A close-up of a logo

Description automatically generated

CMP6200/DIG6200

Individual Undergraduate Project 2024–2025

**A3: Dissertation**

[Document title]



Course: Computer Games Technology

Student Name: Khai Ailyan

Student Number: 22130235

Supervisor Name: Jan Krasniewicz

Abstract

# As more games are produced on newer games systems, older games receive less attention from consumers and their makers and so become obsolete. And in going obsolete these games are often forgotten; therefore, this project aims to create a pipeline, a methodological graphic tool for reverse engineering video games files as a means of games preservation. Said tool will be referred to as GAD (Game Abstract Diagram, an abstract diagram denoting game file reverse engineering).

Therefore, creating the means for future programmers to decompile, reverse engineer and preserve existing obsolete game file formats. Which would allow for video games to be better preserved. This report details the step-by-step process in reverse engineering file formats specifically for this report game asset files, using static analysis techniques as well as techniques that are later detailed in the methodological graphic tool, alongside an example of two artefacts using this methodology to reverse engineer and reconstruct obsolete file formats, i.e. .Str and .Rws.

The results show the benefit in following this diagram, as the contents of .Str/.dff/.rws files were reverse engineered and converted to contemporary file formats, such as .objs for the .rws/.dff and for the .Str files its contents exported for further review for programmers and reverse engineers.

Acknowledgements

This Project would not have been nearly as successfully without the help of Jan Kracniewicz my tutor, with 10+ years of digital forensics experience he aided in the understanding of the .str files as well as the acquisition of the compression algorithm used in .str files, furthermore his pushing for better quality writing with the literature review has improved the report substantially, the improvements in question were introductory pieces for the literature reviews theory and review section to ease readers in.

Furthermore, with the aid of greavsey1889’s Visceral Toolkit “Cauldron” aided in the debugging process of the artefacts made, specifically the .Str tool, further external tools include the dragonFF .dff converter, for its comprehensive syntax and understanding of renderware geometry, this was created by Turk645.

# Introduction

This section details the outline of the scope of the project, the problem the project aims to solve as well as the rationale for why this problem was chosen Furthermore, the report details the creation of the necessary artefacts i.e the .Str extractor/opener, the .rws/.dff model exporter and the pipeline that despite being a part of the methodology is also an artefact, the processes involved in making them as well as tutor feedback and assistance in fixing and or improving them.

## Problem Definition

The problem in question is game preservation. Games are hard to preserve due to the lack of support for old proprietary formats. For games to be preserved, their contents, i.e., files, need to be compatible with new hardware. Game console companies try to offset this problem with “backwards compatibility,” where their new consoles have the necessary hardware requirements to be compatible with old games.

However, this is only done for games one generation earlier. Games from Xbox 360 can be played on Xbox One, but cannot be played on Xbox Series X. As a result, statistically, games will have to be forgotten somewhere along the chronological chain.

To solve this, creating a learning tool to teach people how to reverse these files is crucial. If programmers understand how the file works and its syntax, they can convert its contents to something more applicable or contemporary, in this report it is .Obj files as most 3d software and game engines can open/use .Objs.

## Scope

The scope should encompass the aim of the project; specifically, an artefact needs to be created to verify the legitimacy of the **GAD**. Therefore, only.Str, .Rws, and .Dff were chosen to be decoded, reverse-engineered, and converted. Moreover game asset files were chosen not code, due to code decomplication generally be harder and taking longer, which is not as viable in the time allocated.  
  
In contrast to the initial scope and planning, the inclusion of .Arc files, but were dropped due to time constraints, however, regardless of the absence of .Arc files, the **GAD** has shown to still be a viable pipeline to reverse engineering files, as .Str, .Rws and .Dff were successfully converted and reverse engineered.

## Rationale

Whilst there is considerable effort to preserve video games via emulation and or porting games over to contemporary hardware, there are still ample games that have not been preserved via these methods and so there is still a job to do. Moreover, Creating the **GAD** will ensure that programmers who have a vested interest in an obsolete game, can just convert the game files themselves.

Furthermore, this benefits companies as companies will not need to hire back old staff if they need to remaster their games or to port them to other hardware, an example would be old COBOL programmers who worked for big banks, as current programmers can use the **GAD** to do it themselves.

With the use of the artefact, programmers would find it easier to reverse engineer old proprietary file formats to then preserve, remaster or modify games so that way they do not get forgotten or die out.

