UNIVERSITY OF GOTHENBURG

Bundle.
A Virtual Memory Based Resource System.

Bachelor of Science Thesis in Software Engineering and Management

Jarryd Hall
Taher Odeh

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2012

Bundle
A Virtual Memory Based Resource System

Jarryd Hall
Taher Odeh

Examiner: Helena Holmström Olsson

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2012

Table of Contents

ABSTRACT

Memory related issues such as memory management, limited memory and memory related application crashes, etc. often affect video games. Private game development companies have created solutions to these kinds of issues, however there are no open source solutions available for developers to utilize. This paper presents a solution to solve some of these memory issues using a constructive research approach. Solid State Drives are becoming more common especially in mobile devices. Utilizing the power of SSDs we are able to virtual memory to power the game asset data. The types of gaming systems are diverse and targeting multiple platforms is important for an open source solution. C was used to develop the software as it allows the targeting of multiple platforms. The developed solution is called Bundle and consists of a packaging tool and an Application Programming Interface for developers to use the packaging tool’s output file. Although the solution targets multiple platforms, for the scope of this research the focus was to design, implement and test a C based solution and furthermore to validate the solution’s integration ability with the iOS and Mac OSX platforms

Keywords

Video Games, Memory Management, Cross-Platform, Game Performance, Virtual Memory, mmap, Packaging Tool, API, RAM, SSD, File-System, OSX, iOS

1. Introduction And Motivation

Almost all video games are bound by memory constraints. The system running the game can only allocate a certain amount of usable RAM for the game’s process and this is usually less than what an advanced game needs. The system’s kernel running the game will allocate memory for the game on RAM as well as on disk, which is where the virtual memory is located. A processes’ virtual memory space is numerous times larger than the processes’ available RAM, e.g. on an iPhone 4 there is around 40MB of RAM space, while the virtual memory available to a process is around 700MB.

Large data sizes for game data and assets force developers to spend development time on managing assets in RAM. Game assets need to be dynamically loaded and unloaded as needed when available RAM space runs low. This can lead to the creation of more complex and repetitive code that makes the project harder to manage.

These memory constraints can cause various effects during the development process. These effects could range from longer development cycles due to bug fixing and memory management to the far worse scenario of decreasing game asset quality to reduce the RAM footprint. The development cycle is usually guided by budget and cost, and especially within indie projects and smaller game development projects, the budget could force drastic changes to be made to the project.

There are techniques used to solve these memory constraints, but they are privately owned and there exists little documentation surrounding how the memory problems are solved.

The software market has seen positive changes within the last few years with the rise of smart phones and their capabilities to run complex applications. Software developers have observed the users' needs transform around the mobile operating systems and applications. Apple and Microsoft have considered these mobile-based characteristics and incorporated them into their new operating systems, Apple's OSX Mountain Lion (Viticci, 2012) and Windows 8.

The reason this is relevant is that the mobile software sale platforms such as Apple's App Store, Android's Marketplace and Windows Marketplace are also targeting desktop computing, e.g. Apple's App Store, Google Apps, etc. These sales platforms have created a new age of software development opportunities where anyone is able to develop a product and sell it through these platforms.

Apple's App Store currently has around 111 000 active games and around 635 000 active applications across all categories. (App Store Metrics, 2012) There are currently around 157 000 active publishers on the App Store. (ibid)

With no open source solution for utilizing virtual memory to manage assets within games or asset-driven applications and the amount of developers now developing for various platforms, Bundle was created as a solution to this problem. A further motivation for the need for such a solution was the fact that many developers might not be aware of the possibilities of utilizing a systems virtual memory to enhance performance of the apps when using their assets and resources.

Solid-state drives are becoming the standard on computing systems and are already present on smartphones. Solid state drives have numerous advantages over magnetic based disks, such as lower latency and faster read/write times, especially with random access, as data can be accessed directly rather than waiting for the hardware to adjust the magnetic disks of an HDD.

Virtual memory is stored on disk and these aforementioned advantages of solid-state drives are important if asset data and resources are to be streamed from virtual memory.

This increasing popularity of solid state drives combined with the amount of developers now developing for the various app sales platforms was a strong motivator for creating such a solution.

Game development is a dynamic process with numerous techniques available to develop a product for targeting a variety of platforms. Implementing a solution for this problem and licensing it as an open-source project will allow anyone to be able to increase their game performance as well as improve the overall game design and development by using or modifying Bundle to their custom needs.
The memory problems described above were discussed during an initial interview with Johan Knutzen, a founder of Senri and Phobic Games, mobile application and game development companies based in Gothenburg, Sweden. Johan proposed an idea to use virtual memory to solve these issues.

The solution developed includes a number of elements. The packaging tool compresses all the assets for a game into a single pak file. The pak file format is a file format that contains archived data, with each packaged file stored in either compressed or uncompressed format. The API allows developers to use the generated pak file, handles the virtual memory allocation, file retrieval and removes itself from the system.

The question researched and the results presented in this paper was:

- How can the memory footprint of video games be decreased?

This paper presents Bundle, the open-source solution created to solve these memory related issues and presents the results.

2. Related Work And Literature Review

2.1 Virtual Memory Based Data Streaming In Video Games

The literature on the subject of utilizing virtual memory to enhance memory performance on a gaming system is limited because the concept is usually developed for high quality video games. These solutions are kept private for obvious reasons.

ID Software has done the most notable implementation using a similar concept. John Carmack is the lead developer for ID Software and discusses how he used memory mapping to enhance the iOS version of the game called Rage (John Carmack, 2010).

2.1.1 Texture Streaming

"Texture streaming is a facility mostly used in computer games to reduce memory usage. Instead of loading all textures at once, they are loaded and unloaded on a per-need basis." (Mayer, 2010)

2.1.2 MegaTexture

A MegaTexture is the creation by id Software's technical director John Carmack. A megatexture holds a large number of smaller textures. John Carmack revised the original version to work on arbitrary geometry and it was confirmed in 2006 to work on any geometry. (Mayer, 2010)

2.1.3 Virtual Texturing

"Virtual texturing generalizes previous approaches by adding a level of indirection, i.e., a pagetable, similar to virtual memory subsystems in modern operating systems. The pagetable (texture) allows each fragment to access a suitable tile depending on the re-quired resolution according to standard OpenGL mipmapping calculations, instead of relying on heuristics like distance to viewpoint (as used in clipmapping)." (Mayer, 2010)

Van Waveren, (2009) mentions a number of issues, related to the use of texture virtualization, that were faced during the development of the id Tech 5 game engine, such as:

- *Texture Filtering*
- *Thrashing*
- *Level OF Detail (LOD) Transitions Under High Latency*

2.2 Memory Enhancements for Embedded Systems

CRAMES is a system to enhance memory on embedded systems. (Lekatsas et al., 2005) 'Compressed RAM for Embedded Systems' uses efficient algorithms as a RAM compression technique. CRAMES can double the amount of memory space on an embedded system.(ibid)

"CRAMES takes advantage of an operating system's virtual memory infrastructure by storing swapped-out pages in compressed format. It dynamically adjusts the size of the compressed RAM area, protecting applications capable of running without it from performance or energy consumption penalties. In addition to compressing working data sets, CRAMES also enables efficient in-RAM filesystem compression, thereby further increasing RAM capacity." (ibid)

CRAMES targets low-power embedded systems and offers read and write data access. (ibid)

Cramfs aka. Cram a filesystem onto a small ROM, (Cramfs, 2011) is a compression library to compress files to be used on ROMs. It uses zlib to compress one page at a time and allows random access to these pages. Cramfs targets systems that have a maximum filesystem size of 256MB and offers read only data access. (ibid.)

2.3 File Systems And Archived File Types For Video Games

The pak file is a file that is packaged with many files and works as an archived file. There are numerous games that use pak files as a means of holding game data files in an archived manner. The pak file serves as a means to hide asset data files from extraction as the game assets are archived into a single file and not left inside a resource folder.

McCormick (n.d) wrote a description of the Quake Pak format explaining an example structure for its header and directory information.

id Software released Doom 3's source code (Doom 3, 2011) and the file system's implementation is available. Doom 3's file system uses .pk4 files, which hold the game's resource data. "To save disk space and speed up file loading, directory trees can be collapsed into zip files. The files use a ".pk4" extension to prevent users from unzipping them accidentally, but otherwise they are simply normal zip files. A game directory can have multiple zip files of the form "pak0.pk4", "pak1.pk4", etc. Zip files are searched in descending order from the highest number to the lowest, and will always take precedence over the filesystem. This allows a pk4 distributed as a patch to override all existing data." (Doom 3, 2011)

Id Software games use a number of archived file types for handling game assets. Quake 3 uses PK3 files (File Extensions Library, 2008), Doom 3 and Quake 4 uses PK4 files (File Extension Guide, 2012).

2.4 OSX and iOS Bundles

"A bundle is a directory with a standardized hierarchical structure that holds executable code and the resources used by that code." (Apple, 2010).

3. Method

3.1 The constructive research method

Crnkovic, (2010) describes the constructive research method as a way to turn existing knowledge into novelty or innovation.
This may involve creating artifact design solutions, for example: plans, diagrams, charts or software implementation.

Suhonen, et al. (1991) summarizes the constructive method as a solution oriented method where innovative step-by-step solutions are taken into account, followed by testing the solution and using the data within the testing phase for analysis purposes.

Lindholm (2008), provides three category examples of knowledge gaps to be filled using the constructive research method:

- Feasibility, where a solution to a common problem has not been done yet.
- Novelty, where a unique and new solution is provided to an already solved problem.
- An improvement, where the goal of the research focuses on a preexisting solution and aims to produce better results than the ones available.

The principles of the constructive method were found the most suitable for this type of research, as a solution to a problem is constructed Caplinskas (2004) argues that the constructive research method befits computer science and IT related problems in a usual manner.

