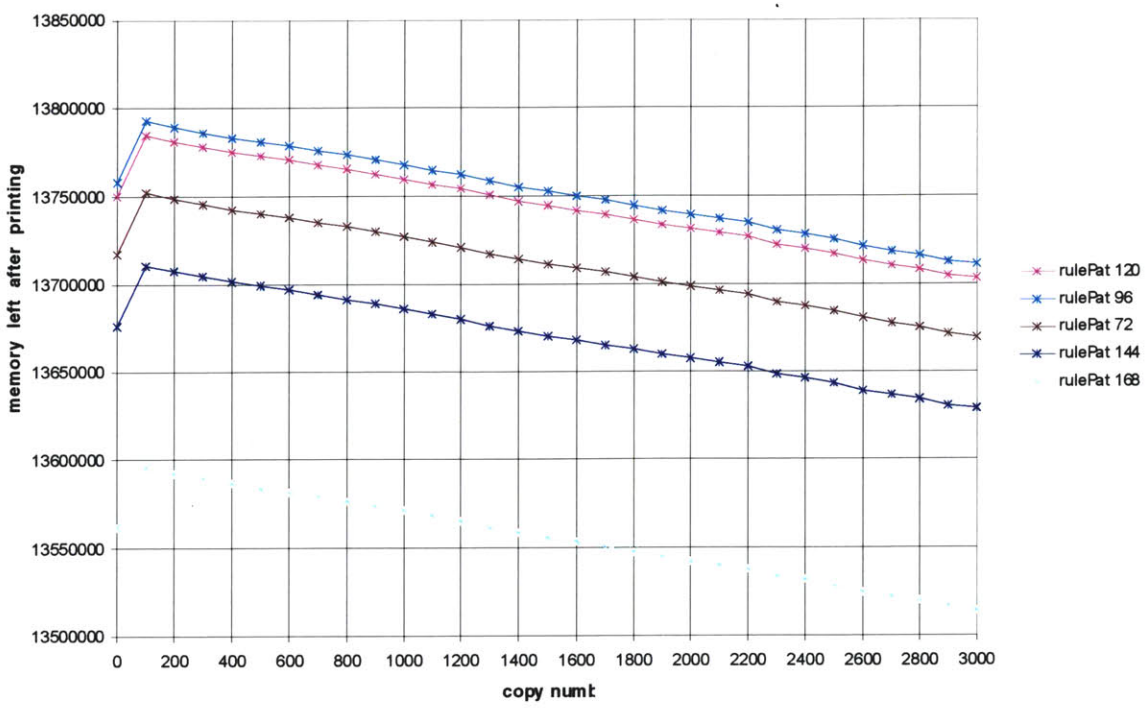
Printing Characters



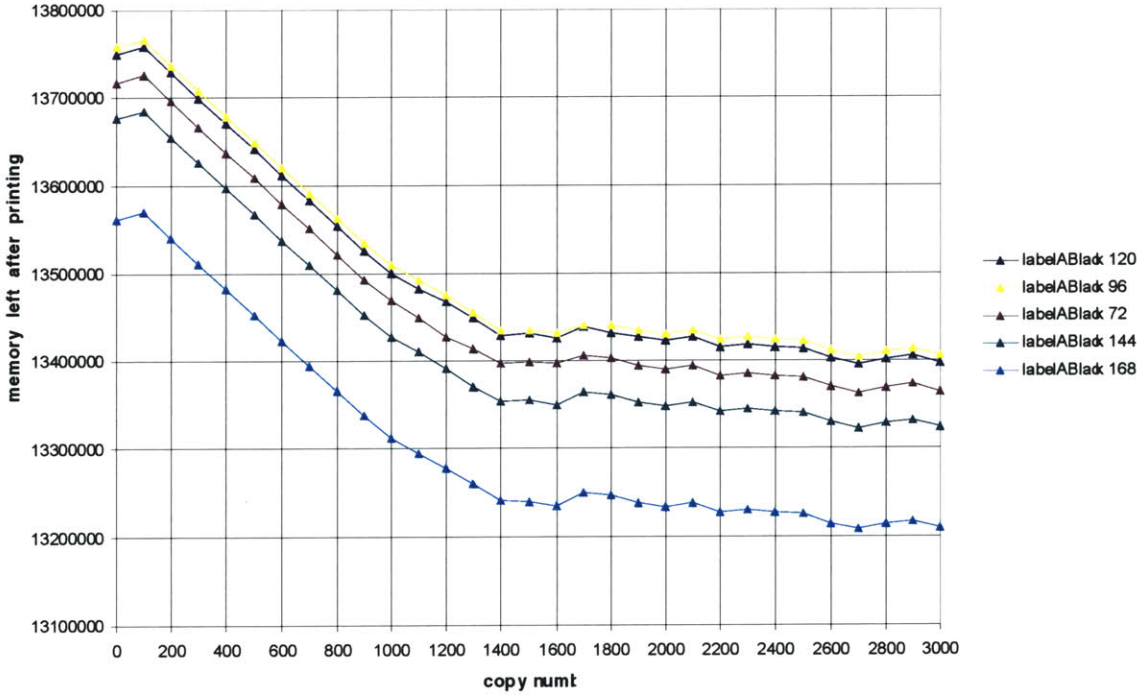
Printing Rules



