**Packaging and streaming assets to decrease the memory footprint in video games**

Jarryd Hall, Taher Odeh

Table of Contents

1. INTRODUCTION 2

2. Related Work 2

3. Method 3

4. Bundle 6

4.1 Bundle and Automatic Reference Counting 7

5. Bundle Design 8

5.1 Design Principles 8

5.2 Design Overview 8

5.2.1 Packaging Tool 8

5.2.2 File Format 9

5.2.3 API 9

6. Bundle Implementation 9

6.1 Tool and Pak File Creation 9

6.2 The API 10

7. Result 11

8. Future Research 11

9. Discussion & Conclusion 11

10. Acknowledgements 12

References: 12

**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 paper the focus was on the OS and Mac OSX platforms

**General Terms**

Video Games, Memory Management, Cross-Platform, Game Performance, Optimization

**Keywords**

Virtual Memory, mmap, Packaging Tool, API, RAM, SSD, File-System

# 1. INTRODUCTION

Almost all video games are bound by memory constraints. The system running the game can only allocate a certain amount of usable memory 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 the HDD, which is where the virtual memory is located, and often the assets and game resources are very large, consuming a lot of the available RAM.  
  
These memory constraints affect game developers and designers working within the development process of a project.

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 RAM footprint. The development cycle is usually guided by budget and cost, and especially within indie projects and smaller game development project, 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. This paper describes the open-source solution created to solve these memory issues and presents the results. An open-source solution was created so that indie game developers who are not aided by proprietary techniques can create better games. Indie developers are often unaware of the technical aspects surrounding memory handling by the system and allowing these memory constraints to impact a development project can yield unwanted results. Game development is dynamic and 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 problem above was pointed out during an initial interview with Johan Knutzen, a founder of Senri and Phobic-Games, mobile application and game development companies based in Gothenburg, Sweden, which are strongly involved in this research paper and it’s requirements and solutions.

The solution developed included 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, which is either compressed or uncompressed. The API allows developers to use the generated pak file, as well as handles the virtual memory allocation and file retrieval.

In order to test the developed solution, a sample game was run numerous times with and without the solution present. The game was profiled and monitored for memory and performance variables and the data was compared.

The questions researched in this paper was:

* How can the memory footprint of video games be decreased?
* Can an SSD offer random data access in an efficient manner?

# 2. Related Work

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 AAA 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 game called Rage.(add reference to this here <http://www.bethblog.com/2010/10/29/john-carmack-discusses-rage-on-iphoneipadipod-touch/>)

CRAMES is a system to enhance memory on embedded systems. (Add reference here to CRAMES paper) Compressed RAM for Embedded Systems uses efficient algorithms as a RAM compression technique.

“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.” (add reference to CRAMES paper)

CRAMES uses a filesystem for compressed data segments in RAM and targets low-power embedded systems and offers read and write data access.

The paper on CRAMES (reference paper here) also acknowledges “Cramfs”, acronym for cram a filesystem onto a small ROM. Cramfs (add reference to https://github.com/wendal/cramfs) 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. (add reference <https://github.com/wendal/cramfs>)

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. (add reference to CRAMES paper) Both of these mentioned systems use compression of file systems themselves and target small memory systems.

Id Software games use a number of archived file types for handling game assets. Quake 3 uses PK3 files and Quake 4 uses PK4 files. (add reference to <http://en.wikipedia.org/wiki/PK3_%28file_extension%29>)

The PAKfile 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 (http://en.wikipedia.org/wiki/PAK\_%28file\_format%29). 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.

Pete McCormick wrote a description of the PAK format explaining its header and directory information. (add reference <http://debian.fmi.uni-sofia.bg/~sergei/cgsr/docs/pak.txt>)

id Software released Doom 3’s source code (add reference to <https://github.com/TTimo/doom3.gpl>) 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 decending 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.“ (add reference to <https://github.com/TTimo/doom3.gpl/blob/master/neo/framework/FileSystem.cpp>)

A MegaTexture is the creation by id Software’s technical director John Carmack. A megatexture holds a large number of smaller textures. MegaTextures allow a game to use a resource pool that is larger than the available memory of the system. This technique involves mapping the megatexture to virtual memory and streaming data on demand for rendering within the game engine. J.M.P. van Waveren mentions a number of issues to the use of megatextures that were faced during the development of the id Tech 5 game engine, (add reference to “From Texture Virtualization to Massive Parallelization” pdf) such as:

* *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 continuously paging data in and out of physical memory, causing the performance of the system to decrease or 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.