According to Lindholm (2008), There are steps to be followed to order to conduct a constructive research:

1. Finding a research problem.
2. Involving an organization in the solution.
3. Obtaining a detailed understanding of the topic researched.
4. Constructing a theoretical solution.
5. Implementing a practical solution and test its usefulness.
6. Examining the applicability of the solution.
7. Showing the theoretical connection and the solution's contribution.

3.2 Work process

The constructive method was adapted to meet the needs of this research.

The fundamental steps above have been borrowed and minimized to best fit a three-phased framework that shall bring a clearer image of development process followed during research. The framework shows the use of the constructive research method steps used in different stages. (See figure 1)

The first phase of the theoretical framework uses step one of the constructive method steps.
A problem is identified and evaluated as one that has no "acceptable" solutions available for. Ellis and Levy (2008) argue that for a solution to be "accepted" it must not only exist, it should be, as well, documented and mentioned in literature.
This phase requires background knowledge about the problem and the systems involved within the problem's scope. Thus detailed knowledge has to be gained while planning the development, prior to the technical process, which is bolder in the next phase.

In phase 2 of the framework, an understanding of the problem is obtained in details. Data relevant to the problem or its solution are collected and analyzed qualitatively then interpreted into a set of requirements for a software solution.
Phase 2 is based on an iterative process. New data is collected and the knowledge scope is enlarged. A set of requirements is defined as an addition to the existing ones, resulting in additional features being expected from the product or optionally, a need to modify previous tasks to better suit the new constraints.

The design and implementation of the software is developed and tested for bugs. Thus, covering steps 3, 4 and 5 of a constructive method.

In the third and final phase, validation, profiling and memory benchmark tests are run against the software implemented to show the established link between the theory and the solution provided. As well, as to point out the level of feasibility and efficiency the application provides.

Data in this phase are collected and analyzed quantitatively. Multiple final steps are taken into account in this stage of the process, as the implementation should assumingly be ready and bug free, covering all the base functionalities deriving from the requirements set during the beginning steps.

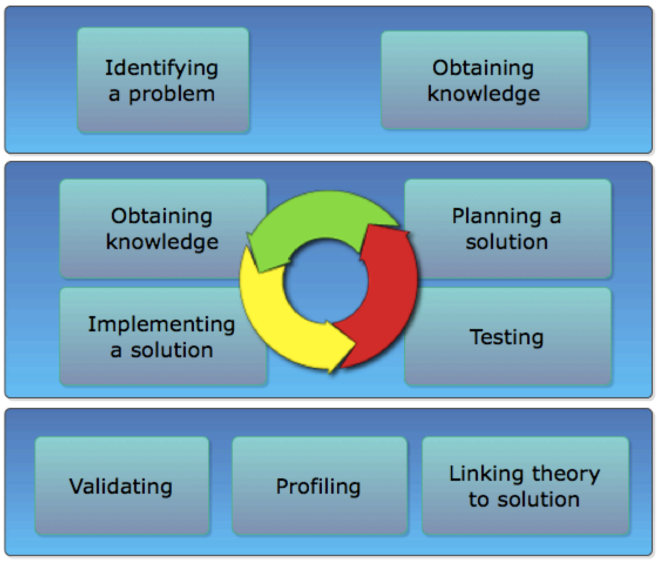


Figure 1. The work process phases (The circle shows the iterative process in phase 2).

3.3 Data Collection

The focus of this research lies on the constructed solution (Lindholm, 2008). Data are collected during all phases of the development process. Enough knowledge must be gained about topics addressing the construction of a solution. A good knowledge base about these topics, which may be theoretical or technical, is necessarily needed in order to initiate a design, and then proceed with the implementation steps. Crnkovic, (2010)

A combination of qualitative methods and quantitative ones were used in this research. Qualitative data was collected through semi-structured interviews with Johan Knutzen as well as via related literature and software manuals. Data of this type is collected mainly during phase 1 and 2 of the development model.

Tests in order to validate the implementation are run after the software is constructed (phase 3).

3.3.1 Semi structured interviews

Harrell and Bradley (2009), argue that interviews are a useful approach in data collecting, since there is a chance to discuss past experiences regarding subjects surrounding the research scope to bring out personal feedbacks for further research.

In a semi-structured interview, Bryman and Bell (2007) argues that the meeting is set to answer a list of questions that are used as the interview guidance.

“In-depth interviewing allows the interviewer to better understand the details associated with the lives of the individuals interviewed. This knowledge can lead to increased understanding of how experience relates to individual interactions with others resulting in the discovery of similarities amongst everyone.” (McClure, 2002)

The interviews mentioned are run with Johan Knutzen, founder and a senior software engineer in Senri. Several ways of data gathering are taken during the interview sessions which are estimated as approximately one hour at a time: dialogues, questions and answers, development guidelines, discussions and suggestions about sources to research further for a better understanding about a specific topic that is relevant to the research.

In phase 1 of the framework, where a problem is identified and a plan to solve it is initiated, qualitative data play an important role. The first semi-structured interview is led by Knutzen, he describes the problem existing in the field. During the first meetings an understanding of why the problem exists is reached in details covering what it addresses, subjects related to the problem and its solution are noted down to be researched later.

Semi-structured meetings are scheduled when there is a need to discuss the next steps in the iteration. Problems faced during previous iterations are discussed and more research-worthy topics are laid out for a further investigation. Eventually possibilities of their integration with the developed product are taken into account.

Meetings contribute in conducting the next steps of the implementation process as discussions focused on deciding the next steps. Before the end of each meeting, the decision to follow a specific technique in the next levels of the plan is agreed on between the participants, considering the time scope of the solution and the technical abilities participants have or are able to produce for the research.

Internet based meetings are held as well during the implementation phase (mostly in phase 2) when there is a need for more information regarding current tasks. These meetings are text based and aiming to cover mostly technical questions where the topic to be discussed is broader and requires more knowledge than what could be gained during the research.

3.3.2 Literature and manuals

Research papers and other literature addressing topics related to computer gaming, packaging assets in games, operation systems, memory and virtual memory management, disks and their types and more, were covered throughout the implementation process. Adding to the above, software manuals are used as the product is getting developed.

Constructivism approaches allow problem-based learning, since an iterative guidance is needed to exist. (Holton, 2010)

Phase 1 of the working model focuses on building a knowledge base in order to initiate a design plan. During this phase, literature and research papers about similar areas of interest are red, covering topics mentioned in these researches in more details.

In phase 2, as the iterative work beings, knowledge is gained from literature and software manuals to carry on with the development.

There is never enough information when it comes to this category of data. With more data collected, there is a better understanding of what is building the research blocks. Thus, more ideas arrive concerning more literature to study about new topics.

3.3.3 Observation

Nunamaker, et. al. (1990) gives an example of virtual memory techniques that were developed by the help of previous implementations.

Design documents and source codes of existing techniques used in the implementation are collected and observed.

3.3.4 Validation tests

As part of the data collected in the last stages of development, validation tests are run against the implementation.

The tests are run with a set of media resources and include:

- Memory usage after mapping and using the files in an archive.
- Virtual memory usage.
- Retrieval time: The time it takes for a file to be retrieved, all the way from the hash table to a pointer to its first address in the virtual memory.

This type of data collected in this stage is numerical are contain statistical information about the system status when running Bundle.

The tests were run on a HDD and on a SSD to point out the efficient alternative solution provided by Bundle, which replaces the use of RAM by the virtual memory.

3.4 Analysis

Information collected during interviews and literature reviews is seen as pieces of information about things that exist, when combined and properly structured, it may bring a clearer view about the bigger system. In this research, it is the product constructed as a solution. (Crnkovic, 2010)

One way to understand how information is transformed into building blocks of a constructive research like this kind is using an approach similar to Information Realism (IR), where objects in reality are seen as informational resources. Nevertheless, interaction with these resources is possible by observing and understanding them and then testing their possible integration. (Floridi, 2004)

Information, be it verbal or written, is studied, discussed and understood, and a possibility of turning it into software requirements is considered. Depending on the information's usefulness and on the current state of development, it is evaluated as data to be turned into software design diagrams, models, programming tasks or future work for an improvement.

The data collected in the profiling stage is used to link the theory addressed in solving the problem to the implemented solution. In other words, whether Bundle really overcomes the RAM constrains in mobile devices when developing a game, or not.

4. Bundle

Bundle was created as a result of using the previously described framework. Bundle is an archiving tool as well as an API for using the produced archive files and allowing the packaged assets to be retrieved by filename. The archiving tool can be used to create pak files and minor enhancements will allow for unpacking of such files.

Bundle can also be applied to asset heavy applications other than games. Media driven applications can greatly benefit from Bundle, especially if video or sound files are read during runtime because the data can be streamed directly from virtual memory and processed byte by byte.

The solution developed was planned from the start to be an open source project (Bundle, 2012). Bundle is licensed under the BSD license. The reason for this decision was the fact that memory management is something every game developer has to deal with and offering an open source solution could help other developers and possibly gains the interest of the game development community to evolve the project.

The focus of Bundle is to target as many platforms as possible because games run a wide range of systems. The focus of this research paper is on the iOS and Mac OSX platforms, although the code itself is C, which allows multiplatform support. Understanding how the current hardware and software operating systems work was crucial in developing Bundle.

Bundle can aid game development projects that are already under development or at any other time during the development phase.

Bundle works by compressing a game's assets into a single archived file, having each file compressed and pushed to the end of the Pak file. A header is constructed to keep track of files in the archive. Then using the API, one may load the file index into a hash table for efficient retrieval of information about the files. Lastly the archive is mapped to virtual memory. The files within the archived file are accessible using the filename itself. The file is retrieved and returned to the game process as a range of bytes, using a pointer and a file size variable.

Game developers can encounter various bugs and issues related to their memory management. Memory also highly affects the performance of games, especially on mobile devices where the available hardware resources are limited.