Furthermore, this is necessary due to more and more game systems becoming more and more obsolete as we go from ps3 to ps4 to ps5, the ps3 has become obsolete and support has been dropped/discontinued

## Project Aim and Objectives

The aim of the project is to demonstrate and create a learning diagram that aids in game file reverse engineering, furthermore, creating artefacts using the methodology outlined in the diagram is done to demonstrate its effectiveness.

Objectives include:

- Construct rough theories/steps on how to reverse engineer game files, I.e through reading offsets, reasonable numbers, observation of repeatable patterns in data, understanding the header file where it starts and ends.

- Secondary Research on .Str files, what are they, who made them, what are they for etc

- Primary Research, going into the .Str via a hex editor, researching the syntax itself by documenting key offsets and behaviours etc

- Find the algorithm required for decompression, then decompress the .Strs contents

- Create the necessary artefacts i.e the **GAD**

## Background Information

The File syntax of a given file has commonalities between files even if said files are completely different to the original file, such as examples would include most files having a header file and that header file being at the top of the file. Said Header file has information relating to the file irrespective of what file it is, such information could be, file offsets, number of files inside the file, unique identifiers and ids, the size of the file, dictionaries, flags such as the case with textures and so much more.

Common file/code reverse engineering methodologies includes the use of static and dynamic analysis, these practices involve reading the data itself raw to discern pattern, where they vary is WHEN these are performed, dynamic analysis is observing the file and its contents when its being run by an exterior program whilst static analysis is just reading the file as it is in a hex editor or equivalent, this form of analysis will be the bulk of the project and HexEdit online will be the tool that is used most.

# Literature Review

## Literature Search Methodology

|  |  |  |
| --- | --- | --- |
| Search Terms: | Rationale: | Database: |
| File Syntax  File Structure  File Format | Each file that is to be decoded to then reverse  engineer has as specific structure and syntax, so  understanding how they work is crucial | ResearchGate |
| Reverse Engineering  Software Reverse Engineering | Understanding the techniques that underpin  reversing software and files. | ResearchGate |
| Games Preservation  Files Preservation | Understanding and techniques used in preserving  games and files | ResearchGate |
| .Rws Models  .Rws Object Formats  .Rws Object syntax  Renderware Object Files | Specifically relating to the.Str file artefact. EA  were using these formats to create their models  and objects | ResearchGate |
| .str file format  .str file syntax  Ps3 stream file syntax  EA stream file syntax  EA object file syntax | Specifically relating to .Str file artefact, EA were  using these formats as storage for all of their  games file. | ResearchGate |

## Themes

## File Syntax - denotes the rules and structures that data is stored as in a file and is a fundamental to reverse engineering files, an example of which would be .Obj where all of the vertices are stored with “v” and the vertex normals “vn”, the relevance to the project being file syntax is needed because the .Str and .Arc files structures are unknown, and needs to be known to then create a file converter. Context drives what type of abstract data structures and structures the file uses in its syntax, for instance 32 bit architectures such as operating windows operating systems that are 32 bit, they will have smaller integer sizes compared to 64 bit operating systems.

## An example of relevant syntax for the files to be decoded would be Obj’s as they contain vertex, normal and uvs necessary for model files to be rendered in games, vertices for geometry and shape, the normal for lighting and uvs for texture mapping. Their structure is very linear meaning there are rows of “v “ (vertices), “vt “(uvs), “vn “(normals) and finally “f “ for face indexes for triangle construction. All the data in these files are stored as floats due to the precision it grants, furthermore all values are in model space/local space due to not being transformed into any world or scene yet because its just a file.

## Reverse Engineering - Reproducing a proprietary product (i.e files, code and data formats) via the understanding of syntax, this involves understanding the File syntax of the specific file to then reverse engineer OR to decompile the software code to convert it to another language or for a newer system. The ethics and legality is purely contextual, due to many aspects needing to be present for the company of the proprietary software to care such as, the age of the software, commercial impact, whether its used commercially, whether it infringes on copyright or not ( this is because if a file reader is made, are said files that are read then used for future use, that would cause the company to actually care).

## Games preservation - Preserving and maintaining games either through physical maintenance and or protection OR through the preservation of the game’s software, code and or files. In this context it is converting their respective contents to more contemporary file formats to make it so future operating systems and systems in general can use / play those games.

## ADT (Abstract Data Type) Syntax - Bytes in code whether a class or struct with specific structures and rules created for storing and manipulating data (via using operations), e.g List, contiguous block of memory containing a list of items of a specific type, if something were to be stored in the list, and the list is full, the size is automatically resized up by allocating more memory to the list.