These issues arise due to the nature of how megatextures are used. Data is streamed in a constant manner from the megatexture and the quality of the system’s hardware is connected to the severity of the issues.

OSX and iOS Bundles:

“Bundles encapsulate related resources in a hierarchical file structure but present those resources to the user as a single entity. Programmatic interfaces make it easy to find resources inside a bundle.” (add reference to https://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/OSX\_Technology\_Overview/CoreServicesLayer/CoreServicesLayer.html)

“Bundles are a fundamental technology in Mac OS X and iOS that are used to encapsulate code and resources.” (add reference to [https://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFBundles/Introduction/Introduction.html#//apple\_ref/doc/uid/10000123i-CH1-SW1](https://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFBundles/Introduction/Introduction.html))

OSX and iOS Bundles organize assets and code files into directory structures for “a more natural type of organization”. (reference [https://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFBundles/Introduction/Introduction.html#//apple\_ref/doc/uid/10000123i-CH1-SW1](https://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFBundles/Introduction/Introduction.html))

These bundles however 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.” (add reference [https://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFBundles/AboutBundles/AboutBundles.html#//apple\_ref/doc/uid/10000123i-CH100-SW1](https://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFBundles/AboutBundles/AboutBundles.html))

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.

# 3. Method

Crnkovic, (2009) describes the constructive research method as a way to turn existing knowledge into novelty or innovation by implementing a solution, to an existing problem, whether it is theoretical or practical, as long as it involves the usage of preexisting knowledge and thinking to produce artifact design solutions, for example, plans, diagrams, charts or software implementation; it may be considered constructive.

Kasanen, 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 of the solution and using the data within the testing phase for analysis purposes.

Additionally, 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; or 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 mentioned method has been found the most suitable for this research, basing on the fact that the latter aims to cover a feasible constructivism in terms of finance and freedom by having the product implemented for this research free of charge as well as it is freely licensed.

Caplinskas (2004) argues that the constructive research method befits the computer science and the IT related problems in a usual manner.

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

1. Finding a research problem.
2. Examining the potential for long-term target organization research co-operation.
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.

Based on the fact that is acceptable in the field of research to use a method’s ground theory and proclaim hybrid mythologies.[ <http://www.nova.edu/ssss/QR/QR12-2/anaf.pdf>]. The fundamental steps above have been borrowed and minimized to best fit a three-phased framework that shall bring a clearer image of the mix of method attributes used within this research.

The framework is mainly concerned with the development process involved within this research, that in order for it to be constructive, it has to follow the principles of the method, though reporting what data collection and analysis techniques and methods are used throughout the process and in what stage. (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. [http://www.inform.nu/Articles/Vol11/ISJv11p017-033Ellis486.pdf page 22] argues 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 details knowledge has to be gained while planning the development, prior to the technical process which is bolder in the next phase.

Step 2 in the list is skipped due to its irrelevancy to this research and its time scope [might want to mention in future work or discussion].

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.

Followed, still within the same phase, an implementation of the software is developed and tested for bugs. Thus, covering steps 3, 4 and 5 of a constructive method.

Phase 2 is based on an iterative process, in each iteration, as 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 form the product or optionally, a need to modify previous tasks to better suit the new constrains.

The goals of the constructive approach are shared with many other similar approaches, like the “Action Research” and the “Design Research” for example. 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.

[describe what is an iteration]

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. [http://cs.joensuu.fi/pages/suhonen/SciMet2009/week5\_R&D\_methods.pdf]

Borrowing this special property from sibling approaches adds agility to the working process followed in the framework. Explain more how agility is connected [http://www.rallydev.com/sites/default/files/mastering\_the\_iteration\_an\_agile\_white\_paper.pdf]

In the third and final phase, [edit] 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, 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 – diagram showing the phases described]

**Data Colletion:**

A mix of both qualitative and quantitative data collection methods are used in this research at different stages to achieve different goals.