These memory issues are experienced with manually allocated memory in RAM. A system also allocates a block of virtual memory to a process. Virtual memory resides on disk and acts similar to a swap file. iOS does not use a swap file. (Apple, 2008). The storage of memory on disk and using this virtual memory during a game's process is important for performance.

Mobile devices such as the iPhone and iPad, which run iOS, have Solid State Drives. This is a reason why they were chosen as the target of this papers research.

Once the file is mapped to virtual memory the addresses are translated as the data is paged in from virtual memory. A Memory Management Unit is used to translate virtual memory addresses. "During virtual memory operation, software-visible memory addresses must be translated to real (or physical) addresses, both for instruction and for data accesses generated by load/store instructions."(Singh, 2006)

As Apple's iOS and OSX platforms were the focus of this research, the memory management was researched on Apple's self- published documentation.

The kernel manages RAM and Virtual Memory segments. When an application launches the kernel allocates a block of memory in RAM, e.g. around 40MB on the iPhone 4 and also assigns a virtual memory data segment, e.g. around 700MB on the iPhone 4. The RAM and Virtual Memory work together by means of paging data segments or pages, in and out, between each other. When a data segment is needed that resides in virtual memory, the kernel pages in the page that holds that data into RAM. (Apple, 2008) . Data is paged into RAM from virtual memory until the calculated max RAM size is reached. Once this occurs the kernel will page out data that is not currently needed, and replace it with the page from virtual memory that holds the currently needed data.

Data allocated to virtual memory rather than strictly allocated to RAM using objective C's alloc method would allow larger segments of data to be allocated due to the size of the virtual memories data block.

4.1 Bundle and Automatic Reference Counting

ARC aka. Automatic Reference Counting is a new LLVM (Low Level Virtual Machine) compiler feature for compiling Objective – C and was introduced in iOS 5 and OSX.

"Automatic Reference Counting (ARC) is a compiler-level feature that simplifies the process of managing the lifetimes of Objective-C objects. Instead of you having to remember when to retain or release an object, ARC evaluates the lifetime requirements of your objects and automatically inserts the appropriate method calls at compile time (Core Services Layer. 2012.)

Although ARC removes the need to use the retain and release method calls, it still works on objects allocated to RAM directly. "Automatic Reference Counting implements automatic memory management for Objective-C objects and blocks, freeing the programmer from the need to explicitly insert retains and releases. It does not provide a cycle collector; users must explicitly manage the lifetime of their objects, breaking cycles manually or with weak or unsafe references." (Automatic Reference Counting. 2012)

This does not affect the use of Bundle within an iOS or OSX project because the objects used for the game assets within this type of application are not strictly allocated to RAM, but have their data retrieved from virtual memory. A game can run on iOS and OSX using Bundle for handling asset data and files, as well as ARC for non-asset data objects.

5. Bundle Design

5.1 Design Principles

The goal of Bundle is to increase memory performance when using assets in games and media driven applications, while supporting multiple platforms. Bundle also has to be easy for developers to use at any stage of the development process.

The following principles achieve this goal:

1. *Use a low level language to target multiple platforms.* Using C allows Bundle to be integrated into many projects that run on different platforms.

2. *Package assets according to the user's needs.* The command line tool allows users to specify which files they need compressed within the pak file.

3. *Provide a simple API to use the generated pak files within an application.* Bundle's API provides three easy methods for starting Bundle, retrieving files within the pak file and also stopping Bundle.

4. *Allow file retrievals in a random access manner.* A memory mapped pak file will contain a number of files throughout the data block. These files need to be accessed in a random manner according to the execution of the application process. Random access of these data segments should not cause unwanted overhead for the CPU.

5.2 Design Overview

Video games can run on many different platforms so targeting as many platforms as possible was a design decision from early on. Using C to develop Bundle would allow this as it supports multiple platforms. C compiles correctly with Objective-C, the language used in iOS and OSX, which is a C superset.

Integrating Bundle into an existing project or a fresh project has been made as easy as possible for the developers using it.

Bundle consists of three parts.

1. File Format with .pak extension
2. Packaging Tool
3. API for using the pak file

5.2.1 File Format

As mentioned before, the pak file format is an archive file type without a standardized content format. Bundle's pak file contains a header of variable size depending on the number of assets to be archived, as well as a data section that holds the files. The files that are placed within the data section include byte padding, which is sized depending on the architecture of the system e.g. 32 bit or 64 bit. The header contains the index information for the contents of the data section. Each archived file is indexed with the following data:

- The key value[1] of the filename *
- The file's offset within the pak file
- The size of the file within the pak file
- A compression flag. 1 if the file is compressed and 0 if it is not compressed.

5.2.2 Packaging Tool

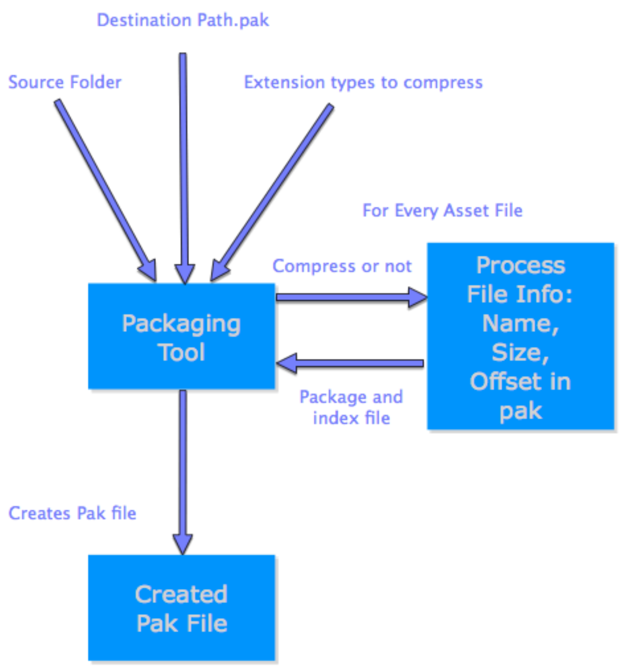
The packaging tool is a command line tool. The following arguments need to be supplied:

- Source Folder Path
- Destination Path with .pak extension
- File types to be compressed

The flagged file types to be compressed are sent to the packaging process and stored. The tool iterates through the source folder and counts the number of files, ignoring non-asset file types, such as the .DS_Store. The asset file count is used to calculate the header length for the pak file being created. The tool checks if the destination file exists, if it does, the user is asked if they would like to overwrite it, otherwise the pak file is created. The tool packages each individual file into the pak file, compressing the file if the user specified compression for the file type. Each file added to the pack is indexed with the file information into the header segment.

When the packaging process is complete the pak is located at the destination path and ready for use.

Figure 2 below shows the packaging process to create a pak file.



5.2.3 API

Bundle's API is simple and easily extendable. The core API is C based like the rest of Bundle's source code. The goal was to create an easy to use solution for developers by abstracting the inner workings and providing the least amount of functions as possible.

It was also important to allow the extendibility of Bundle, as the project is open source and targets many platforms.

6. Bundle Implementation

6.1 Tool and Pak File Creation

The pak file format is an archive file format that is not standardized on its contents. Pak files are usually used for games, where game assets are archived into a single file.

The tool takes the source folder, traverses the entire directory to count the files, ignoring DS_Store files that are present on Apple's platforms. Once the file count is obtained, the value is used to calculate the variable header length.

The pak file header consists of offsets holding information about the files compressed within the pak, in addition to an integer in the beginning of the file representing the number of files compressed. Each offset in the header holds 21 bytes of information needed to be used to locate files in the archive, the first 4 bytes in an offset hold the hash value of the filename, the next 8 bytes hold the index of the beginning of the data of the that file within the archive, the next 8 bytes in an offset hold the size of the that file in bytes, and the last byte indicates whether the file is compressed or not. The above information is stored in a binary form in the header of the pak file to allow the data compressed to be located by the filename of the original file.

The tool outputs a pak file at the user specified destination path. Once this pak file is created it is ready to be used within the application.

[1] The key value is initially the sum of the characters of the filename, the sum is increased in case of duplication until a free cell is found in the hash.

6.2 The API

The base API was developed in C allowing for future evolution of the product, which is especially needed for an open source project where many people might have ideas to evolve the product.

For part the focus of this research paper, the operating systems being Apple's iOS and Mac OS X, wrapper methods were needed to support Apple's Objective-C language, which forms the basis of the Cocoa and Cocoa Touch frameworks. Objective-C is a superset of C, allowing it to integrate seamlessly.

Usability was always an important attribute of the solution as it is an open source project to be used my many developers.

The usage of the API is split into 3 important functions.

1. *int bundle_start(char *pakFile, struct mappedData *mData)*
2. *offset_p bundle_getIndexDataFor(char *fileName)*
3. *int bundle_stop(struct mappedData *mData)*

The *bundle_start* function starts Bundle by loading a given pak file, memory mapping the file to virtual memory, creates a hash table holding the offsets and returns success or failure.

The *bundle_getIndexDataFor* function retrieves the file offset , the size ad the compression flag for a developer with the ease of only needing a filename as an argument.

The *bundle_stop* function stops Bundle by destroying the hash table and removing the mapped file from virtual memory.

Allowing such ease of use of the Bundle system developers do not need to lose valuable development time, which adheres to an important attribute of Bundle, to improve the development process in various ways. Developers can integrate Bundle quickly into existing games. Including the bundle.h file and linking the static library allows the user to use the interface functions available, for loading a Pak file, retrieving offsets by filename and freeing the memory used by Bundle.

The offsets type retrieved is defined in the *header.h* file:

```
" struct{
  khint_t hash;
  long int offset_start;
  size_t size;
  char compressed;
} header_offset "
```

The variable *hash*, holds the hash value of the filename in the archive, offset_start holds the address of the data in the virtual memory, size is the size of the fle before compression in bytes and compressed is a flag corresponding to whether the file is compressed in the pak or not.