More examples would be the stack, a stack is a last in first out data container, that allows the programmer to push data to the stack using the “Push” method/operation to store data, the last thing to be put into the stack is the first one to be pushed out when the “pop” method/operation is called, this is done for stuff like ai prioritization, where if a new player attacks the enemy, they are then prioritized because they were the last added to the stack.

## Compression Algorithms - Algorithms created to make data and or files smaller than they are usually, this is for transportation between devices or storage capacity optimization, compressed items can then be uncompressed to their original size for further use, such examples would include run length encoding compression, where it looks for repetitive data sets, combines them into one value, therefore condensing the data. However games that are to be preserved are old, and so run on oldhardware such as ps3 games running on the ps3 hdd, from this compression algorithm were designed to be very quick, due to how slow HDD contrast with SDD, such algorithms were likely to be LZ77

## Review of Literature

**File Syntax:** File syntax originated in (Russel Kirsch 1957) via a picture of his son, and from that the first file syntax was created alongside it having its own rule set. File syntax varies between different types of files due to each type of file having a different purpose (Kauthar Abdulazeez, sohit agarwal 2021; Samiya Khan; Mansaf Allam 2019), coupled with different structures inside different files (Mike Folk 2003) each type of file is unique in its rules, structure and therefore purpose. The limitations are that there are little research documents/journals that go into proprietary formats and more look into general syntax and older simpler formats like text or image files.

**Reverse Engineering:** Reverse Engineering's existence dates to the first thing ever made. However, in the context of software, software reverse engineering began in 1975 with the first text editor WYLBUR (Kathi Hogshead Davis, Peter H Aiken, 2000), features of reverse engineering include disassembler, debugger, etc (Abigail A 2021), whilst in contrast (Alessandro Mantovani , Simone Aonzo, Yanick Fratanatonio), refer to the techniques of reverse engineering through their behaviour and performance in time. (Anand Gadwal 2011) talks in detail on the step-by-step process of reversing software from “context parsing” through to “Design Reconstructing Phase”, (Ramandeep singh 2013) refers to the types of reverse engineering such as “data” and “code” reverse engineering, detailing the different types depending on what is being reversed. There needs to be a greater focus on breaking down the individual processes, with example of high-level reverse engineering in software, most of the research refers to very low level or do break down the steps enough, as well as on Data Reverse Engineering specifically.

**Games Preservation:** Games preservation has existed as long as games have needed to be stored, as time progressed the games software separate to its hardware needed to be maintained (Harkai Istvan 2022)(Haydock Christopher 2018) as well as the hardware (Dany Guay-Belanger, may 2021)(Widget, Megan A, 2009) , moreover the importance of games preservation cannot be understated due to its cultural impact (Todd Benjamin C 2019)(Henry Lowood 2009) as games and games systems are being threatened with obsolescence such as the discontinuation of ps3. Moreover, the importance is shown with people's desire to preserve games, but can’t be due to legal restraints (Bachel, Alasdair and Barr 2014). Most games preservation today is hardware based and comes in the form of “Archiving” (Johansson 2023). There lacks research on a software approach to Games preservation, most of the research refers to hardware and the minimal research that refers to software rarely accounts for the legal issues, especially when it comes to international law, not just America.

**Abstract Data Types:** Abstract Data types or ADT’s have existed since the 70’s created by (Barbara Liskov, 1974), a standard example of an ADT being a “Stack” (John V Guttag 1977) which contains items of a certain data type and allows for the storage of multiple items in a first in last out order of execution. In Contrast to ADO’s (Abstract Data objects) which are rawer and have little functionality (Jean Francois, Koschke 2000), Vectors in games would be an ADO, another ADT a Queue is commonly used for AI behaviour in games (John v Guttag June 1977). Limitations are that the general research goes back to the 60’s and 70’s and the majority of the ADT’s that are common today date that era, although the process in making the ADT’s as well as the logic they use is shown, so they could be replicated or expanded upon that way.

**Compression:** Data Compression’s origins come from (Robert Fano, Claude Shannon 1949), then (Huffman Coding 1952) with “Huffman Coding” and (Lempel-ziz-welch) with LZW, Different compression algorithms are used for specific files and data structures (Tito Waluyo Puroboyo 20,17). This in turn leads to the most common use of compression algorithms, compressing text (Amandeep Singh Sidhy 2014). Custom Compression algorithms are created for specific data structures and object types like model files (Mustafa, Ammar 2017; JingLiang Ping 2005; siddeeq mohammed m, rodrigues marcos 2016), said algorithms are specialized to be better optimized for 3d objects. There lacks research into more proprietary compression algorithms, companies often make their own, as a result these compression algorithms that have been documented and research are often non-applicable for reverse engineering and thus for games preservation.

