The architecture and evolution of computer game engines

University of Oulu
Department of Information Processing Sciences
B.Sc thesis
Rainer Koirikivi
12.3.2015
Abstract

In this study, the architecture and evolution of computer game engines are analyzed by means of a literature review on the academic research body on the subject. The history of computer games, from early 1960s to modern day is presented, with a focus on the architectures behind the games.

In the process, this study will answer a selection of research questions. The topics of the questions include identifying the common parts of a game engine, identifying the architectural trends in the evolution from early to present-day games and engines, identifying ways the process of evolution has affected the present state of the engines, and presenting some possible future trends for the evolution.

As findings of the study, common parts of a game engine were identified as the parts that are specific to every game, with the suggestion that more detailed analyses could be made by concentrating on different genres. Increase in the size, modularity and portability of game engines, and improved tooling associated with them were identified as general trends in the evolution from first games to today. Various successful design decisions behind certain influential games were identified, and the way they affect the present state of the engines were discussed. Finally, increased utilization of parallelism, and the move of game engines from genre-specific towards genre-neutral were identified as possible future trends in the evolution.

Keywords
computer game, video game, game engine, game, software architecture, architecture, evolution
Foreword

I'd like to thank my thesis supervisor Jouni Lappalainen for his continued support on what turned to be an epic journey into the fields of game engines and academic writing. Especially, I'd like to thank him for directing me to dig in the history of game engines outside the norm of the academic research, namely the FPS genre.

Writing this thesis has been a long project, but a worthwhile one. In the process, I have learned a great deal about not only computer games and the engines behind them, but also about academic writing, the process of a scientific literature review, and, most importantly, perseverance.

Rainer Koirikivi
12.3.2015
Oulu, Finland
# Contents

1. Introduction..............................................................................................................5  
   1.1 Research problems............................................................................................5  
   1.2 Method................................................................................................................5  
   1.3 Focus and limitations........................................................................................6  
   1.4 Structure.............................................................................................................6  
2. What is a game engine?..........................................................................................7  
3. The architecture and evolution of computer game engines....................................9  
   3.1 Early games.........................................................................................................9  
   3.2 Adventure games...............................................................................................10  
   3.3 Platformers and other third-person games.......................................................14  
   3.4 First-person shooters.......................................................................................15  
   3.5 Role-Playing Games..........................................................................................18  
   3.6 Games of other genres.....................................................................................21  
   3.7 Multi-genre engines.........................................................................................22  
4. Analysis....................................................................................................................24  
   4.1 What are the common parts of a computer game engine?...............................24  
   4.2 How have the architectures of game engines evolved over time?..................25  
   4.3 How has the evolution affected the present state of game engines?..............26  
   4.4 What future trends can be identified in the evolution of game engines?.......28  
   4.5 Advances in parallel architectures...................................................................28  
   4.6 Decrease in genre-specificity............................................................................29  
5. Conclusion...............................................................................................................30  
   5.1 Limitations and ideas for further research.......................................................30  
6. References...............................................................................................................31
1. Introduction

Computer games have come a long way, from Spacewar of 1962 (Anderson et al., 2008) to securing their place among the biggest media of the 21st century, even surpassing the movie industry in terms of gross revenue (Fairclough, Fagan, Mac Namee & Cunningham, 2001). Technologically, a computer game is just like any other computer program and is as such made up from modules of source code. A game's engine, as defined by Bishop et al. (1998) and Lewis & Jacobson (2002), refers to that part of the modules that's separated from the game's content: behavior or environment. Like a car's engine, the game engine is the technological core of a game, on which the game's content is built.

As pointed out by Anderson et al. (2008), it's apparent that there's a need for research in the domain of game engine architectures. The purpose of this research is to do just that: to study the architecture and evolution of computer game engines.

1.1 Research problems

This research aims to answer the following research questions:

- What are the common parts of a computer game engine?
- How have the architectures of game engines evolved over time?
- How has the evolution affected the present state of game engines?
- What future trends can be identified in the evolution of game engines?

1.2 Method

This research is conducted as a literature review on the published material on the subject.

The material for the review will be collected from the databases ACM Digital Library1, IEEE Xplore2, Scopus3 and Google Scholar4. The keywords used will be game, engine, architecture and structure. The keywords will be combined as seen fit.

The search results will be searched for articles with relevant-looking titles. From these, the articles with abstracts that seem to be about either the architecture of games or game engines or a supporting subject will be selected for full text review.

Additional reference material may be found by looking at the sources of the found ones, by further searching on topics that emerge from the already found material, or by peer recommendation.

---

1 http://dl.acm.org/
2 http://ieeexplore.ieee.org/Xplore/
3 http://www.scopus.com/
4 http://scholar.google.com/
According to Anderson et al. (2008), the existing literature on the subject is scarce and, where it exists, tends to be limited, for an instance focusing too much on the low level implementation of the engines, dismissing the high-level architectural description. However, it's the firm belief of the author that an understanding of the present theoretical knowledge is an absolute necessity before conducting any new research.

1.3 Focus and limitations

This research will focus solely on the technological side of games' architectures. As such, articles focusing on game theory, the educational use of games, game studies (ludology) or other unrelated subjects will not be considered to a great extent (though they may be searched for information related to the research questions, or for other sources). Studies focusing on highly specific technological parts of the engines (e.g. path finding or optimization) will be considered similarly.

This research paper will also not contain an in-depth analysis on the history of computer games, rather focusing on the engines of the games. Hence, many notable events in the history of computer games will without a doubt be omitted from the “Architecture and evolution” section.

1.4 Structure

The next chapter will feature a definition of a game engine. After that follows a look into the architectures of game engines at different times, and of the evolution process. This will be followed by an analysis chapter, where the research questions will be answered, and a conclusion chapter.
2. What is a game engine?

There is some confusion about the exact meaning of a game engine (Anderson et al., 2008). A common definition seems to be that of Lewis and Jacobson, where a game engine is defined as the “collection of modules of simulation code that do not directly specify the game’s behavior (game logic) or game’s environment (level data)” (Lewis & Jacobson, 2002). Among the engine's modules, Lewis and Jacobson list input handling, output (3D, 2D and sound) and generic physics and dynamics for game worlds. A similar definition is made by Bishop et al. (1998), of the game engine as the modules that have no effect on a game's actual content, including input, audio, graphics, dynamics and an event loop.