Mixing of approaches aids in evaluating results that are not theoretical or that does not rely on literature. [http://www.sagepub.com/upm-data/15565\_Chapter\_4.pdf] Thus making data being collected and analyzed in a more flexible manner, prior, during, and after the development process.

[show why its ok to mix in constructive research]

Any data collection method is acceptable as long as it seen helpful for gathering information about the constructed project or in its early phases prior to development, which is an important step in the process. The research is less likely to be ready to take off without having solid ground information about the subject researched. [http://www.mrtc.mdh.se/~gdc/work/MBR09ConstructiveResearch.pdf]

Qualitative data in this research is collected

using in-depth semi-structured interviews, which are described by [http://www.sagepub.com/upm-data/15565\_Chapter\_4.pdf ] as a useful and efficient way to find literature about hypothesis or claims heard or read by the interviewee.

[http://www.rand.org/content/dam/rand/pubs/technical\_reports/2009/RAND\_TR718.pdf] argues 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 to be further researched.

“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.” [http://www.evokedevelopment.com/uploads/blog/commonData.pdf]

The interviews mentioned are run with the CEO of “Senri”. Several ways of data gathering are taken during the interview sessions which are estimated approximately an hour time: dialogues, questions and answers, development guidelines, discussions and suggestions about sources to look into 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 the [company] to describe the problem existing in the field. During the first meetings an understanding of why the problem exists is reached in details covering the scope of it, subjects related to the problem and its solution are noted down to be researched later, similar solutions in the market are talked about and an explanation of how their approach is given for a later research.

During phase number two of the framework, semi structured meetings are scheduled every two weeks 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 accoun.

[write more about the data collected during interviews]

Case study observation is also another qualitative data collecting method used within the all the phases of the process framework. [http://www.ijazconsulting.com/uploads/Qualitative\_Research\_by\_Ijaz\_and\_Dr.Shea.pdf] stresses that data collected during a case study is often structured and organized since data sources may be various and vary in context.

Through-out the iterative process in phase 2, knowledge is gained as the product gets developed. [http://www.mrtc.mdh.se/~gdc/work/MBR09ConstructiveResearch.pdf] argues, the constructive research method leads to creating an artifact and learning by building it, thus making researches gain more knowledge and information about topics involved in solving the problem.

Since this research follows the constructive method principles, learning from of technical experiences plays an important role in data collecting. As the problem involves multiple areas of research, more knowledge is gained as the learning gap increases for more topics to be discovered and more skills to be gained.

As mentioned previously, a mix of methods is used for data collection during this research as part of solving the problem discussed in this paper.

Quantitative data is collected mainly in the last phase of the process framework. [http://www.insites.org/CLIP\_v1\_site/downloads/PDFs/TipsQualQuanMthds.4B.8-07.pdf] describes quantitative data as numerical and statistical aiming to answer a question and coming up with conclusions.

“Quantitative methods help to answer questions such as who, how much, and how many.” [http://documents.wfp.org/stellent/groups/public/documents/ko/mekb\_module\_13.pdf]

One of the goals of quantitative data collection methods used in a mix is to show whether statistical outcomes validates or invalidates variables. [http://documents.wfp.org/stellent/groups/public/documents/ko/mekb\_module\_13.pdf]

[Explain what quantitative data collectios are used]

**Analysis:**

**[Analysis of qualitative data goes here]**

**[Analaysis of quantitative data goes here]**

# 4. Bundle

Bundle is an archiving tool as well as an API for using the produced archive files and allowing the archived files 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 (add reference to github here), releasing the product under a Bsd License or similar (add reference to Bsd License here). 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 gain the interest of the game developers community and 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. 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.

Bundle can aid game development projects that are already under development or at any other time during the development phase. Developers can import and use it from the start of a project or integrate it during development.

Bundle works by compressing a game’s assets by filename into a single archived file. The file is then loaded into virtual memory using the Bundle API. The files within the archived file are then 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.

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.(add swap file reference here).

