Hochschule Kempten
University of Applied Sciences

MASTERARBEIT IM STUDIENGANG INFORMATIK - GAME ENGINEERING UND VISUAL COMPUTING

PROTOTYPING AN OPEN WORLD 3D ENGINE ON A MODERN DUAL-CORE $1 MICROCONTROLLER

Kurzbeschreibung

As the technological capabilities of modern microcontrollers improves, applications such as open world 3D games, once reserved for more capable PC hardware, become more viable on these restricted platforms. This thesis investigates the possibility of running an open world 3D game by implementing a prototype on a microcontroller-based gaming handheld. An analysis of existing software-based 3D engines and open world games is documented along with an overview of the complete implementation of the engine and prototype game. The implementation mainly consists of a software rasterizer, world chunk system as well as miscellaneous game components commonly found in other open world games.

Verfasser:	Bernhard Lee Strobl
Aufgabensteller/Prüfer:	Prof. Dr. Bernd Dreier
Arbeit vorgelegt am:	07.01.2023
Fakultät:	Fakultät für Informatik
Studiengang:	Game Engineering & Visual Computing
Matrikelnummer:	343117

VORWORT:

Ich möchte mich herzlich beim Prof. Bernd Dreier für die Betreuung und Unterstützung dieser Masterarbeit bedanken. Ein Dank geht auch an die Frau Janine Scheiterbauer für die Ausstellung der Konsole an der Gamescom sowie an meine Freundin und Kommilitonen für das Testen und Feedback.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 OPEN WORLD GAMES

Open world games have over the years become a de facto standard to showcase the development ability of game studios, both in fictional world creation as well as graphical benchmarks. Their complexity and large worlds frequently come with high hardware requirements pushing most modern PCs and consoles to their limits.

FIGURE 1: GRAND THEFT AUTO V (LEFT) AND THE ELDER SCROLLS: SKYRIM (RIGHT) ARE PROMINENT EXAMPLES OF MODERN OPEN WORLD GAMES. (COURRÈGES, 2015) (BETHESDA SOFTWORKS LLC, N.D.)

Although definitions vary, an open world game usually allows players to explore a large, interconnected game world without loading screens and artificial obstacles. They can freely decide to complete quests or simply move around and explore, interact with non-player characters (NPCs) or amuse themselves with side activities. (Hughes, 2021)

Well known developers of open world games include Bethesda Softworks (The Elder Scrolls and Fallout series), CD Projekt Red (The Witcher series and Cyberpunk 2077) and Rockstar Games (Grand Theft Auto and Red Dead Redemption series).

1.2 OPEN WORLD GAMES ON HANDHELD CONSOLES

Early open world games on handheld gaming devices like the Nintendo Game Boy were limited to simple 2D worlds. Game releases such as Grand Theft Auto for the enhanced Game Boy Color attempted to provide a world which could be accessed by the player from all directions without loading screens but suffered from middling reviews. (Gamer Network Limited, 2000)

With rapidly increasing performance on later devices more ambitious open world games could be published. Hardware acceleration allowed gaming handhelds such as the PlayStation Portable to produce a variant of Grand Theft Auto III's Liberty City in the game Grand Theft Auto: Liberty City Stories. This came at the cost of additional extensive optimizations to assets and code to keep performance and battery life within acceptable levels. (Castro, 2005)

FIGURE 2: GRAND THEFT AUTO 2 FOR THE GAME BOY COLOR IN 1999 (LEFT). GRAND THEFT AUTO: LIBERTY CITY STORIES RELEASED IN 2005 (RIGHT). (GAMER NETWORK LIMITED, 2000) (PHILLIPS, N.D.)

Eventually, sufficient performance of gaming handhelds like the New Nintendo 3DS allowed previously large open world console titles such as Xenoblade Chronicles to be completely ported without compromises in gameplay. (Helgeson, 2015)

The release of the Nintendo Switch pushed this concept further due to its combined handheld and home console design. Open world games such as The Witcher 3 and The Legend of Zelda: Breath of the Wild proved to be largely successful on the platform. (Awan, 2020) (Orland, 2017)

FIGURE 3: THE WITCHER 3 (LEFT) AND THE LEGEND OF ZELDA: BREATH OF THE WILD (RIGHT). (CD PROJEKT S.A., N.D.) (HOOKSHOT MEDIA, N.D.)

The popularity and sales of the Nintendo Switch has contributed to competitors releasing PC based handhelds to great demand. The most notable of these is the Steam Deck from Valve Corporation. (Heaton, 2022)

1.3 MICROCONTROLLERS

Microcontrollers (or Microcontroller Unit - MCUs) are primarily low-cost chips. They are usually designed with all necessary components such as a CPU, RAM, ROM and Input/Output integrated on a single die. This allows them to be used as single chip solutions when compared to typical microprocessors, which require several support chips to function. As RAM is mostly fully integrated no system buses are exposed further reducing pin count and packaging cost. Clock speed is significantly lower when to save power and to allow the use of older manufacturing nodes. (List, 2020) (Osborne, 1978)

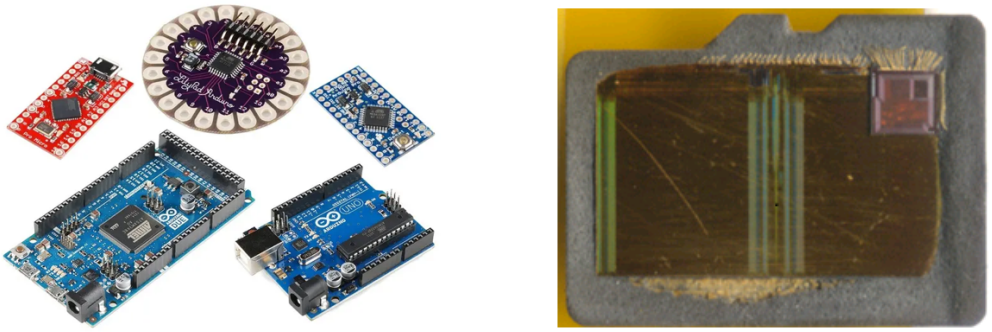

FIGURE 4: A NUMBER OF DEVELOPMENT BOARDS USING MICROCONTROLLERS (LEFT). A MICROCONTROLLER BUILT INTO THE CORNER OF A MICROSD CARD (RIGHT). (GHARGE, 2022) (HUANG, 2013)

Their simplicity and cheap price per unit often affords them usage in single-task or dedicated applications including:

- Household appliances like washing machines, refrigerators and microwaves.
- In computers as additional controllers for components like disk drives and peripherals (e.g., keyboards and mice).
- In cars including engine control units (ECUs) and Anti-lock braking systems (ABS).
- Electronic Toys such as RC-cars, robots and low-cost gaming handhelds.
- Smart cards such as credit/debit, SIM and identity cards.

(Brain, n.d.) (Gupta, 2021)

1.4 MICROCONTROLLERS IN HANDHELD CONSOLES

Microcontrollers were commonly used in previous decades to run low-cost handheld games. Numerous models have been manufactured by different companies (along with soviet clones) up to the modern day. (Branagan, 2022)

One of the first gaming handhelds is the Microvision introduced in 1979. It utilized exchangeable cartridges with integrated 4-bit TMS1100 Microcontrollers produced by Texas Instruments. The TMS1100 chip itself is a RAM/ROM doubled variant of the first microcontroller ever released, the TMS1000 released in 1975. (Boris, n.d.) (List, 2020)

Later releases such as the Nintendo Game & Watch series and Tamagotchi frequently made use of cheap 4-bit microcontrollers designed for watches such as the SM5 family of MCUs from Sharp. These chips contain internal LCD controllers suitable for driving the custom segments of the LCD Panels available at the time, with clock speeds as low as 32,768 Hz. (Sharp Corporation, 1990) (Rubio, n.d.)

FIGURE 5: MICROVISION RELEASED IN 1979 (LEFT). ORIGINAL RELEASE OF GAME & WATCH BALL IN 1980 (RIGHT). (VIDEO GAME KRAKEN, N.D.) (SHERRILL, 2020)

Over the decades, the introduction of 8-bit microcontrollers have allowed more advanced low-cost game handhelds to be implemented, including the well-known Brick Game clones. In turn, reduced costs have allowed formerly more costly 4-bit gaming units to be included as an example in cheap fast-food meals. (sonnyboy, 2017) (Inhibit, 2021)

Although more advanced handhelds such as the original Game Boy came in 1989 (itself inspired by the Microvision), it used a dedicated microprocessor and required additional RAM and supporting chips. The additional components and higher quality drove up the cost of the console to its original US$ 89.99 price. (Brian, 2016)

The introduction of 32-bit capable microcontrollers in recent years has made releases of more premium handheld consoles possible. One such example is the Nintendo Game & Watch: The Legend of Zelda, which is itself a reimagining of prior Game & Watch titles. It uses emulation to provide older console titles to retro-enthusiasts. The Playdate is another example, with a built-in hand crank as a unique input method, showcasing newer form-factor innovations. Both handhelds are manufactured using available off the shelf microcontrollers. (Nintendo Co., Ltd., 2022) (Panic Inc., 2022)

FIGURE 6: THE NINTENDO GAME & WATCH: LEGEND OF ZELDA (LEFT) AND THE PLAYDATE (RIGHT). (NINTENDO CO., LTD., 2022) (PANIC INC., 2022)

Cheaper component costs have also contributed to further advancements in low-cost handhelds such as the frequent addition of color screens and more complex games commonly found in older 8-bit PCs and consoles. Several consoles use a NES-on-a-chip

(NOAC) along with copied or modified titles and homebrew games often produced for the original Nintendo Entertainment System. (Black, 2019) (Basinger, The Oregon Trail Electronic Handheld Game!, 2018)

1.5 MOTIVATION

While not comparable to a modern Desktop CPU when it comes to performance, microcontrollers have evolved regardless to provide increasing computational speed relative to their cost. The capability of these chips to be used for more than simple 2D game handhelds, including large open world 3D games, should be investigated. This would allow new gameplay experiences on inexpensive hardware, which is especially relevant for price sensitive markets. Many stores to this day still retail low-cost handheld gaming devices like the ones in Figure 7.

FIGURE 7: BRICK GAME CLONE (LEFT), AND RETRO CONSOLE LIKELY BASED ON A NES-ON-A-CHIP (RIGHT) FOUND IN LOCAL STORES.

The high cost (easily exceeding millions of US Dollars) of otherwise producing dedicated application specific integrated circuits (ASICs) for low-end 3D graphics is usually beyond the reach of most production budgets for these devices. (Lankshear, 2019)

1.6 AIM

The goal of this thesis is therefore to create a functional open world 3D prototype on a microcontroller-based handheld. This will require extensive research in previous engine implementations and graphic techniques.

The resulting engine itself should mimic modern open world 3D engines and:

- be reasonably simple to understand and modify, as the combination of microcontroller and 3D graphics knowledge is not necessarily very common.
- not have any loading times or screens when traversing the world to prevent breaking immersion by the player.
- have tooling to allow simple import of meshes from Blender (3D application).
- contain typical optimizations found in other rendering pipelines which can be quickly performance tested and applied.
- run with an acceptable frame rate on common graphical workloads.

2 ANALYSIS OF SOFTWARE RENDERERS

Before the advent of affordable hardware accelerated GPUs, early 3D games made use of the general-purpose CPU found in their respective platforms to render 3D graphics. This chapter focuses on existing engines and games which historically used a software rasterizer, some of which at the time already included open world gameplay.

2.1 PC GAMES

3D wireframe rendering became common on early 8-bit hardware with the release of games like Elite on the BBC Micro in the 1980s. A significant bottleneck in early 8-bit hardware was still the CPU, and therefore its ability to effectively fill the generated polygons. (Loguidice, 2009)

One of the first dedicated 3d engines was the Freescape engine by Incentive Software, used in games such as Driller and Castle Master. It was one of the first engines to implement filled flat-shaded polygons. More powerful 16-bit computers allowed the engine to operate at a much more acceptable framerate. (Fahs, 2008)

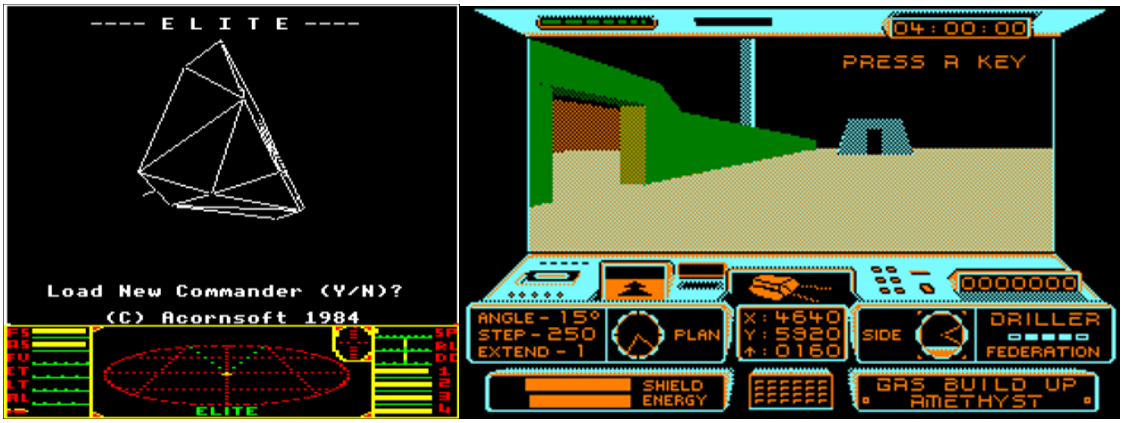

FIGURE 8: ELITE RELEASED IN 1984 (LEFT). DRILLER, BASED ON THE FREESCAPE ENGINE, IN 1987 (RIGHT). (LOGUIDICE, 2009) (VIDEOSPIELHALBWISSEN, N.D.)

Later games like The Terminator from Bethesda and Hunter from Activision, both released in 1991, were among the first to combine typical open world gameplay elements with filled 3D polygons. 3D engines progressed from there with features including texture mapping introduced in games like Ultima Underworld in 1992. (McMullen, 2019) (Moss, 2017) (Maher, 2019)

Games such as The Elder Scrolls II: Daggerfall released in 1996 extended these principals to one of the biggest open worlds in gaming measuring over 160,000 square kilometers in size. This was achieved with mainly procedurally generated and repeating content to fill the world. Polygon counts were still heavily limited in an open environment, leading to the use of 2D sprites for characters and objects to maintain performance. (Burgar, 2022) (Zao, 2012)

FIGURE 9: HUNTER RELEASED BY ACTIVISION IN 1991 (LEFT), ULTIMA UNDERWORLD (RIGHT) RELEASED IN 1992. (MOSS, 2017) (PAYNE, 2018)

A significant milestone was the introduction of Quake in 1996. Created by id Software (known previously for Wolfenstein 3D and Doom) Quake utilized a true 3D engine including fully modelled enemy characters. Features such as pre-baked lightmaps and dynamic lighting were implemented and showcased then advanced techniques which would persist to the modern day.

The main software rasterizer as described by Abrash (1997) utilizes a technique usually referred to as scanline rendering (or scanline conversion). This method consists of sorting triangles in the vertical order in which they appear. Each scanline can then be rendered by proceeding horizontally and rendering triangles as their edges start and end, while keeping overdraw to a minimum (i.e., only the frontmost triangle at a given pixel is rendered).

When combined with binary space partitioning (previously used in Doom) and other optimizations, the performance of the engine became notably efficient on hardware at the time. However, implementing a working solution was error-prone and time-consuming and did not work well with moving objects (Enemy models used different rasterizers).

Quake also made extensive use of the features of the original Intel Pentium processor, such as the ability to have both floating-point and integer operations operating concurrently. (Abrash, Graphics Programming Black Book, 1997)

FIGURE 10: THE ELDER SCROLLS II: DAGGERFALL (LEFT) AND QUAKE (RIGHT), BOTH RELEASED IN 1996. (ZAO, 2012) (SANGLARD, THE STORY OF THE RENDITION VÉRITÉ 1000, 2019)

Quake along with several other notable titles at the time such as Tomb Raider also introduced support for one of the very first dedicated 3D GPUs for consumer PCs, the

Rendition Vérité 1000. (Linneman, DF Retro: Quake - The Game, The Technology, The Ports, The Legacy, 2021)

The Vérité 1000 utilized a RISC based processor core with additional graphic instructions and hardware blocks, making the card act more like an additional CPU dedicated to graphic operations. Later GPUs such as the Voodoo Graphics by 3dfx instead focused on dedicated fixed functions units to achieve significantly faster output. (Sanglard, The story of the Rendition Vérité 1000, 2019)

Rapid adoption of these and more advanced GPUs led to the eventual deprecation and complete removal of most software rendering implementations in 3D engines, as dedicated GPUs rapidly outpaced CPU rendering performance.

Even though the use of software rendering has practically vanished for most gaming applications, software renderer fallbacks are still provided for modern APIs like DirectX (Windows Advanced Rasterization Platform) and Vulkan (LLVMpipe). (Microsoft, 2022) (The Mesa 3D Graphics Library, n.d.)

Increasing programmability of shader cores in GPUs of the last decade has also progressed to the point where a software rasterizer itself can be implemented and programmed on these increasingly general compute units. (Laine & Karras, 2011)

2.2 4TH GENERATION HOME CONSOLES

4th generation game consoles mainly consisted of 16-bit machines which relied on graphical workarounds like Mode-7 effects to provide pseudo 3D environments.

Two notable exceptions appeared using the same approach for producing 3D graphics. The first was Star Fox for the Super Nintendo Entertainment System (SNES), while the second was Virtua Racing for the Sega Mega Drive (known as the Sega Genesis in North America). Both games made use of a coprocessor on the game cartridge itself to produce the needed 3D graphics. In the case of Star Fox this was the Super FX chip. Virtua Racing made use of a custom Samsung DSP processor called the Sega Virtua Processor (SVP). (McCloud, n.d.) (David, 2007)

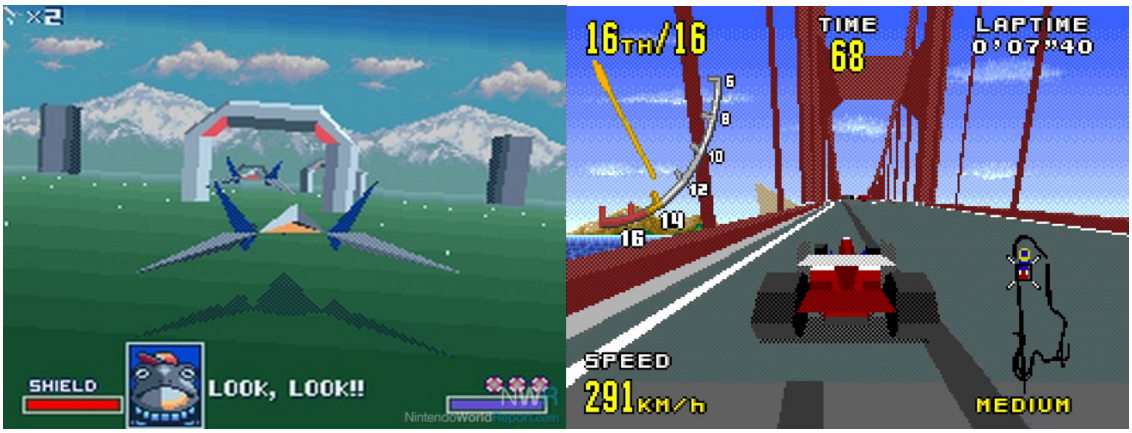

FIGURE 11: STAR FOX ON THE SNES (LEFT) AND VIRTUA RACING ON THE MEGA DRIVE/GENESIS (RIGHT). (HERNANDEZ, 2011) (LINNEMAN, DF RETRO: VIRTUA RACING SWITCH VS EVERY CONSOLE PORT VS MODEL 1 ARCADE!, 2019)

Although both co-processors were 16-bit capable and clocked higher than their respective base consoles, neither chip contained true hardware acceleration for 3D

rendering. Custom programming along with rudimentary mathematical operations were used to perform vertex calculations and rasterization.

Further games were developed for the SNES utilizing the Super FX chip in one form or another, while Virtua Racing remained the only game on the Mega Drive due to its high production cost as a result of the additional SVP.