The term “data-driven” is sometimes used to describe the separation of game content from game software (Demers, Gehrke, Koch, Sowell & White, 2009). Gregory (2009) argues, with some caution, that a data-driven architecture is what differentiates a game engine from a piece of software that is a game but not an engine, and suggests reserving the term “game engine” for extensible software that can be used for different games without major modification. Demers et al. (2009) distinguish “the abstract game engine”, programmed by software engineers, from content and character AI created by designers who often have little programming experience.

Other research, such as that by Munro, Boldyreff & Capiluppi (2009), doesn't explicitly define the engine or its separation from the game's content. Anderson et al. (2008) point out that the boundary between a game engine and game logic is still not clearly defined. A similar view is offered by Gregory (2009), who also points out that some engines make the distinction between the engine and the game more clear than others.

The reuse of game engines between different games seems to be a common theme on the written literature, and a key motivation for separating the engine of the game content (Lewis et al., 2002; Bishop et al., 1998; Blow, 2004; Anderson et al., 2008; Gregory 2009). Motivations for reuse are plentiful. Designing and developing software components is a costly and time-consuming process, and as such developers of a game are usually economically better off investing in a ready-made engine instead of developing their own (Bishop et al., 1998; Gregory, 2009). Reusing good-quality components also improves the overall quality of the whole product (Mili, Mili & Mili, 1995). There is also plenty of room for reuse since building new software inevitably results in “reinventing the wheel”, duplication of code estimated as high as 85% with previously created software (Mili et al. 1995). This is also true with game engines, and indeed, at the time of writing, the Wikipedia “List of game engines” page lists over 250 different game engines, of which over 120 are free to use (“List of game engines”, n.d.).

In spite of the theme of reuse, Bishop et al. (1998) point out that in practice game engines are usually tuned to a specific game's content style, which impairs the reuse of the engine in games of a different style. Gregory (2009) agrees, saying that most engines are crafted to run a particular game on a particular platform, and adds that the more general purpose an engine is, the less optimal it is for running a particular game on a particular platform. More generally, Brooks (1975) estimates that building reusable software components takes three times the effort of building single-use components. As such, while it often makes sense to reuse existing game engines, it does not make sense
for everyone to build their own reusable engines, or for every game engine to be reusable (or, indeed, even separate from the other parts of the game).

This research will focus on the architectures of game engines as separate entities from the game content, but will fall back on the architectures of whole computer games as necessary, for an example with older games where the engine doesn't necessarily exist as a separate entity.
3. The architecture and evolution of computer game engines

This chapter provides an in-depth look to the history of computer games and the architectures behind them. The chapter is laid out in a semi-chronological manner, with the games analyzed grouped by genres.

3.1 Early games

The title of the first computer game is usually attributed to Spacewar (1962), which featured two player-controlled spaceships represented on primitive 2D display (Fleming, 2007; Anderson et al., 2008), though simpler games existed even before (Graetz, 1981). Early computer games consisted of simple 2D graphics on low-resolution screens (Bishop et al., 1998), a trend that continued until the 1990s.

The hardware on which the early games were run was very limited. This forced developers to take every bit of processing power out of the system, for an example by programming the games in assembly language. (Bishop et al., 1998) Blow (2004) describes game engineering in the past being mainly about low-level optimization.

Because of the limited hardware, the architectures of early computer games were minimal, consisting – as described by Lewis and Jacobson (2002) – of little more than an event-loop, state tables and graphic routines. The games shared little to no reusable elements with each other, which meant that new games were usually programmed from scratch (Anderson et al., 2008; BinSubaih, Maddock & Romano, 2007).
3.2 Adventure games

A glimmer of early engine-like architecture can be seen in early adventure games, the first of which was Colossal Cave Adventure. Originally released by Will Crowther in 1975 and later expanded by Don Woods in 1977, “Adventure” marked the start of the text-based interactive fiction genre, in which the gameplay consists of reading a textual representation of the setting and typing simple commands to interact with it. Text-based adventures were immensely popular among early games, and later, with graphics incorporated, spawned the graphical adventure genre. (Jerz, 2007)

The architecture of Adventure was, as can be suspected, very simple, with Bishop et al. (1998) describing it consisting of “little more than game logic in the form of a state table, minimal level data […] and a wrapper hardly recognizable as an event manager.” Indeed, the game was comprised of two files, one 700-line file of FORTRAN code and another of textual data. The data file includes most of the text displayed to the player, vocabulary keywords, game states, and map data. The code file is, of course, responsible of getting input from the player, parsing it and reacting according to the data file. (Jerz, 2007)

Despite the simplicity of Adventure's architecture, one can clearly see elements of the data-driven design, described by Demers et al. (2009), typical to game-engines: The game's content is clearly separated from the code and indeed, it can be seen that one could in principle even create modified versions of the game just by altering the data and leaving the code alone.

Some successors to Adventure, all heavily influenced by it, were Adventureland (Adventure International, 1978), Zork (originally written by a couple of MIT students in 1977, later released as three different games by Infocom in 1980-1982), and Mystery House (On-Line Systems, 1980). (Jerz, 2007; Veugen & Quérette, 2008; Anderson, 1985). While the mechanics of Adventureland were almost identical to the original Adventure, Zork and Mystery House spawned two branches in the development of adventure games (Williams, 1996). Zork and other Infocom's games were still text-only, but featured a greatly improved command parser (Williams, 1996), while Mystery House featured rudimentary graphics (though player commands were still input as text), starting the genre of graphic adventure games (Williams, 1996; Jerz, 2007). Still, Infocom's text-based games were, for a while, more popular, and Ken Williams of On-Line Systems later remarked that the graphics processing on the limited hardware of the
time took away from the depth of game play that could have otherwise been presented (Williams, 1996).

3.2.1 Zork, ZIL and the Z-Machine

While the original Zork ran on a mainframe computer and was written in a dialect of the Lisp programming language called MDL, the commercial port to the personal computers of the time prompted Infocom to separate the game's platform from its content. The platform became the Z-Machine, a virtual machine that ran the content written in a specifically crafted scripting language called the Zork Implementation Language (ZIL). (Gouveia, 2012; Bartholomew, 2008).