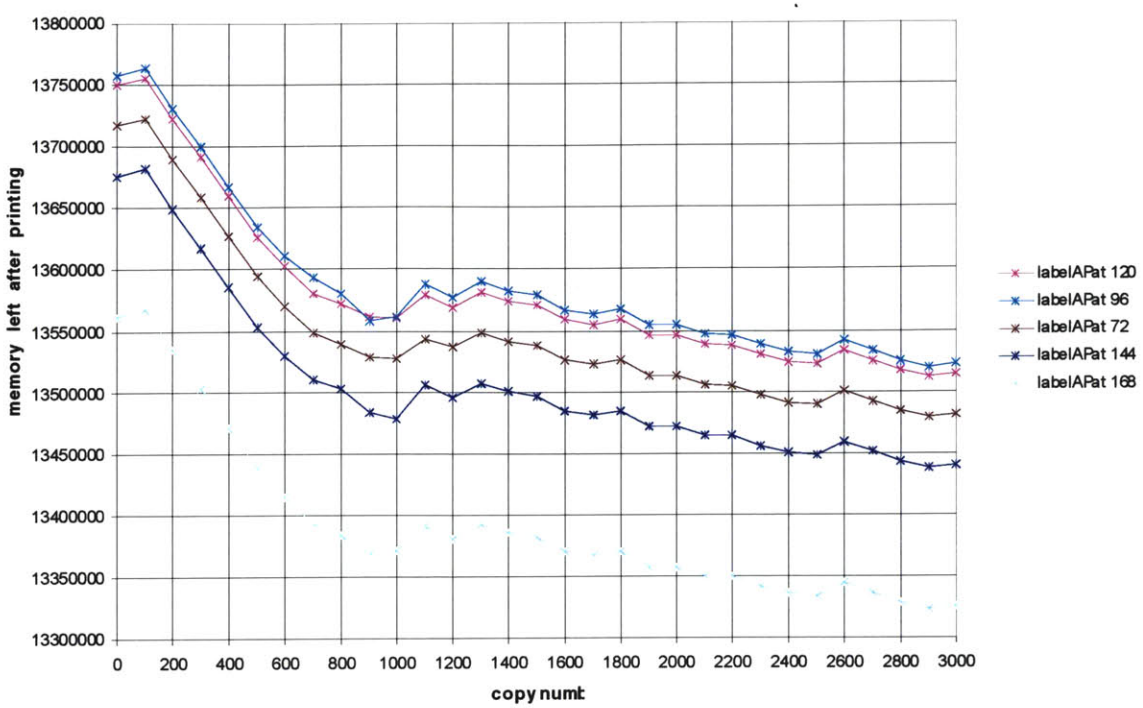
Printing Rules



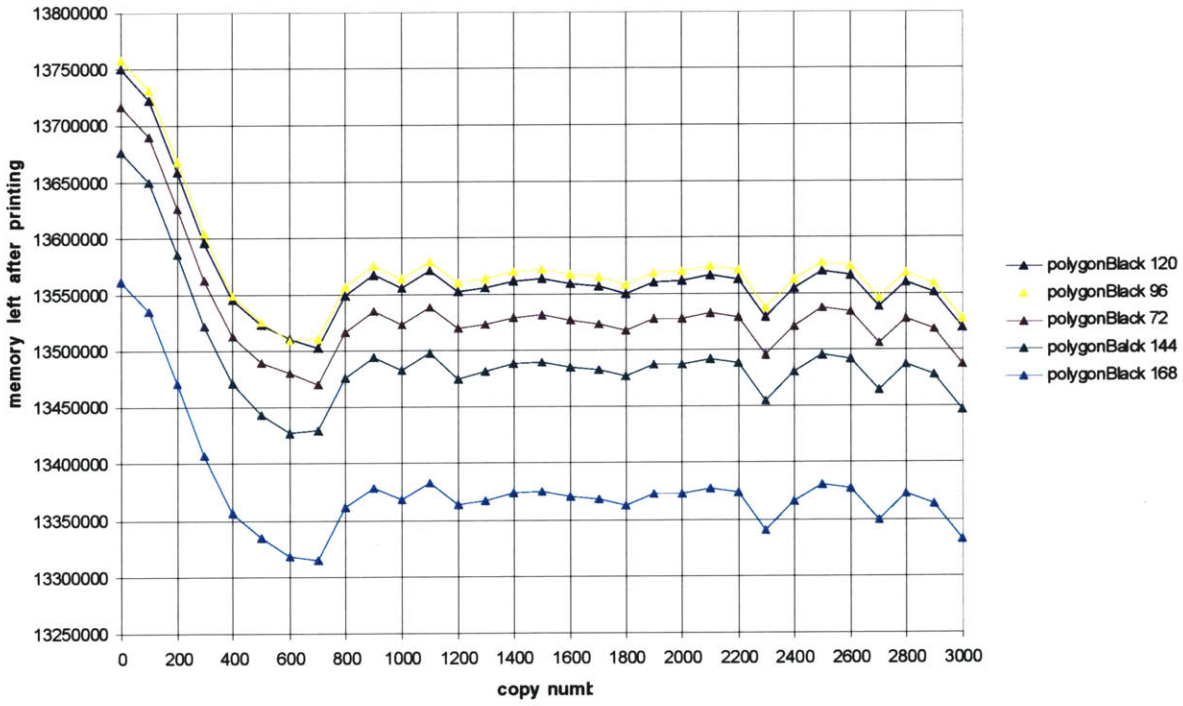
Printing Labels