### **Theory**

**Games Preservation:** Games Preservation includes the physical storage of games and their associated hardware (Haydock, Christopher april 2018) (Dany Guay-belanger may 2021), it includes the maintenance of games/games systems themselves through maintaining hardware or Copying contents of games to newer external storage due to bit rot (Haydock, Christopher april 2018), bit rot being when data is stored in a hard drive for too long, the data deteriorates. Also includes the maintenance of software via reverse engineering the games/games systems software to accommodate newer hardware. (Johansson, spring semester 2023), so that it can be played on contemporary systems. This is the approach used in the project, software maintenance is the method of which the game/game files will be preserved, not hardware maintenances.

**Reverse Engineering:** Reverse Engineering in a general sense is when a product/system has been understood mostly in its entirety and recreated. However, in this context, Reverse Engineering consists of understanding the syntax of code, a file, hardware architecture/structure (Aremo, Abigail, 2021) etc and trying to recreate it, it's about understanding and recreation. In this context its about understanding the header file of a file, reading the values inside of the header to ascertain data about the file, locating offsets in the file to determine where to go to in the file and where to stop, reading the header file to determine key features of the file such as, the amount of files, of what memory policy/allocation, what structures exist within the file so forth. From this, creating code ( an artefact) to then read the file and write it or its contents into more understandable/contemporary sub-files, if applicable.

**Abstract Data Types:** Abstract Data types allow for data to be stored with specific functionality depending on what ADT it is, a list allows for continuous addition of items of a type, and once fully allocates more memory to the list for further use, arrays are finite and need redeclaring if the array is to full, sets are immutable and so the values inside cannot be modified directly. For this project ADT’s are necessary due to a file effectively being a contiguous block of memory that functions like an array/list, therefore reading line by line and assigning to an array or list variable ADT is crucial in creating an artefact to read/write files.

**Compression:** Compression is the practice of taking in data and making it smaller for the sake of storage and transport. Compression is either lossy, data is lost in compression, first lossy algorithm (ahmed 1974), or is lossless and so data is not lost on compression, the first and commonly used LZW (Lempel-ziv-welch). Depending on the type of data certain compression algorithms are more ideal e.g audio, model files, textures etc (Purboyo, prasasti 2017; Sidhu 2014), this is especially true with model files that have specific compression algorithms ideal for triangles (Mustafa, Ammar, 2017; Rodrigues 2016; Ping 2005).

**File syntax:** File syntax is the structures and rules of the file itself normally shown using a “header file”, which is at the top of the file, file syntax is important as by understanding its structures and rules a programmer can read its data correctly without mishap. Understanding file syntax is crucial for extracting models, audio files, textures etc from games to reverse engineer them as the programmer is then able to determine where they are in the file, there size etc.

## Summary

## Games Preservation: Games preservation in the form of hardware preservation is seemingly a problem that has already been fixed, in terms of there is a legal viable solution, however from a software maintenance approach there seems to still be a lack of information and problems with legality, making the project more important due to a lack of emphasis on both software maintenance as well as a more legal avenue.

## Reverse Engineering: The research shows the methodologies and types of reverse engineering, as a result there is a detailed blueprint as to what type of reverse engineering is to be used and the steps to which software can be reversed. Furthermore, for the project, it is data reverse engineering, using text editing tools, reading offsets in memory to memory and identify key structures and repetition in the files.

## Abstract Data Types: Abstract data types are shown to be useful containers for data with additional functionality, artefacts will have to be read as arrays to process line by line characters for binary analysis and data reverse engineering File syntax: File syntax research has shown the importance of header files due to their relation to the rest of the file, moreover file syntax is different for each file type, given this information the project going forward requires better understanding of custom files due to their custom syntax as shown by their respective header files, this will be crucial in decoding the files.

## Compression: The research can be summarized as each data type and or file having its own ideal compression algorithm designed to deal with that specific file, this is important because companies tend to design their own file systems and as a result have their own compression algorithms to better optimize space efficiency. Said algorithm needs to be understood before reverse engineering

# Method and Implementation

The Method And implementation covers both the development life cycle of the artefacts as well as the use of the **GAD** in the report, as the artefacts were created using the methodology the **GAD** shows, therefore validating the diagrams use.