In-depth descriptions of the Z-machine, or rather it's implementation, the Z-language interpreter program (ZIP), and ZIL are provided by Berez, Blank & Lebling (1989) and Infocom (1989) respectively. For this paper, a shorter description is sufficient:

Programming in ZIL deals mostly with two tasks: describing objects and programming routines. Objects are defined in a declarative syntax, such as in the following example, courtesy of Infocom (1989):

```plaintext
<OBJECT LANTERN
  (LOC LIVING-ROOM)
  (SYNONYM LAMP LANTERN LIGHT)
  (ADJECTIVE BRASS)
  (DESC "brass lantern")
  (FLAGS TAKEBIT LIGHTBIT)
  (ACTION LANTERN-F)
  (FDESC "A battery-powered lantern is on the trophy case.")
  (LDESC "There is a brass lantern (battery-powered) here.")
  (SIZE 15)>
```

Routines, on the other hand consist of sets of instructions (opcodes) with which the language communicates with the interpreter program. (Infocom, 1989). The interpreter, ZIP, then is responsible for providing these instructions and thus abstracting the operating system from the programs written in ZIL (Berez, Blank & Lebling, 1989).

Zork's architecture enabled Infocom to both reduce the size of the game program (important with the limited hardware) and to ease the porting of the games to new hardware platforms, which only required re-writing the Z-Machine, leaving content files untouched. (Gouveia, 2012; Bartholomew, 2008). The engine was also reused for other Infocom's games (see for example Infocom, 1989).

The development of the Z-Machine stopped with the demise of Infocom, but it was later reverse engineered by Graham Nelson, which resulted in the creation of the Inform language, which began a path of its own. Later developments of Inform have concentrated on making the language resemble natural English even more. (Bartholomew, 2008).

3.2.2 AGI and SCI

While Infocom concentrated on text-based adventure games, On-Line Systems, eventually to be known as Sierra On-Line and later Sierra Entertainment, continued to work on their graphic adventure games. A major development in the genre was the 1983 release of King's Quest (Sierra On-Line, 1997), born out of a deal with IBM and Sierra On-Line to showcase the improved hardware of IBM's new personal computer, the PCjr.
This allowed King's Quest to feature advanced graphics, animations, sound effects and music. (Williams, 1996; Veugen & Quérette, 2008).

King's Quest was also the first game to feature the game engine known as Adventure Game Interpreter (AGI), which would be eventually used by many of the games by Sierra (Gouveia, 2012; Black, 2012). Like the Z-machine, AGI simplified porting the King's Quest and other games built on top of it to other platforms (Gouveia, 2012).

The literature review didn't yield any results for peer reviewed or official documentation of AGI's internal architecture. However, a community effort⁵ to reverse-engineer the engine and to continue development of fan-made games was found, which includes an in-depth specification of the engine's internals by Kelly et al. (1999), which we can use with some considerations.

AGI's architecture, as described by Kelly et al. (1999) consists of an interpreter (the engine) and resource files (the game content). The resource files can be multimedia (backgrounds, animated pictures, sounds), game objects, dictionary words (used by the interpreter), or code (written in the Logic programming language). During startup, the interpreter loads the first Logic (code) resource in memory and enters the main loop, during which it gets input from player, updates objects in memory, executes game logic and updates the screen for the player.

AGI was followed by Sierra's Creative Interpreter (SCI) (Black, 2012). As is the case with AGI, neither official nor peer reviewed documents of the architecture of SCI were found during the literature review, but luckily again, a community effort provided fruitful specifications (by Skovlund et al., 2002) of the engine's internals.

From the outside, SCI has many resemblances with AGI, such as the fact that game content was still handled as resource files. In contrast to AGI, however, SCI was built as a full stack-based virtual machine, on top of which higher-level functions (e.g. to receive input and handle complex logic) were implemented. The programming language used in code resources was also upgraded to an object-oriented version from AGI's procedural one. (Skovlund et al., 2002)

Readers interested in more in-depth specifications of the AGI and SCI engines are advised to Kelly et al. (1999) and Skovlund et al. (2002), or the community efforts centered around the engines.

### 3.2.3 SCUMM

Meanwhile, another company called Lucasfilm Games (and later, LucasArts) was working on a graphic adventure game, Maniac Mansion (released 1987 by Lucasfilm Games) (Black, 2012). While the development of Maniac Mansion was originally started in an assembly language, it was quickly deemed too complex, and thus an engine called SCUMM (Script Creation Utility for Maniac Mansion) was written to aid development (Edge Staff, 2012).

In contrast to earlier adventure games, games built on top of SCUMM didn't feature a text parser to get input from player, but instead offered a point-and-click interface (Black, 2012; Gouveia, 2012). Similar interfaces were later incorporated by other companies, including Sierra On-Line (Shiels, 2008; Williams, 1996).

---

Architecture-wise, SCUMM featured a similar split between game content and the engine as the engines analyzed before. Game data consisted of resources (such as rooms) and an index (used for navigation), stored either in a single file or split in two. (Black, 2012)

As was the case with Infocom's Z-Machine and Sierra's AGI and SCI, SCUMM didn't just make development of games easier – it greatly simplified both porting games created on it to new platforms and creating new games altogether (Black, 2012). Through its lifetime, SCUMM was revised and improved multiple times and numerous games were built on top of it, some examples including The Secret of Monkey Island and Day of The Tentacle (Black, 2012; Gouveia, 2012).

The end of SCUMM finally came in 1998, when LucasArts switched SCUMM to their new engine, GrimE, built for the game Grim Fandango. GrimE supported 3D graphics and a keyboard-based controller scheme (Gouveia, 2012). However, SCUMM still lives today in the form of the reverse engineered open-source ScummVM implementation (Black, 2012).

* * *

In retrospective, its easy to see why adventure games adopted data-driven architectures and the use of reusable engines early on: The genre was relaxed of several requirements of other games at the time, such as fast-moving graphics or varying game play mechanics. Since adventure games can be very different (in terms of story and puzzles) while having identical game play, it also made sense to separate the core functionality from the content.

Later incarnations of adventure games have varied with the style of their graphics and user interface, with some incorporating first-person 3D graphics and others keeping the traditional point-and-click 2D style. Various toolkits for adventure game creation, such as Adventure Game Studio and Visionaire, have also appeared on the market. (Furini, 2008). Other styles of 2D and 3D games and their engines, architectures of which may be relevant when adventure games too, are discussed in a later parts of this chapter, while more discussion about the present state of game engines is provided in the next chapter.
3.3 Platformers and other third-person games

The term “platformer” refers to the variety of games, generally in the third-person perspective, where the game play consists of jumping from platform to platform (Gregory, 2009).