The storage of memory on disk and using this virtual memory during a game’s process is especially important on devices using Solid State Drives. Solid State Drives work similar to RAM. Solid State Drives have lower access times and lower latency than older hard drives. (add SSD reference here)

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. Solid State Drives thus allow faster access to the virtual memory mapped files as compared to traditional hard drives.

Once the file is mapped to virtual memory the addresses are translated as the data is paged in from virtual memory.

ADD info here about Paging etc.

A MMU(add MMU reference here) is used by the CPU to translate virtual memory addresses.

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. (add profiling diagrams? Or some reference). 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. (add virtual memory reference to apple’s site here) . 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.

Virtual memory became the focus at this point to be the basis of the solution. 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, e.g. around 700MB on iPhone 4, compared to the iPhone 4’s 40MB RAM.

A POSIX function called mmap (reference mmap here) is a low-level virtual memory mapping technique that allows one to allocate a file on disk into virtual memory addresses.

As mmap could map a file to virtual memory for use within a running process, creating a file for a game’s assets became the focus at this point.

## 4.1 Bundle and Automatic Reference Counting

ARC aka. Automatic Reference Counting is a new LLVM 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.” (add reference here to ARC <https://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreServicesLayer/CoreServicesLayer.html>)

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.” (add reference to <http://clang.llvm.org/docs/AutomaticReferenceCounting.html>)

This does not affect an iOS or OSX application that uses Bundle because the objects used for the game assets within this type of application are not strictly allocated, 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 to use for developers 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, stopping Bundle.
4. *Allow file retrievals in a random access manner.* Using the *madvise* function with the MADV\_RANDOM argument allows the virtual memory mapped pak file to be accessed randomly with readahead turned off. (add reference to http://man.he.net/man2/madvise)

## 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. (add reference here to C platform target) C compiles correctly with Objective-C, the language used in iOS and OSX. (add reference to C compiling with obj c)

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

### 5.2.1 Packaging Tool

The packaging tool takes a source folder and a destination pak file name, without the need for the pak file to exist beforehand. The tool packages each individual file into the pak file and indexes the file information into the header. (add info about header here and file format design)

Figure 1 shows the packaging process to create a pak file.



figure 1.

### 5.2.2 File Format

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 number of archived assets as well as a data section that holds the files with byte padding which is sized depending on the architecture of the system, 32 bit or 64 bit. The header contains the filename, offset and file size for each archived file within the pak file.

### 5.2.3 API

The API is a static library that developers can include and use within their projects. The usage of the API is split into 3 basic 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 and returns success or failure.

The *bundle\_getIndexDataFor* function retrieves the file offset and size 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.

A wrapper is also available to developers who need Objective-C specific objects. The function

*id bundle\_useFile:(NSString \*) fileName forObject:(id) object*

takes a filename and an object pointer as arguments, then retrieves the data from virtual memory using the Bundle API. The retrieved data is then passed to a native function for NSData and creates an object, which the developer can then use to hold their game assets. NSData is an object that holds binary data and can be used as arguments to methods requiring other object types. (add reference to NSData here and also an example maybe of UIImageView using NSData)

ADD diagram here to show how bundle works on a high level.

# 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. (add reference to pak file here) (possibly add info about other games that use this as well as battlefield that uses it for every level, having multiple pak files)

A tool was then developed to archive these game assets into a single pak file. This was created as a command line tool that takes two arguments, the source folder and the output 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(reference DS\_Store). 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 20 bytes of information needed to be used to locate files in the archive, the first 4 bytes in an offset holds 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 remaining 8 bytes in an offset hold the size of the that file in bytes. 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.

Each asset within the source folder, e.g. sound files, textures, images etc. is compressed using zlib(reference zlib here) and indexed within the pak file, with its index information stored in the header segment. (reference file format creation link)

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.

## 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 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.(add reference here to objc and cocoa/touch) Objective-C is a superset of C, allowing it to integrate seamlessly.

The wrapper function was only needed to pass the needed data’s pointer, the pointer to the data in virtual memory, to the object needing it within an objective-C runtime environment.

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. (add reference here to the NSData class)

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

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. (add a possible function call and small explanation of the process)

This feature would make it as simple as possible for developers to use the API as they can use their filenames as usual without needed an understanding of how the file’s data is passed to the called NSData method.