## Development Life Cycle

### General development

This section covers the artefacts that were made, their general development, tutor feedback, basically an overview of the planning, creation of the artefacts in a macro sense which is to say the general overview of the **GAD**, in contrast the micro sense is going into detail of each node in the **GAD,** explaining what each node does, what stage of development etc.

Four Artefacts were created, the .str extractor, the .rws converter, the .dff converter and finally the actual aim and purpose of the project the **GAD**.

The artefacts development was very linear and had very little delays in its creation, the complexity was quite big, due to no available synax of the .str file bar the compression algorithm, however apon reflection with the project tutor at key deadlines, all of them were met in time.

Moreover, the development of the .rws and .dff file converters were less time constrained, due to the syntax of the .rws already being available in a github repository i.e “the turk”. As a result the complexity was smaller and was therefore quicker to accomplish.

Review with supervisor who is well versed in file syntax and decomplication, pushed for the contents to be uncompressed BEFORE the files were extracted.

from that further development led to the sections being uncompressed via the implementation of the “RefPack decompression” algorithm, this enabled files to be successfully extracted as the contents of the files were now actualized, furthermore the output of the sections was compared with similar software that does the similar file viewing, and in contrast the files were identical, meaning this was successful.

Further development came in then reverse engineering the .rws and .dff files contained in the .str to a more suitable format .obj. .Rws and .Dff files are 3d geometry files created for the game engine renderware. They used triangle strips instead of triangle faces, they contain vertices, uvs, texture references, where they differ is that .Dff have a skin mesh plg component attached to it, which enables animations and bone weights to be applied to the file to animate it in game.

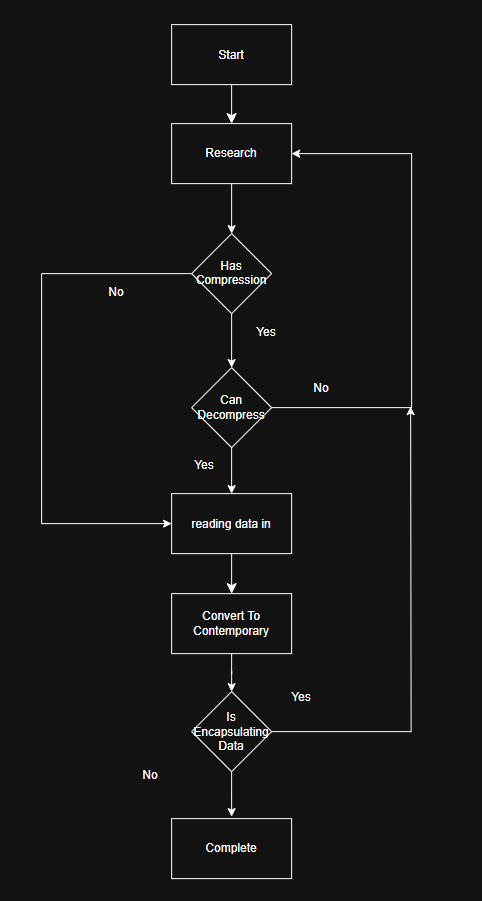
This in part was since there is no 3d model software that will open and or view the .rws as a model. Instead RW analyzse was used to view the file as a hierarchy, which in turn enabled the syntax of the .rws file to be laid bare, this enabled me to create my own file converter due to the syntax being expressed.

The development of the .rws converter was quick however during development there were problems with the normals that were calculated from the vertices, with guidance from my secondary supervisor, the normals were calculated as vertex normals not face normals.

This lead to the successful creation of .objs.

Their relevance to the project is that they are needed in order to preserve the Simpsons game, due to them being very important aspects of the Simpsons game, that being the characters, maps and environments. Without them the game does not exist. So preserving them was crucial to preserving the game.

### **GAD** – The proposed learning tool



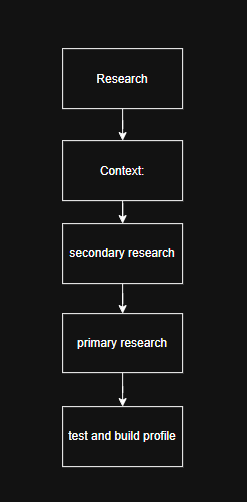
**Figure 1. Shows the Learning tool at its macro-level**

The diagram above is the general process of creating the reports artefacts as well as being the focus of this project I.e. the **GAD**, this diagram enables the aim of the project to be fulfilled as it allows for the successful reverse engineering and conversion of video game assets as a means of games preservation. This diagram was created through the process of understanding, reverse engineering and then converting the .Str, .Rws and .Dff files, in effect this diagram represents the process of how these files were converted to contemporary formats.