3D hardware acceleration became standard with 5th generation video game consoles such as the Sony PlayStation and Nintendo 64. (Copetti, Architecture of Consoles | A Practical Analysis, n.d.)

2.3 NINTENDO GAME BOY ADVANCE

One of the most commercially successful 2D handheld gaming consoles, the Game Boy Advance (GBA), arrived in 2001 and featured a 16.78MHz 32-bit Arm core along with a total of 384KB of RAM. (Copetti, Game Boy Advance Architecture | A Practical Analysis, 2021)

While only designed to handle 2D graphics and some mode-7 effects like the SNES, certain developers were able to overcome these limitations and produce 3D games. For example, VD-dev (having already developed a suitable 3D engine for the platform with V-Rally 3) produced the open world game DRIV3R in 2005. (Stop Skeletons From Fighting, 2016)

FIGURE 12: DRIV3R BY VD-DEV (LEFT) AND AN UNRELEASED PORT OF QUAKE (RIGHT). (VD-DEV, N.D.) (FOREST OF ILLUSION, 2022)

While commercial development has long ceased for the console, homebrew developers still frequently use the console due to its well documented architecture. This has in the past led to developments such as an unreleased port of Quake and a port of Tomb Raider (OpenLara). Both games mostly utilized assembly programming as well as other optimizations like look-up tables for mathematical functions at the cost of available memory. (Giannakis, Tomb Raider on the Nintendo Game Boy Advance is incredible | MVG, 2022)

The Game Boy Advance was succeeded by the Nintendo DS in 2004, which featured a form of scanline rendering (similar to Quake) but accelerated in hardware. (Copetti, Architecture of Consoles | A Practical Analysis, n.d.)

2.4 NOKIA N-GAGE

Before the appearance of smartphones featuring hardware acceleration, games on mobile phones were usually strictly limited to 2D graphics. In an attempt to enter the handheld gaming market, Nokia produced the N-Gage. Although it featured a 104 MHz Arm core coupled with 16MB of RAM to allow significantly better-looking games, the N-Gage was a commercial failure. (Hardware-Aktuell, n.d.)

Several games for the device were released with 3D graphics. Among them were ports of popular PS1 titles like Tomb Raider and Tony Hawk. The Elder Scrolls Travels: Shadowkey is an example of an open world title on the platform. However, 3D titles on the N-Gage gained a reputation for somewhat inconsistent performance, low framerates and short draw distances. (Palley, 2004) (Leeper, 2005)

FIGURE 13: PORT OF PLAYSTATION TITLE TOMB RAIDER(LEFT), CALL OF DUTY (MIDDLE) AND BETHESDA'S OPEN WORLD THE ELDER SCROLLS TRAVELS: SHADOWKEY (RIGHT). (STELLA, N.D.) (PALLEY, 2004) (LEEPER, 2005)

Similar processing power and system on a chip (SoC) architecture existed in the form of several Palm handhelds and Windows Mobile PDAs. However, their small market share meant larger development studios refrained from providing any notable 3D games. (Basinger, Tapwave Zodiac: The Failed 2003 Gaming PDA, 2018)

The later introduction of smartphones such as the iPhone came with 3D hardware acceleration as standard to help offload graphical tasks from the main CPU. (Patterson, 2008)

3 ANALYSIS OF ASSET LOADING SYSTEMS

An important aspect of open world games is the ability to seamlessly load in parts of the world as the player changes position in the game world.

3.1 LEVEL-BASED ASSET LOADING

Early 3D games loaded environment meshes and other data for a single game level completely into RAM. The CPU was often incapable of processing other loading tasks when already busy with rendering and handling game logic.

The systems on which these games ran on were also heavily affected by sorting of visible triangles before the common use of Z-buffers. Simpler indoor scenes could solve the problem of visible surface determination (VSD) using binary spaces partitioning (BSPs) to store levels. This allowed significantly faster and more efficient rendering by providing a correct triangle order, while significantly reducing overdraw when given a player position. (Target, 2019)

Polygonal presorting could also be used in linear games where the player was ever only going in one of two directions. Triangle order could be precalculated on more powerful hardware at a slower rate and quickly retrieved later in the game in the correct order. Crash Bandicoot itself made use of this technique along with a paging system to load in batches of triangles dynamically. This allowed significantly larger and much more detailed levels than in any other PlayStation games at the time. (Gavin, 2011) (Ars Technica, 2020)

FIGURE 14: DOOM RELEASED IN 1993 (LEFT). CRASH BANDICOOT IN 1996 (RIGHT). (FUNKE DIGITAL GMBH, N.D.) (KÜPPER, 2020)

The simplest and most reliable method of solving triangle sorting issues as polygon counts increased was the later use of Z-Buffers. However, this still limited most levels to available RAM on the system along with the additional memory required for the Z-Buffer itself.

One approach to circumvent loading screens in such instances is the use of airlocks (also called door and portal technique). This consists of tunneling players through loading areas which in turn trigger loading of the next level. Level designers must ensure players are within a region shared by both larger areas in memory before the old level can be unloaded upon the player leaving it completely. (Ruskin, 2015)

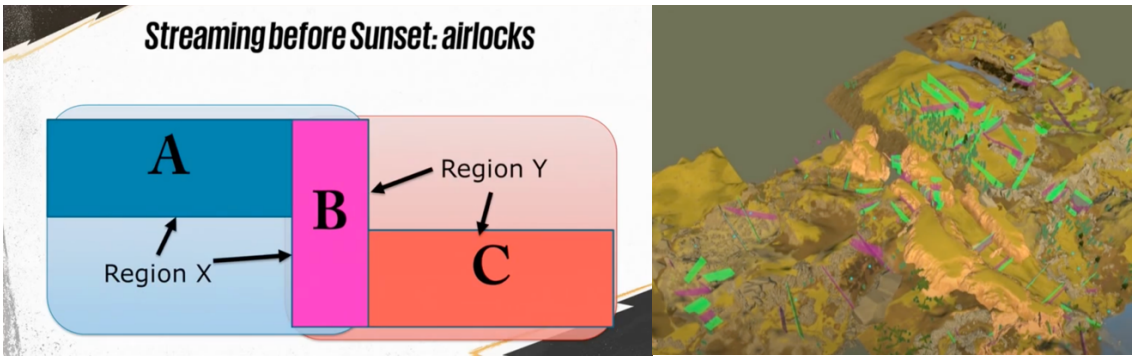

FIGURE 15: LOADING USING INTERMEDIATE LEVELS OR CORRIDORS (LEFT). LOADING ZONES IN FIREWATCH (RIGHT). (RUSKIN, 2015) (NG, 2016)

While this approach can be used to the extent of a pseudo-open world experience with larger sections for the players to explore, it is unsuitable for a true open world game where a player is not restricted to chokepoints and may enter a new area from any direction. (Ruskin, 2015)

3.2 OPEN WORLD ASSET STREAMING

A common setup for the loading of assets in open world games is to split up the game world into a two-dimensional grid consisting of chunks (also commonly referred to as tiles, sections, sectors, patches or cells) containing landscape and scene geometry. Chunks do not necessarily have to be square in size, if they are capable of being repeated in a pattern such as with triangles and hexagons.

Only the chunks closest to the player are loaded in, thereby alleviating the issue of loading the complete game world into memory. This is especially important on mobile devices and RAM limited consoles. (Epic Games, Inc., n.d.) (Bilas, 2003) (Ender, 2017)

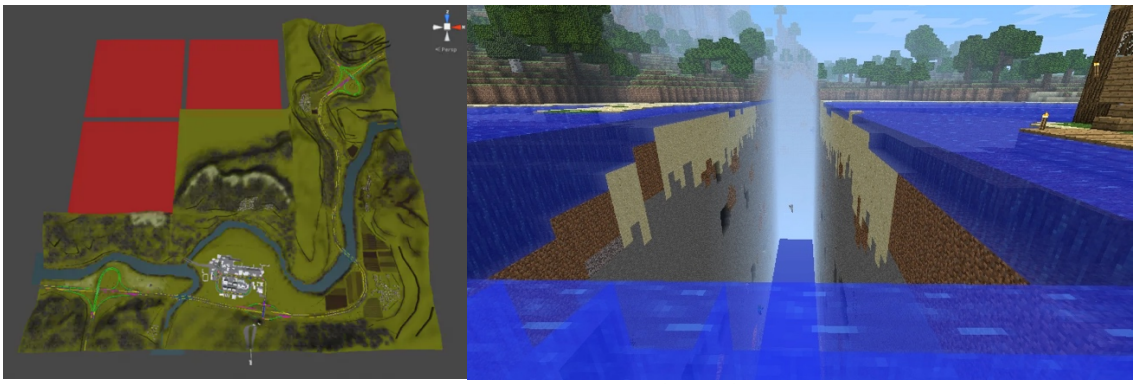

FIGURE 16: PATCHES IN TRUCKSIMULATION 16 (LEFT). MISSING CHUNKS DUE TO LOADING ERROR IN MINECRAFT (RIGHT). (ENDER, 2017) (MINECRAFT WIKI, N.D.)

Games such as Horizon: Zero Dawn utilize 2D maps streamed in a similar way to generate procedural landscapes and vegetation. This allows rapid editing of scenery by artists and automated tools. (Muijden, 2019)

Depending on hardware, there are limitations in how quickly a game world can be streamed in. Games like the Legend of Zelda: The Wind Waker on the GameCube restricted the speed of the player to ensure assets could be loaded in on time. This was resolved with a later HD remake on the Wii U, which could load the entire ocean into memory at once. (Nintendo of America, 2013)

FIGURE 17: HORIZON ZERO DAWN (LEFT) AND THE LEGEND OF ZELDA: THE WIND WAKER HD (RIGHT). (SONY INTERACTIVE ENTERTAINMENT EUROPE LIMITED, N.D.) (HOOKSHOT MEDIA, N.D.)

Certain games like Cyberpunk 2077 go a step further, enabling vertical streaming of assets due to increased complexity of geometry caused by high-rise buildings and stacked levels. (Cryer, 2019)

3.3 LEVEL OF DETAIL

The use of a chunk loading system is frequently combined with the reduction in complexity of meshes at a distance. Chunks which are further away from the player may use lower detail meshes, thereby reducing graphical workload and improving performance. Lowering details in models can either be performed manually or using automated tools. (Ruskin, 2015)

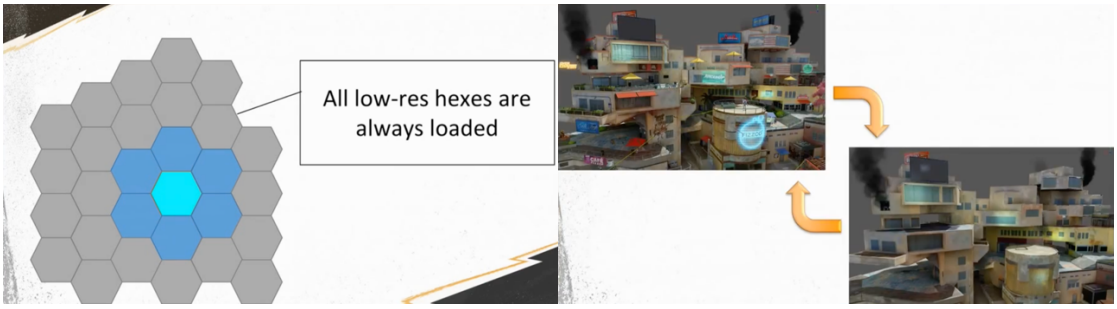

FIGURE 18: MESHES ARE STORED IN LOWER DETAIL AND LOADED IN CHUNKS FURTHER FROM THE PLAYER (LEFT). LOWER LOD LEVELS CAN BE ACHIEVED THROUGH AUTOMATED MEANS TO SAVE ON ARTIST TIME (RIGHT). (RUSKIN, 2015)

Far objects may also be dropped in favor of impostors or fake point lights to give the impression of activity in distant areas. (Persson, Creating vast game worlds, 2012)

More advanced techniques may also be used to merge low level of detail meshes into larger chunks. This is combined with the application of more advanced data structures such as quadtrees when loading fresh assets into memory. (Meta, n.d.)

3.4 NUMERIC PRECISION AT LARGE DISTANCES

As the player moves further away from a world's central origin, graphical glitching is likely to occur due to floating-point number precision reducing with higher values. If

integers are used, overflow and precision errors may also result in graphical artifacts and errors. (Bilas, 2003)

A solution is the use of additional offsets to move player and models back to the origin whenever a player exceeds a certain distance. This ensures only the lower range of a floating-point type is utilized reducing graphical artifacts. (O'Neil, 2002)

Another solution is to fix the player position at world origin and move objects around them according to their movements. This comes with additional drawbacks in development complexity. (Wooden, 2015)

The use of double-precision floating-point numbers for distant objects is also possible. However, this is generally avoided for actual graphical tasks as most GPU hardware lacks full speed support for 64-bit datatypes. (Battaglia, 2020) (Persson, Low-level Shader Optimization for Next-Gen and DX11, 2014)

Precision issues also heavily affect Z-Buffers, with very distant objects in the world often encountering Z-fighting and early culling. (Persson, Creating vast game worlds, 2012)

4 DEVELOPMENT HARDWARE

4.1 PICOSYSTEM

A small gaming device called the PicoSystem (manufactured by Pimoroni) is used for development of the engine and prototype. Unlike other commercial handheld game consoles where software is not meant to be modified, the PicoSystem is designed for custom software development by the end user.

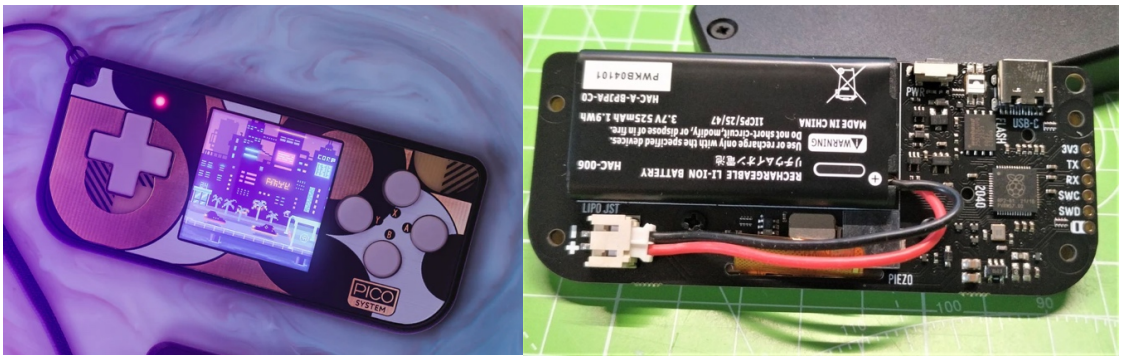

FIGURE 19: THE PICOSYSTEM (LEFT), WITH MAINBOARD SHOWING INTERNALS (RIGHT). (PIMORONI LTD., N.D.) (POUNDER, 2021)

Among its core specifications are:

- RP2040 microcontroller with 264KB of RAM (overclocked to 250 MHz from a native 125 MHz)
- 16MB of QSPI Flash
- 240x240 pixel display (with support for pixel doubling to produce 120x120 pixel output)

The PicoSystem is among the recent influx of dedicated hobby-development handhelds like the Arduboy, Gamebuino and PyGamer. (Kenney, 2019)

4.2 RP2040 MICROCONTROLLER

The PicoSystem uses an RP2040 microcontroller. This chip was released in 2019 along with a reference development and prototyping board, the Raspberry Pi Pico.
The RP2040 contains among other components:

- two Cortex M0+ CPU cores (labeled core 0 and core 1 respectively)
- 256 kB of main RAM (separated into 4 banks of 64kB each).
- two small RAM allocations of 4kB, designed for use as stack space by each individual core.
- several common input/output methods including GPIO ports, UART, SPI and I2C. Used in the PicoSystem to drive components like the display, buttons etc.
- An eXecute-In-Place(XIP) cache for caching previously accessed data from flash.

Although many microcontrollers integrate flash or other ROM into the chip directly, the RP2040 was created as a “flashless microcontroller”, requiring an external chip to hold program code and data. This was done to increase the available SRAM in the design, lower cost and give some flexibility in utilizing different flash memory sizes (up to a maximum of 16MB). (Adams, Fraser, & Wren, 2021)

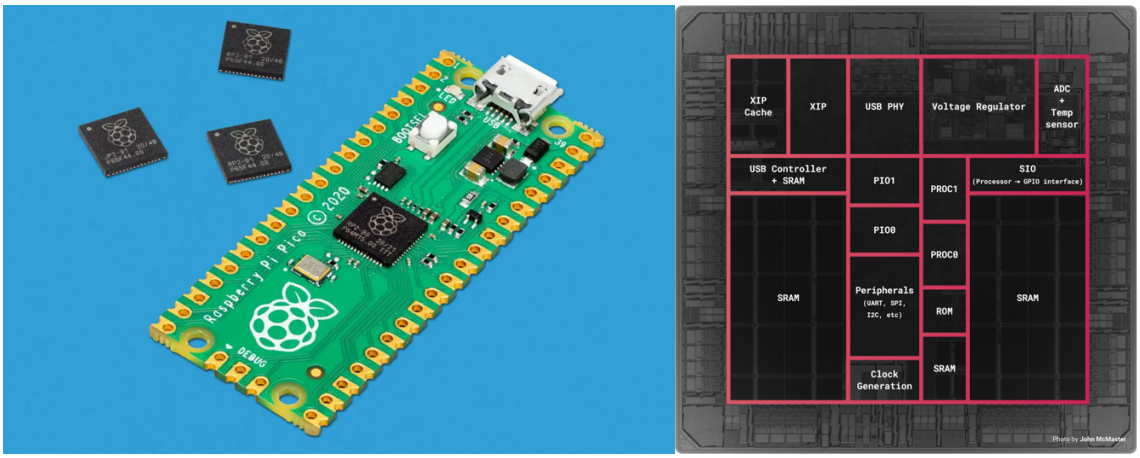

FIGURE 20: THE RASPBERRY PI PICO (LEFT) AND FLOORPLAN OF THE RP2040 (RIGHT). (RASPBERRY PI LTD, N.D.)

The two primary Cortex M0+ cores as used in the RP2040 belong to the smallest available core designs in Arm's Cortex M-Series of embedded 32-bit cores. They retain a 32-bit address space and register size while utilizing more compact 16-bit Armv6-M Thumb instructions. This allows twice as many instructions to fit into memory when compared to other Arm instruction sets. As with other modern microcontrollers and those in the M-series, a discerning feature of the cores is the lack of a memory management unit (MMU). This prevents operating systems and applications which rely on virtual memory from executing without the use of emulation. (Shimpi, 2014)

Unlike larger cores in the M-series, M0+ does not feature acceleration of floating-point instructions. The cores also do not come with an official integer divider. However, an additional hardware block has been added by Raspberry Pi Ltd to compensate for this, allowing divisions to be performed with an 8-cycle latency. The hardware block also provides dedicated FIFO queues to allow intercommunication between both cores. Both cores in the RP2040 are fitted with a single cycle integer multiplication unit (an optional feature for M0+ cores). (Raspberry Pi Ltd, 2022)

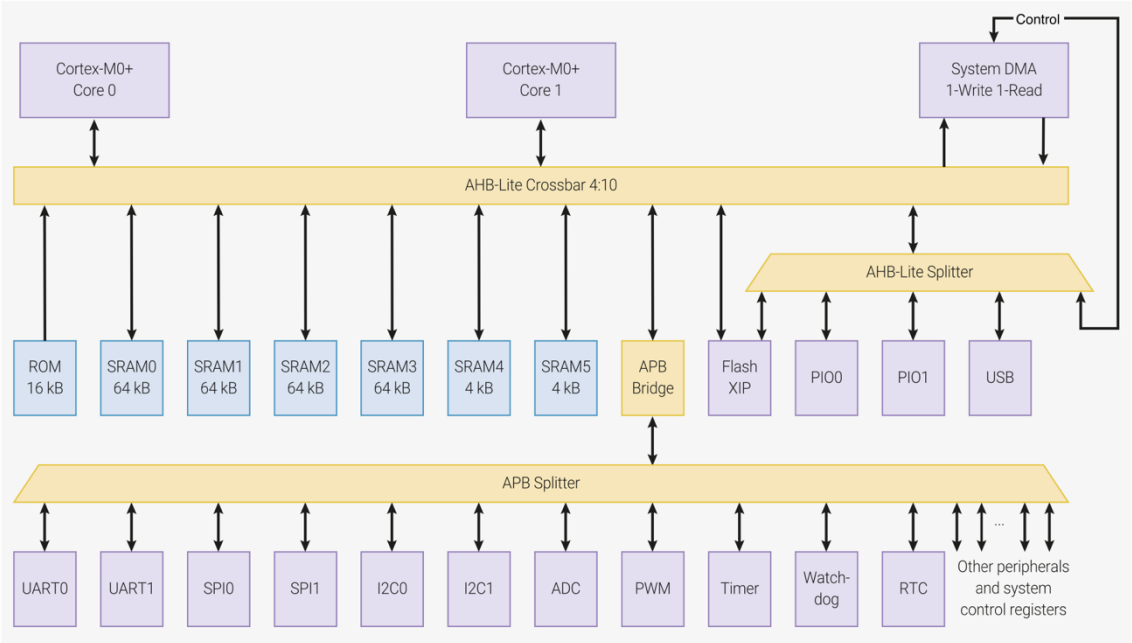

FIGURE 21: COMPONENTS OF THE RP2040. (RASPBERRY PI LTD, 2022)

The four main banks of RAM (256KB total) are configured by default together in a striped fashion. Every 128-bits of linearly accessed data is thereby split into four consecutive 32-bit reads handled by each RAM bank. As only a single core can access one bank of RAM concurrently, accessing any memory location in main RAM with both cores thereby comes with a 25% chance of a collision. This will cause stalling for one of the cores and reduced performance.