A hashmap implementation was developed based on khash(C)(add reference) 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 hashmap is initialized globally on the stack and is structured to hold the hash values of the filename (add reference to hashing function) as the key, and the offset copied from the header as the value.

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

The use of the API was minimized to 3 important functions:

* int bundle\_start(char \*pakFile, struct mappedData \*mData)
* offset\_p bundle\_getIndexDataFor(char \*fileName)
* int bundle\_stop(struct mappedData \*mData)

The first starts Bundle by hashing and memory mapping a given pak file. The second function is the one that is always used directly or from within a wrapper method to retrieve the needed data pointer and size, that, if called from a wrapper can be passed to the native languages method. The third function stops Bundle, by destroying the hashmap and unmapping the file from virtual memory. Allowing such ease of use of the Bundle system developers do not need to lose valuable development time, as another important attribute of Bundle is to improve the development process in various ways. Developers can integrate Bundle quickly into existing games.

Add information here about the crossroads we came to with the different options regarding compression and decompression, or will that come in the discussion?

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 hashmap 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, decreased memory size, memory fragmentation which would lead to no memory being available of a large enough size for a needed block, even if the total free memory was more than enough. 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, the stack would be the option. If a game had n objects all with their source data memory mapped in virtual memory and a duplicate of the decompressed data in memory, it would not be a solution to a problem as much as it is reading from a file and using the same amount or memory.

This led to a solution where filetypes needed to be defined as to how they are used. Defining which file types needed to be decompressed or not during runtime would allow the most dynamic and efficient way to handle the data for use by objects.

The game assets would be packaged as compressed files into the pak file and when the bundle\_start function is called, the assets filetypes would need to be analyzed and depending on the type, either decompressed into virtual memory or left compressed.

The reason for this is that certain functions or methods that use this data work more efficiently with compressed data; others work more efficiently with uncompressed data.

Give some examples of the file types?

# 7. Result

# 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 to undertake.

* *Compression of all assets and “hooking” in to FILE IO while decompressing on the fly*.

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 a file when decompressing a file. 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.

(add section on C File IO hookin)

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.

* *Header encryption using private keys*

Project based private key encryption of the pak files header would allow the archived game assets to be sold as a file downloads. This would protect the header from being accessed and the internal files from being extracted. Whether this file is downloaded and the application using the file allows the user to load it on demand or if the file is downloaded using an in-app purchase system, the file will be secure from theft and only the application loading the file can open and use it with the provided private key.

* *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 dirction.

* *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.

* *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.

* *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 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.

Android uses apk files to deliver content to an application. “Android application package file (APK) is the file format used to distribute and install application software and middleware onto Google's Android operating system. To make an APK file, a program for Android is first compiled, and then all of its parts are packaged into one file. This holds all of that program's code such as (.dex files), resources, assets, certificates, and manifest file. As is the case with many file formats, APK files can have any name needed, but must end with the four character, three letter extension, .apk.” (reference <http://en.wikipedia.org/wiki/APK_%28file_format%29>)

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++.” (add reference to <http://developer.android.com/sdk/ndk/overview.html>)

# 9. Discussion & Conclusion

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 will 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

> Compression of game assets

> 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.

Compression of game assets is done during the packaging stage. The source folder containing all the game assets is traversed and each file is compressed, indexed and placed within the pak file. Reducing the game assets size also decreases the memory mapped file size.

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 adapt to 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 for this. Developers can extend, modify, extract parts of the Bundle source code and use it to their custom needs.

The growing availability of Solid State Drives at affordable prices will see the number of devices using Solid State Drives increase which means Bundle can be used on more devices. The number of Solid State Drives in mobile devices is increasing which makes them perfect deployment targets for Bundle as mobile devices are usually affected by the RAM limitation due to smaller data storage capabilities. 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 images, sound files, textures, stage data 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.

# 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.

# References:

[ref2 = Kasanen et al. (1991)]

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

Crnkovic, G., D., 2009. *Constructivist Research and Info-Computational Knowledge Generation*

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

<http://www.inform.nu/Articles/Vol11/ISJv11p017-033Ellis486.pdf>

Android Developers, “What is the NDK?,” Available: <http://developer.android.com/sdk/ndk/overview.html>.