The first steps are research into the file that is to be converted, then it must be determined whether the contents are compressed, then is it able to be uncompressed for further data analysis, do that, the data has to be read into acceptable data structures and ADT’s such as lists prepping it for conversion.

Then it needs to be converted to something contemporary such as .Obj for geometry files, if the files encapsulate more data, then they need further decompiling and reverse engineering to convert those sub-files, so repeat the process, then after that its finished.

The diagram was created by documenting the process in which the .Str/.rws/.dff files were converted, then deconstructing the process into sub-steps, the **GAD** is the culmination of this report’s artefacts.



**Figure 2 shows the elaboration of the “Research” node in figure 1**

For the research part of the project as well as the artefacts pertaining to .str/.dff/.rws files. Both secondary and primary research are needed to build a profile of the syntax for the .str/.dff/.rws files.

What this means is conducting secondary research in the form of forum and wiki searching for the syntax of the given file formats to then build a profile.

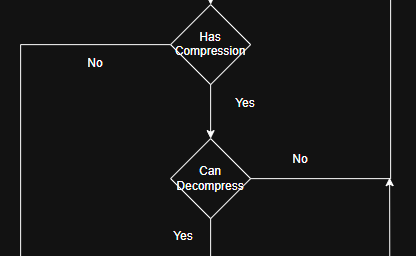
This is done first because there might already be a format available to copy from, why reinvent the wheel, this is outcome orientated, which is to say games preservation is the goal and so if the syntax already exists it doesn’t matter if the programmer takes what already exists. Moreover this is the rationale behind doing secondary research first, if it already exists use it.

Primary research will be the bulk of the research depending on how little secondary research there is, primary research involves opening the files into a hex editor, and reading the data expressed as bytes, this is done in order to read patterns in the data to discern their purpose as well as identifying key offsets and flags that will lead to building a profile of its syntax.

The first step in the Primary Research stage is determine where the header file is, how big it is, whether its size changes depending on how big the file is, an identifier to tell the programmer and programs reading the file what type of file it is, dictionary tables that lead to other places in the file, offsets and flags.

For the .str file the identifier is “SToc”, then 4 proceeding bytes of what seems to be padding, then an integer 8 that denotes how many sections there are in the file, followed by the integer 8 hardware identifier flag which tells the program what hardware it’s supposed to run on, a big endian 32 integer denoting the start of the section table, then after that it’s just each section of which each is 24 bytes denoting the compressed and uncompressed sizes of the sections.

Once a profile is built, it is to be tested, this is done by using the profile that is constructed and applying it to other files that are also .str. If the same profile works by virtue of having the same offsets, dictionary tables, 24 byte size of the sections. Then it is a success. This logic also applies to any other type of file, comparing the syntax profile of a file to other files of the same type allows for cross examination between the files, and serves as validation.



**Figure 3 denotes the compression stage in the reverse engineering pipeline**

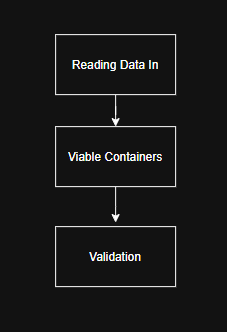
This section of the diagram refers to the compression used by companies to reduce the size of their game files to reduce the size of their games on hardware, the compression of a file can be ascertained in a variety of ways, the quickest yet unreliable way it so read the magic numbers I.e key byte identifiers of sections of the file, to determine the compression algorithm, this is unreliable because if there isn’t already a resource for this, as a result the programmer cannot do anything with this magic number. However, if there is, the programmer has a link back to the original compression algorithm and can replicate it.

The second way is to read the entropy of the file to determine what kind of compression it uses, what that means is to read the file in a binary visualizer and looking at how the binary is displayed, if it is highly concentrated it is compressed, the types of entropy are normally keyed depending on the binary visualization software.

Moreover experimenting with already existing compression algorithms and types of compression algorithms on the file, then measuring the entropy after the fact, is also a viable way to determine the type of compression, moreover, there are only so many types of compression algorithms, Huffman, lzw, lx77, deflate, lzma, so experimenting isn’t too hard, although tedious.

Within the context of the project the .Str files were compressed and using a compression algorithm called the “RefPack Algorithm” denoted by the magic number 0x10FB, this magic number tells the program that not only it is compressed but that it is using the refpack algorithm.