Using the above offset, one may access the data in VM and use it freely, decompress it if it is compressed, save it to a file, view or play it using a higher level language, etc.

6.2.1 Memory Mapping with the POSIX mmap Function

A POSIX function called mmap (Kernel.org, 2010) is a low-level virtual memory mapping technique that allows one to allocate a file on disk into virtual memory addresses. The mmap function provides a number of ways to map a file to virtual memory.

Bundle assumes that the mapped file is read-only. The PROT_READ macro was used to provide this read-only feature. Bundle also maps the file using the MAP_PRIVATE macro. This private mapping disables the mapped file from being accessed within other processes, a feature which is needed on an operating system that supports multitasking. Bundle calls the mmap function as follows:

- *mmap(NULL, mData->fileSize, PROT_READ, MAP_PRIVATE, fileDescriptor, offset);*

It is important to note that developers needing to bundle scripts into their pak file and allow those scripts to be executed directly from virtual memory can use the PROT_EXEC macro as an argument for the mmap function call.

Bundle also uses the madvise() function, which advises the kernel on how to use the memory. Using the madvise function with the MADV_RANDOM argument notifies the kernel that the virtual memory mapped pak file is to be accessed randomly with readahead turned off. This was chosen as one of the previously mentioned design principles is to provide random access to the archived data within the memory mapped pak file without unwanted overhead to the CPU. Disabling readahead avoids this unwanted CPU overhead (madvise, 2008).

6.2.2 Hash table

The hash table implementation is developed to hold offsets red from the header of the pak file, thus allowing easy and fast access to information about packed files in the archive. The hash table is initialized globally on the stack and is structured to hold the hash values of the filename as the key, and the offset copied from the header as the value. At first, a simple and efficient hash table implementation that has the same number of items as the files in the pak, thus not allowing duplicates, but might provide slower access to objects within the hash.

The hash table wrapper provides the following functions:

- int hash_init(char *filename
- offset_p get_offset(char *filename)
- void hash_read()
- void hash_destroy()

The first function takes a filename and initializes a hash table with the header offsets existing in the header. The keys of the hasmap are the hash value of the filename, and the value points to a *header_struct* type representing the offset.

The *get_offset* fuction returns the offset corresponding to the hash value of the filename passed.

The *hash_read* function, iterates through the elements stored in the hash table and returns an array of offsets.

The *hash_desotroy* function frees the created table from the stack.

6.2.3 Objective-C Wrapper

Bundle provides an Objective-C wrapper class called *BundleCocoaWrapper.m.* This class contains two methods:

- *(NSData *) bundle_useFile:(NSString *) filename*
- *(NSNumber) isFileCompressed:(NSString *) filename*

The *useFile* method converts the NSString based filename to a C based char *. The char * is then used with Bundle's C based function.

offset_p *bundle_getIndexDataFor(char *fileName)*

An NSData object is created and the offset_p data is passed to an Objective-C method.

The Objective-C NSData class has a method for passing a pointer to an address as well as the length of bytes to process, with the alternative option of freeing the data when done or not (Apple, 2011).

*+ (id)dataWithBytesNoCopy:(void *)*bytes
*length:(NSUInteger)*length
*freeWhenDone:(BOOL)*freeWhenDone

The data object is assigned with offset_p's data using the following method call:

data = [NSData
dataWithBytesNoCopy:offset->offset_start
length:offset->size
freeWhenDone:NO];

Each object that requires asset data to execute will thus have an associated NSData object that is not copied from virtual memory and allocated to RAM. The NSData pointers created by the developer will point to the memory segment in virtual memory and be paged in and out by the kernel.

This allows an object to be created using the virtual memory pointer and size of bytes of the segment, without the need to allocate memory in RAM for the object. This perfectly integrated with the developed base C API that allowed retrieval of a file's index data within the pak file, using the filename itself.

This *useFile* method then returns the NSData object pointer for further use by the developer.

This feature would make it as simple as possible for developers to use the API as they can use their filenames as usual.

The second method provided within the *BundleCocoaWrapper* class allows a developer to determine whether or not a file is compressed within the memory mapped pak file. Developers can then use this data appropriately.

6.2.4 Static Libraries

Bundle also provides a static library for developers to link to within their projects. This simplifies the process for developers using Bundle and adheres to the design principle of providing a cross platform solution.

The following options are available to build the static libraries and use Bundle's functions in a C compatible environment:

- Using the *make* command from src/:
  - make header_lib: to create a static library for the header functions.
  - make hash_lib: to create a static library for the hash table functions.
  - make mmap_lib: to create a static library for the memory mapping functions.

- Manually, by compiling the .c files in the src/ folder and use a library tool like *ar* to create the libraries.

Including the appropriate .h files and linking the libraries when building the code is one way of using Bundle's libraries.

Figure 3 displays the process of using Bundle with a system.

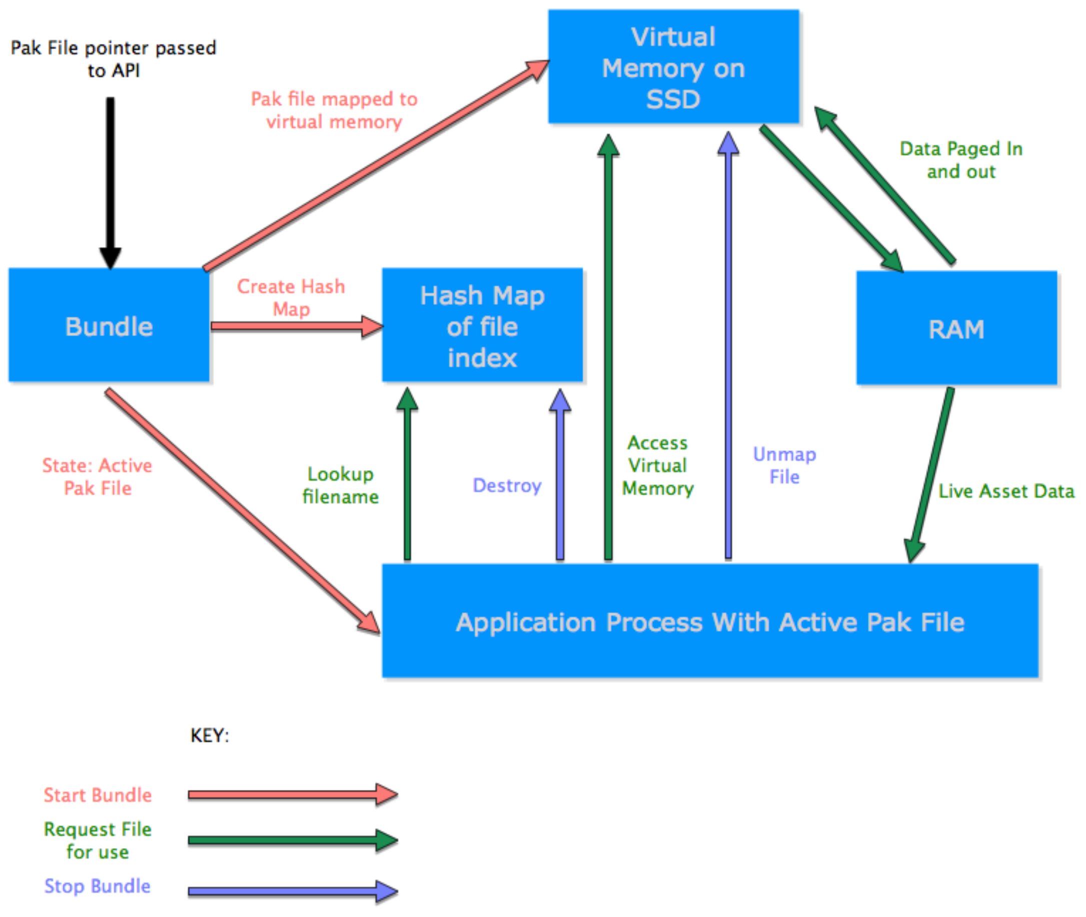


Figure. 3

7. Validation

Testing occurred at every stage of development. Code review and testing the code was done during each development phase.

During the system test phase a number of images were packaged using the packaging tool. The generated pak file was used with Bundle's API via the command line. The pak file was loaded, hashed and mapped to memory. Each files' data was then retrieved from memory and saved to disk. The images were then compared based on size of the file and the rendered image itself.

The final test for Bundle was to compare the SSD vs HDD performance when using Bundle. A prototype iOS application targeting iPad was used to visually test Bundle. One thing to note here is that we did not test the prototype on an iPad device, but rather in the simulator. Using the simulator allowed the Bundle and the prototype to be testing on systems with different hard disks. Two computers running OSX were used for testing the prototype. Both computers had 2.4ghz i5 processors and 8GB RAM.

The test itself was loading and mapping a pak file with 49 images. The prototype consists of 49 UIImageViews that hold UIImages, which contains an NSData Pointer to the data segment in virtual memory. Two tests were done, random and sequential. The random test was a randomized set of numbers within the 49 image limit. The random test was the when running it on both systems as the image order between 1 and 49 was randomized once and used on each random test. The sequential test retrieved images 1 – 49. A time stamp was place before the data retrieval call and after the NSData object contained the non-copy version of the appropriate image data.

The retrieval time contains the access-time and seek-time for a call to retrieve an image. Access time being the time it takes to locate or access the data in virtual memory. Seek time being the time it takes to seek through the data segment until it reaches the last byte of a file. The other variable being tested with is the time is takes for retrieval; this was in microseconds, one millionths of a second.

The results were significant as they showed the SSD outperforming the HDD on the random and sequential tests.

One should note that the SSD and HDD benchmarks for the random tests were each faster compared with their sequential test. The reason for this is due to the madvise function advising the kernel to use MADV_RANDOM to optimize for random access.

The longest retrieval time in all tests was the initial retrieval, which is due to the first access to the virtual memory block.