A unique feature of the RP2040 is the availability of small state machines called Programmable Input-Output (PIO). These state machines can execute 32 instructions and are designed to quickly process data on the I/O pins. One PIO is used by the PicoSystem to copy framebuffer data to the display and provide native pixel doubling capabilities, thereby alleviating the primary cores of this task. (Raspberry Pi Ltd, 2022)

4.3 COMPARISON OF THE RP2040 TO OTHER SYSTEMS

The years of CPU-only 3D rasterization spanned roughly the mid 80s to mid 90s, while mobile hardware utilized CPU rasterization between the early 2000s and mid 2000s before the transition on both platforms to hardware accelerated 3D processing.

When compared to the previous architectures in Chapter 2, the RP2040 more closely has the performance profile of a desktop CPU from the mid-90s due to its high clock speed and dual processor cores. Robust overclocking capabilities used by the PicoSystem allows stable operation at twice the native clock speed.

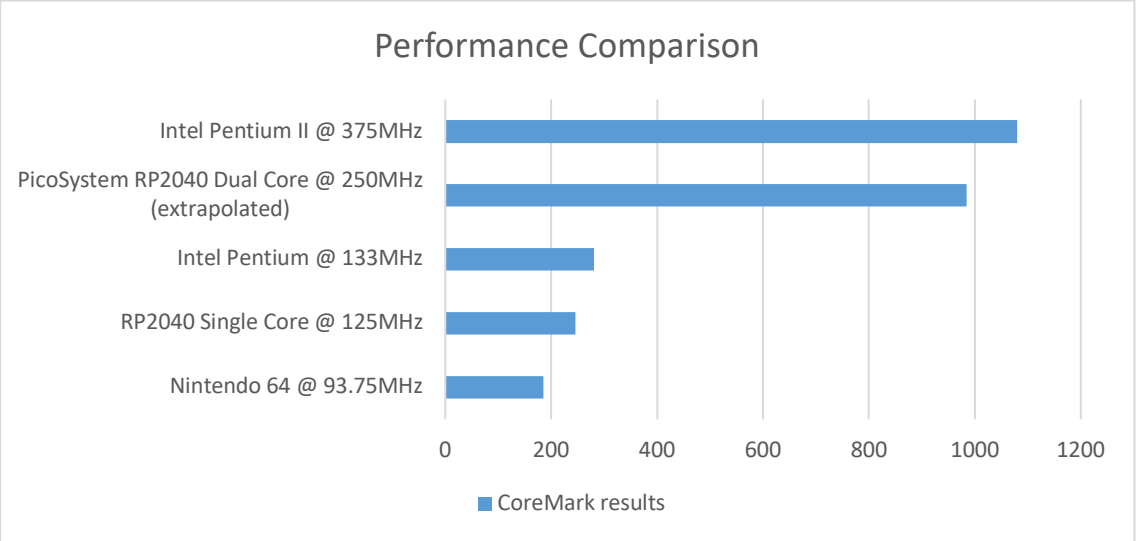

FIGURE 22: CPU PERFORMANCE COMPARISON USING COREMARK. DATA REFERENCED FROM (ZHANG, N.D.).

While the processing capability of the RP2040 is comparatively high, available RAM is one order of magnitude lower and more comparable to the Game Boy Advance, 4th gen consoles or older 16-bit computers. 5th generation consoles like the PlayStation and Nintendo 64 along with desktop PCs released at roughly the same time contain substantially more RAM.

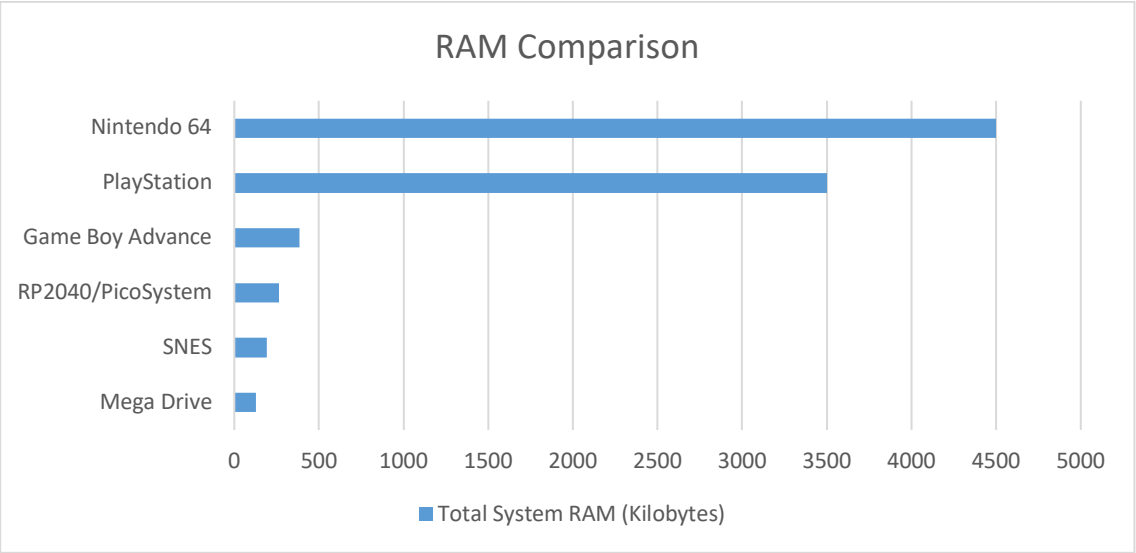

FIGURE 23: AVAILABLE RAM ON COMPARABLE AND MORE POWERFUL SYSTEMS. (COPETTI, ARCHITECTURE OF CONSOLES | A PRACTICAL ANALYSIS, N.D.)

The limited amount of RAM is compensated somewhat with flash memory directly being mapped into the RP2040's address space. This allows program instructions and read only data to reside on flash and be loaded only when directly needed through execute in place. The XIP cache (16kB in size) masks some of the much longer access times to flash if a successful cache hit takes place. (Raspberry Pi Ltd, 2022)

Flash bandwidth is still a bottleneck necessitating placement of frequently used data in RAM to prevent excess stalling of the cores. Infrequently used or large data sets can be kept in flash with the use of the *const* keyword for read-only variables. (Sanderson, Making It Run Fast And Fit in RAM, n.d.)

4.4 DEVELOPMENT ENVIRONMENT

Development on the PicoSystem can be done using either:

1. A native C/C++ cross-compiler
2. MicroPython
3. CircuitPython (itself a simplified variant of MicroPython)
4. A 32-blit SDK offering compatibility with other platforms

(Pimoroni Ltd., n.d.)

The use of the native C/C++ compiler was decided early on to maximize performance and allow optimizations toward the target hardware. Compiled C/C++ programs in the form of .uf2 binary files can be drag and dropped into the device when connected to a PC via USB.

Development Software is provided by the PicoSystem SDK which builds on the Raspberry Pi Pico SDK for the RP2040 microcontroller. The SDK provides an API with basic graphical functions and a base framework for game development including example tile maps and font.

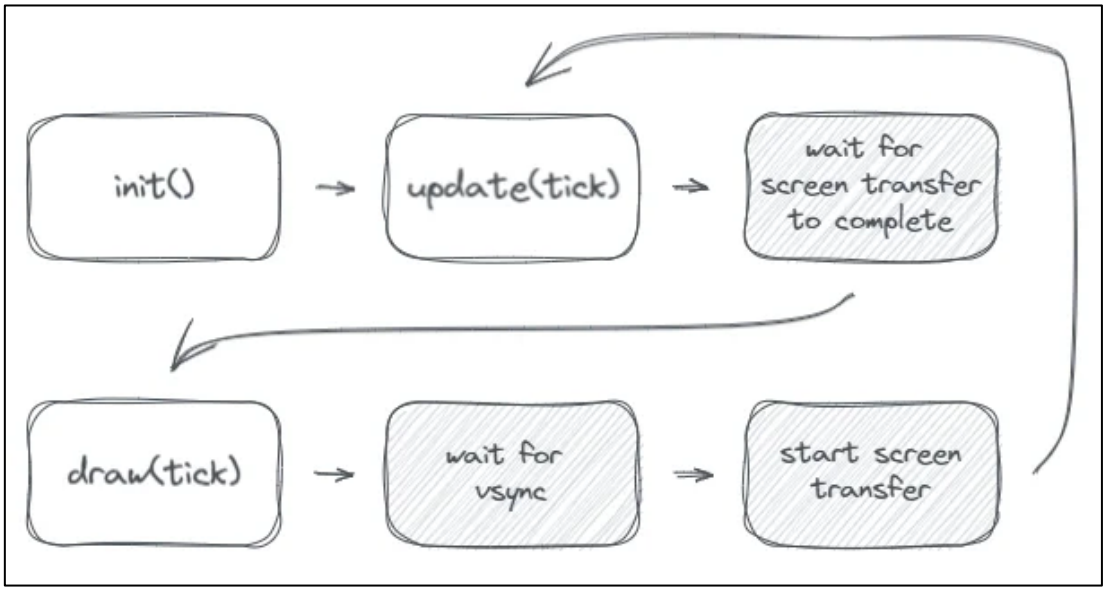

FIGURE 24: BASE FUNCTIONS PROVIDED BY THE PICOSYSTEM SDK FOR DEVELOPMENT. (WILLIAMSON, 2021)

Three main functions are given to assist the developer in getting started:

1. init() for initialization of game code.
2. update() designed for logic code while the framebuffer is copied to the display.
3. draw() for safe modification of the framebuffer itself.

The screen is refreshed at a rate of 40 FPS giving a frame time of 25000 microseconds in which the microcontroller can process all input, logic and display output. This is evenly split between the update() and draw() functions.

CMake is used to in compiling the final binary and Microsoft Visual Studio Code is used as an IDE for development along with Git for version control.

5 DESIGN

The engine and game prototype itself contain several core systems which need to be implemented. These can be split up into three main components:

- A complete software-based rendering pipeline.
- An asset loading system to stream in scenery meshes along with custom 3D models to showcase the ability of the engine.
- A game prototype containing several functional elements from an open world game.

5.1 RENDERING PIPELINE

As there is neither an existing GPU nor any 3D API available for the RP2040/PicoSystem, a complete rendering pipeline including software rasterizer must be implemented. This pipeline is heavily tied to the dual-core nature of the RP2040 in order to fully utilize the microcontrollers' full performance.

An initial design plan is to move all rendering tasks onto core 1. Core 1 would thereby act as more of a dedicated GPU and run independently of core 0 to create a full frame before passing the frame back to core 0 for any further processing.

Similar to the concept of display lists found in older 3D consoles and command buffers in modern APIs such as Vulkan and DirectX 12, core 1 would be fed with a list of triangles. These "triangle lists" will only handle triangles and leave final UI and post-effects rendering to core 0, which is able to make use of the PicoSystem API and its drawing functions to provide a GUI and other post-processing effects. This also reduces rendering load on core 1 and allows the rendering pipeline to be kept compact for future optimizations, as it is likely to be the primary bottleneck of the engine.

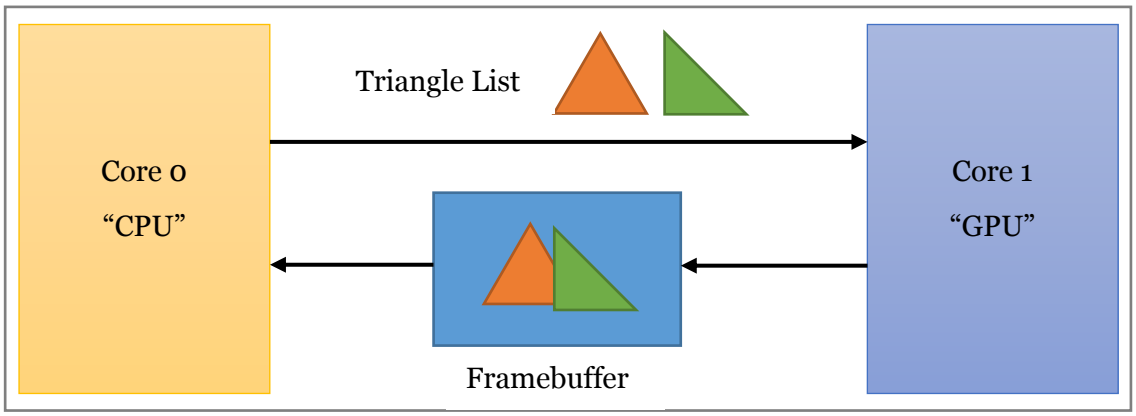

FIGURE 25: SPLIT OF TASKS BETWEEN CORES

Although the RP2040's cores contain optimized floating-point operations in ROM, the lack of a dedicated floating-point unit is likely to result in insufficient performance. The use of fixed-point integer calculations is therefore most likely necessary.

A Blender add-on capable of exporting meshes needs to be created to allow the testing of more complicated models.

5.2 CHUNK SYSTEM

As with most other open world games, a form of asset streaming is necessary due to the scenery meshes not being able to fit in the limited amount of RAM of the RP2040.

Additionally, a chunk cache may be implemented, as constantly loading meshes from flash memory is likely to hamper performance due to limited bandwidth and invalidate existing entries in the XIP cache causing further performance degradation.

This includes the extension of the Blender add-on to automate export of chunks.

A prototype world should be modeled which includes environments inspired by existing open world games such as:

- Cyberpunk 2077 – known for the large scale of environments including high-rise buildings
- Yakuza Series of games – known for the relatively small but highly dense city district of Kamurocho.
- The Legend of Zelda: Breath of the Wild - known for large natural environments filled with foliage.

The different environments are chosen to showcase the engine's ability to handle different geometry and density scales.

FIGURE 26: ENVIRONMENTS OF CYBERPUNK 2077 RELEASED IN 2020 (LEFT) AND YAKUZA 0 RELEASED IN 2015 (RIGHT). (CD PROJEKT S.A., 2022) (SEGA HOLDINGS CO., LTD., N.D.)

5.3 OPEN WORLD PROTOTYPE

A prototype open world game is created along with common open world components to demonstrate the capabilities of the engine. Along with scenery provided by the chunk system, the prototype world should have:

- Blocking physics to prevent the player from entering all areas of the game world
- At least one type of foliage
- NPCs and one form of enemy
- A simple form of gameplay, consisting of defeating enemies
- A day and night cycle

Each component is grouped together and executed procedurally, with each component deciding on how to add its own triangles to the triangle list. A budgeting system is to be implemented to limit the maximum triangles which can be added by any single component. As an example, only a limited number of NPCs can be displayed on screen at a time, with the remaining triangle budget being reserved for the chunk system to

add its own triangles. This is to prevent graphical artifacts from occurring when the chunk system is unable to load sufficient triangles to display the scenery correctly.

5.4 DEVELOPMENT ESTIMATE

Each core task consisting of implementing the rendering pipeline, chunk system and the completion of the prototype is given a rough estimate of 1.5 months to complete.

Due to the research-heavy nature and high overlap of the three implementation phases, this estimate contains sufficient margins to ensure enough time is available for testing, bug-fixing and documentation of results.

5.5 PROGRAMMING METHOD

Programming focuses on using a data-oriented design. This is done due to the tight RAM constrains of the development hardware, often requiring careful data alignment to 4-byte boundary. (Acton, 2014)

The engine should only use fixed memory allocations to allow for more predictable and consistent performance and memory use, as dynamic allocation can lead to memory fragmentation or allocation failure. Also, excess calling of functions can be detrimental to performance due to the required overhead and should be avoided. (Gregory, 2014)

Data structures are kept consistent, with in-game objects stored in arrays of structs modified and read out by functions dedicated to single tasks looping over these arrays.

An example of this is the NPC code. Code is split into a logic function, dedicated to keeping operations like movement and NPC decision making, and a dedicated rendering function, designed to produce triangles from existing mesh arrays and inserting them into the triangle list for rasterization. This is somewhat comparable to the update() and draw() routines of the PicoSystem SDK. Both functions loop over an array of structs containing NPC information such as position, their current state etc.

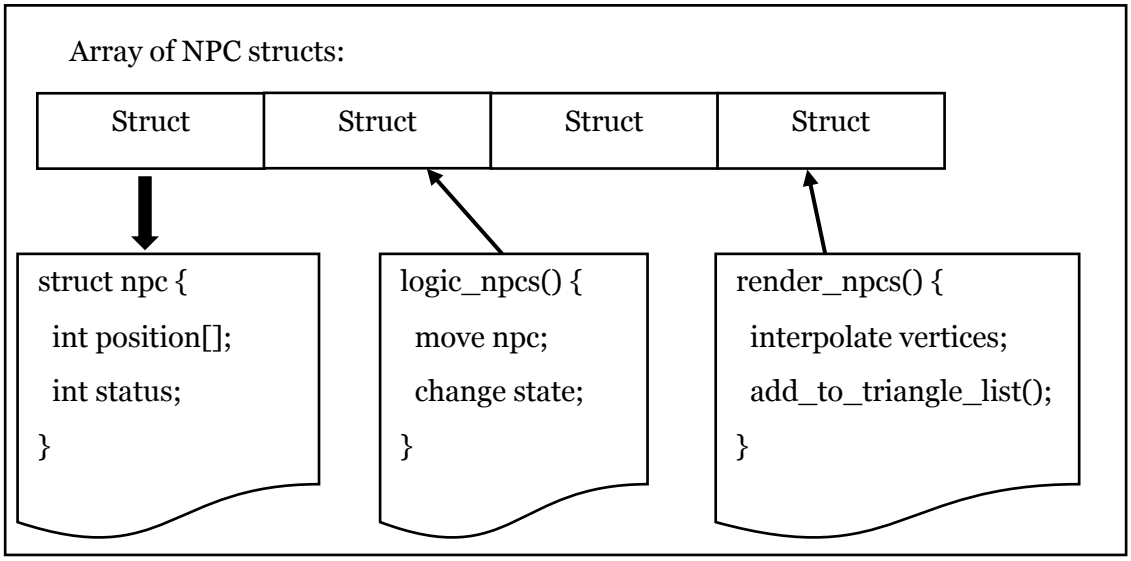

FIGURE 27: AN ARRAY OF STRUCTS HOUSING NPC INFORMATION IS MODIFIED BY SEPARATE LOGIC() AND RENDER() FUNCTIONS.

Decoupling logic operations and rendering functions allows for logic to be independently updated from rendering, ensuring constant updates on tasks that are also more time critical and less likely to be dropped. For example, an NPC may no longer be visible to the player long before it stops being simulated in the game world. (Llopis, 2009)

An approach to data-oriented design is commonly found in existing engines such as Unity under terms like entity component system (ECS). (Unity Technologies, 2019)

6 RENDERING PIPELINE IMPLEMENTATION

6.1 DUAL CORE UTILIZATION

The PicoSystem SDK only utilizes the first core by default. Enabling the use of the second core is done by adding the pico_multicore flag to the CMake project.