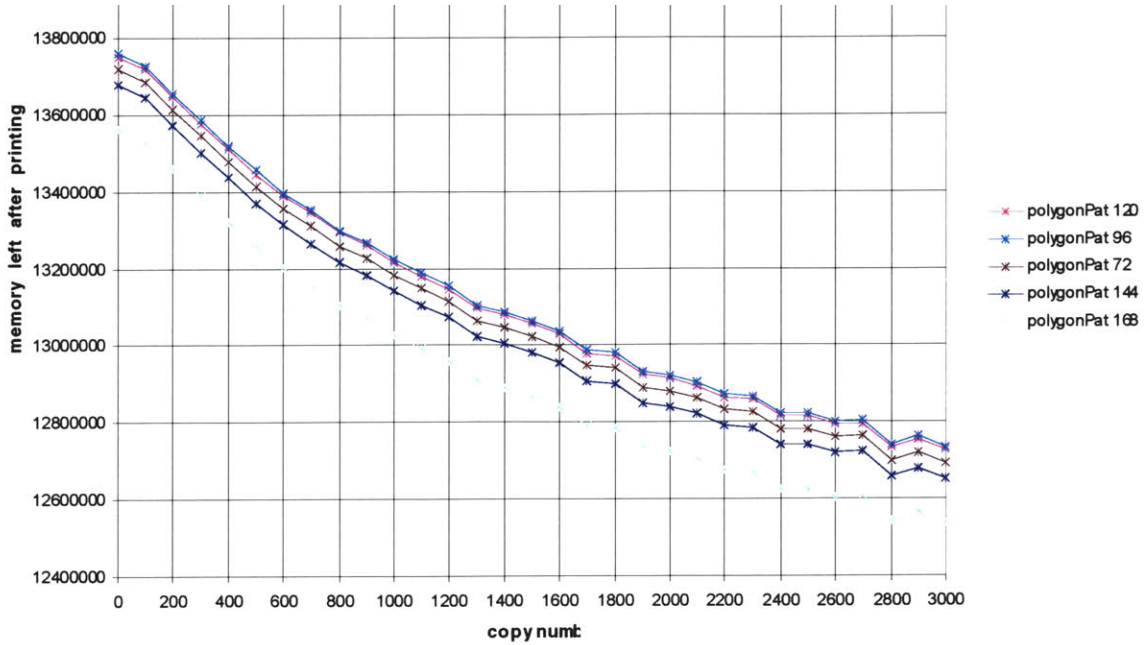


Printing Labels

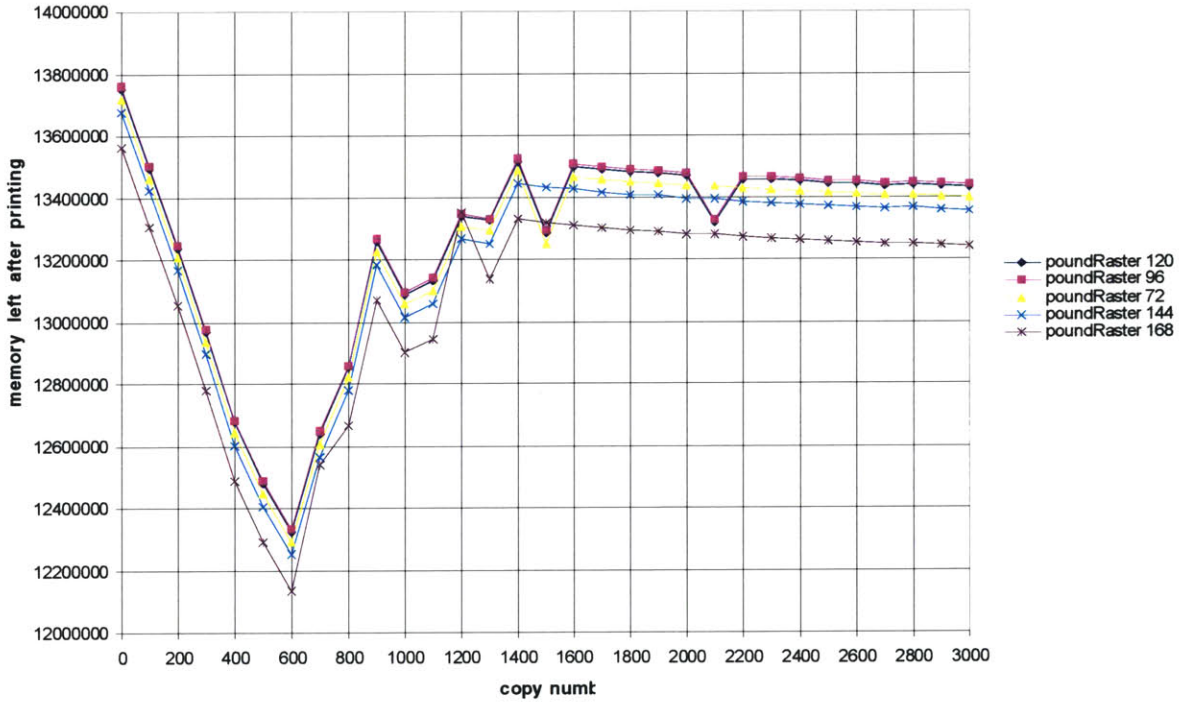


Printing Polygons





Printing Raster



7 Appendix C

This appendix contains information for the actually use of MemUse.