The genre arguably began with Space Panic (Universal, 1980), which featured a character that moved on a single screen, dodged aliens, climbed up and down ladders and dug holes in the ground. Following games like Donkey Kong (Nintendo, 1981) added additional features, like the ability to jump. Even further developments occurred with games like Super Mario Bros. (Nintendo, 1985), that replaced the single-screen mechanics with side-scrolling ones and added other features. (Crawford, 2003)

As is often the case with the architectures of early games (with the exception of games in the adventure genre), literature on the architectures of early platforms is scarce. Blow (2004) offers some insight, naming the following parts of "one 1990s 2D game", while pointing out that games of other genres would likely consist of different parts, gaining for an instance a part for artificial intelligence and perhaps losing the fast graphics node.

- streaming file I/O
- sound
- main / misc.
- simulation
- fast 2D graphics

With the dawn of 3D graphics, the platformer genre evolved too, with games like Super Mario 64, Crash Bandicoot and Jak and Daxter (Gregory, 2009).

Gregory (2009) lumps the technical requirements of platformer games together with other third-person games, putting specific emphasis on the movement of camera as an engine requirement.
3.4 First-person shooters

As we can see from the previous parts of the chapter, examples of recognizable computer game engines can already be found in adventure games from as early as the 1980s. However, what many authors consider a turning point for computer games – and especially for game engines – was the birth of the FPS genre with the releases of Wolfenstein 3D (1992) and Doom (1993) by id Software (Gregory, 2009). Doom, especially, became immensely popular, starting the still continuing trend of realism in games (Lewis & Jacobson, 2002).

The architecture of Doom is also worth considering. Doom's core software components, such as graphics rendering, collision detection and audio, were separated from the game content, such as art assets, game worlds and rules of play, which meant that users could add new levels, models and other assets to the game, effectively creating new games of their own (Gregory, 2009). While the degree of Doom's programmability was minimal, and games derived from it continued to play just like the original game (Bishop et al., 1998; Lewis & Jacobson, 2002), multiple companies licensed id Software's Doom engine to produce commercial games of their own (see e.g. Arsenault, 2009). Thus, while not the first game to utilize a data-driven architecture, it is perhaps not far-fetched to attribute its popularization to Doom.

3.4.1 The id Tech series of game engines

Doom was followed by Quake (id Software, 1996), which picked up where Doom left, featuring true 3D graphics and a client/server architecture. Quake also provided a true game-independent game engine, with its inclusion of a level editor and QuakeC, a scripting language with which the game's behavior could be altered. This was a major improvement from Doom's approach of adding data files to a game executable, with minimal programmability. (Lewis & Jacobson, 2002).

Quake's engine is composed of two parts, the client and the server. All input (keyboard, mouse, joystick) and output (3D rendering, 2D drawing and sound) take place on the client, and all of the actual gameplay, e.g. player movement and physics simulation, AI and QuakeC scripts, take place on the server. So to reiterate, the client takes input from the player and sends it to the server, which processes the game forward for a fixed slice of time and then sends the game's state back to the client, which provides audiovisual output for the player. (Abrash, 1997).
Both single player and multiplayer games of Quake are built on this architecture. In multiplayer games, multiple clients communicate with a single server (usually run on a different machine) through a networking layer, whereas single player games use shared memory buffers for the communications, and each frame of processing is split between the getting input from the client, processing gameplay on the server and outputting it on the client. (Abrash, 1997).

In addition to the simplification of synchronization issues in multiplayer games (in contrast to Doom, where all players ran their own gameplay simulations and machines proceeded in lockstep), Abrash (1997) points out that the client-server architecture enforces a modular design, simplifies debugging and keeps the single player and multiplayer codes identical.

Quake was soon followed by Quake II (id Software, 1997) and Quake III Arena (id Software, 1999). The architecture and evolution of the game engines in the series was analyzed by Munro et al. (2009). It was found out, that as the engine evolved from Quake 1 engine to its third iteration, id Tech 3 (the engine for Quake 3), it grew in size both according to the source lines of code (SLOC) count and the amount of source code modules. The engine also grew more organized over time, as the source code files in the first version simply existed in one folder, whereas subsequent versions had the source files organized into a tree of subfolders.

A couple of modules were found in all versions of the engine (Munro et al., 2009):

- Common functionality module.
- Game logic module.
- Server specific functionality module.
- Client-side functionality module.
- Sound and video output module (separated in later versions).

All the engines analyzed in the study were written in the C programming language. However, it was pointed out that the fourth version of the engine, id Tech 4, has been
completely redesigned using C++ and the object oriented paradigm (Munro et al., 2009).

The latest version of the engine at the time of writing is id Tech 5, which utilizes software architecture taking advance of parallelization – utilizing multiple CPU cores – to gain even more performance and to feature even better graphics, such as support for very large textures (van Waveren, 2009). The next generation of the engine, id Tech 6, is still under research, but planned features include the use of new rendering techniques such as ray tracing, and the utilization of future-level hardware (Shrout, 2008; Rautenbach, 2012).

The Quake engines are an especially relevant subject of study, since they and their derivatives have been used in multiple other high-profile computer games. For an example, GoldSrc, the engine behind Half-Life (1998), Counter-Strike (2000) and many other prolific games released by Valve Corporation, is a heavily modified version of the Quake 1 engine (Birdwell, 1999; Bokitch, 2002). GoldSrc eventually became Source, the engine behind Half-Life 2, Counter-Strike: Source (both Valve Corporation, 2004) and numerous other games (Johnson, 2005). Thus, it can be assumed that the influences of Quake's architecture are still present in today's games.

3.4.2 Unreal Engine

A major competitor for the Quake technology is Unreal Engine, first released with Unreal (Epic Games, 1998) and subsequently used in numerous different games (Gregory, 2009; BinSubaih, Maddock & Romano, 2007).

Among the components of Unreal Engine, Väänänen (2008) identifies the server, the client, a graphics component and multiple helper components. The engine offers programmability with a custom scripting language called UnrealScript (Väänänen, 2008; BinSubaih, Maddock & Romano, 2007).

Unreal Engine was followed by subsequent releases of versions 2, 2.5, 3 and 4. The latest iteration of the engine, Unreal Engine 4, is written in C++, runs on multiple platforms and offers a full development suite for game creation (“Unreal Technology Roadmap”, n.d.; “About Unreal Engine 4”, n.d.).