The “Refpack Algorithm” is a type of compression algorithm that uses key size identifiers to dictate how much it compresses per byte section, it is very similar to LZ77 or LZSS, in which it scans the data and replaces it with a reference depending on if certain flags are met I.e 0x80, 0x40, 0x20 then default, all these numbers mean is that they are identifiers used for if statements to tell the program how much to copy.



**Figure 4 shows the Data reading section, relates to how data is stored**

This section refers to how the data is read by virtue of using of the constructed profile that is built from the research stage, the syntax now being understood, algorithms can be developed to extract the data and parse the results into suitable ADT’s for conversion.

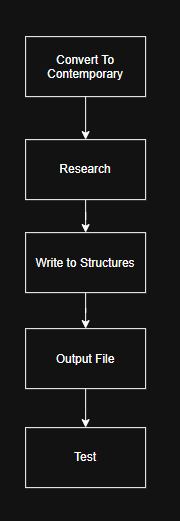
An example would be with .rws and .dff files. They contain geometry and so when converting them to contemporary geometry/model files such as .Obj files, there contents need to be read as Vector3’s and Vector2’s, which are viable containers for their xyz values I.e vertices and normals, then their contents need to be stored in row like ADT structure such as a list or array.

The data can be validated by comparing against multiple different instances of .rws and .dff files, if the algorithms designed to parse the data, do not crash apon multiple instances of the same file, that would show that the boundaries for reading the vertices, uvs and triangle strip tables are valid. Moreover, if the read values for the vertices, are not astronomically big or small and are instead “reasonable values”, that also acts as a form of validation. Furthermore, if the offsets for how big the vertices size are equal to how many vertices there actually are, that also acts as a form of validation.

Therefore, in the process of reformatting the data, the data needs to be read in appropriate containers and stored in appropriate containers, whether that be Vectors, lists, arrays or other array/row like structures.

The .rws and .dff were contained in the .str files, so they needed no further decompression. The .rws and .dff used the renderware file structure, this is due to them being renderware objects. The renderware header contains the id for what type of data it is, the size of that type of file and then the renderware id.

An example might be 1:200000:4912312526. Each .rws and .dff file will contain multiple of these renderware headers denoting specific types in the file like geometry, EA specific plugins and clumps. Where they differ is that .dff are used for props that are reused and have animation weights to them, whilst .rws are for static objects that are used once with no animations for map data.

****

**Figure 5 shows the final stage of the reverse engineering pipeline**

The most important stage of the diagram is the conversion stage, once the profile has been created and code has been read to transfer the contents into viable containers, the last step is to determine what file format the formatted data will be outputted to.

Specifically for .Str each of the 24 byte sections were read linearly, each of the uncompressed sections were added to a going offset of 2048, this number was chosen because it was the size of the header file an so the beginning of the actual file, from that a start and end point for each section of the .str could be constructed.

From that each of these sections could have each file exported from it by virtue of reading the header files of each section, moving to that sizes offset, and generating a start and end point for each file in the section, this could be read into a list ADT and then exported.

Each of the files were checked to see if they were .rws or .dff and from that there own profile was used to create an algorithm to parse the data into a list ADT to then be exported.

This involves research into other similar file types to determine the best fit, for example .rws and .dff files are geometry files that use triangle strips, so we can convert them to triangle faces then parse that data into .objs, objs are used because they are very simple to use and contain vertices, uvs and normals which are all of the necessary values inside .rws/.dff files.

From the research the Formatted data needs to follow its syntax, but because these are being converted to proprietary formats, this part is far easier, due to the ease and access of the syntax of newer file formats such as .obj, in the case of .obj the vertices are stored as “v X Y Z”, the uvs “vt X Y”, the normals “vn X Y Z” and the face indexes are “f vert1/uv1/normal1 vert2/uv2/normal2 vert3/uv3/normal3”.

A diagram of a computer

AI-generated content may be incorrect.

**Figure 6 denotes the encapsulation part of the diagram in more detail**

What this section refers to is the question of does the file encapsulate more files in itself, an example of which would be .Str, the .Str file is a ps3 stream file created by EA, it in itself has no use other than to store more files and to have its own compression algorithm on top of it, the actual files inside of the .str are what matters, as a result the .str file encapsulates more data.

This needs to be known by the reverse engineer because if it encapsulates more files inside of it, those files need to be reverse engineered to now, as a means of game preservation, just extracting the sections and files of the .str does nothing.