As two cores accessing the same data on the microcontroller is likely to cause corruption or stale information, two data sets are created to allow both cores to work on their own data set without contention. Each data set consists of a framebuffer (effectively creating a double buffered setup) and a triangle list containing all triangles to be rendered.

Synchronization of both cores is achieved by using the integrated FIFO queues each core contains to communicate with the other core. This synchronization is performed at the start of the draw() function on core 0 when framebuffers can be safely swapped without corrupting display output.

Core 0 handles all tasks related to game input and logic as well as preparation of the view/projection matrix and filling of the triangle list for later rendering on core 1.

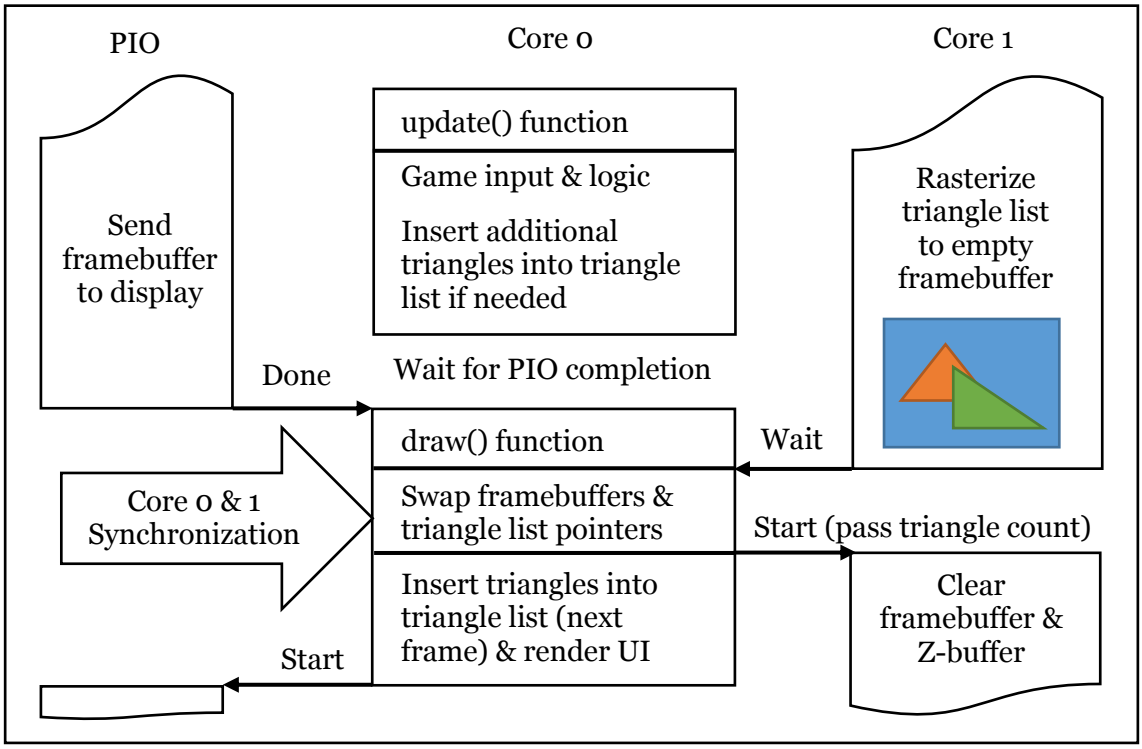

FIGURE 28: OVERVIEW OF PIO, CORE 0 AND CORE 1 TASKS AND SYNCHRONIZATIONS DURING A SINGLE FRAME.

If core 1 is unable to complete rasterizing a frame in time, the previous framebuffer and triangle list are retained and the draw() function on core 0 resumes after a short timeout. This ensures game logic can progress at a steady pace regardless of graphics slowdowns caused by high rendering workloads.

Core 1 in turn handles all rasterization tasks utilizing the triangle list created by core 0 and producing output in the assigned framebuffer. Once complete, core 1 waits for core 0 to finish copying its framebuffer to the display (start of draw() function) and returns the microseconds needed for completing the frame (to assist in performance profiling) before beginning again.

6.2 TRANSFORM PIPELINE

Triangles that are to be rendered can be passed to a dedicated render_triangle() function once they are in world space, which eases development by decoupling and encapsulating rendering functionality.

The render_triangle() function handles all the steps needed in a minimal fixed vertex transform pipeline. Its purpose is comparable to early dedicated fixed-function systems like the Silicon Graphics Geometry Engine. (Fuchs, 1987)

The render_triangle() function's responsibilities include:

- World to view/projection vertex transform.
- Near-plane clipping to prevent inversion of triangles beyond the camera. Produces an additional triangle if needed. (Kenwright, n.d.) (Gambetta, n.d.)
- Complete culling of triangles beyond the view frustum in Z direction.
- Back face culling.
- Per-vertex lighting based on distance to light source.
- Copying the resulting triangle into the currently assigned core 0 triangle list.

(Foley, Dam, Feiner, & Hughes, 1996)

Clipping of triangles on the sides of the view frustum is skipped to save performance, as most triangles fall within the values of the 16-bit guard-band on the sides of the viewport. (Giesen, 2011)

The function returns without adding a triangle to the triangle list if a triangle is either not visible in the scene or the triangle list itself is already full, thereby preventing any overflows.

As with other open world games, the view matrix is configured for a first-person perspective and player inputs modify the matrix directly. (Oosten, 2011)

6.3 FIXED-POINT ARITHMETIC

The RP2040 does not contain any form of dedicated hardware to perform calculations using floating point numbers. This is somewhat remedied through optimized floating-point functions burned into the ROM of the RP2040.

Initial attempts to use floating-point numbers for all calculations ended up with poor performance. This was subsequently resolved by adopting a fixed-point integer representation, allowing a boost in performance by more than 700%.

To prevent overflows especially when using multiplications, a fractional component of 10-bits was decided upon for the whole engine. This gives a rough distance granularity of one millimeter (or 1024 steps) for every full integer component representing one meter in the game world. (Bikker, 2003)

The fractional component is also ensured to be a power of two, allowing the compiler to apply bit-shifts instead of using the slower hardware divider.

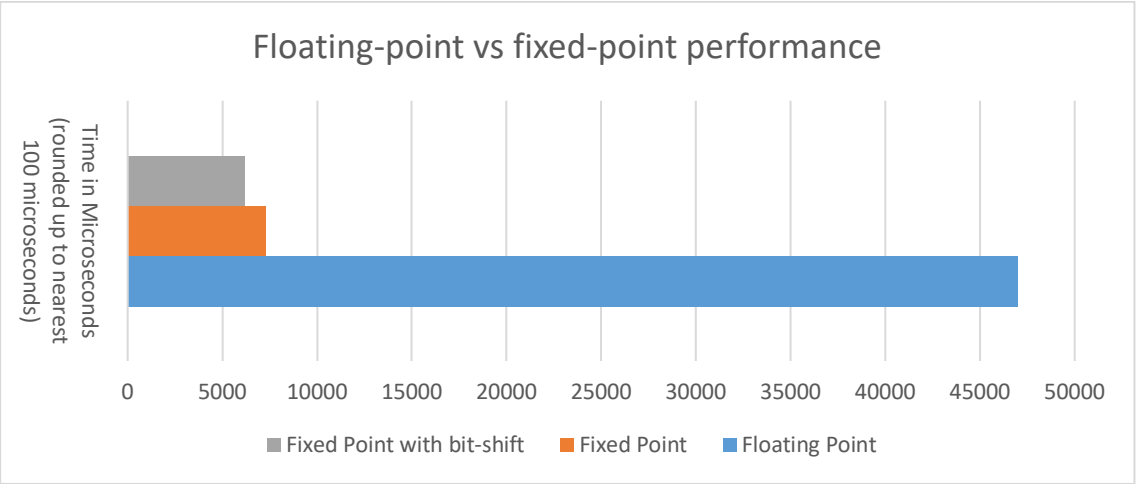

FIGURE 29: FLOATING-POINT VS FIXED-POINT PERFORMANCE. LOWER IS BETTER.

While most mathematical operations in the engine are performed using fixed-point arithmetic, certain variables are kept in floating point and converted when needed. This includes the camera position to ease developer modifications.

Another advantage of the use of 10-bit fixed point integer arithmetic is the ability to keep vertex points in a 16-bit integer format if their position does not exceed the 6-bit integer component. Objects can therefore have a size of around 64 meters in each direction before needing a larger integer type, thereby allowing meshes to consume less memory by around 36%. This also alleviates flash storage consumption and needed bandwidth from flash along with reducing XIP cache pressure.

Objects can then be transformed into world space when needed before the view-projection transform is applied. After culling and transform, triangles are likely to be in a view space which can be represented by 16-bit values, allowing the triangle lists to use 16-bit vertex positions as well.

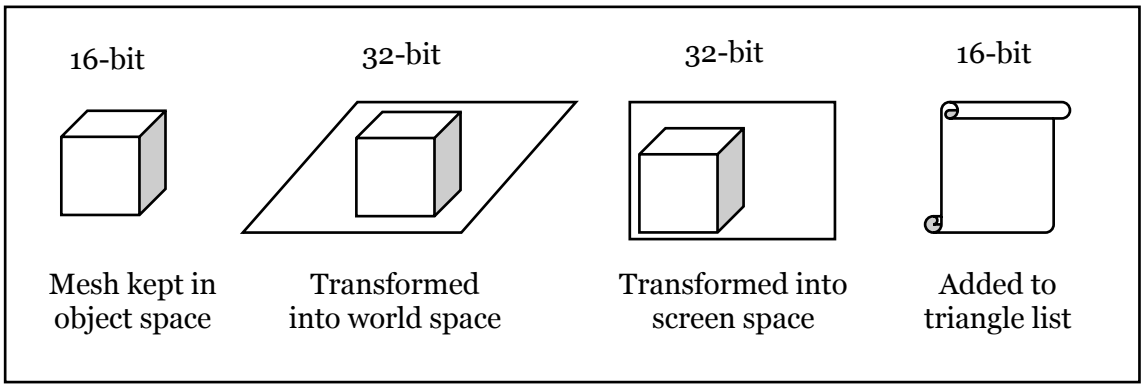

FIGURE 30: MESHES CAN BE KEPT USING 16-BIT INTEGER VERTICES IN STORAGE AND WHEN PASSING TO CORE 1 FOR RASTERIZATION TO SAVE MEMORY.

6.4 RASTERIZER

Triangles passed to core 1 need to have sufficient information to allow rasterization to proceed independently of core 0. Each triangle therefore consists of a struct with which all necessary information is passed on, consisting of:

- vertex position of the three vertex points.
- corresponding vertex parameters. This is either color or, if a texture is used, UV coordinates.
- a shader ID, to select which shader is executed when a pixel is visible.
- a texture ID, to select which texture is used in a shader, if needed.

As core 1 runs independently of core 0, the timing requirement for a frame change to 25000 microseconds, assuming realistic graphical workloads.

Unlike the scanline converters used in early 3D games such as Quake, the engine prototype rasterizer utilizes edge functions for finding pixels in a triangle. Individual edge functions define a value for a given pixel on a plane (or in this case the framebuffer) as either positive, zero or negative. Any values equal or above zero determines the half of the plane where a triangle resides. The combination of all three edge functions/sides of a triangle gives the definitive answer as to whether a pixel is within the triangle boundaries. (Pineda, 1988) (Fuchs, 1987)

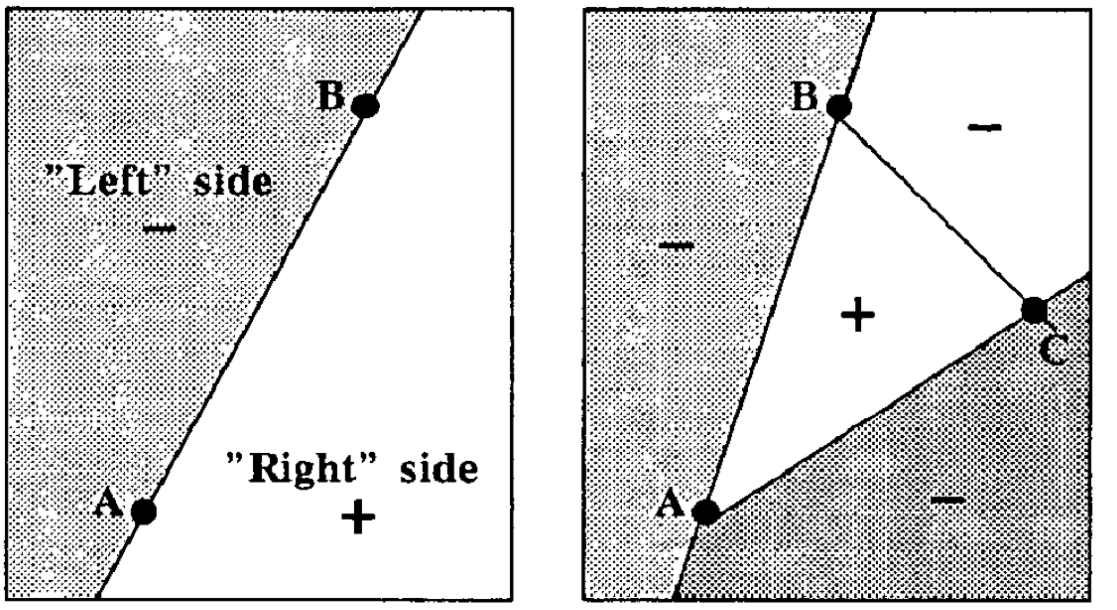

FIGURE 31: EVALUATING THE EDGES OF A TRIANGLE IS SUFFICIENT TO DETERMINE IF A PIXEL IS IN IT. (PINEDA, 1988)

The decision to use edge functions for a rasterizer mainly resulted from:

- Significantly faster implementation time.
- Lower memory consumption as buffers for sorting edges are not needed.
- No sorting of input triangles.

- Potential to allow multiple cores to process different parts of the screen without interference (e.g., using both cores of the RP2040 if needed to improve performance).
- Barycentric coordinates are simple to implement, allowing interpolation of vertex parameters such as color blending and UV coordinates.

(Scratchapixel, n.d.) (Abrash, Sponsored Feature: Rasterization on Larrabee -- Adaptive Rasterization, 2009)

Modern GPU architectures may utilize a combination of edge functions on a coarse level to bin groups of triangles into tiles, which in turn perform fine rasterization. This is mainly due to their easily parallelizable nature and ability to be extended for subpixel precision. As there are no dependencies on other pixels in a scene, tile-based renderers are possible instead of requiring rasterization to be performed in a scanline fashion. (Kramer, 2020) (Abrash, Sponsored Feature: Rasterization on Larrabee -- Adaptive Rasterization, 2009)

An overview of the rasterization process when looping through each triangle in the dedicated render_rasterize() function is therefore:

1. The extremes of the vertex points of a triangle are calculated and a bounding box created.
2. If needed, texture information in the form of UV coordinates are preloaded.
3. Area of triangle and inverse Z values are calculated for barycentric coordinates
4. Triangle is then filled by looping over vertical lines and horizontally over individual pixels.
5. Each pixel is checked whether it is in a triangle using edge functions.
6. If a pixel is in a triangle, check whether a value closer to the camera has already been written in the Z-Buffer.
7. If not, select a shader which can fill the pixel using the given shader ID of the triangle.

A Z-buffer is implemented to allow rasterization without the need to sort polygons in painters or reverse-painters order. This solution significantly reduces graphical artifacts from intersecting triangles as well as copies the most common approach used in modern 3D graphics.

The low 10bit Z-buffer does come with a relatively low resolution, causing occasional Z-fighting especially at a distance due to the reciprocal nature of stored values. This also prevents the final projection matrix from containing a near-plane too close to the camera or a far plane too distant. (Baker, n.d.) (Reed, 2015)

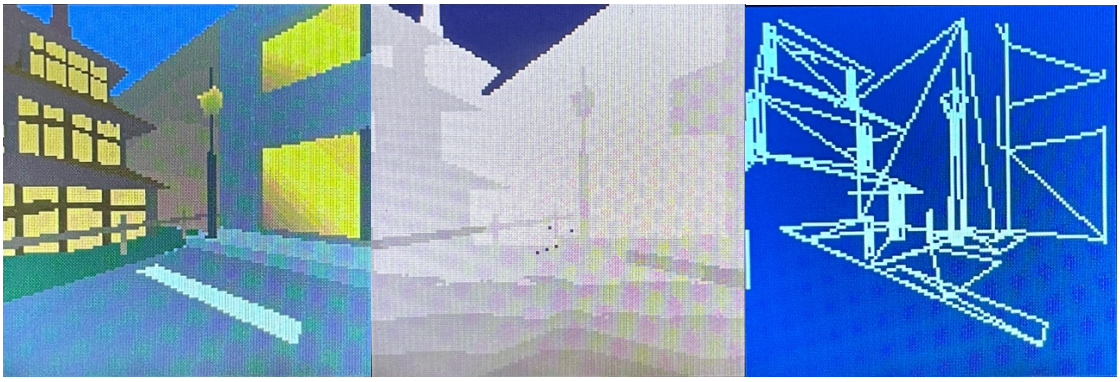

FIGURE 32: GAME OUTPUT(LEFT), Z-BUFFER OUTPUT(MIDDLE) AND WIREFRAME OUTPUT ON THE RIGHT.

Z-buffer output along with several other debug shaders are provided to help with debugging graphical output.

6.5 OPTIMIZATIONS

Several minor optimizations to the pipeline were integrated to improve performance. This was done as the rasterizer was the primary bottleneck of the application.

Among the optimizations were:

- Culling of triangles beyond viewport using Cohen-Sutherland algorithm. (Foley, Dam, Feiner, & Hughes, 1996)
- Back-face culling (triangles in counterclockwise direction)
- Early return when single edge function fails
- Early return upon encountering first pixel which is no longer shown on a horizontal line
- Increased core 1 priority to the RP2040 bus, eliminating the chance of contention when accessing a RAM bank at the cost of stalling other bus masters like core 0 trying to access the same bank.

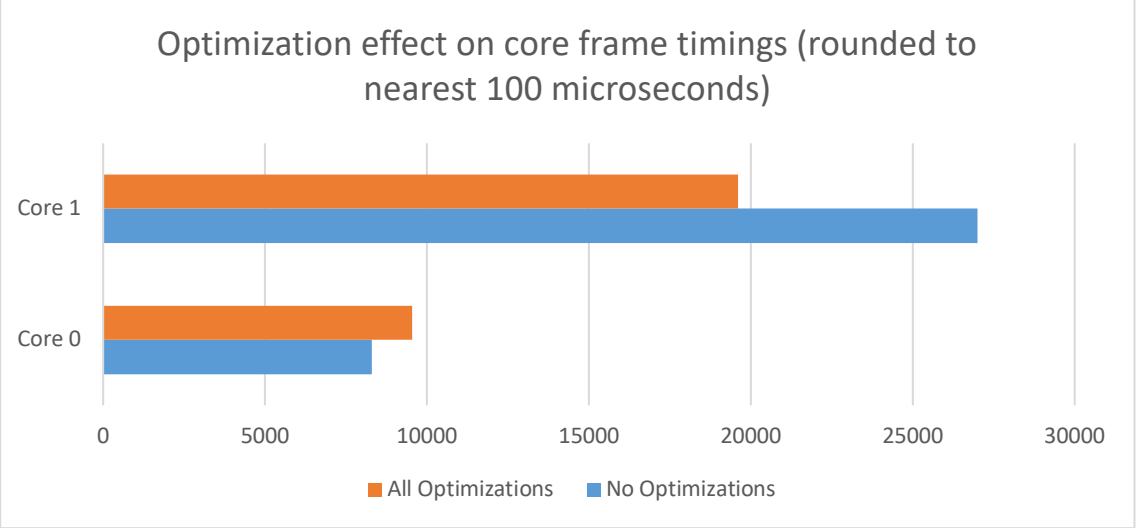

FIGURE 33: OPTIMIZATION EFFECT ON CORE FRAME TIMINGS (ROUNDED TO NEAREST 100 MICROSECONDS)

When combined, the performance profile of both CPU cores changes. This is due to additional workloads on core 0 for performing culling operations and facing higher contention when accessing RAM as core 1 is given higher priority. The benefits to core 1 are significant enough however to outweigh the cost, as rasterization in the default starting scene can be completed within the allocated time frame of 25000 microseconds for a single frame. Another benefit of performing culling operations on core 0 is the ability to ensure only visible triangles are added to the triangle list, thereby reducing required memory for both lists.