Although originally designed for FPS games, Unreal Engine has also been used successfully in games of different genres (Gregory, 2009). As such, we see that the distinction of the genre of the engine starts to get blurry, and it could be argued that Unreal Engine is indeed an example of a multi-genre engine. Other multi-genre engines, many of which have been used for FPS games too, are discussed at the end of the chapter.
3.5 Role-Playing Games

Computer role-playing games (RPGs or CRPGs) blend different genres and gameplay mechanics, often resembling adventure games, but concentrate on the development of character statistics (Barton, 2008).

3.5.1 Early role playing games

The origins of RPGs can be traced back to Dungeons & Dragons, an influential desktop role playing game that spawned the system of combat statistics, which many computer games either borrowed from or copied directly (Achterbosch, Pierce & Simmons, 2008; Barton, 2008). Other influences include desktop sport simulation games and early adventure games, like Will Crowther's Colossal Cave Adventure. Indeed, many early RPGs can be seen as a combination of Adventure and the ruleset of Dungeons & Dragons, and like Adventure, were played on mainframe computers. (Barton, 2008).

Among the first mainframe RPGs Barton (2008) lists pedit5 (Rusty Rutherford, date unknown), Dungeon (Don Daglow, 1975-1976), dnd (Gary Whisenhunt and Ray Wood, mid 1970s), Moria (1978), and Rogue (Michael Toy and Glenn Wichiman, 1980s). Rogue, especially, is an interesting game to consider. Its graphics consist entirely of the character set of a computer – letters, numbers and special signs – and the dungeons in the game are procedurally generated, providing unique experiences each playthrough. Rogue became immensely popular, spawning a whole subgenre of RPGs called roguelikes. Examples of which include Hack (1982), Larn (1986) and NetHack (1987). (Barton, 2008).

Interestingly, many of the early RPGs offered features years ahead of their time, such as first-person perspective and multiplayer gameplay (Barton, 2008).

As time passed, mainframes lost popularity in favor of personal computers, the prices of which dropped while performance raised. Computer role-playing games also began to move from mainframe computers towards the personal computer market. Among the first RPGs for personal computers, Barton (2008) lists Wizard's Castle, Eamon, Space and Akalabeth: World of Doom. (Barton, 2008)
3.5.2 MUDs

One interesting development in the history of RPGs was the creation of MUD (also known as MUD1) by Roy Trubshaw and Richard Bartle in 1978. MUD, short for Multi-User Dungeon, was a text-based game that ran on a mainframe computer and resembled earlier RPGs in that gameplay consisted of fighting monsters and leveling up the skills of a character. The gist of MUD was that multiple users could play at the same time and interact with each other, both socially and in combat. (Achterbosch et al. 2008; Barton, 2008).

Earlier adventure games were an influence in the creation of MUD, and one of the key reasons for its creation was to make a multi-player adventure game. The other reason was to write an interpreter for a database definition language, and compared to Crowther & Woods's Adventure, the architecture of MUD resembles more of a “real” engine, much like those of early adventure games like Zork. MUD's game data was contained in a database file, written in a language called MUDDL (Multi-User Dungeon Definition Language), that was read by the game's interpreter. Hence, MUDDL was in a way a game engine for MUD, and indeed, other games were later produced on top of that engine. (Bartle, 1990).

The popularity of MUD resulted in the birth of a whole genre of similar games, also called multi-user dungeons or, more commonly, MUDs. Later MUDs, examples of which include Mirrorworld, GemStone, and TinyMUD, added additional features and sometimes changed the focus from combat to socializing. (Achterbosch et al. 2008; Barton, 2008).

MUDs would eventually go on to be an influence for the popular MMORPG genre, which will be discussed in a while, but to get a sense of the evolution of role-playing games, modern “regular” RPGs have to be considered first.

3.5.3 Modern role playing games

As games and graphics evolved, so did role-playing games. Modern incarnations of the genre feature not only improved graphics, but innovative new features. Some examples listed by Achterbosch et al. 2008 include replacing turn-based mechanics with real-time ones in Dungeon Master; open-ended game mechanics in Wasteland; and fully rotational 3D camera in Ultima Underworld.

A few modern RPGs also offer multiplayer game mechanics. Among early examples, Neverwinter Nights (different from Bioware's later game of the same title) featured a server supporting 200 simultaneous players and simple 2D graphics, while Diablo offered randomly generated dungeons and items. (Achterbosch et al. 2008).

The technical requirements of modern RPG games don't necessarily warrant genre-specific engines. In his comparison of engines of different genres, Gregory (2009) lumps role-playing games into the "Other genres" category, and mentions that improved hardware (making low-level optimization less of a concern) makes it easier to reuse engines for different games or genres. One engine that has been used in modern RPGs, for an example in the Elder Scrolls series, is the Gamebryo engine (former NetImmerse), which will be discussed in a later chapter ("Published titles", n.d.).
3.5.4 Massively Multiplayer Online Role Playing Games

Massively multiplayer online role-playing games (MMORPGs) are a subset of RPGs that are played over the internet with countless other players in a persistent world. Today, MMORPGs represent a fast-growing and popular section of modern computer entertainment, with games like World of Warcraft racking millions of subscribers. (Achterbosch et al., 2008).

Having their roots in both regular RPGs and MUDs, features of both can be seen in today's MMORPGs. Some examples include the class-based character development systems of RPGs (that can be ultimately traced back to D&D) and the multiplayer aspect of MUDs with its social flavors. (Achterbosch et al., 2008).

Achterbosch et al. (2008) categorize MMORPGs into two generations. First-generation MMORPGs – including games like Meridian 59 (3DO, 1996; often credited as the first MMORPG), Ultima Online (ORIGIN, 1997), and EverQuest (Sony, 1999) – were an experimental phase where a new genre was built on top of existing ones, incorporating the graphics and user-friendly interfaces of modern computer RPGs to text-based MUDs. Second-generation MMORPGs, on the other hand, generally focused on technological improvements, such as advanced graphics and interfaces, more than innovation. Examples of commercially successful second-generation MMORPGs include Eve Online, City of Heroes and World of Warcraft. (Achterbosch et al., 2008).

Architecturally, MMORPGs and massively multiplayer (online) games (MMGs or MMOGs) in general represent a major challenge in engine development. Blow (2004) characterizes MMGs as "[t]he largest endeavor we currently attempt", describing the rough architecture of one game with a complex web of 27 different game modules, and adding that there are no market-proven engines for them (though the present-day validity of that claim can perhaps be challenged considering the age of the paper). Achterbosch et al. (2008) identify MMORPG architectures as one of the four big research areas in the field of MMORPG research, describing research questions related to the amount of simultaneous players on a server, the amount of control developers of a game can expect to maintain, and up-front costs of launching.