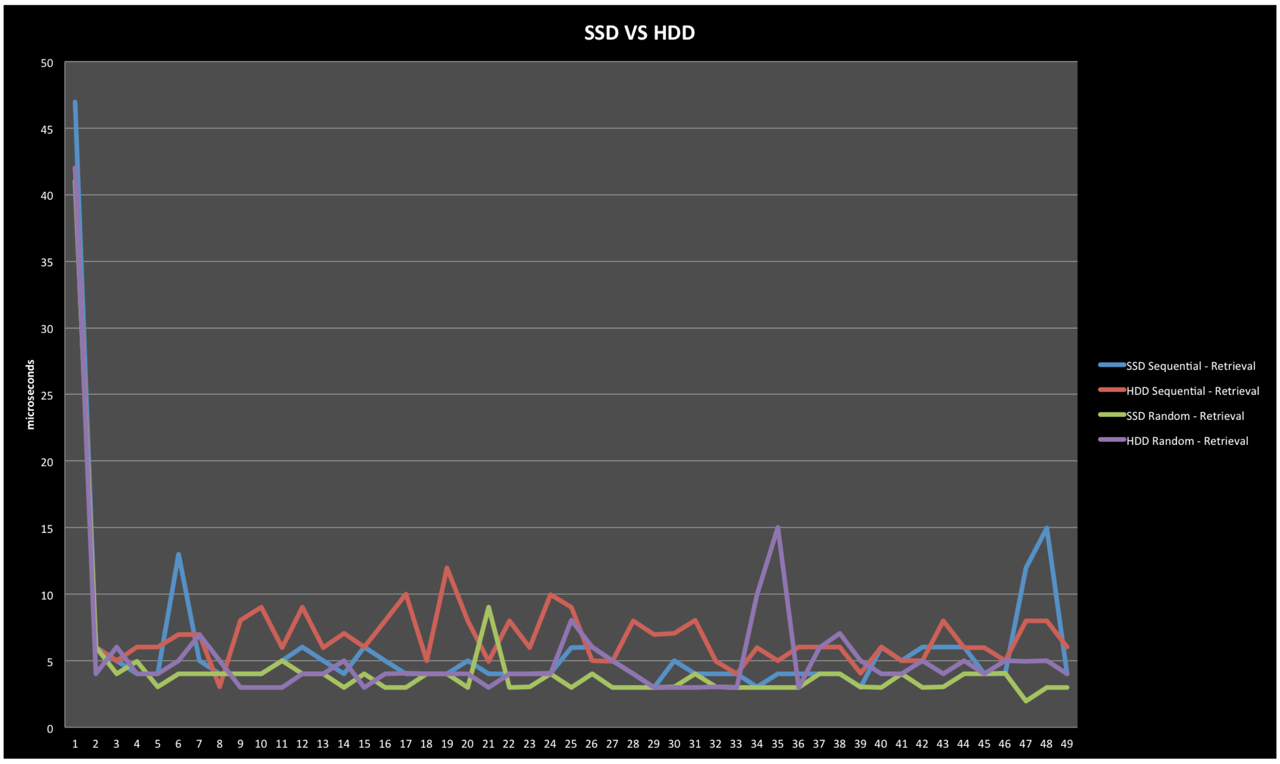
Three figures below display the test results when comparing the performance speed on a Solid State Drive and a Hard Disk Drive. The first figure shows all 4 tests combined.

These results were significant and proved part of the hypothesis that SSDs would aid in the performance of Bundle and also be an important part of why Bundle was even created. With these performance gains applications can use Bundle with significantly lower latency and faster read speeds when using memory-mapped assets.

The prototype was also testing using Instruments, which is packaged with Apple's Xcode development suite. Instruments is a powerful profiling tool for applications and offers many features and testing possibilities. Bundle was profiled checking for memory allocations in order to observe the RAM usage when using Bundle and whether Bundle does in fact decrease the memory footprint, which was the purpose of this research.

This test was loading and mapping 2 pak files, one with the 49 images and another with 7 sound files. The same test was run on the images to load 49 from virtual memory. The extra tests on for this profiling was to play a sound file from memory as well as animate the second row of the 7 X 7 grid of UIImageViews and animate each of the seven images. Each of the seven images ran an animation sequence containing the 49 images in virtual memory.

So the profiling tested all of these things at the same time to push Bundle as much as possible and see how the system performed.



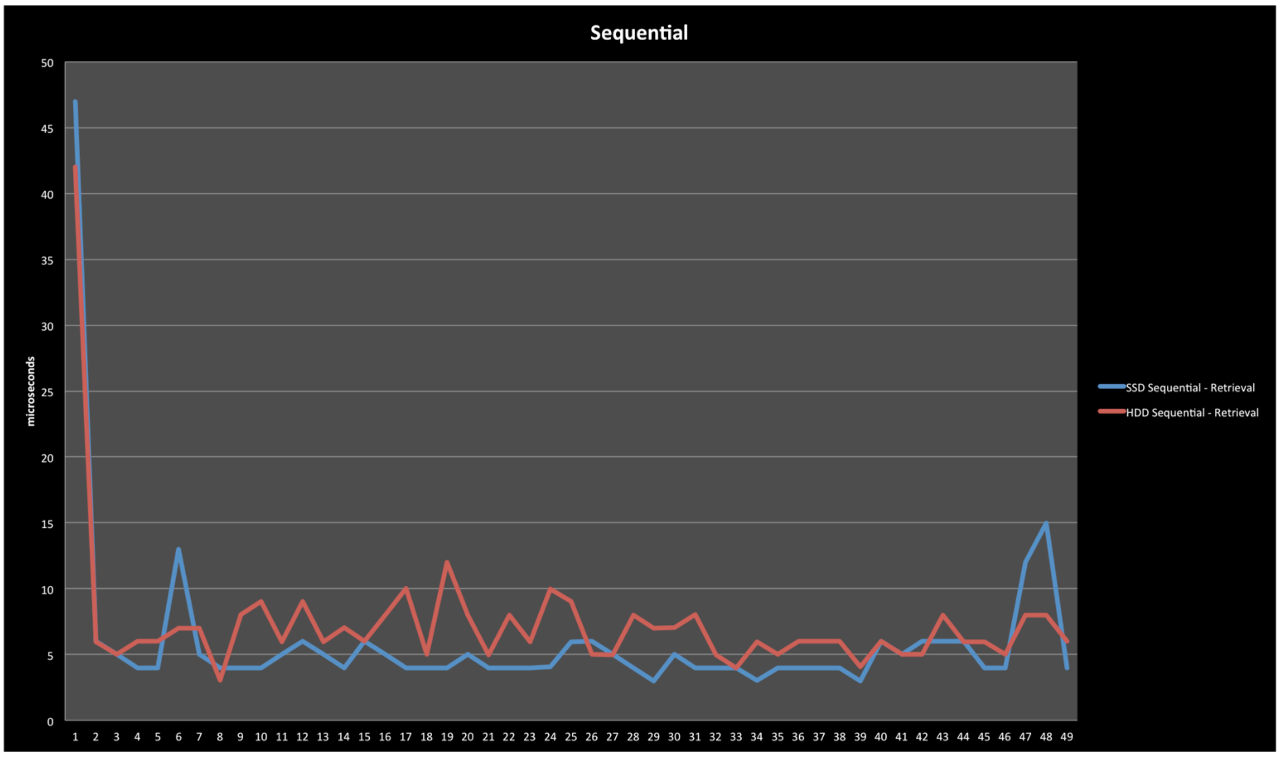


Figure 4. (top)
Figure 5. (below)