The RP2040 can execute code from RAM instead of the flash and the related XIP cache. This is usually faster when factoring in cache misses and contention if code is stored in a ram bank only accessed by a single core.

Attempts to utilize the speed improvement by moving the complete rasterization function to the scratchpad/stack RAM of core 1 resulted in slower performance. By

receiving help from one of the engineers at Raspberry Pi, Graham Sanderson, the problem was found to be the use of function veneers which translate function calls between RAM and flash. As the rasterizer extensively utilized divisions, which themselves are transparently turned into function calls during compilation to utilize the separate hardware divider, a noticeable performance hit was incurred. This was solved by moving division functions into RAM using a PICO_DIVIDER_IN_RAM=1 target compile definition. (Sanderson, Functions from RAM run slower than from XIP cache, 2022)

Moving the rasterizer instructions to scratch_x RAM (i.e., core 1 dedicated bank) resulted in an additional 3% performance improvement on core 1. Additionally, core 0 no longer needs to share the XIP cache with instruction fetches intended for core 1, reducing cache pressure and contention. This itself improves performance by 30% on core 0.

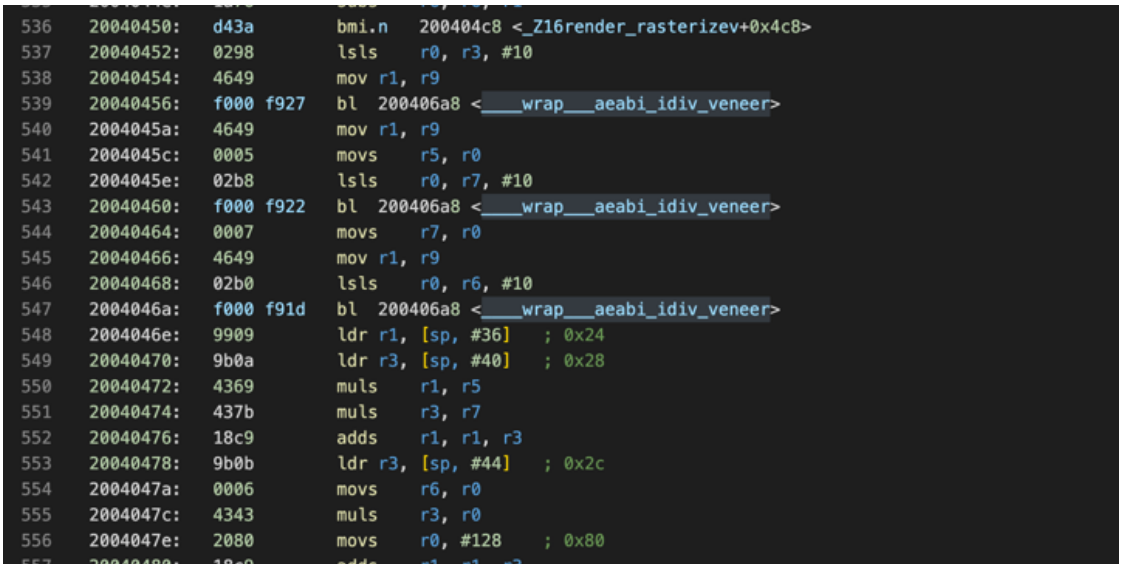

FIGURE 34: COMPILER PRODUCED ARM ASSEMBLY CODE SHOWING DIVISION FUNCTION CALLS WRAPPED IN VENEERS.

Other attempts to improve performance such as using a more advanced rasterizer failed. This is likely due to additional optimizing logic consuming processing cycles on already small triangles due to the low screen resolution. The XIP cache also does not provide the ability to prefetch instructions, which leads to long waits when new instructions cannot be fetched directly from the cache. This is more likely with larger amounts of code, thereby causing stalling if logic complexity is too high. Small loops are therefore preferable over common desktop optimizations like loop unrolling.

7 CHUNK SYSTEM IMPLEMENTATION

7.1 CHUNK LOADING SYSTEM WITH LEVEL OF DETAIL

The game world is split into a grid of square cells, each containing a mesh of the environment. As the Player moves from one chunk to the next, chunks are loaded around the player to ensure the player is always surrounded by visible chunks.

The chunk sizing and layout was determined through tests and subjective appearance of draw distance to yield a suitable compromise in scene complexity and size.

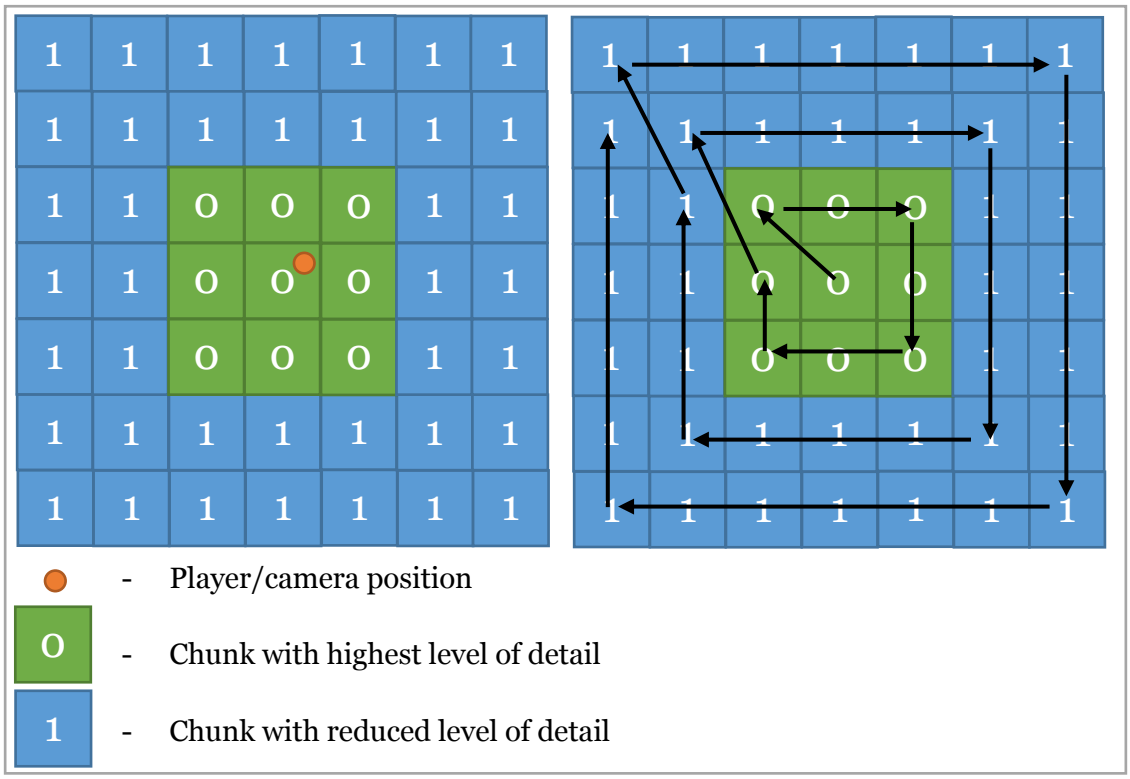

FIGURE 35: LOADING OF CHUNKS IS PERFORMED IN AN ONION LAYER LIKE MANNER TO REDUCE OVERDRAW AND POSSIBLE Z-FIGHTING.

To reduce overdraw, chunks closest to the player are loaded in first, with chunks furthest away being loaded in last. This also reduces the chance of Z-fighting in meshes which are close to each other.

Chunks are also stored in two variants. One variant is modelled with a full level of detail, and one with a reduced level of detail. This allows meshes far away to be loaded in with significantly lower vertex count reducing pressure on the rasterizer and increasing performance.

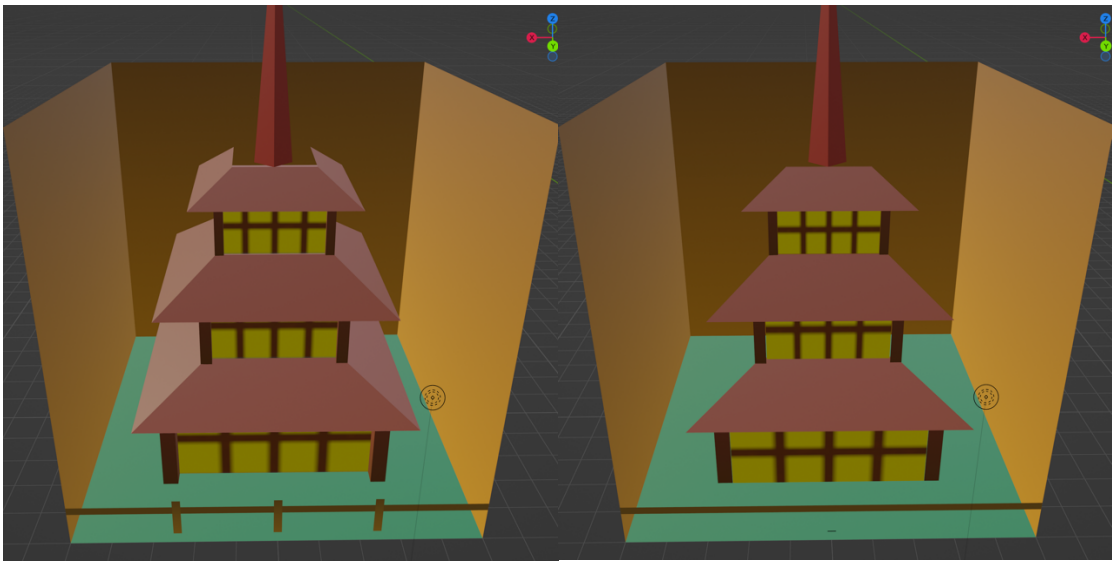

FIGURE 36: CHUNK WITH FULL LEVEL OF DETAIL (LEFT) AND REDUCED LEVEL OF DETAIL (RIGHT).

7.2 CHUNK CACHE

While a chunk system with LODs works to reduce vertex load, a significant bottleneck is still encountered when loading chunks constantly from flash for each frame. To reduce this impact, a chunk cache consisting of a separate triangle list is created. This buffers all chunk triangles when the player is not actively moving to a different chunk.

As each chunk is only 10 meters in size in each direction from object origin, they can be stored as 16-bit vertex models. Chunk locations are stored separately and used when adding meshes to the triangle list to calculate final world coordinates of each triangle.

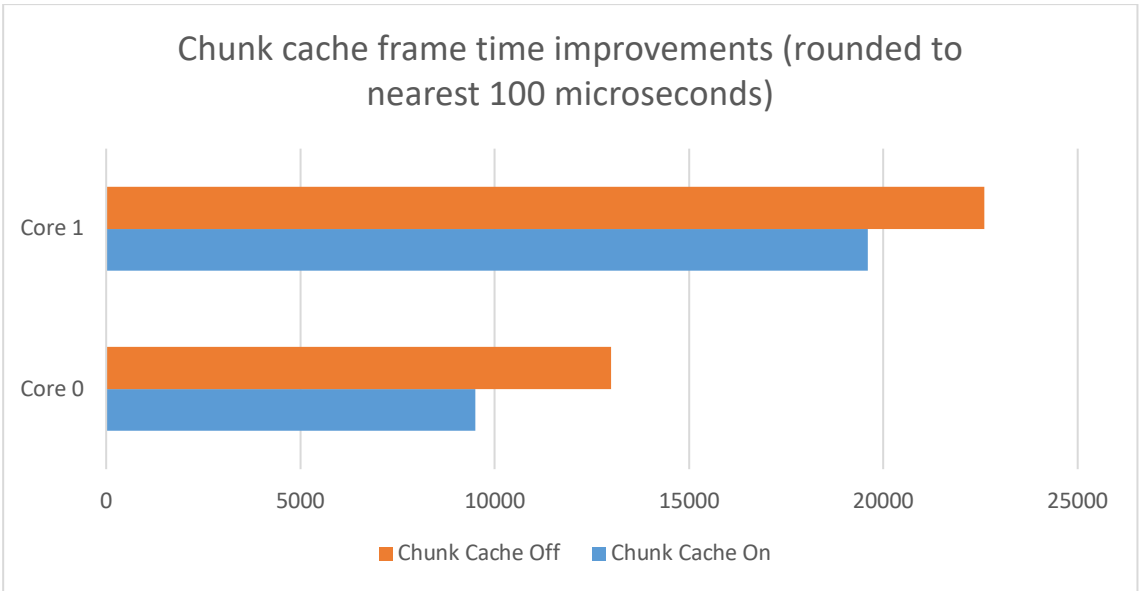

FIGURE 37: CHART SHOWING PERFORMANCE IMPROVEMENTS WHEN UTILIZING CHUNK CACHE (LOWER IS BETTER).

The use of a chunk cache significantly benefits both cores, as the XIP cache does not need to flush out active instructions or data when existing chunks can instead be loaded

from RAM. This leads to a 27% cut in frame time for core 0 and a 13% cut in frame time for core 1.

Frame jittering when entering a new chunk does exist as the chunk cache has to be refilled with updated meshes. This is deemed to be an acceptable tradeoff for the increase in performance in all other frames where a user is not entering a different chunk.

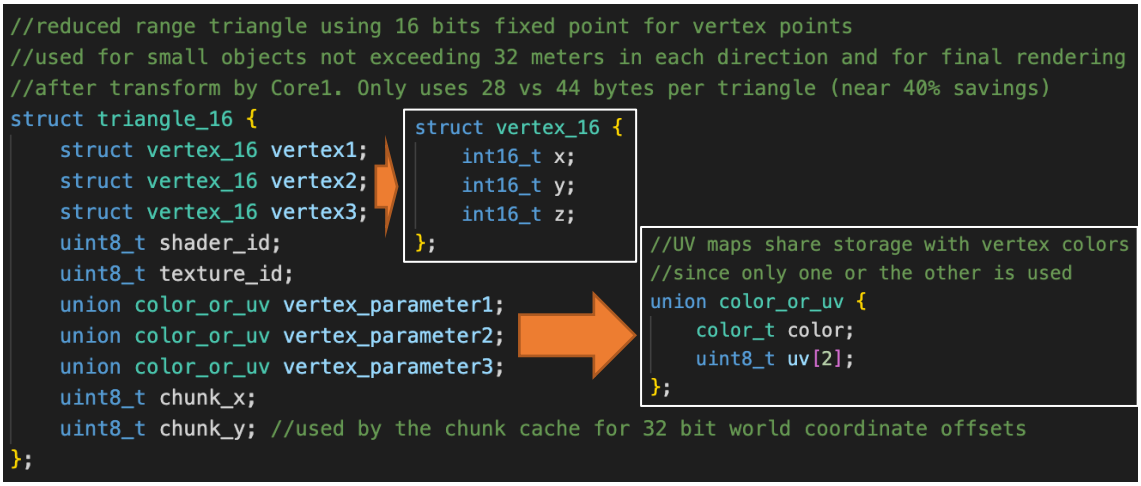

FIGURE 38: 16-BIT TRIANGLE STRUCT USED TO STORE MESH TRIANGLES IN FLASH, THE CHUNK CACHE AND TRIANGLE LISTS.

A more advanced chunk cache was considered, utilizing the ability to shift chunks and only reload new chunks which were not already residing in the cache. This was dropped however, as the additional code complexity and the additional required memory to handle worst case scenarios outweighed the benefits of a simpler reloading mechanism.

7.3 BLENDER ADD-ON

As manual entry of vertex points for larger amounts of meshes is impractical an add-on for Blender, the open-source 3D modelling application, is created. Written in Python and using the Python API and scripting environment found in Blender itself, the add-on converts meshes automatically into compatible C header files which can then be included in the engine.

The add-on also natively exports chunks into a single large header file containing both LOD stages along with light positions for each chunk for dynamic lighting calculations. Chunk metadata such as the number of triangles and a pointer to the associated triangle list are stored in a large 2D array, with one array for each LOD stage.

To ease modelling work, chunks can also be optionally repeated and are consequently only exported as a single mesh, saving storage.

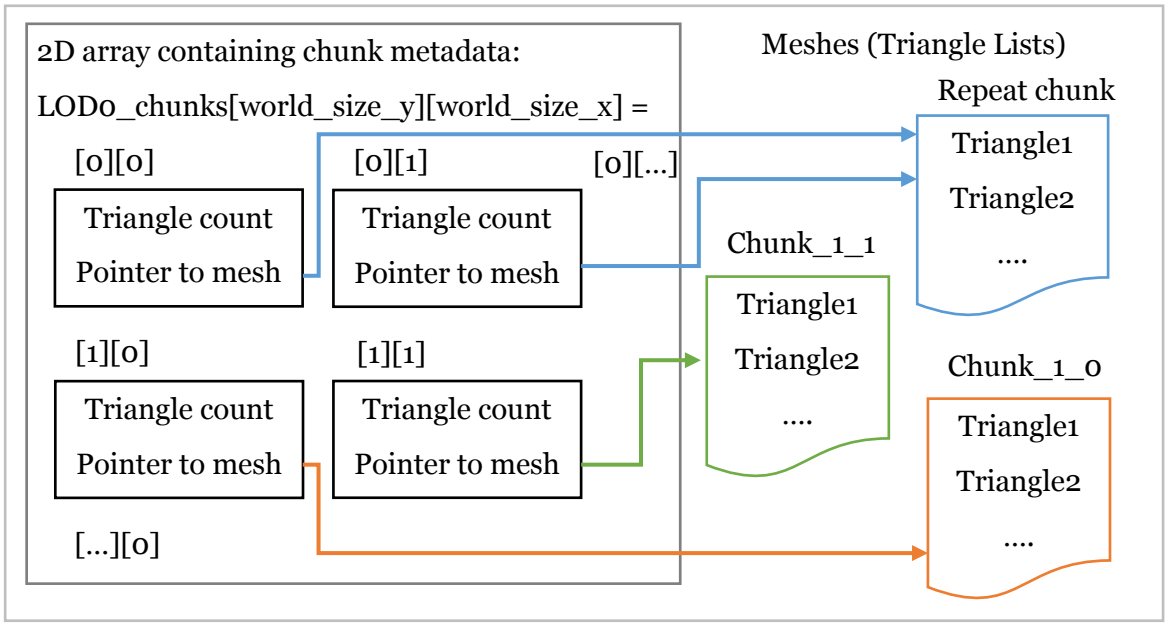

FIGURE 39: CHUNK METADATA IS STORED IN A 2-DIMENSIONAL ARRAY WITH POINTERS TO MESHES. THIS STRUCTURE IS REPEATED FOR LOD1.

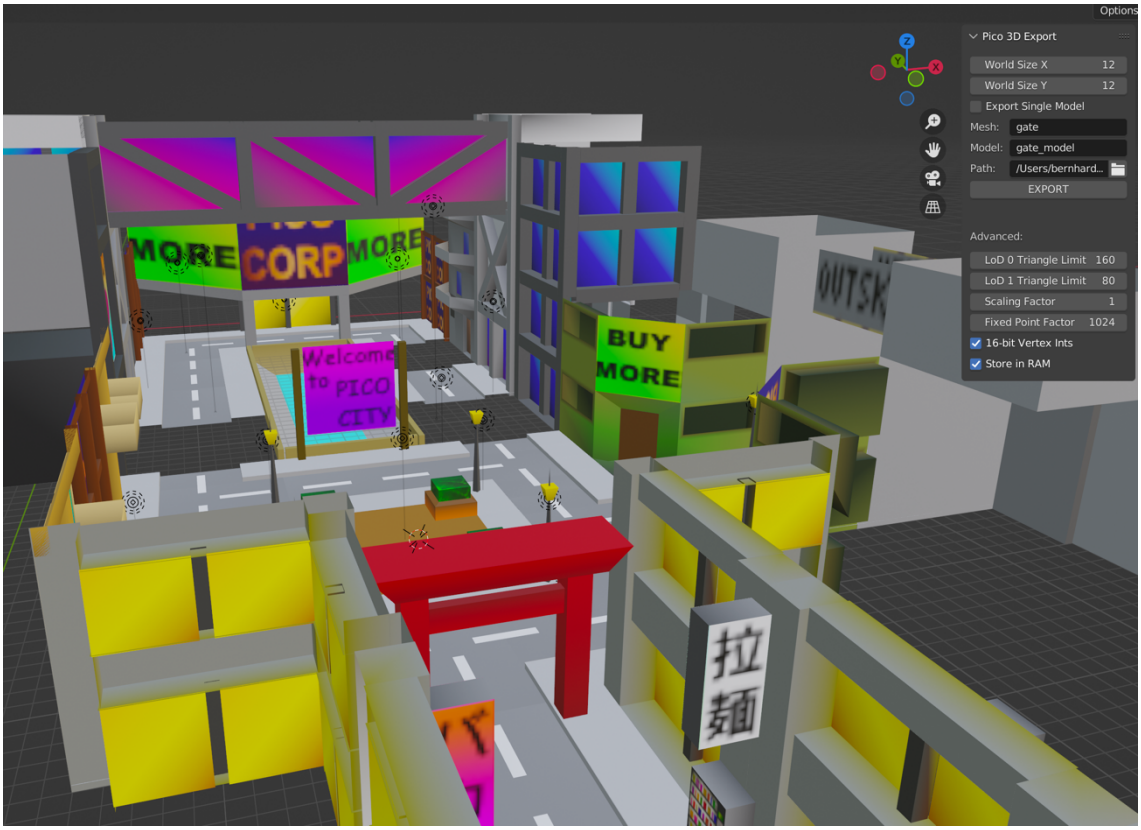

FIGURE 40: BLENDER WITH ADD-ON INTERFACE AND PROTOTYPE GAME WORLD DIVIDED IN CHUNKS.