Much of the technical challenge in building an MMGs stems from the management of network-related code, especially with a large amount of players. Gregory (2009) places emphasis on the network of servers as a part of the technical requirements of MMGs; another big research category defined by Achterbosch et al. (2008) is that of the affect of latency on an MMORPG; and Kabus, Terpstra, Cilia & Buchmann (2005) explore the option of using peer-to-peer architectures to manage server loads. In fact, it seems that the focus of architectural discussion in the context of MMGs has shifted from the client-side game engine with realistic graphics to server and networking related issues. Certainly a major development from the simplistic client-server architecture of Quake!
3.6 Games of other genres

In addition to the genres discussed above, numerous other genres of computer games exist. Among them, an early taxonomy by Crawford (1984) lists combat games, maze games, sports games, paddle games, race games, adventures, D&D games, wargames, games of change, educational and children's games, interpersonal games and miscellaneous games. A newer listing by Gregory (2009) identifies fighting games, racing games, real-time strategy games, sports games, God games, simulation games, puzzle games and conversions of traditional games (like chess). Both authors admit that the listings don't even aim to be conclusive ones, and that other genres exist as well.

In any case, a thorough look into the engines of every genre – or even most of the genres for that matter – is, if even possible, outside the scope of this study. The genres discussed above were deemed most interesting for the study, because they offered interesting viewpoints to the field of game engines in general, and were coincidentally the ones about which most material could be found from the academic research body. For an example, FPS games initiated the aggressive push towards better and better graphics (Lewis & Jacobson, 2002), while adventure games provide an interesting example of even earlier data-driven architectures.

Luckily, as Gregory (2009) points out, increasing hardware power is enabling the use of game engines across different genres. Engines not aimed for a specific genre are discussed in the next (and final) part of the chapter.
3.7 Multi-genre engines

The previous parts of this chapter have focused on the evolution of the architectures of engines and games in the context of specific genres. However, some engines are flexible and widely used enough that it would not make sense to categorize them with specific genres.

3.7.1 NetImmerse/Gamebryo

One example of an engine that has been used multiple different games – with genres ranging from role-playing through strategy to racing – is the Gamebryo engine, formerly known as NetImmerse ("Published titles", n.d.; Cohn, Schmorrow, Lyons, Templeman & Muller, 2004). NetImmerse, published by Numerical Design, Limited, is described in great detail by Bishop et al. (1998) and Eberly (2005).

The development of NetImmerse was guided by several design goals, the most important being speed, which resulted in the authors using simple techniques throughout the engine and basing a lot on a scene graph data structure, which allows multiple performance enhancing techniques such as culling game objects that are not visible to the player (Bishop et al, 1998; Eberly, 2005).

Other design goals were providing compatibility with standard graphics and audio interfaces, and providing a high-level programming interface. The authors of the engine claim that the goals were attained, pointing out separately that a finding from creating the engine was that "a high-level programming interface need not compromise performance." Providing a high-level interface was deemed important because it allows developers to cut development costs, and because the engine needs to be optimized for games of different genres. (Bishop et al, 1998).
3.7.2 Unity

Another widely used package for creating games of various genres is Unity which features a game engine and a visual integrated development environment (IDE) to create games with (Xie, 2012; "Made with Unity", n.d.; "Unity – Game engine, tools and multiplatform", n.d.). Unity supports multiple different platforms and has become a popular choice for game developers, especially in the mobile games market (Xie, 2012; "Unity – Fast facts", n.d.).

Architecturally, Unity uses a hierarchy of three kinds of objects, scenes, entities (called GameObjects) and components, to represent a game. Scenes reside at the top level of this hierarchy, with a game containing one or more scenes and each scene containing zero or more GameObjects. GameObjects, representing entities in a scene, are in turn composed of components or other GameObjects. Components reside at the bottom level of the hierarchy, providing the very basic building blocks for game logic, such as the physical properties of an object or custom scripts that control an object's behavior. (Xie, 2012).

Unity's architectural model provides an obvious opportunity for code reuse, as previously created components and GameObjects can be used in other games with little to no configuration. Indeed, Unity comes with a predefined library of components that can be used to kickstart development of a game with little to no actual programming. (Xie, 2012).

3.7.3 Middleware

In addition to full-blown game engines, there exists a plenty of software packages that implement just parts of the functionality of an engine. These third-party packages are often referred to as middleware, and they are used as parts of many contemporary games and game engines. (Gregory, 2009).

Gregory (2009) identifies a number of middleware categories, including graphics, collision and physics and artificial intelligence. Examples of popular middleware include the Edge graphics engine and the Havok physics and collision engine.

Blow (2004) warns that while middleware can be crucial in lowering the workload in creating a game, the packages themselves are often complex pieces of software and may require lots of work and glue code to use, and as such a cost-benefit analysis is in place when picking a new package.

** Anderson et al. (2008) put forth the question whether it is possible to create a so-called Überengine, a game engine that suits all genres equally well. While outside the scope of this paper to analyze thoroughly, the engines in this part of the chapter (especially Unity), and the middleware approach – if one is comfortable with twisting the definition of an "engine" just a bit – certainly provide innovative solutions to the problem as well as real-world examples where engines have been used for vastly different games. **
4. Analysis

While the previous chapter outlined the history of computer games and the architectural decisions behind them, this chapter will focus on tying the ends together by answering the research questions based on the previous chapter.

4.1 What are the common parts of a computer game engine?

Various authors have tried to list typical parts of a game engine. Commonly identified modules include input handling, graphics, sound, and physics/dynamics (Bishop et al., 1998; Lewis & Jacobson, 2002). An event handler (Bishop et al., 1998) or a main/misc. (Blow, 2004) module that connects everything together is also often specified as a part of the engine. However, these lists are vague at best, and the genre of a game has a great influence on the engine behind it (Bishop et al., 1998; Anderson et al. 2008). Is there a physics module in LucasArts' SCUMM? Aren't there many modules in the FPS engines of today that are not adequately identified in the lists, such as networking and artificial intelligence?

Another complication is that the definition of a game engine is a vague one, and a clear separation between a game engine and a game can't always be made (Anderson et al., 2008). This is especially the case with older games, where game logic was often tightly interwoven with the platform due to performance considerations and lack of reusable modules.