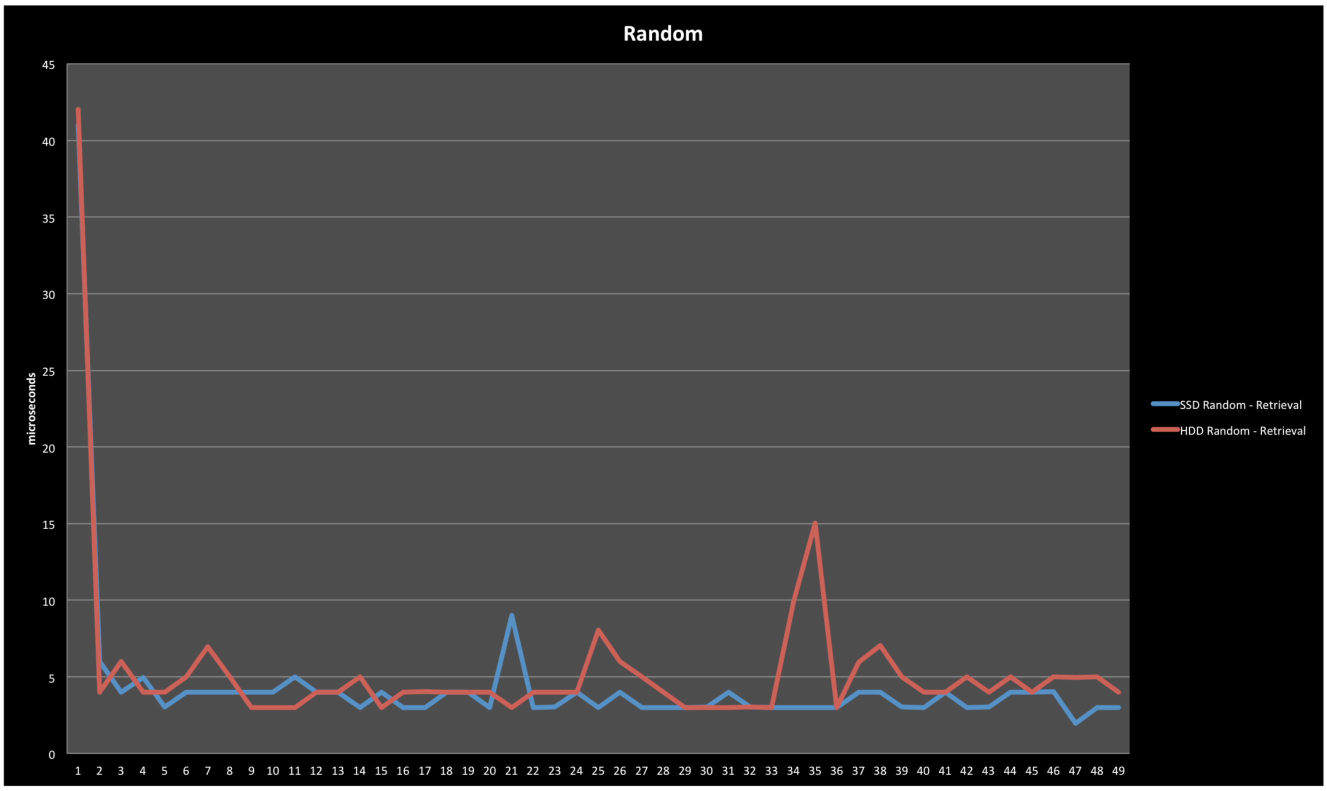
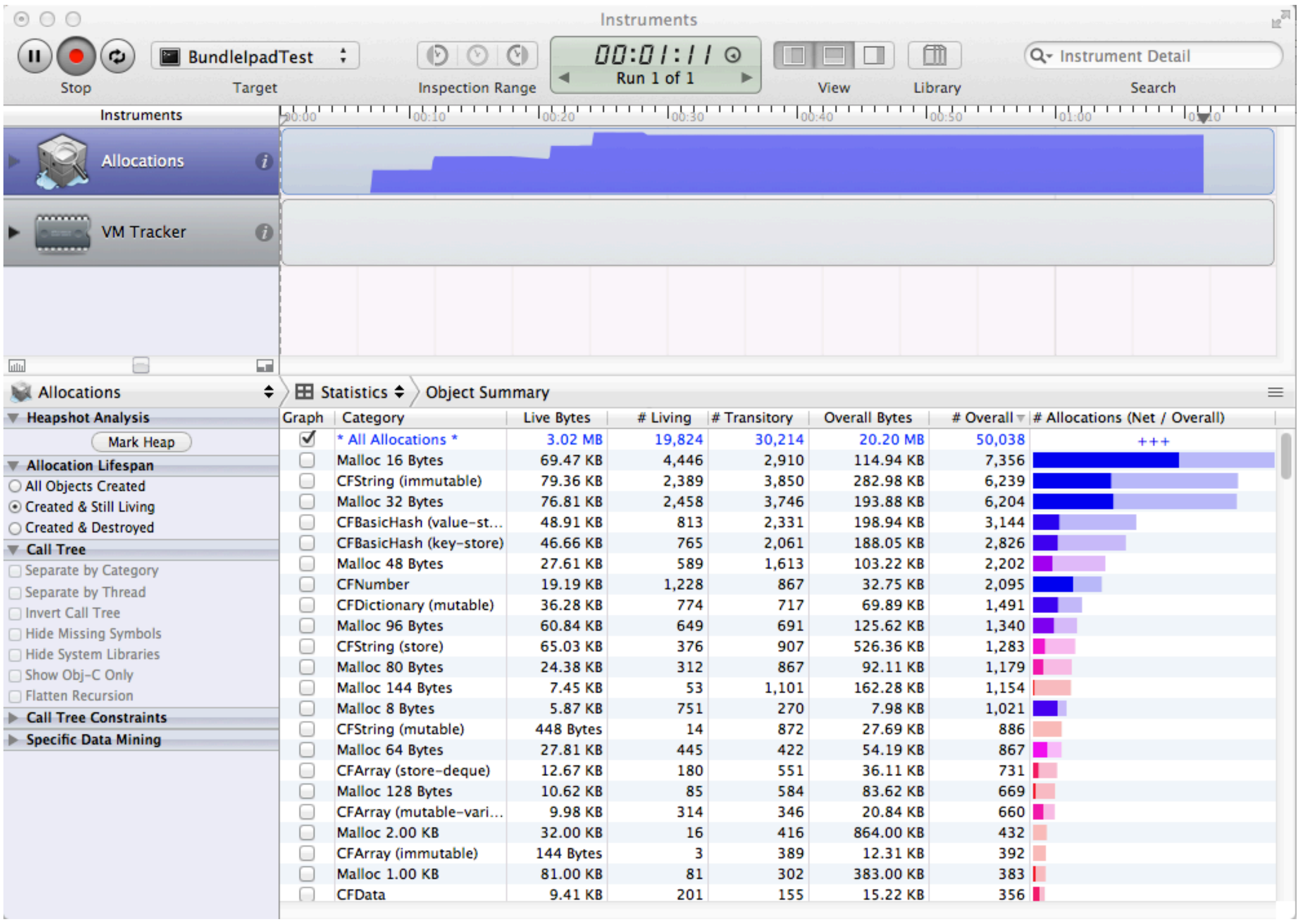


Figure 6.



Heapshot Analysis
Mark Heap
Allocation Lifespan
○ All Objects Created
[● Created & Still Living]
○ Created & Destroyed
Call Tree
☐ Separate by Category
☐ Separate by Thread
☐ Invert Call Tree
☐ Hide Missing Symbols
☐ Hide System Libraries
☐ Show Obj-C Only
☐ Flatten Recursion
Call Tree Constraints
Specific Data Mining

Graph	Category	Live Bytes	# Living	# Transitory	Overall Bytes	# Overall	# Allocations (Net / Overall)
✓	* All Allocations *	3.02 MB	19,824	30,214	20.20 MB	50,038	+++
☐	Malloc 16 Bytes	69.47 KB	4,446	2,910	114.94 KB	7,356	
☐	CFString (immutable)	79.36 KB	2,389	3,850	282.98 KB	6,239	
☐	Malloc 32 Bytes	76.81 KB	2,458	3,746	193.88 KB	6,204	
☐	CFBasicHash (value-st...	48.91 KB	813	2,331	198.94 KB	3,144	
☐	CFBasicHash (key-store)	46.66 KB	765	2,061	188.05 KB	2,826	
☐	Malloc 48 Bytes	27.61 KB	589	1,613	103.22 KB	2,202	
☐	CFNumber	19.19 KB	1,228	867	32.75 KB	2,095	
☐	CFDictionary (mutable)	36.28 KB	774	717	69.89 KB	1,491	
☐	Malloc 96 Bytes	60.84 KB	649	691	125.62 KB	1,340	
☐	CFString (store)	65.03 KB	376	907	526.36 KB	1,283	
☐	Malloc 80 Bytes	24.38 KB	312	867	92.11 KB	1,179	
☐	Malloc 144 Bytes	7.45 KB	53	1,101	162.28 KB	1,154	
☐	Malloc 8 Bytes	5.87 KB	751	270	7.98 KB	1,021	
☐	CFString (mutable)	448 Bytes	14	872	27.69 KB	886	
☐	Malloc 64 Bytes	27.81 KB	445	422	54.19 KB	867	
☐	CFArray (store-deque)	12.67 KB	180	551	36.11 KB	731	
☐	Malloc 128 Bytes	10.62 KB	85	584	83.62 KB	669	
☐	CFArray (mutable-vari...	9.98 KB	314	346	20.84 KB	660	
☐	Malloc 2.00 KB	32.00 KB	16	416	864.00 KB	432	
☐	CFArray (immutable)	144 Bytes	3	389	12.31 KB	392	
☐	Malloc 1.00 KB	81.00 KB	81	302	383.00 KB	383	
☐	CFData	9.41 KB	201	155	15.22 KB	356	

Figure 7.

The profiling results of figure 7 show that the RAM usage was 3.02MB and the overall bytes were 20.20MB. This overall bytes value is the combined RAM and virtual memory bytes. It is important to note that the virtual memory bytes within this variable are the bytes used by the system for virtual memory data. This could be sound caches done internally when playing sounds, image caches for the animations etc. The pak data itself that is backing the images, sounds and animation sequences resides on disk and it read in as needed.

This profiling showed that the RAM usage was at maximum of 3.02MB when running the prototype with Bundle as the resource system for the assets. The loaded pak files totaled 126.6MB, which itself is enough to easily cause a crash on an ipad device as the allowed RAM is well below this. That is also the reason why the testing was not done on an application running bundle as well as not running bundle. The focus was to determine how low the memory footprint could be decreased on a system that uses Bundle. The 3.02MB of RAM usage is mostly for the User Interface, UIImageViews, UIImages and the NSData objects that are needed to reference and use the data within virtual memory.

8. Future Research and Development

Bundle was developed within a short time frame and although developed for multiple platforms, the focus was on the iOS and OSX platforms. There are a number of possible directions for the Bundle project.

8.1 "Hooking in" to FILE IO

Zlib offers stream-based decompression. In some cases decompressing on the fly and reading the data byte by byte is faster than decompressing the entire file first. Bundle includes a function that creates another file when decompressing the source. Currently this decompressed file will reside in RAM, either on the stack or heap. Solving this issue by storing the file in virtual memory was considered, however both techniques store an extra version of the source file, the compressed source and the decompressed additional file. Zlib streams would allow this data to be used as it is being compressed rather than writing all decompressed data to a separate file first. Researching how to "hook in" to C's File IO functions would allow for the single copy of the asset data to reside in virtual memory and due to the data being executed as a stream, maintain a low RAM based memory usage for compressed files.

Compressing all the game assets into the archived file would decrease the pak file size. It is important to note that Bundle currently offers compression support for user selected file types because certain file types such as image and sound files that are already optimized in size, e.g. png, ogg, mp3 etc. will not benefit from further compression. Implementing these research findings would also speed up the pak file creation, as the user does not need to enter all the file types needed for compression.