7.1 Primitive File

The primitive file contains all the primitives that the user wants to use with the tool. The file has one primitive per line. For comments the user must start the line with '//'. For each line the following format is used:

Primitive type | print command

E.g.:

```
charA|^[(s#HA
labelABlack|^]%0BINSP1PA~,~SD3,#DT*CF0LBA*;
```

The type will be used to identify the primitive. This exact name must appear in the size file and will be used in any data output produced by the tool. Both the primitive type and print command must be present on the line; one can not be present without the other.

The print command is the exact command need to print the primitive. Above you will notice '#' and '~' in the print commands. These characters will be replaced. After replacing these characters this command should be able to be placed in a file with the appropriate heading, the required cursor placement, and proper exit commands. The result should be a file that will print the primitive on a page.

The '#' in the print command are placeholders for the size. These characters will be replaced with the sizes provided in the size file. Each file produced will contain only one size primitive, but there will be a file for each size provided in the size file for the primitive.

The '~' in the print command are placeholders for locations. Before the print command is placed in the file, the cursor is positioned. For some print command like the one above, this cursor positioning is ignored; therefore actual positioning is required in the print command. These characters are replaced with one coordinate of a location. Therefore the '~' should appear two at a time to receive a x and y location.

7.2 Size File

The size file holds the sizes the user wants to use with the primitives in the primitive file. Each primitive in the primitive file must be represented in this file with a size. If the primitive does not require size replacement, the primitive must still be represented with a size, but during file construction the size will be ignored. The size number is apart of the filename whether the size is used in the print command or not. Therefore it is recommended that if the number in the size file is not being used in the print command, the number should model the actual size of the primitive being printed.

Comments in this file start with '//'. Each primitive has a 'size section'. The section begins with '!!primitive name' with each line after containing a size. Each primitive can have one or more sizes in its section. Whatever size put on a line will be entered in the print command: a number, a group of numbers, a letter, or a word. Therefore it is important to have the correct format. Wrong size formats will result in an error in simulation; the file will produce a print output that is not what was expected.

If multiple '#' are in one print command each size line for that primitive should contain the same number of sizes separated by commas.

Example:

If primitive file is :

```
charA|^[(s#HA
labelABlack|^]%0BINSP1PA~,~SD3,#DT*CF0LBA*;
```

The size file could be:

```
!!charA
2
3
!!labelABlack
10
```

If the primitive file is

```
rollA|^[$(#f##HA
box|^[*y
```

The size file could be:

```
!!rollA
2,3,4
5,6,7
!!box
2,3
```

*The size for box, '2,3', will not be entered into the print command for box. But the size placed in the size file is close to the size the print command uses. When this number shows up in the filename, it will be a reminder of what size the primitive is, instead of just being a random number.

7.3 Shell Scripts and Control Files

Sim.start: script to start the simulator. Currently the tool is using the LaserJet 5si firmware simulator. This file should do all the necessary operations to start the simulator allowing its progress to be monitored and allowing it to be exited if needed. This file currently contains the following:

```
MEMUSE/SIM/sim -X -d MEMUSE/SIM -D MEMUSE/TMP > MEMUSE/TMP/sim.out&
echo $! > MEMUSE/SIM/process.id
```

This starts the firmware simulator assigning 'TMP' as directory for output files. The simulator is run in text mode '-X'. By running the simulator in text mode the simulator's working status can be redirected to the file sim.out. Therefore, sim.out is the file monitored for readiness. Since this simulator outputs READY when it is waiting for further input, 'READY' is the word being looked for in sim.out.[7] The last line outputs the process id number associated with starting the simulator to a file called process.id. This file is then used by Sim.end.

Sim.input: script to send files to the simulator. If the format of this file is changed, it must be set up to take one variable. The code in the Simulator class passes this script the filename of the file being sent. Currently the file contains the following:

```
cat $* > MEMUSE/TMP/parallel
```

Sim.end: script to exit the simulator. This file causes the simulator process to die. Currently this file contains the following:

```
kill -s SIGINT `cat MEMUSE/SIM/process.id`
```

MemUse.on: this is a printable page file that has the PCL commands to turn on the tools output in the firmware simulator. Its contents should not be changed unless the mechanism for turning the tool's code on is changed.