With these considerations in mind, is it possible to identify the common parts of a game engine? One way to look at the issue is to concentrate on the definition of the game engine as everything but the content. Parts common to every game engine would then include:

- A main loop (or an alternative construct)
- Modules to process game data files or link to game data written in code
- Modules to handle game logic, which can, but is not required to, be specified by data files
- Modules to get input from the player
- Modules to present output to the player
- Auxiliary modules, such as networking and menu handling

While parts outside the engine could include such things as level data and behavioral scripts for non-player characters. This listing requires a somewhat pedantic view on the nature of a game engine, but since a similar definition was proposed by Bishop et al. (1998) and Lewis & Jacobson (2002), and since a focus on data-driven architectures when talking about game engines was suggested by both Demers et al. (2009) and Gregory (2009), and since all the engines analyzed in the previous chapter support it, the view of the author is that it's a reasonable one.
The problem with this listing is that it's not very specific, which brings us right back to the lament of Anderson et al. (2008) about vague and broad definitions of game engines. More specific listings of common engine modules could be made by concentrating on engines of specific genres, such as those of adventure games or first person shooters. Unfortunately, such detailed analyses are outside the scope of this study. Eager readers are advised to take a look at Munro et al. (2009) for an example of such study, and to refer to the other sources in chapter 3.

4.2 How have the architectures of game engines evolved over time?

A few common trends can be identified in the evolution of game engines and games in general:

4.2.1 Increase in size

The limited hardware of early computers forced early games to be small. Program size itself was limited, and the computers couldn't display sophisticated graphics or handle other complicated logic. In contrast, modern games are faced with multiple challenges from the box: Wonders can be worked with graphics, the gameplay is complex and networking brings its own issues in the equation. As such, modern games have grown into complex systems, consisting of multiple subsystems interacting with each other. This growth is reflected in the increase of both the SLOC and module counts of the code bases of games over time (Munro et al., 2009; Blow, 2004).

4.2.2 Increase in modularity

The architectures of games have also become more modular over time, a testament to which is the rise of the term "game engine" itself. This change has been propelled by various reasons. On the one hand, as the sizes of the code bases of games have risen together with the hardware capabilities of computers, focus from game engineering has shifted from low-level optimization to better organization of code (Blow, 2004). On the other, modularity is generally seen as a positive property of a code base (see e.g. Abrash, 1997) and tends to result in perks such as easier reuse of code. Finally, as we see with the case of early adventure games, modularity can aid in porting a game to new platforms increasing the possible market for the game.

For adventure games, key events in the evolution of modularity include the very architecture of Adventure (which was already somewhat data-driven), the creation of Zork's Z-Machine and ZIL and the creation of Sierra's AGI and SCI. For first-person shooters, key events include the creation of Doom and the move to a client-server architecture in Quake.

A manifestation of modularity, and the benefits of reuse, is the fact that the engines for today's games are usually not built, but reused. And even if one was to build an engine, various middleware, such as the Havok physics package, are readily available. (Gregory, 2009). The move to reusable engines and middleware resonates nicely with the thoughts of Gamma, Helm, Johnson & Vlissides (1995):

“Frameworks are becoming increasingly common and important. They are the way that object-oriented systems achieve the most reuse. Larger object-oriented applications will end up consisting of layers of frameworks that cooperate with each other. Most of the design and code in the application will come from or be influenced by the frameworks it uses.” (Gamma et al., 1995, p. 28)
Notably, in the case of computer games, reusable game engines can be seen as a form of frameworks.

Another way the increase in modularity can be seen in the games is by looking at the programming languages they have been written in. The first games were usually written directly in machine code, which is error-prone and cumbersome and causes serious issues in porting to different platforms. Doom and many other games at the time were written (mostly) in C instead, which offers many benefits over assembly while still retaining high speed. The latest id Tech engines use C++ instead of C, which adds support for object-oriented architectures. On an interesting note, Haskell was proposed by Sweeney (2006) as the next big programming language.

4.2.3 Improved tooling

Blow (2004) lamented that we didn't, at the time, have excellent tools for developing games with. The situation seems to have changed quite a bit, with many modern engines such as Unity, Unreal Engine 4 and Adventure Game Studio resembling full-blown development environments more than the traditional sense of game engines. As an example, Unity offers an in-game editor that enables both editing and testing a game using the same interface (Mehm, Reuter, Göbel & Steinmetz, 2012).

4.2.4 Increased portability

Finally, many of today's engines support multiple platforms, allowing games created using them to be run on multiple different machines and operating systems. For an instance, the platforms supported by Unity include Windows, Mac, Xbox 360, PlayStation 3, Wii, iPad, iPhone, Android, and (using a plugin) the web browser (Xie, 2012).

In a way, this is not a new development but more a result of the increase in available platforms, as we see that even early games that ran on top of engines (such Zork and the Z-Machine) reaped the benefits of increased markets due to portability. Still, especially with the inclusion of mobile devices, the “build once, run everywhere” model can be considered an ongoing evolution trend for game engines.

As a curious note, some authors (see BinSubaih et al., 2007; BinSubaih & Maddock, 2008) have taken the idea of modularity to a rather extreme, suggesting a “game-engines-independent” development approach, where a game would be runnable on multiple different engines (such as a Quake game on the Unreal Engine). However, the conducted literature review didn't yield any examples of this idea being put in practice, nor much mention about the idea in general. Whether we see such examples in the future remains an open question.

4.3 How has the evolution affected the present state of game engines?

To answer the question, let us consider a couple of case studies, from which various trends can be drawn.

4.3.1 FPS games and the Quake series

The engines behind various games of today can be traced back to their origins. The Quake (id Tech) series is a good example of a long line of engines with noticeable points of evolution, that's used as a basis for many of the games today. Due to its open-
source nature, the series also has the most research behind it (most notably that of Munro et al., 2009). As such, it serves as a useful lens through which to observe the question.

As outlined in chapter 3, the series began with Doom, which featured pseudo-3D graphics and a network play added as an after-thought, where a slow connection on one player's side would slow the game for everyone. The game also allowed limited amounts of modification, and while a couple of other games were based on top of it, they played pretty much like Doom.

With the creation of Quake, id Software made some major architectural decisions. First, the game featured true 3D graphics: every object was represented with polygons instead of 2D sprites in a pseudo-3D realm, which meant, for an instance, that players could freely look around and the graphics were better. Perhaps even more importantly, the game was designed for multiplayer from ground-up, with a client-server architecture being used even for the single player campaigns. Quake also offered vastly better modification, which resulted in its engine being used for games not necessarily all that similar it.