7.4 PER-VERTEX DYNAMIC LIGHTING

Open world games frequently employ changes in daylight to give players a sense of time progression. This is achieved with changes in lighting to the scenery and sky box or background.

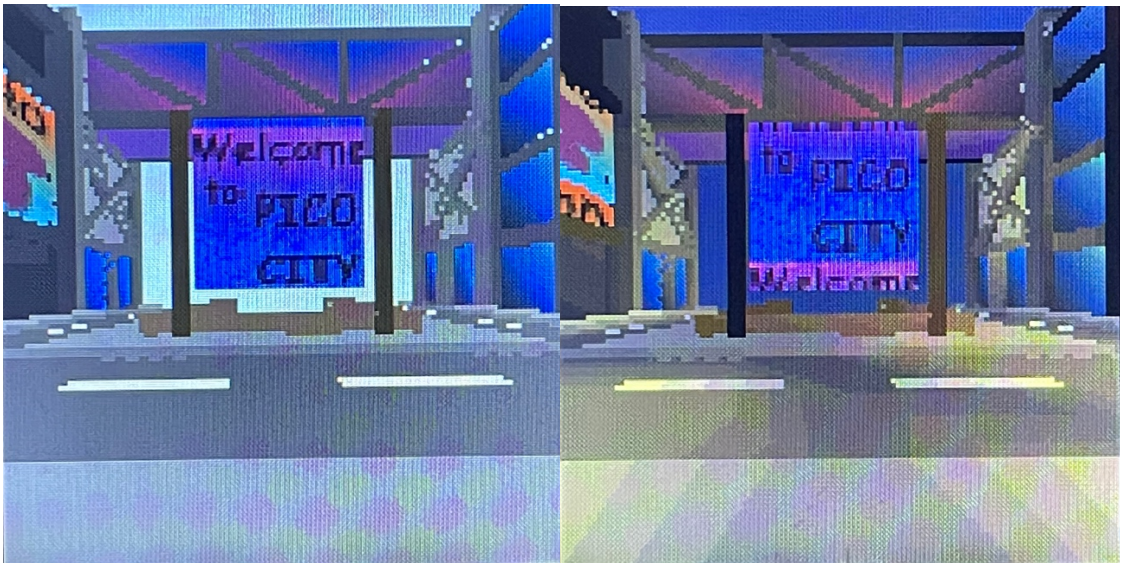

FIGURE 41: DEFAULT STARTING SCENE, ONE VARIANT WITHOUT DYNAMIC LIGHTING APPLIED (LEFT), ONE WITH DYNAMIC LIGHTING (RIGHT).

The use of pre-baked and dynamic lightmaps is a significant consumer of RAM and storage, the use of which should be avoided (Abrash, Graphics Programming Black Book, 1997). An alternative is to prebake color values on vertices. This reduces the ability to dynamically change environmental lights as is the case for example with streetlamps only being turned on at night.

Utilizing the chunk loading system, however, allows lights to be exported along with mesh data. This allows lighting to be calculated at runtime to affect color values of individual vertex points depending on time of day in-game.

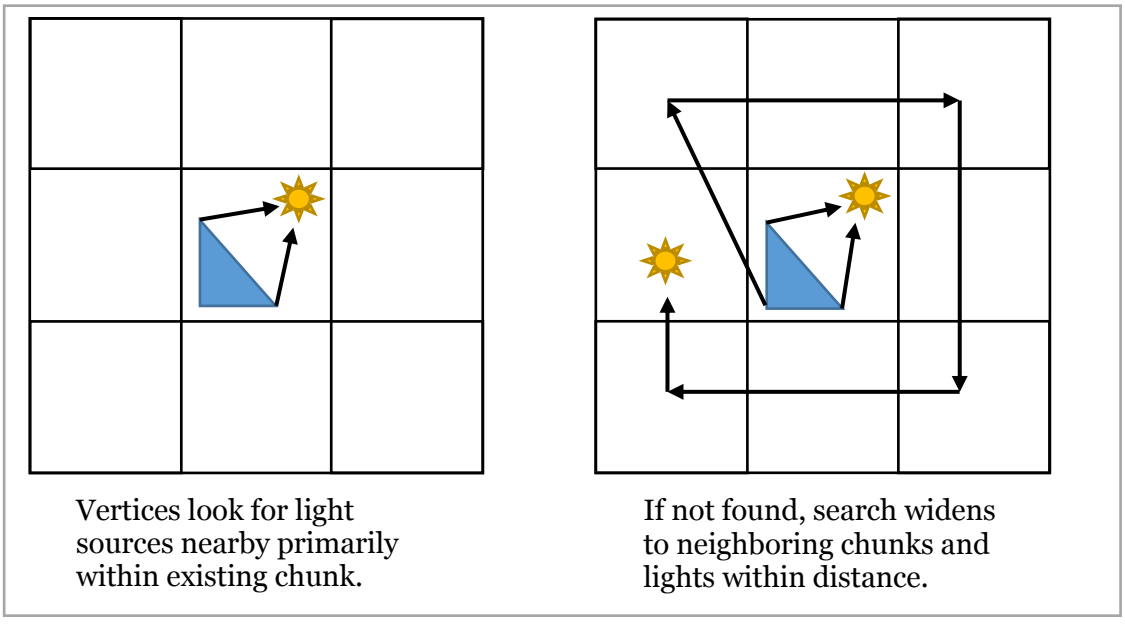

FIGURE 42: VERTEX POINTS CALCULATE DISTANCES TO NEARBY LIGHT SOURCES.

The disadvantage is lowered performance, as dynamically lighting each vertex point consumes resources to calculate distances (skipping square root calculations due to performance cost) and comparing them for each light source. This is especially problematic if light sources are not close enough or simply do not exist in the current chunk the vertex is in.

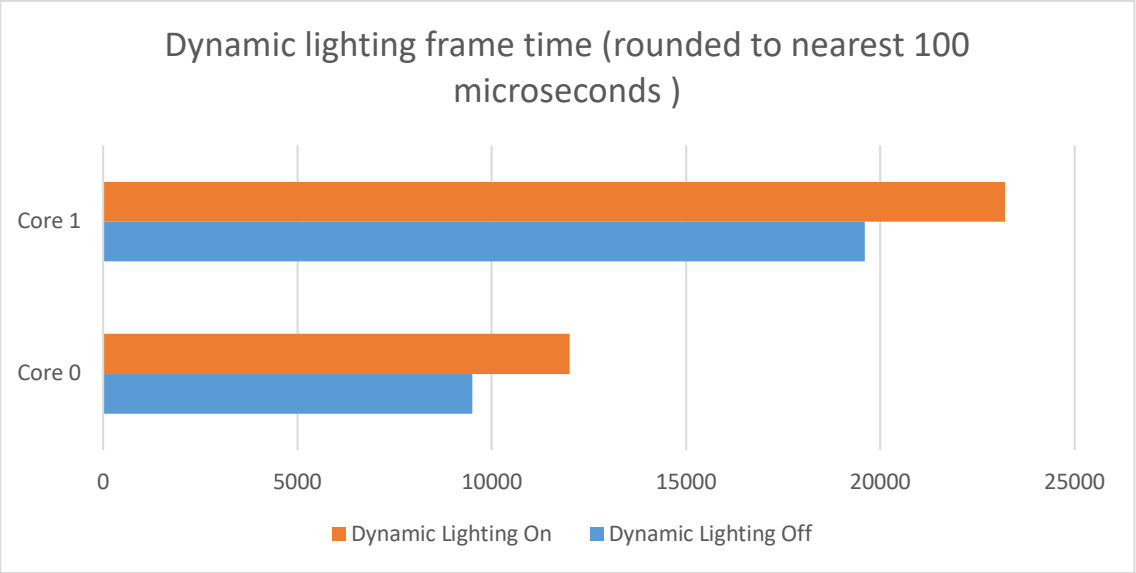

FIGURE 43: PERFORMANCE COMPARISON OF ENABLED/DISABLED DYNAMIC LIGHTING

8 PROTOTYPE IMPLEMENTATION

8.1 MODEL LOADING & TECHNIQUES

Models which are to be displayed are loaded each frame into the triangle list for rendering. Smaller models such as NPCs, objects and foliage are given priority before meshes from the chunk system are loaded in. This reduces the chance of objects being partially obscured due to low Z-buffer bit depth and lowers overdraw as most objects are likely to be close to the player.

The low bit depth of the Z-buffer also increases the likelihood of Z-fighting, especially when in combination with the near-plane clipping code due to the low accuracy of the resulting calculated reciprocals. This necessitates careful modelling to ensure large surfaces are not too close to each other in parallel or multiple layers of geometry are applied to meshes.

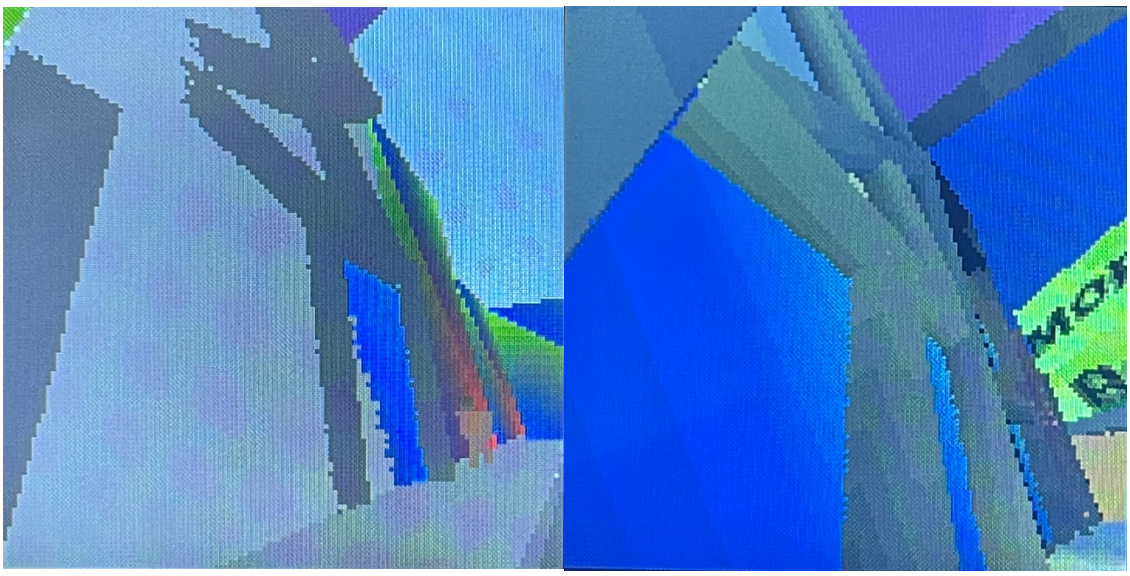

FIGURE 44: CLOSE SECONDARY SURFACE ON A BUILDING CLIPPING THROUGH (LEFT), CORRECT APPEARANCE ON THE RIGHT.

A side benefit of ensuring unseen surface removal is increased performance, as surfaces which are only present once do not need to be overdrawn.

Another modelling optimization is to completely remove chunks which are not accessible or visible to the player during normal gameplay. Existing chunks can in return have an increased polygon count without exceeding the triangle list limit. This allows more detail to be presented for example in alleys or inside buildings.

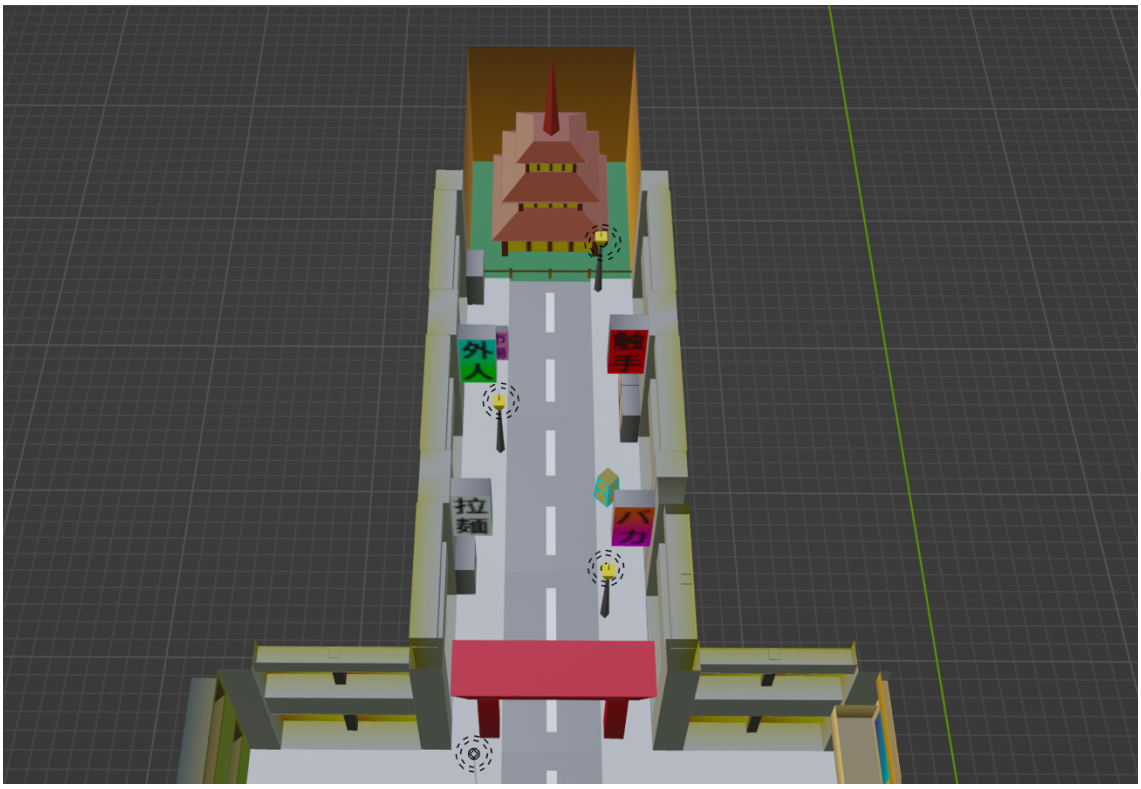

FIGURE 45: NOT MODELLING INACCESSIBLE OR NOT VISIBLE CHUNKS REDUCES GRAPHICAL WORKLOAD.

8.2 NON-PLAYER CHARACTERS

Open world games frequently make use of non-player characters (NPCs) and enemies to provide the player with entertainment. These characters further need to be animated to give them a live-like appearance. However, the use of advanced techniques like vertex weighted bones and skeletons are too computationally expensive to be effectively done on a microcontroller. (Ars Technica, 2020)

A viable solution to this problem is the use of linear interpolation between already posed meshes to animate characters. The use of linear interpolations usually comes with computationally slow integer divisions. This can be resolved by ensuring timings in character animations are a power of two to allow the compiler to apply bit shifts instead of actual divisions to increase performance.

Rotation matrices can also be skipped in favor of simpler mathematical negations of vertices if characters only need to face in one of the cardinal directions.

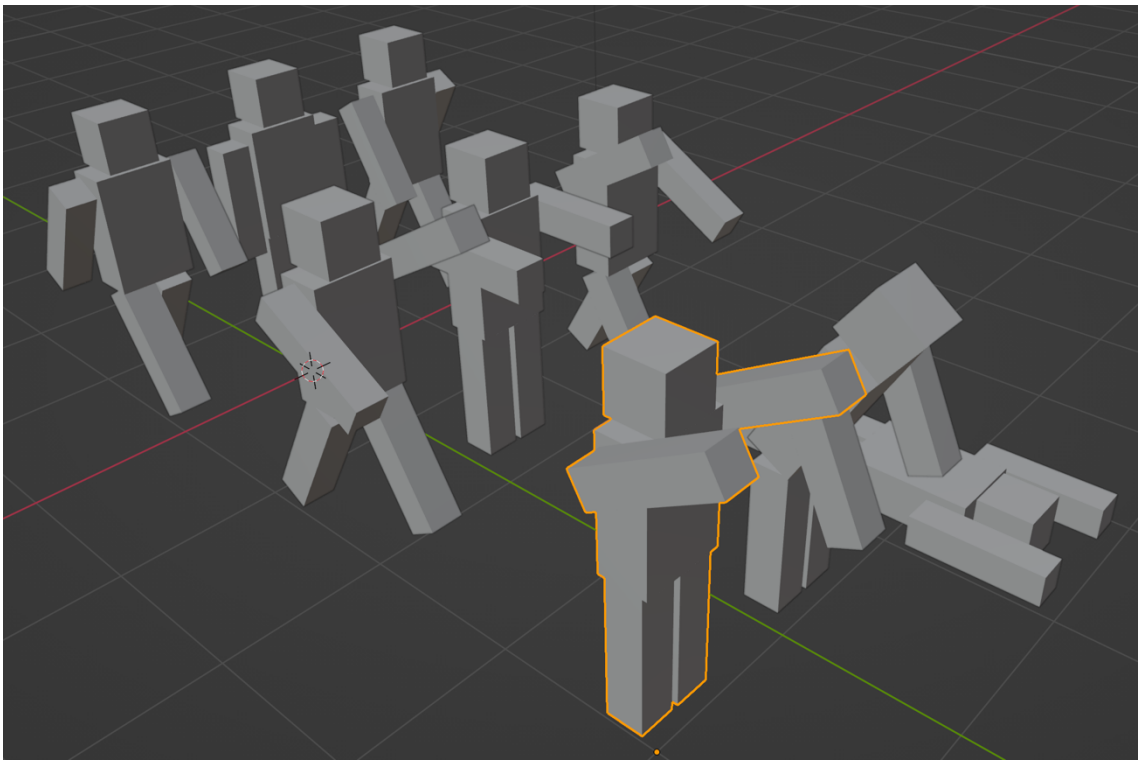

FIGURE 46: NPC MODELS IN VARIOUS POSES FOR LATER INTERPOLATION BY THE ENGINE.

As the process of blending meshes is expensive to calculate, significant performance gains are achieved using view frustum culling. This allows NPCs to be completely skipped when not in view of the player.

In the prototype, a variant of NPCs is produced as enemies to provide gameplay. These "zombies" differ in containing additional animation targets as they react to the player and their actions.

8.3 FOLIAGE

Scenery containing larger and simpler objects such as buildings and landscapes can be handled using chunks. Foliage (Grass, trees, bushes etc.) is a common graphical addition unsuited for this method due to higher graphical requirements. Commonly implemented using many impostors drawn close to the player, newer games have progressed to using actual geometry for elements like grass blades. (Wohllaib, 2022)

For the prototype, grass is implemented by feeding the triangle list with a grid of individual grass elements. These elements consist of simple pyramids to maintain performance, while the top vertex element is moved to simulate wind activity. To reduce overdraw and improve performance, as with the chunk system, grass elements closest to the player are filled first with further foliage added according to distance.

To improve performance, dynamic lighting is skipped in areas with foliage and a custom vertex shader (modifying vertex colors) is applied to simulate day- and nighttime changes.