8.2 Virtual memory based script execution

The POSIX mmap function, which was used for memory mapping in Bundle, offers the *PROT_EXEC* option to allow memory-mapped data to be executed. Scripts archived within the pak file could be executed directly from virtual memory.

Extending the Bundle API to allow execution of various script types existing in virtual memory during runtime is a possible future research direction.

8.3 Wrappers for various frameworks

Bundle targets multiple platforms that support C. Currently developers using Bundle can obtain a pointer to a memory location, the size of the file segment and a compression flag. Extending Bundle by creating wrappers for various frameworks would attract more developers to the project.

Wrappers for the following frameworks could be considered for future research.

- openGL
- openAL
- Cocos2d
- Android

Extending iOS wrappers is a highly recommended future research area.

8.4 Gui Packaging Tool for various platforms

Research and development of Graphical User Interfaces for the packaging tool is an important extension, as it will speed up the archive file creation process.

Developing the GUI as a cross platform solution is the most suitable direction for this research direction.

8.5 Automated iOS project conversion

A useful extension to be researched is an automated process of detecting asset usage within an iOS project. A script could scan the source code of a project and look for filenames that match files within the archived pak file. The script could then replace the native language's methods passed in argument with the wrapper function for retrieving the file data from virtual memory, with the return of the wrapper function as the argument to the native method.

Researching the different methods that use asset filenames as arguments is necessary to achieve effective automation of this process.

9. Discussion & Conclusion

During this research a software solution was created to answer for the original research question, "How can we decrease the memory footprint?" Bundle was implemented and validated showing results that prove the hypothesis and can reduce the memory footprint of the resources or assets using in a process, as close to the size of a page 4KB.

The fact that there are no open source solutions for the problem Bundle was intended to solve, Bundle fills a gap in the software development field. The open source release of Bundle is also a contribution to the community, as the source code, documentation and development are made directly available to the development community.

The constructed solution developed in C was also integrated with iOS and OSX to validate Bundle's usage with a media driven application. The scope of the project had to be limited to go beyond iOS and OSX.

An important characteristic of Bundle was to allow the developers who use it to have the freedom to develop games as usual, with the difference that Bundle would handle their asset memory management. Offering the memory address pointer and file size of the requested filename allowed this to be maintained as a characteristic of Bundle.

Bundle offers developers the following:

> Reduced memory management
> File name based data retrieval
> Lower memory footprint
> Larger memory size available
> Selected file types are compressed
> Open source project allows modifications for custom changes

Reduced memory management is achieved by removing the need for developers to strictly allocate and deallocate memory for the game assets. The game assets are mapped to virtual memory as a single file, and the individual game assets are located within this single memory mapped pak file. All memory management after mapping the file to virtual memory is handled by the kernel. The kernel will page data in and out as needed by the game's process.

File name based retrieval works by passing the given filename to the Bundle API and retrieving the data offset and file size to the developer for an intended use. Simplifying this process hides the implementation from developers and does not waste development time due to complications.

Lower memory footprint is due to the game assets being stored in virtual memory that resides on disk. The kernel will manage the RAM usage of game asset data as it pages data in and out whenever the process demands data on a given page. Strict game asset based object allocation and deallocation of memory by the developer during runtime is removed and handled by the kernel.

Larger memory size is available because of the allocated virtual memory data segment done by the kernel. On the iPhone 4, the physical memory is around 40MB depending on what other applications are running on the device, such as e.g. Apple's Mail, Safari, Address book applications as well as other applications with live processes on the device. The virtual memory on the iPhone 4 is around 700MB. Utilizing this memory space directly means the application can allocate a substantially larger data amount, which can be used as needed, which the kernel will page the data in and out of RAM.

Selected file types are compressed during the packaging stage. This has been made easily extendable to allow for any file types to either be compressed or left as is during the packaging process. Certain game development APIs offer various functions that could either work with compressed files or uncompressed files for highest efficiency. Allowing the developer to customize what file types can be compressed or not allows Bundle to integrate with the project under development.

Open source projects allows for custom modifications depending on developer needs. Game development is a very dynamic topic and there are numerous ways of developing games. Maintaining the freedom to work according to the project at hand is important and releasing Bundle as an open source project allows this. Developers can extend, modify, extract parts of the Bundle source code and use it to their custom needs.

Bundle is intended for use on systems with solid state drives and the growing availability of Solid State Drives at affordable prices will see the number of devices using Solid State Drives increase. This increase in SSDs allows Bundle to be used on more devices. Systems that do not use Solid State Drives are not ideal targets for Bundle as these systems' drives would have a longer latency and loading times and have negative effects on game performance.

It is important to note that Bundle works with game assets such as image files, sound files, textures, stage data, video files etc. These assets are stored within virtual memory and the objects using this data do not need to allocate memory for these objects. The developer might allocate E.g. A UIImageView object that can hold a UIImage to display is manually allocated by the developer and resides on RAM. The UIImage object to be placed within the UIImageView for display will have its data backed by the virtual memory segment for the named file.

Figure 8 displays an example of the objective-c object hierarchy when using Bundle.

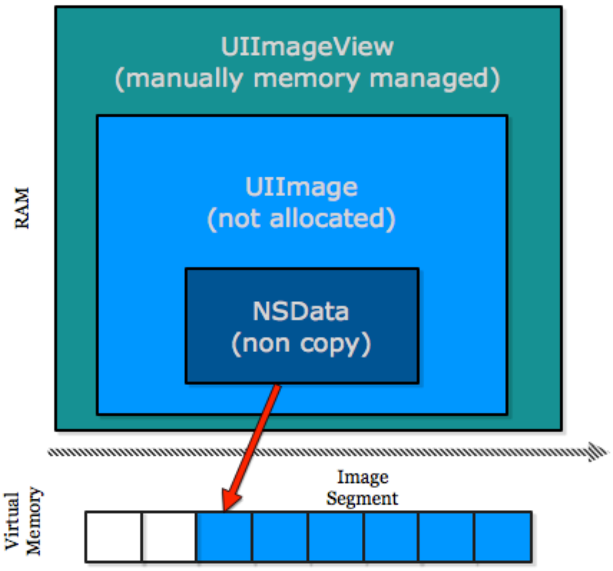


Figure 8.

Once the tool was packaging assets correctly into a pak file and the API could correctly place the pak file's header info into a hash table and memory map the file to virtual memory, the focus was on reading this data and making it available to the caller. An issue arose regarding what to do with the needed compressed data. The issue was that if a number of compressed files are within the pak file, and mapped to virtual memory, when using this data for objects, the data should be used directly from virtual memory, rather than copying the data temporarily to the stack, whether it was decompressed or not. A number of options were considered.

1. Package the game assets into the pak file without compression.
2. Package the compressed assets into the pak file, and decompress the entire file to virtual memory, re indexing an internal structure.
3. Package the compressed assets into the pak file, and decompress data as needed.
4. Package the compressed assets into the pak file, and decompress and memory map individual files on demand.

All of the above had side effects for the solution such as loading times or decreased memory size. The main issue with the mentioned options was the fact that they all forced a temporary duplicate of the virtual memory mapped data to be placed somewhere else in memory, either the stack or the heap. Keeping with the focus of reducing strictly allocated RAM usage to a minimum.

Bundle can also be used for in-app purchasing systems or other content distribution systems. Storing the pak files on a server and delivering this content to your applications as needed. An application can receive content directly from the server, which could then be used with bundle for performance enhancements, using the pak files as needed. Dividing content according to the design can benefit an application in numerous ways.

Game assets or application resources can be separated into pak files for e.g. characters, levels, maps, UI elements with accompanying plists or xml files for dynamic UI transitions, sound bundles, video resources, etc, can all be delivered on a demand based manner.

This allows server based updates of application content all benefiting from the packaging process and virtual memory management of the pak file. The pak files can be provided from servers as a product through purchasing systems.

This is important, as there are limits on the size of an application in order to provide it as a 3G based download. Apple limits an applications size to 20MB over 3G networks (Panzarino, 2012), anything larger will force the entire application to be downloaded over wireless connection.

Creating the application logic and system design under the 3G max download size limit and delivering content to the application using pak files allows the application to be downloaded over 3G. This size limit is not applied to the pak files to be downloaded via a server. In a mobile market being able to provide content to all users is important and having an application over the 20MB size limit will affect the amount of downloads.

Android uses apk files to deliver content to an application. "Android packages contain all the necessary files for a single Android program. These include the code (.DEX) files, resource (.ARSC) files, and the AndroidManifest.xml file." (File Info, 2011)

This allows Bundle to package a file and give it the apk extension. This apk can then be hosted on Google's server which an android game or media driven application that will use Bundle can retrieve the apk file and use it with Bundle's API.

Android uses a Java based virtual machine known as the Dalvik Virtual Machine, which is a JVM enhanced for mobile devices. Android developers can use the Android Native Development Kit (NDK) to run Bundle within their android game or application. "The NDK allows you to implement parts of your applications using native-code languages such as C and C++" (Android Developers, n.d).

It's important to note that CRAMES works on a lower level than the target of Bundle. CRAMES registers itself with the kernel as a memory block and offers a function by which the kernel can access the compressed data whenever there is a read or a write operation. (Lekatsas, H et al., 2005) Both of these mentioned systems use compression of file systems themselves and target small memory systems.

Another related work discussed is Apple's bundles. These allow the assets and files to be retrieved and used to allocate data to RAM. This differs to how Bundle maps the pak file, which is similar to an OSX or iOS Bundle, to virtual memory and allows similar retrieval of files within that pak file, using a filename based system. Another difference is that an OSX and iOS bundle " is a directory with a standardized hierarchical structure that holds executable code and the resources used by that code." (About Bundles, 2010) and is not a system where the data is made readily available to the developer within memory by using filename retrieval. The pak file created by the archiving tool will be added to an Xcode project and will be placed within the projects bundle. Retrieving the pak file by name using its filename will fetch the pak file within the bundle during runtime and allow the Bundle API to memory map it and use its archived files.