Better graphics and network play obviously became more important for id Software, so they decided to reflect for them in the architecture of their flagship product. Since the engine has been reused (though often with modifications) for a huge number of modern games, such as the Half-Life series, these games share the same design decisions too.

Can it then be said that without id Software's design decisions we would have poorer graphics and network play in Call of Duty: Advanced Warfare? Of course not. Better graphics and network play evolved out of a need and the developers of other games simply chose the engines that suited them best. However, the pioneering nature of the Quake series and its engines meant that they were likely candidates to be chosen for new games. Additionally, it seems likely that other engines built for similar games, while not directly inheriting the code base of the id Tech engine, have been influenced by the battle-proven design decisions made with it.

4.3.2 Sierra Entertainment and adventure games

As discussed before, adventure games prove an early example of data-driven, engine-like architectures even before the creation of first FPS games. Like with the Quake series, we can trace the origins of virtually all adventure games to a single game, namely Crowther and Woods' Adventure. This is especially true with the games by Sierra Entertainment (previously On-Line Systems and Sierra On-Line), as Adventure is cited as an explicit influence that lead to the creation of the company (see e.g. Jerz, 2007).

Most of the early adventure games already offered examples of data-driven architectures. Architectural advancements were made when distinct engines started appearing, Adventure Game Interpreter (AGI) in the case of Sierra Entertainment. The successor for AGI, Sierra's Creative Interpreter (SCI) featured even more modular decisions, such as the fact that it worked essentially as a virtual machine and that the programming language used to control it was upgraded to an object-oriented one.

To put this another way, we might postulate that there was an early influence (Adventure) that resulted in a motivation for data-driven architectures in later adventure games (such as Mystery House). This in turn, coupled with other motivating factors such as portability and ease of development, resulted in the creation of distinct game engines (such as AGI), which evolved into even more modular ones (SCI).
The drive for better graphics, as seen with FPS games, is also visible in the history of
adventure games. The games of On-Line Systems spawned the graphic adventure genre
with the rudimentary graphics of Mystery House, and continued to feature even
improved graphics with later games. Perhaps even more interesting, however, is the
breaking away from the traditional text parser to a point-and-click interface by games
such as those by Lucasfilm Games (now LucasArts), because it provides an example of
a move towards better user interface, not only towards better graphics. The benefits of
the point-and-click interface were also realized by On-Line Systems, who incorporated
it into later iterations of the King's Quest series (Gouveia, 2012).

Later adventure games made even more advancements with their graphics and styles of
play, such working in the first-person 3D perspective, while others have kept the simple
2D point-and-click style (Furini, 2008). As such, examples of both technological
advancement and influences of earlier games can be seen in the games of today.

4.3.3 In conclusion

Finally, to answer the research question on a rather general level, we can identify at
least the following ways the evolution of game engines has affected the state of the art:

• The drive for better and better graphics has resulted in the move from pixel-
based spaceships and text-based adventure games to today's hyperrealistic
graphics.

• Successful design decisions in the architectures of early, influential games are
still visible in today's games. These include the move to a client-server
architecture and the way network code was structured in Quake and later FPS
games, and the move to data-driven architectures and better user interfaces in
adventure games.

• The increased modularity of game engines has resulted in the huge market of
reusable engines and middleware today.

Future research should probably concentrate on the evolution of engines inside a single
genre, or of a single lineage, to answer the question in a more detailed and technological
manner.

4.4 What future trends can be identified in the evolution of game
engines?

Finally, to conclude the research questions put forth at the start of the study, let us
consider some of the future trends in the evolution of game engines that can be
identified from the way the engines have evolved over time. Predictions about the future
are, of course, cautious at best, but still a couple of trends seem worth a mention.

4.5 Advances in parallel architectures

Parallelism, the use of multiple CPU cores, was introduced to the id Tech series of
engines with id Tech 5 (as Tulip, Bekkema and Nesbitt [2006] point out, the previous
version of the engine didn't support multiple CPU cores, and neither did the other games
tested in the study). However, efficient utilization of parallelism in game engines is still
a new field of study, and there are still many advances to be made. As CPU
manufacturers include move continually towards substituting more CPU cores for
increased power of a single CPU, game engines will eventually have to utilize them to
continue the trend of extracting maximum performance out of the system (Tulip et al., 2006).

4.6 Decrease in genre-specificity

Increase of hardware capabilities has enabled games to move from low-level number crunching towards higher-level, modular architectures. This has also led into the creation of engines that are not tied to games of specific genres. As hardware capabilities rise over time, it should not be a surprise if we see even more of such engines, and conversely, less engines that are tied to specific genres. Though it has to be noted that in many cases, it may actually be preferable for a developer to pick a genre-specific engine, if such an engine offers easier development for a given game.
5. Conclusion

This study was conducted to explore the architectures of computer game engines and the way they have evolved over time, through an extensive literature review on the subject. It was the hope of the author that the research could provide a brief historic on the architectures of computer games and the engines behind them, and to answer the research questions set forth in the beginning.

In the end, these goals were achieved satisfactorily. Dwelling in the literature on the subject provided much new information for the author, from which the historic could be constructed. After some labor, answers to the research questions were also be constructed to a satisfactory degree, with the caveat that the nature of the questions was rather open-ended. Common parts of a game engine were identified as the parts that necessarily appear in every game. Increase in the size, modularity and portability of game engines, and improved tooling were identified as general trends in the evolution of game engines through time. Various successful design decisions behind certain games were identified as influences of still present in games today. Finally, increased utilization of parallel architectures, and the shift of focus from genre-specific towards genre-neutral game engines were identified as possible future trends.

5.1 Limitations and ideas for further research

The scope of research surpassed the expectations of the author by a great degree. After spending some time with the academic research body on the subject, it became apparent that games and engines are numerous and that papers of much larger lengths have been written on games of specific genres of games while this one tried to include all of them. On the other hand, it would have been a shame to restrict the study of game engines to a specific genre, especially the FPS one since that is a subject the field is already too narrowly focused on. In the end, many games and engine families are undoubtedly not represented in the study, even though they would provide interesting perspectives on the subject.

Literature on the subject was also hard to find at times, especially from peer-reviewed publications. Thus, sources from less reliable sources had to be used at times, though care was still taken to double-check them and to back them up with peer-reviewed ones where possible.

Further studies on the subject should pick a narrower field. An example of a good topic could be "the evolution of lag-correction in first-person shooter engines, and directions for future development." Various pointers for open research subjects on the field of game engines are provided by Anderson et al. (2008).
6. References