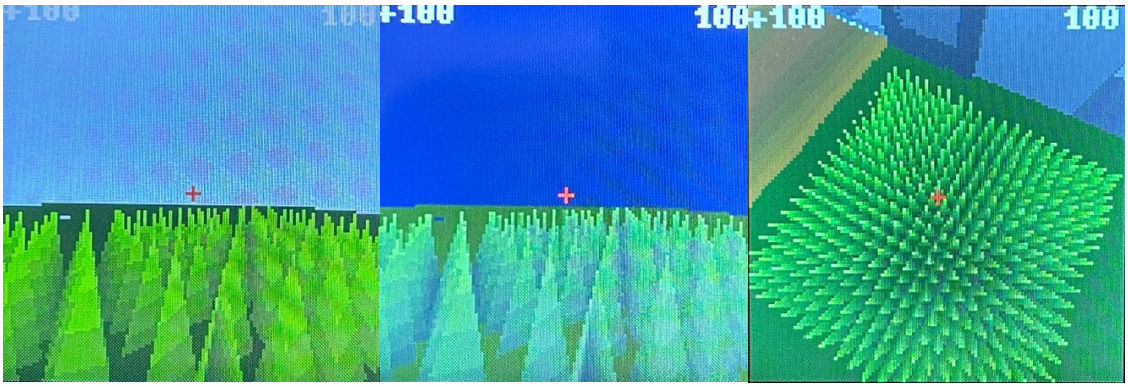

FIGURE 47: FROM LEFT TO RIGHT: GRASS WITH DAYLIGHT LIGHTING, NIGHTTIME LIGHTING AND ARRAY OF GRASS AS SEEN FROM ABOVE.

8.4 MEMORY CONSUMPTION & PERFORMANCE

The completed engine and prototype demonstration world come with the following memory consumption based on the total available RAM of the main banks (256KB) of the RP2040 (not including both 4KB RAM banks dedicated for each core):

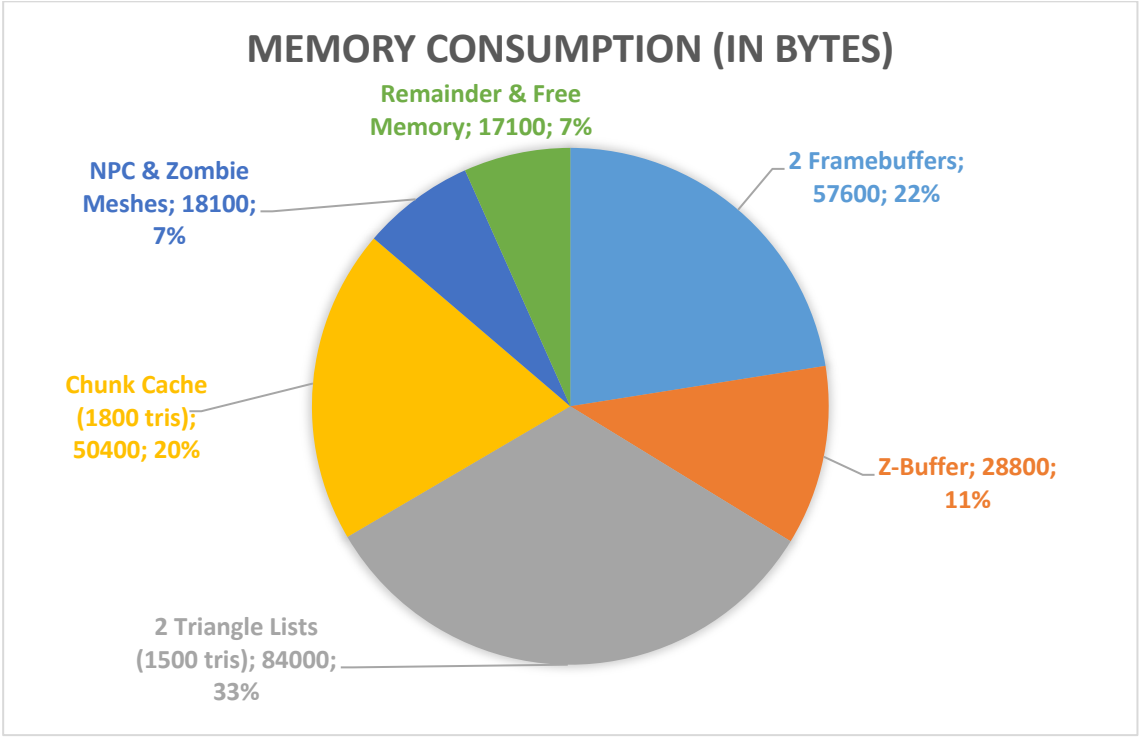

FIGURE 48: GRAPH SHOWING MEMORY CONSUMPTION OF MAIN VARIABLES IN RAM.

Performance is measured by implementing a benchmark utilizing the demo mode of the prototype game. This measures average frame time and frames rendered along with categorizing frames based on frame time.

Due to the method of implementation of the engine, synchronization of both cores only happens at the beginning of the draw() routine. Frame rate is therefore effectively the

supported display refresh divided by a whole number, representing the time taken to render the frame. As an example, if core 1 can fully render a frame within 25 milliseconds (i.e., one frame of time), the effective frame rate is 40/1 = 40FPS.

However, if core 1 requires slightly more time, for example 26 milliseconds, the next synchronization only occurs at the next V-Sync another frame later. This produces the effect of a 2-frame lag and therefore a frame rate of 40/2 = 20FPS. Further slowdowns cause effective drops to 13.3FPS (40/3 or 50-75 milliseconds required core 1 time), 10FPS etc.

Running the benchmark utilizing a final prototype version produces the results seen in Table 1.

TABLE 1: BENCHMARK RESULTS

Condition	*Value*	*Note*
Frames rendered within 25,000 microseconds	3,123	Effectively 40FPS
Frames rendered within 50,000 microseconds	1,964	Effectively 20FPS
Total frames rendered	5,087	Theoretical maximum of 7000 frames
Average frame time in microseconds	25,366	Total rendering time divided by total frames rendered

Frame rate drops below 20FPS are possible during actual gameplay when the player for example looks straight at a wall. This is caused by the software rasterizer being heavily fillrate limited, and scenes which fill up the display likely to cause the highest performance degradation.

The engine as configured for the prototype world can output up to 1,500 triangles per frame, leading to an effective maximum triangle count of 60,000 triangles per second at 40 FPS. This number only factors in visible triangles as culling operations on core 0 effectively remove non-visible triangles before rasterization takes place.

8.5 PROTOTYPE GAME ARCHITECTURE

Most important functions of the prototype game, including the rendering pipeline, can be mapped out in a call graph as shown below in Figure 48. Mesh data is mostly retrieved from header files as individual triangles, transformed as needed, and passed on for final rendering using the render_triangle() function.

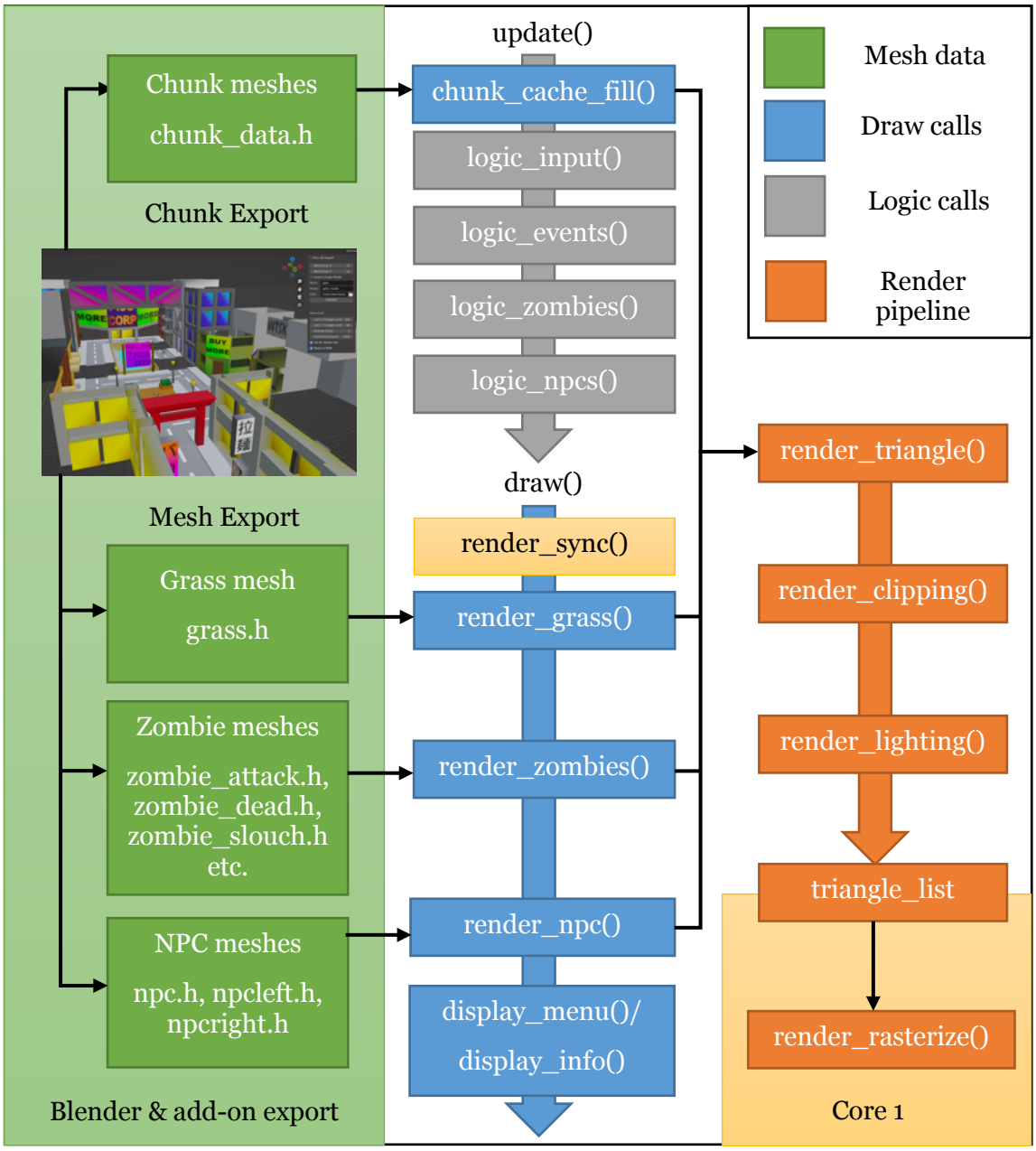

FIGURE 49: FUNCTION CALL GRAPH WITH MESH DEPENDENCIES

9 CONCLUSION

The completed prototype successfully runs, proving the viability of an open world engine and game on a microcontroller-based handheld. Between 20 and a maximum of 40 frames per second are achieved to provide a relatively consistent gameplay experience depending on graphical workload. Varying environmental models found in the prototype also showcase the flexibility of the engine in providing different aspects and variants of an open world game.

The engine itself is also kept relatively simple, allowing other developers to produce custom games and modify it according to their needs. Coupled with the Blender add-on for exporting meshes and chunks, the engine allows for rapid development of new content and rapid reuse of existing components.

Issues were encountered early on due to a lack of floating-point hardware acceleration, however. Fixed-point calculations often contained issues when dealing with overflows and reciprocals limiting precision in areas such as the Z-Buffer. While many are resolved to an acceptable extent, a few graphical issues and common artifacts persist due to these limitations.

FIGURE 50: SCREENSHOTS OF THE FINAL PROTOTYPE GAME.

Noticeable difficulty was also had in applying more complex algorithms to speed up code sections such as the rasterizer. This was caused by the lack of prefetch capability of the XIP cache, punishing excessive use of CPU instructions when not run from RAM. The implemented rasterizer is therefore straightforward and without additional complexity.

Producing sufficient graphical content for the prototype also consumed a significant amount of time. The resulting prototype therefore contains a rather constrained world measuring only 12 by 12 chunks in size (120m by 120).

9.1 POTENTIAL IMPROVEMENTS

During and after development, several components of the engine were noted to be likely candidates for further improvements in the future. These include:

- Unifying coordinate systems to be more in line with Blender. This would reduce developer confusion when dealing with differing orientations both in the engine and when modelling.
- Improved or additional tooling for components such as NPCs, mesh imports and texture support.
- Modification of rasterizer to perform after Z comparison for partially transparent textures.
- The rasterizer itself does not contain additional rules to handle gaps and holes between triangles which should be resolved.
- Implementation of a dedicated texture cache to prevent stalling the rasterizer during a cache miss.
- Core 1 rasterizer function reprogrammed or optimized to improve performance. A scanline converter or coarse-grained binning rasterizer may be of use here.
- Custom linker file to separate variables into separate ram banks, thereby preventing excess collision of cores when accessing memory. (Styger, 2012)
- Higher resolution or faster rendering using alternating scanline rasterization. This potentially requires DMA to transfer completed lines to framebuffer.
- Exporting all lights which may affect vertices in a single chunk to reduce the search space for dynamic lighting.
- Pre-calculated vertex lighting for vertices affected by fixed scene lighting.
- Additional world offset to reduce graphical artifacts due to distance from the world origin.
- Procedural generation and placement of chunks, as modelling work is a significant use of development time, allowing faster reuse of assets.
- Changes to the chunk system allowing easier reuse of existing chunks to save placement time and save memory.

Several of these optimizations are likely to be implementable only once a game's requirements are fully known and developed in order to specifically target bottlenecks which may hamper performance.

9.2 OUTLOOK

As microcontrollers become more performant over the next several years, their ability to perform more complex 3D rendering tasks is likely to increase. Higher resolutions as well as more complex shaders and graphical features are expected to work on cost-sensitive gaming handhelds which may not have been possible before, potentially opening a new market.

A significant hurdle for this however is still likely to be developer and artist time, both expensive even at the scale of a small open world game. The sale of open world games on handheld gaming consoles, either as a separate software product or included on a dedicated device, may therefore not be financially viable. This is compounded by the fact that low-cost and general-purpose smartphones capable of better graphics and gameplay already have significant market share. On the other hand, a successful niche (for example as merchandise or gaming trinkets) may still be carved out in a market already heavily saturated with open world gaming titles and significantly more powerful devices.

In the end, even though microcontrollers may not have the performance of larger desktop and mobile CPUs their ability to perform matching tasks on a smaller scale allows their use in future low-cost applications which have not been considered before.

9.3 SOURCE CODE

The source code of the prototype engine and game (including Blender files, documentation and precompiled binaries) have been made publicly available on GitHub: https://github.com/bernhardstrobl/Pico3D

List of Figures

List of Tables

10 BIBLIOGRAPHY

Abrash, M. (1997). *Graphics Programming Black Book*. Coriolis Group.

Abrash, M. (2009, November 13). *Sponsored Feature: Rasterization on Larrabee -- Adaptive Rasterization*. Retrieved from Game Developer: https://www.gamedeveloper.com/programming/sponsored-feature-rasterization-on-larrabee----adaptive-rasterization-helps-boost-efficiency

Acton, M. (2014, September 30). *CppCon 2014: Mike Acton "Data-Oriented Design and C++"*. Retrieved from YouTube: https://www.youtube.com/watch?v=rX0ItVEVjHc

Adams, J., Fraser, L., & Wren, L. (2021, February 8). *#529 – Embedded Hardware with the Raspberry Pi Team*. Retrieved from The Amp Hour Electronics Podcast: https://theamphour.com/529-embedded-hardware-with-the-raspberry-pi-team/

Ars Technica. (2020, February 27). *How Crash Bandicoot Hacked The Original Playstation | War Stories | Ars Technica*. Retrieved from YouTube: https://www.youtube.com/watch?v=izxXGuVL210&t=1468s

Awan, S. (2020, November 28). *The Witcher 3 On Switch Was a Nice "Revenue Driver" For CDPR, Performed "Really Well"*. Retrieved from Twisted Voxel: https://twistedvoxel.com/the-witcher-3-switch-nice-revenue-driver-nintendo-switch/

Baker, S. (n.d.). *Learning to Love your Z-buffer*. Retrieved from Steve Bakers' Home Page: https://www.sjbaker.org/steve/omniv/love_your_z_buffer.html

Basinger, C. (2018, March 16). *Tapwave Zodiac: The Failed 2003 Gaming PDA*. Retrieved from YouTube: https://www.youtube.com/watch?v=Mz3nNKQRnNQ

Basinger, C. (2018, March 5). *The Oregon Trail Electronic Handheld Game!* Retrieved from YouTube: https://www.youtube.com/watch?v=WHQK9qeKavM

Battaglia, A. (2020, February 9). *Star Citizen tech in-depth: seamless scaling from gas giants to detail-rich alien worlds*. Retrieved from Eurogamer.net: https://www.eurogamer.net/digitalfoundry-2020-star-citizen-tech-focus-alpha-3-point-8

Bethesda Softworks LLC. (n.d.). *Skyrim*. Retrieved from The Elder Scrolls: https://elderscrolls.bethesda.net/en/skyrim

Bikker, J. (2003, August 16). *flipcode - 3D Graphics on Mobile Devices*. Retrieved from flipcode - game development harmony: https://www.flipcode.com/archives/3D_Graphics_on_Mobile_Devices-Part_2_Fixed_Point_Math.shtml

Bilas, S. (2003). *The Continuous World of Dungeon Siege*. Retrieved from Game Development Articles, Publications, Papers, Resources - GameDevs.org: https://www.gamedevs.org/uploads/the-continuous-world-of-dungeon-siege.pdf

Black, A. (2019, November 16). *Can a $10 8-bit handheld game console be any good?* Retrieved from YouTube: https://www.youtube.com/watch?v=XrieCn9-9GU

Boris, D. (n.d.). *Dan B's Atari Microvision Tech Page*. Retrieved from Dan B's Videogame Tech Web Site: https://atarihq.com/danb/MicrovisionCarts.shtml

Brain, M. (n.d.). *How Microcontrollers Work*. Retrieved from HowStuffWorks: https://electronics.howstuffworks.com/microcontroller.htm

Branagan, N. (2022, November 20). *Soviet Game and Watch: The Elektronika IM-32*. Retrieved from Nicole Express: https://nicole.express/2022/in-soviet-russia-game-watches-you.html

Brian. (2016, December 29). *Former Nintendo designer on the creation of Game & Watch, Game Boy, DS*. Retrieved from Nintendo Everything: https://nintendoeverything.com/former-nintendo-designer-on-the-creation-of-game-watch-game-boy-ds/

Burgar, C. (2022, June 5). *Every Elder Scrolls Game Ranked By Map Size*. Retrieved from Game Rant: https://gamerant.com/every-elder-scrolls-game-ranked-map-size/

Castro, J. (2005, October 14). *GTA: Liberty City Stories Interview*. Retrieved from IGN Deutschland: https://www.ign.com/articles/2005/10/14/gta-liberty-city-stories-interview

CD PROJEKT S.A. (2022). Retrieved from Cyberpunk 2077 — from the creators of The Witcher 3: Wild Hunt: https://www.cyberpunk.net/de/en/

CD PROJEKT S.A. (n.d.). *The Witcher 3: Wild Hunt - Official Website*. Retrieved from https://www.thewitcher.com/en/witcher3

Colson, D. (2021, November 30). *Building a PS1 style retro 3D renderer*. Retrieved from David Colson's Blog: https://www.david-colson.com/2021/11/30/ps1-style-renderer.html

Copetti, R. (2021, May 18). *Game Boy Advance Architecture | A Practical Analysis*. Retrieved from Rodrigo's Stuff | Rodrigo Copetti: https://www.copetti.org/writings/consoles/game-boy-advance/

Copetti, R. (n.d.). *Architecture of Consoles | A Practical Analysis*. Retrieved from Rodrigo's Stuff: https://www.copetti.org/writings/consoles/

Courrèges, A. (2015, November 2). *GTA V - Graphics Study*. Retrieved from Projects - Adrian Courrèges: https://www.adriancourreges.com/blog/2015/11/02/gta-v-graphics-study/

Cryer, H. (2019, August 27). *Cyberpunk 2077 is Adding a New Dimension to Asset Streaming*. Retrieved from USgamer: https://www.usgamer.net/articles/cyberpunk-2077-is-adding-a-new-dimension-to-asset-streaming

David. (2007, December 4). *David's Video Game Insanity! - Editorials - Inside Virtua Racing*. Retrieved from ClassicPlastic.net: http://www.classicplastic.net/dvgi/editorials-davidvirtuaracing.html

Ender, C. (2017, February 23). *Open World on Mobile with Unity*. Retrieved from Game Developer: https://www.gamedeveloper.com/programming/open-world-on-mobile-with-unity

Epic Games, Inc. (n.d.). *World Partition in Unreal Engine*. Retrieved from Unreal Engine 5.1 Documentation: https://docs.unrealengine.com/5.1/en-US/world-partition-in-unreal-engine/

Fahs, T. (2008, October 22). *Exploring the Freescape*. Retrieved from IGN: https://www.ign.com/articles/2008/10/22/exploring-the-freescape

Foley, J. D., Dam, A. v., Feiner, S. K., & Hughes, J. F. (1996). *Computer Graphics: Principles and Practice.* Addison-Wesley.