It should be noted that iOS allows multitasking which keeps applications threads alive in a suspended state. These suspended applications retain their processes' memory blocks within RAM and virtual memory. This retained memory can decrease the amount of memory available to a launched application. If an active application requests memory within the allowed amount and it is not available the kernel will deallocate the suspended applications as needed. Using Bundle to retrieve the needed data from the virtual memory block and using that data as a stream would minimize the RAM usage. RAM usage would only be as large as the data buffer used for streaming.

Van Waveren, (2009) mentioned a number of issues experienced when using texture virtualization.

They are explained below:

- *Texture Filtering*

Texture filtering is a technique that uses nearby texels (pixels of a texture) to determine the color for the current texture pixel being rendered. Texture filtering is related to computational complexity and the quality of the rendered image.

- *Thrashing*

Thrashing occurs when a system's physical memory is completely occupied and the virtual memory subsystem is continuously paging data in and out of physical memory. This causes the performance of the system to decrease or can lead to a system crash.

- *Level OF Detail (LOD) Transitions Under High Latency*

When streaming texture data from virtual memory, high latency can cause the detail of the rendered texture area to fluctuate depending on the latency level.

Although the above-mentioned issues relate to a large texture file, the same lessons can be applied to a project that uses Bundle, as the pak file can hold numerous files of various types, all of which can be streamed.

Insufficient Memory Addresses within virtual memory is also an issue to be aware of. In order for a file to be memory mapped to virtual memory there needs to be enough address space available for the file to fit. A file will not be successfully mapped to virtual memory if there are not enough memory addresses available within a continuous data block, even if the total amount of free virtual memory is larger than the size of the file to be mapped.

Another caveat regarding memory mapping is that memory mapping files smaller than the page size on a system wastes space. A page size is usually 4kb in size. A file smaller than 4kb will still consume 4kb of RAM space when paged in by the kernel.

Packaging two files with the same filename into one pak is possible, though, this causes the duplicates in the hash table, resulting in retrieving the last file corresponding to the filename requested, ignoring equally named files in the pak.

The constructive research method contributed greatly to this research. Its principles were found suitable for this kind of study, as it focuses on the solution and fills the learning gap throughout the process of development. (Crnkovic, 2010)

The flexibility provided by the constructive approach allows mixing social and scientific approaches, (bid). On the other hand, freedom in interpreting knowledge during the construction of the solution made it harder to try to adapt to a suitable research method or framework that describes the exact development process.

The goals of the constructive approach are shared with many other similar approaches. The "Action Research" and the "Design Research" both aim to creating a link between theory and practice. Nevertheless, several attributes that exist in the previously mentioned approaches do not seem to appear in the constructive approach steps.

The iterative nature that exists in the Design and Action research approaches enables knowledge gaps to be filled iteratively by reflected lessons learned during the implementation phase. (Suhonen , 2009)

Borrowing this special property from sibling approaches adds flexibility to the working process followed in the framework (especially in phase 2).

The Semi structured interviews contributed in gaining knowledge in a continuous and a flexible manner. Topics related were discovered iteratively during every meeting as open discussions were taken place.

Analysis of data is continuous as more meeting sessions are set and more topics are outlined. Interviews are limited in time thus resources cannot be covered fully during meetings. Topics that may contribute to the implementation were discussed in depth, though, leaving detailed specifications and further knowledge to be gained individually from written literature about these topics, observations from experiments, questions and answers during the meetings.

As the constructivism used in this research requires knowledge to be translated into novelty (Crnkovic, 2010). Knowledge collected was interpreted as software requirements for the implementation process.

The transformation from data collected to software requirements specifications is done by understanding the data using the search of building technical knowledge about the topics involved, and using the existing knowledge and technical skills, build diagrams, setup a development strategy using scheduled milestones tied to programming tasks.

In the first steps of the research, an understanding of an unsolved problem in the field of computer gaming is achieved, and a solution to this problem is proposed during the meeting. The solution is not discussed in details; several techniques to solve the problem were discussed rather than a provided detailed list of implementation steps.

A different view of the constructive research method is described by (Nunamaker et al. 1990), and reflected in steps which might somewhat look more stabile to this research. The steps structured seem to lack social data collection methods, which are necessary steps in the development of this research.

This paper presented the research and implementation of a solution to a current issue within the software engineering field. The benchmark data also shows the importance of SSDs in terms of performance.

10. Acknowledgements

This work was done with guidance from Senri. We would like to thank Johan Knutzen for his guidance and support throughout the research and development of Bundle.

We would like to thank Lennart Petersson for supervising our thesis.

References:

About Bundles. 2010. Available: https://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFBundles/AboutBundles/AboutBundles.html#//apple_ref/doc/uid/10000123i-CH100-SW1. Last accessed 24th April 2012.

Android Developers. (n.d). What is the NDK?. Available: http://developer.android.com/sdk/ndk/overview.html. Last accessed 21st May 2012.

App Store Metrics. 2012. Available: http://148apps.biz/app-store-metrics/. Last accessed 10th May 2012.

Apple. 2011. NSData Class Reference. Available: https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSData_Class/Reference/Reference.html. Last accessed 21st May 2012.

Amit Singh 2006. Mac OS X Internals: A Systems Approach. Massechusetts: Pearson. 181.

Apple. 2008. About the Virtual Memory System. Available: https://developer.apple.com/library/mac/#documentation/performance/conceptual/managingmemory/articles/aboutmemory.html. Last accessed 21st May 2012.

Automatic Reference Counting. 2012. Available: http://clang.llvm.org/docs/AutomaticReferenceCounting.html. Last accessed 8th March 2012.

Bundle. 2012 Bundle Available: http://jarryddev.github.com/Bundle/. Last accessed 21st May 2012

Crnkovic, G., D. 2010. Constructivist Research and Info-Computational Knowledge Generation.

Caplinskas, A., and Vasilecas, O. 2004.I nformation systems research methodologies and models.

Core Services Layer. 2012. Available: https://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreServicesLayer/CoreServicesLayer.html. Last accessed 19th May 2012.

Cramfs. 2011. Cramfs. Available: https://github.com/wendal/cramfs. Last accessed 21st May 2012.

Doom 3. 2011. Doom 3. Available: https://github.com/TTimo/doom3.gpl. Last accessed 21st May 2012.

Doom 3. 2011 FileSystem.cpp. Available: https://github.com/TTimo/doom3.gpl/blob/master/neo/framework/FileSystem.cpp. Last accessed 21st May 2012.

Ellis, T., J., and Levy, Y. 2008. Framework of Problem-Based Research: A Guide for Novice Researchers on the Development of a Research-Worthy Problem. Volume 11. Fort Lauderdale, Florida, USA. P.22

File Extension Guide, 2012, “File extension pk4“. Available: http://pk4.fileextensionguide.com/. Last accessed 19th May 2012.

File Info. 2011. .APK File Extension. Available: http://www.fileinfo.com/extension/apk. Last accessed 21st May 2012.

File Extensions Library, 2008, “File extension pk3 information”. Available: http://pk3.file-extension-library.com/. Last accessed 19th May 2012.

File Extension Guide, 2012, “File extension pk4“. Available: http://pk4.fileextensionguide.com/. Last accessed 19th May 2012.

File Info. 2011. .APK File Extension. Available: http://www.fileinfo.com/extension/apk. Last accessed 21st May 2012.

Bryman, A., and Bell, E. 2007. Business research methods.

Floridi, L. 2004. Informational Realism. Third move: the concept of a structural object is empty. P.6.

Harrell, M. C., and Bradley, M. A. 2009. Data Collection Methods: Semi-Structured Interviews and Focus Groups.

Holton, D., L. 2010. Constructivism + Embodied

Cognition = Enactivism: Theoretical and Practical Implications for Conceptual Change. P.2.

John Carmack. 2010. John Carmack discusses RAGE on iPhone/iPad/iPod. Available: http://www.bethblog.com/2010/10/29/john-carmack-discusses-rage-on-iphoneipadipod-touch/. Last accessed 16th March 2012.

Lekatsas, H.; Dick, R.P.; Chakradhar, S.; Lei Yang; , "CRAMES: compressed RAM for embedded systems," Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on , vol., no., pp.93-98, Sept. 2005

Lindholm, A., L. 2008. A constructive study on creating core business relevant CREM strategy and performance measures. Showing the theoretical contributions. P.356.

Lindholm, A., L. 2008. A constructive study on creating core business relevant CREM strategy and performance measures. P.345.

Mayer, J, A 2010, ‘Virtual Texturing’, M.A. thesis, Technischen Universität Wien, Wein

madvise. 2008. madvise. Available: http://man.he.net/man2/madvise. Last accessed 21st May 2012.

Matthew Panzarino. 2012. ipad 3s retina display mans trouble for many apps due to apples 20mb 3g download limit. Available: http://thenextweb.com/apple/2012/03/02/ipad-3s-retina-display-mans-trouble-for-many-apps-due-to-apples-20mb-3g-download-limit/. Last accessed 20th May 2012.

Pete McCormick. (n.d). Quake PAK Format. Available: http://debian.fmi.uni-sofia.bg/~sergei/cgsr/docs/pak.txt. Last accessed 1st May 2012.

Kernel.org. 2010. mmap(2). Available: http://www.kernel.org/doc/man-pages/online/pages/man2/mmap.2.html. Last accessed 21st May 2012.

Van Waveren, J.M.P, From Texture Virtualization to Massive Parallelization, SIGGRAPH 2009, New Orleans, pp. 1 – 18.

Suhonen, J. 2009. Week 5: R&D methods in computing: action research, development research, design research and constructive research. Introduction

McClure, R., D. 2002. Common data connection strategies effective in qualitative studies using action research in technical/operational training programs.

Nunamaker, J., F.,Chen, M., and Purdin, D., M., T. 1990. Systems development in information systems research. .