MemUse.off: this is a printable page file that has the PCL command to turn off the tools output in the firmware simulator. Its contents should not be changed unless the mechanism for turning the tool's code off is changed.

Iobuffer.pjl & iobuffer.auto.pjl: these are printable page files that have PCL commands to produce a powercycle event in the simulator. The contents of these files should not be changed unless the mechanism for producing a powercycle changes.

---

[7] The tool options menu asks for the filename of the file monitored for readiness and the word that signals readiness. For this example, the file is 'sim.out' and the word is 'READY'.

7.4 Default Directory Structure For MemUse

The tool should reside in a directory called MEMUSE. The default format of this directory:

- ANAL : contains all files produces from the analysis. Default name of data chart is 'data', and the default name for the analysis report is 'report'.
- JAVADOC : contain html files to display class documentation via the web. To begin open the file Package-SRC.html. To regenerate the documentation, from the MEMUSE directory type:
  javadoc –d JAVADOC –classpath .:/opt/java:/opt/java/lib/classes.zip:MEMUSE SRC.
- OPT : contains size and primitive file. See section 7.1 and 7.2 for an explanation of these to files.
- SIM : contains the files needed to run the simulator. See section 7.3 for an explanation of the shell scripts and control files. The user should know what system files are needed to run a local version of the simulator. These system file must be in this directory.
- SRC : contains all the source code. To recompile source code, from the MEMUSE directory type: *javac –O –d EXE SRC/*.java.* The –O in the compile command is optional; it optimizes the class files.[8]
- EXE : contains all the class code.
- TMP : simulator output files will be placed in this directory.
- FILES : files generted by the TestPageGenerartor class will be placed in this directory.
- HELP : contains the help files.
- DOC: contain the tool's manual.

The default options of the tool assumes you are running the tool from the EXE directory with the above directory format.

[8] Optimization of class files allows static, final, and private methods to run faster but it results in larger class files. It also prevents Java from adding line number debugging information to class file.

7.5 Example Of MemUse's Tool Options Menu

```
****************SET OPTIONS MENU****************[9]
---OPTION-----------------------------------CURRENT VALUE---
1.Primitive List File:                      ../OPT/Primitives
2.Size List File:                           ../OPT/Sizes

3.Simulator executable directory:           ../SIM
4.Simulator input directory:                ../TMP
5.Simulator output directory:               ../TMP
6.Simulator memory data output file:        ../TMP/sim_wp

7.System execution command:                 sh
8.File to start simulator:                  ../SIM/mobux.start
9.File to send a file to simulator:         ../SIM/mobux.input
10.File to exit simulator.:                 ../SIM/mobux.end

11.Analyzer data chart file:                ../ANAL/data
12.Analysis Report file:                    ../ANAL/report

13.Word signaling simulator is ready:       READY
14.File to monitor simulator readiness:     ../TMP/sim.out
15.File delimiter for memory data file:     **Memory Output**
16.Field delimiter for memory data file:    :

17.Language:                                PCL
18.Extention for language:                  .pcl

19.Printer DPI:                             300
20.Logical Page Width:                      2400
21.Logical Page Length:                     3300
22.Resolution:                              600
23.Orientaion:                              PORTRAIT

24.Smallest copy number:                    0
25.Largest copy number:                     1
26.Increment for copy numbers:              1
27.Number of copy repeats:                  1
28.The memory in the printer:               16000000

29. TESTMODE(0:false, all other:true)       0

R Reset to default values
H Help
B Back to Main Menu
Enter '*' to cancel selection

Please Make a Selection
>
```

[9] This output was produced by the text user interface.

8 References

[1] Hewlett-Packard Company, "Met Characterization Theory, Implementation and Characterization Tool Usage", 1997

[2] Hewlett-Packard Company, "IP Characterization FW Architecture and Implementation", 1998

[3] Hewlett-Packard Company, *PCL 5 Printer Language Technical Reference Manual*, Hewlett-Packard Company, USA, 1992