Forest of Illusion. (2022, June 9). *Forest of Illusion on Twitter: "Today we have preserved a cancelled Quake port for the GBA! It was originally being developed by @RandalLinden. If that name rings a bell, Linden is a veteran game developer best known for the creation of the BLEEM emulator*. Retrieved from Twitter: https://twitter.com/forestillusion/status/1534884292035112961

Fuchs, H. (1987, July 4-17). *An Introduction to Pixel-planes and other VLSI-intensive Graphics Systems*. Retrieved from The University of North Carolina at Chapel Hill: http://www.cs.unc.edu/~fuchs/publications/IntroPixel-Planes87.pdf

FUNKE DIGITAL GmbH. (n.d.). *Screenshots zu Doom (Oldie): Alles zum Shooter-Spiel Doom (Oldie)*. Retrieved from 4Players.de: https://www.4players.de/4players.php/screenshot_list/PC-CDROM/28557/Screenshots/58932/0/Doom_Oldie.html

Gambetta, G. (n.d.). *Clipping - Computer Graphics from Scratch*. Retrieved from Gabriel Gambetta: https://gabrielgambetta.com/computer-graphics-from-scratch/11-clipping.html

Gamer Network Limited. (2000, November 22). *Grand Theft Auto 2*. Retrieved from Eurogamer.net: https://www.eurogamer.net/r-gta2-gbc

Gavin, A. (2011, February 4). *Making Crash Bandicoot – part 3*. Retrieved from All Things Andy Gavin: https://all-things-andy-gavin.com/2011/02/04/making-crash-bandicoot-part-3/

Gharge, P. (2022, January 25). *Microprocessor vs Microcontroller: The Differences*. Retrieved from All3DP: https://all3dp.com/2/difference-between-microprocessor-and-microcontroller/

Giannakis, D. (2022, January 17). *Tomb Raider on the Nintendo Game Boy Advance is incredible | MVG*. Retrieved from YouTube: https://www.youtube.com/watch?v=_GVSLcqGP7g&t=180s

Giannakis, D. (2022, June 13). *The Story of Quake on the Game Boy Advance | MVG*. Retrieved from YouTube: https://www.youtube.com/watch?v=R43k-p9XdIk

Giesen, F. (2011, July 5). *A trip through the Graphics Pipeline 2011, part 5*. Retrieved from The ryg blog: https://fgiesen.wordpress.com/2011/07/05/a-trip-through-the-graphics-pipeline-2011-part-5/

Gregory, J. (2014, March 11). *XXI SINFO - Jason Gregory - Dogged Determination*. Retrieved from YouTube: https://youtu.be/f8XdvIO8JxE

Gupta, D. (2021, August 7). *The hardware inside your credit card: this is how Smart Cards work*. Retrieved from TechUnwrapped: https://techunwrapped.com/the-hardware-inside-your-credit-card-this-is-how-smart-cards-work/

Hardware-Aktuell. (n.d.). *Lexikon - Nokia N-Gage*. Retrieved from Hardware-Aktuell: https://www.hardware-aktuell.com/lexikon/Nokia_N-Gage

Heaton, A. (2022, October 5). *Steam Deck Has Shipped More Than One Million Units*. Retrieved from Game Rant: https://gamerant.com/steam-deck-1-million-shipped/

Helgeson, M. (2015, April 15). *Interview: Inside The Development Of Xenoblade Chronicles 3DS*. Retrieved from Game Informer: https://www.gameinformer.com/b/features/archive/2015/04/15/interview-inside-the-development-of-xenoblade-chronicles-3ds.aspx

Hernandez, P. (2011, August 12). *Star Fox - Feature*. Retrieved from Nintendo World Report: https://www.nintendoworldreport.com/feature/27387/the-snes-20-star-fox

Hookshot Media. (n.d.). *The Legend of Zelda: Breath of the Wild (Nintendo Switch) Screenshots*. Retrieved from Nintendo Life - Nintendo News & Reviews 24/7: https://www.nintendolife.com/games/nintendo-switch/legend_of_zelda_breath_of_the_wild/screenshots

Hookshot Media. (n.d.). *The Legend of Zelda: The Wind Waker HD (Wii U) Screenshots*. Retrieved from Nintendo Life: https://www.nintendolife.com/games/wiiu/legend_of_zelda_the_wind_waker_hd/screenshots

Huang, B. (2013, December 29). *On Hacking MicroSD Cards*. Retrieved from bunnie's blog: https://www.bunniestudios.com/blog/?p=3554

Hughes, N. (2021, April 14). *What IS an Open World Game?* Retrieved from Nathan Hughes: https://ngjhughes.com/post/what-is-an-open-world-game/

Inhibit. (2021, April 30). *The McSega: Teardown of McDonalds Sega Handheld Game*. Retrieved from PCBurn: https://pcburn.com/mcsega-teardown-of-mcdonalds-sega-handheld-game/

Kenney. (2019, May 23). *Handhelds for developers*. Retrieved from Pixeland: https://pixeland.io/library/handhelds-for-developers

Kenwright, B. (n.d.). *Rasterization : Clipping*. Retrieved from The Maths of 3D: https://xbdev.net/maths_of_3d/rasterization/clipping/index.php

Kramer, L. (2020, December 16). *All the Pipelines - Journey through the GPU*. Retrieved from GPUOpen: https://gpuopen.com/videos/graphics-pipeline/

Küpper, S. (2020, August 19). *Crash Bandicoot (Review)*. Retrieved from Gaming-Village: https://gaming-village.de/wp/2020/08/19/crash-bandicoot-review/

Laine, S., & Karras, T. (2011, August 1). *High-Performance Software Rasterization on GPUs*. Retrieved from NVIDIA Research: https://research.nvidia.com/sites/default/files/pubs/2011-08_High-Performance-Software-Rasterization/laine2011hpg_paper.pdf

Lankshear, I. (2019, July 15). *The Economics of ASICs: At What Point Does a Custom SoC Become Viable?* Retrieved from Electronic Design: https://www.electronicdesign.com/technologies/embedded-revolution/article/21808278/ensilica-the-economics-of-asics-at-what-point-does-a-custom-soc-become-viable

Leeper, J. (2005, January 3). *GameSpy: The Elder Scrolls Travels: Shadowkey - Page 1*. Retrieved from IGN Deutschland: http://wireless.gamespy.com/n-gage/spider-man-2/575960p1.html

Linneman, J. (2019, May 1). *DF Retro: Virtua Racing Switch vs Every Console Port vs Model 1 Arcade!* Retrieved from YouTube: https://www.youtube.com/watch?v=uEbmTfSxt9Q

Linneman, J. (2021, May 30). *DF Retro: Quake - The Game, The Technology, The Ports, The Legacy*. Retrieved from YouTube: https://www.youtube.com/watch?v=0KxRXZhQuY8

List, J. (2020, February 18). *THE TMS1000: THE FIRST COMMERCIALLY AVAILABLE MICROCONTROLLER*. Retrieved from Hackaday: https://hackaday.com/2020/02/18/the-tms1000-the-first-commercially-available-microcontroller/

Llopis, N. (2009, December 4). *Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP)*. Retrieved from Games from Within: https://gamesfromwithin.com/data-oriented-design

Loguidice, B. (2009, April 7). *The History of Elite: Space, the Endless Frontier*. Retrieved from Game Developer: https://www.gamedeveloper.com/design/the-history-of-elite-space-the-endless-frontier

Maher, J. (2019, January 18). *Life Off the Grid, Part 1: Making Ultima Underworld*. Retrieved from The Digital Antiquarian: https://www.filfre.net/2019/01/life-off-the-grid-part-1-making-ultima-underworld/

McCloud, F. (n.d.). *A Super FX FAQ*. Retrieved from Welcome to anthrofox.org!: http://www.anthrofox.org/starfox/superfx.html

McMullen, C. (2019, November 8). *The Best Terminator Game Was Made By Bethesda in 1991*. Retrieved from Fanbyte: https://www.fanbyte.com/games/features/terminator-bethesda/

Meta. (n.d.). *World Games and Asset Streaming*. Retrieved from Oculus Developers: https://developer.oculus.com/documentation/unity/po-assetstreaming/

Microsoft. (2022, October 11). *Windows Advanced Rasterization Platform (WARP) Guide - Win32 apps*. Retrieved from Microsoft: https://learn.microsoft.com/en-us/windows/win32/direct3darticles/directx-warp

Minecraft Wiki. (n.d.). *Chunk*. Retrieved from Minecraft Wiki: https://minecraft.fandom.com/wiki/Chunk

Moss, R. (2017, March 3). *Roam free: A history of open-world gaming*. Retrieved from Ars Technica: https://arstechnica.com/gaming/2017/03/youre-now-free-to-move-about-vice-city-a-history-of-open-world-gaming/

Muijden, J. v. (2019, December 27). *GPU-Based Run-Time Procedural Placement in Horizon: Zero Dawn*. Retrieved from YouTube: https://www.youtube.com/watch?v=ToCozpl1sYY

Ng, J. (2016, May 20). *Making the World of Firewatch*. Retrieved from YouTube: https://www.youtube.com/watch?v=hTqmk1Zs_1I

Nintendo Co., Ltd. (2022). *Game & Watch: The Legend of Zelda*. Retrieved from Nintendo: https://www.nintendo.de/Hardware/Game-Watch-The-Legend-of-Zelda/Game-Watch-The-Legend-of-Zelda-2039375.html

Nintendo of America. (2013, June 11). *Wii U Developer Direct - The Legend of Zelda: The Wind Waker HD @E3 2013*. Retrieved from YouTube: https://www.youtube.com/watch?app=desktop&v=DZRTakeKTbw&t=207s

O'Neil, S. (2002, July 12). *A Real-Time Procedural Universe, Part Three: Matters of Scale*. Retrieved from Game Developer:

https://www.gamedeveloper.com/programming/a-real-time-procedural-universe-part-three-matters-of-scale

Oosten, J. v. (2011, July 6). *Understanding the View Matrix*. Retrieved from 3D Game Engine Programming | Helping you build your dream game engine : https://www.3dgep.com/understanding-the-view-matrix/

Orland, K. (2017, April 4). *Why Zelda: Breath of the Wild is the biggest system seller in history*. Retrieved from Ars Technica: https://arstechnica.com/gaming/2017/04/why-zelda-breath-of-the-wild-is-the-biggest-system-seller-in-history/

Osborne, A. (1978). *An introduction to microcomputers: Some real microprocessors, Volume 2*. Adam Osborne and Associates, Inc.

Palley, S. (2004, November 19). *Call of Duty Review*. Retrieved from GameSpot: https://www.gamespot.com/reviews/call-of-duty-review/1900-6113659/

Panic Inc. (2022). Retrieved from Playdate: https://play.date

Patterson, B. (2008, July 07). *Under the Hood: The iPhone's Gaming Mettle*. Retrieved from TouchArcade: https://toucharcade.com/2008/07/07/under-the-hood-the-iphones-gaming-mettle/

Payne, C. (2018, January 16). *GamesIndustry.biz*. Retrieved from Ultima Underworld and the freedom to make bad choices: https://www.gamesindustry.biz/ultima-underworld-and-the-freedom-to-make-bad-choices

Persson, E. (2012, August). *Creating vast game worlds*. Retrieved from ACM SIGGRAPH 2012 Talks: https://dl.acm.org/doi/10.1145/2343045.2343089

Persson, E. (2014). *Low-level Shader Optimization for Next-Gen and DX11*. Retrieved from GDC Vault: https://www.gdcvault.com/play/1020352/Low-Level-Shader-Optimization-for

Phillips, C. (n.d.). *Liberty City Stories PSP Screenshots*. Retrieved from The GTA Place: https://m.thegtaplace.com/gtalcs/psp-screenshots/?page=1

Pimoroni Ltd. (n.d.). *PicoSystem*. Retrieved from Pimoroni: https://shop.pimoroni.com/products/picosystem?variant=32369546985555

Pineda, J. (1988). A Parallel Algorithm for Polygon Rasterization. *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, 17-20.

Pounder, L. (2021, October 23). *Pimoroni PicoSystem Review: Tiny Console for Big Ideas*. Retrieved from Tom's Hardware: https://www.tomshardware.com/reviews/pimoroni-picosystem-review-tiny-console-for-big-ideas

Raspberry Pi Ltd. (2022, June 17). *RP2040 Datasheet*. Retrieved from Raspberry Pi: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

Raspberry Pi Ltd. (n.d.). *Buy a RP2040*. Retrieved from Raspberry Pi: https://www.raspberrypi.com/products/rp2040/

Reed, N. (2015, July 15). *Depth Precision Visualized*. Retrieved from NVIDIA Developer: https://developer.nvidia.com/content/depth-precision-visualized

Rubio, L. (n.d.). *Tamagotchi Tech Specs*. Retrieved from Luc Rubio · Software Engineer: http://tama.loociano.com

Ruskin, E. (2015). *Streaming in Sunset Overdrive's Open World*. Retrieved from GDC Vault: https://www.gdcvault.com/play/1022268/Streaming-in-Sunset-Overdrive-s

Sanderson, G. (2022, November 17). *Functions from RAM run slower than from XIP cache*. Retrieved from Raspberry Pi Forums: https://forums.raspberrypi.com/viewtopic.php?t=343023

Sanderson, G. (n.d.). *Making It Run Fast And Fit in RAM*. Retrieved from RP2040 Doom | rp2040-doom: https://kilograham.github.io/rp2040-doom/speed_and_ram.html

Sanglard, F. (2011, September 16). *Quake 2 Source Code Review*. Retrieved from Fabien Sanglard's Website: https://fabiensanglard.net/quake2/quake2_software_renderer.php

Sanglard, F. (2019, April 1). *The story of the Rendition Vérité 1000*. Retrieved from Fabien Sanglard's Website: https://fabiensanglard.net/vquake/

Scratchapixel. (n.d.). *Rasterization: a Practical Implementation (An Overview of the Rasterization Algorithm)*. Retrieved from Scratchapixel: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

SEGA Holdings Co., Ltd. (n.d.). *World*. Retrieved from Yakuza 0: https://yakuza.sega.com/yakuza0/world.html

Sharp Corporation. (1990, September). *Microcomputers Data Book*. Retrieved from The bitsavers main page: http://bitsavers.informatik.uni-stuttgart.de/components/sharp/_dataBooks/1990_Sharp_Microcomputers_Data_Book.pdf

Sherrill, C. (2020, November 13). *All Nintendo Game and Watch Consoles, From Ball to Super Mario Bros Anniversary*. Retrieved from Esquire: https://www.esquire.com/lifestyle/a34659956/every-nintendo-game-and-watch-edition-photos/

Shimpi, A. L. (2014, August 18). *ARM's Cortex M: Even Smaller and Lower Power CPU Cores*. Retrieved from AnandTech: https://www.anandtech.com/show/8400/arms-cortex-m-even-smaller-and-lower-power-cpu-cores

sonnyboy. (2017, October 31). *THE BRICK GAME [THE HIT GAMING CONSOLE OF LATE 80s to EARLY 90s] - REVIEW sonnyboy*. Retrieved from Steemit: https://steemit.com/gaming/@sonnyboy/the-brick-game-the-hit-gaming-console-of-late-80s-to-early-90s-review-sonnyboy

Sony Interactive Entertainment Europe Limited. (n.d.). *Horizon Zero Dawn - PS4 Games*. Retrieved from PlayStation® (UK): https://www.playstation.com/en-gb/games/horizon-zero-dawn/

Stella. (n.d.). *Tomb Raider for Mobile Devices | Stella's Site*. Retrieved from Stella's Walkthroughs - Strategy Guides for the Tomb Raider Series: https://tombraiders.net/stella/trmobile.html

Stop Skeletons From Fighting. (2016, September 7). *PW Hall of Fame: V.D.-Dev (aka Velez & Dubail) | Punching Weight [SSFF]*. Retrieved from YouTube: https://www.youtube.com/watch?v=rfOR3XqrJc4

Styger, E. (2012, November 1). *Defining Variables at Absolute Addresses with gcc.* Retrieved from MCU on Eclipse: https://mcuoneclipse.com/2012/11/01/defining-variables-at-absolute-addresses-with-gcc/

Target, S. (2019, November 6). *How Much of a Genius-Level Move Was Using Binary Space Partitioning in Doom?* . Retrieved from Two-Bit History: https://twobithistory.org/2019/11/06/doom-bsp.html

The Mesa 3D Graphics Library. (n.d.). *LLVMpipe.* Retrieved from The Mesa 3D Graphics Library latest documentation: https://docs.mesa3d.org/drivers/llvmpipe.html

Unity Technologies. (2019). *Entity Component System*. Retrieved from Package Manager UI website: https://docs.unity3d.com/Packages/com.unity.entities@0.0/manual/index.html

VD-dev. (n.d.). *VD-dev Games.* Retrieved from VD-dev Games: http://www.vd-dev.com/Games.html

Video Game Kraken. (n.d.). *Microvision by Milton Bradley*. Retrieved from The Video Game Kraken: http://videogamekraken.com/microvision

Videospielhalbwissen. (n.d.). *Driller (1987).* Retrieved from Videospielhalbwissen: https://videospielhalbwissen.de/timeline/driller-1987/

Williamson, J. (2021, October 5). *PicoSystem API Cheatsheet.* Retrieved from Pimoroni: https://learn.pimoroni.com/article/picosystem-api-cheatsheet

Wohllaib, E. (2022, July 14). *Procedural Grass in 'Ghost of Tsushima'.* Retrieved from YouTube: https://www.youtube.com/watch?v=Ibe1JBF5i5Y

Wooden, R. (2015, April 13). *Overcoming Floating Point Precision Errors!* Retrieved from Game Developer: https://www.gamedeveloper.com/design/overcoming-floating-point-precision-errors-

Zao, R. D. (2012, February 4). *Blog 486: The Elder Blogs: Daggerfall.* Retrieved from Rao Dao Zao – The shape and the power of the blog.: https://raodaozao.net/2012/02/04/blog-486/

Zhang, W. (n.d.). *Coremark.* Retrieved from Wenting's Web Page: https://zephray.me/coremark/

11 ERKLÄRUNGEN

Die Unterzeichnung der Selbstständigkeitserklärung ist obligatorisch. Die Unterzeichnung der Ermächtigung ist optional.

11.1 SELBSTSTÄNDIGKEITSERKLÄRUNG

Hiermit erkläre ich, dass ich die Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe.

Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Immenstadt, 7. Januar 2023

Ort, Datum Unterschrift

11.2 ERMÄCHTIGUNG

Die Urheberin/Der Urheber der studentischen Arbeit kann (muss nicht) erklären, dass die Hochschule Kempten folgende Nutzungsrechte erhält.

☒ Hiermit ermächtige ich/wir die Hochschule Kempten zur Veröffentlichung einer Kurzzusammenfassung sowie Bilder/Screenshots und ggf. angefertigte Videos meiner studentischen Arbeit z. B. auf gedruckten Medien oder auf einer Internetseite der Hochschule Kempten zwecks Bewerbung des Bachelorstudiengangs „Game Engineering“ und des Masterstudiengangs „Game Engineering und Visual Computing“.

Dies betrifft insbesondere den Webauftritt der Hochschule Kempten inklusive der Webseite des Zentrums für Computerspiele und Simulation. Die Hochschule Kempten erhält das einfache, unentgeltliche Nutzungsrecht im Sinne der §§ 31 Abs. 2, 32 Abs. 3 Satz 3 Urheberrechtsgesetz (UrhG).

Immenstadt, 7. Januar 2023

Ort, Datum Unterschrift