Moreover this is shown in the project as well, the .Str files contained sub files inside, those were exported out by means of reading the offsets of each section in the .Str files, and then exporting start to end in each section, this resulted in each of the files of the .str e.g textures, models, binary space partition files, sounds etc, to be successfully exported.

Then from that, the .rws and .dff could then be reverse engineered, of which they were, but that’s later.

# Evaluation

## Evaluation Methodology

### Evaluation Metrics

Success Metrics include:

- .Str reading/input, can .str files be inputted and have its contents successfully read i.e files are valid, files are made valid through being able to be successfully open through other software

- .Rws reading/input, can .rws files be inputted and have its contents converted to .obj successfully I.e can said .objs be loaded in 3d viewing/editing software

- Speed of application, when the program is run, it should not take a tremendous amount of time.

- replicability, the diagram ought to reflect the development progress of the .Str/.Rws artefacts, as that is the aim of the project, therefore from that, future files that are to be reverse engineered need to be able to replicate this progress.

### Baseline systems

The current base lines include, Greavsey1889 visceral toolkit that processes .Str files, reads them and allows for selected files to be exported, allows for reading and exporting in very quick time.

### Dataset

**Performance Statistics:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File Amount / Sizes | Project Artefacts  (.str) | Visceral Toolkit Cauldron  (.str) | Project Artefacts (.rws and .dff of each .str) | Python Plugin  ( .rws and .dff of each .str) |
| 3 files:  - zone01.str (12768 kb)  - zone02.str (9666 kb)  - zone03.str ( 8748 kb) | 28173 ms + 38421 ms + 24282 ms  Medal of homer | 2000ms | 2969 ms + 4941 ms + 3687 ms | 1000ms |
| 10 files:  - zone01.str (822 kb)  - zone02.str (1510 kb)  - zone03.str(1114 kb)  - zone04.str(668 kb)  - zone05.str(230 bk)  -zone06.str(572 kb)  -zone07.str( 304 kb)  -zone08.str(612 kb)  -zone20.str(446 kb)  -zone22.str(492 kb) | 2834 ms + 5588 ms + 3900 ms + 2674ms + 1050ms + 1859ms + 1353ms + 2509ms + 1317ms + 1694ms  Mob rules | 2000ms | 7248 ms + 13264 ms + 1734 ms + 4475 ms + 1498ms + 2030ms + 1753ms + 4231ms + 4253ms + 1806ms | 1000ms |
| -zone01.str (3298 kb)  -zone02.str(1846 kb)  -zone04.str(2872 kb)  -zone05.str(246 kb) | 8332ms + 5148ms + 8174ms + 506ms  Never quest | 1000ms | 2027ms + 1037ms + 3865ms + 171ms | 1000ms |
| -zone01.str(4074 kb)  -zone02.str(120kb)  -zone03.str(796kb)  -zone04.str (2906kb)  -zone05.str(10 kb)  -zone06.str(10 kb) | 10017ms + 498ms + 2009ms + 7727ms + 17ms + 16ms  rhymes | 1000ms | 36527ms + 1278ms + 9527ms + 19060ms + 6ms + 4ms | 1000ms |
| -zone01.str(5574 kb)  -zone02.str(864 kb)  -zone04.str(642 kb) | 16281ms + 3814ms + 3080ms  Meet thy player | 1000ms | 4764ms + 2244ms + 2543ms | 1000ms |
| -zone01.str(10 kb)  -zone02.str(3138 kb)  -zone03.str(2798 kb)  -zone04.str(2228 kb) | 19 ms + 10707ms + 10847ms + 9902ms  Day of dolphin | 1000ms | 5ms + 10646ms + 1262ms + 1629ms | 1000ms |
|  |  |  |  |  |

## Results

### .Str Input/Output:

.Str files can be put into an “AllAssets” Folder and the .Str files are read, “de-sectioned”, uncompressed and the files extracted in bulk. Each file has its file extension preserved, each file can then be processed individually depending on context I.e xmls are preserved as xmls and can be read as xmls, meaning files are successfully preserved.

### .Rws Input/Output:

.Rws files are automatically converted to .Obj file formats instantly after .Str files are processed, this ensures user usability, .Rws have their triangle strips converted to face geometry, each of its indices, vertices and Uv’s are transcribed into text with the correct prefixes e.g “v” for verts, “vt” for Uvs, which are then converted to .Objs, more over Normals are calculated instead of pulled from .Rws files, because .Rws do not contain normals. An example of which is below.

A grey tank with wheels

AI-generated content may be incorrect.

**Figure 7 shows the second half of a tank from Medal of Homer.**

**A black screen with white text

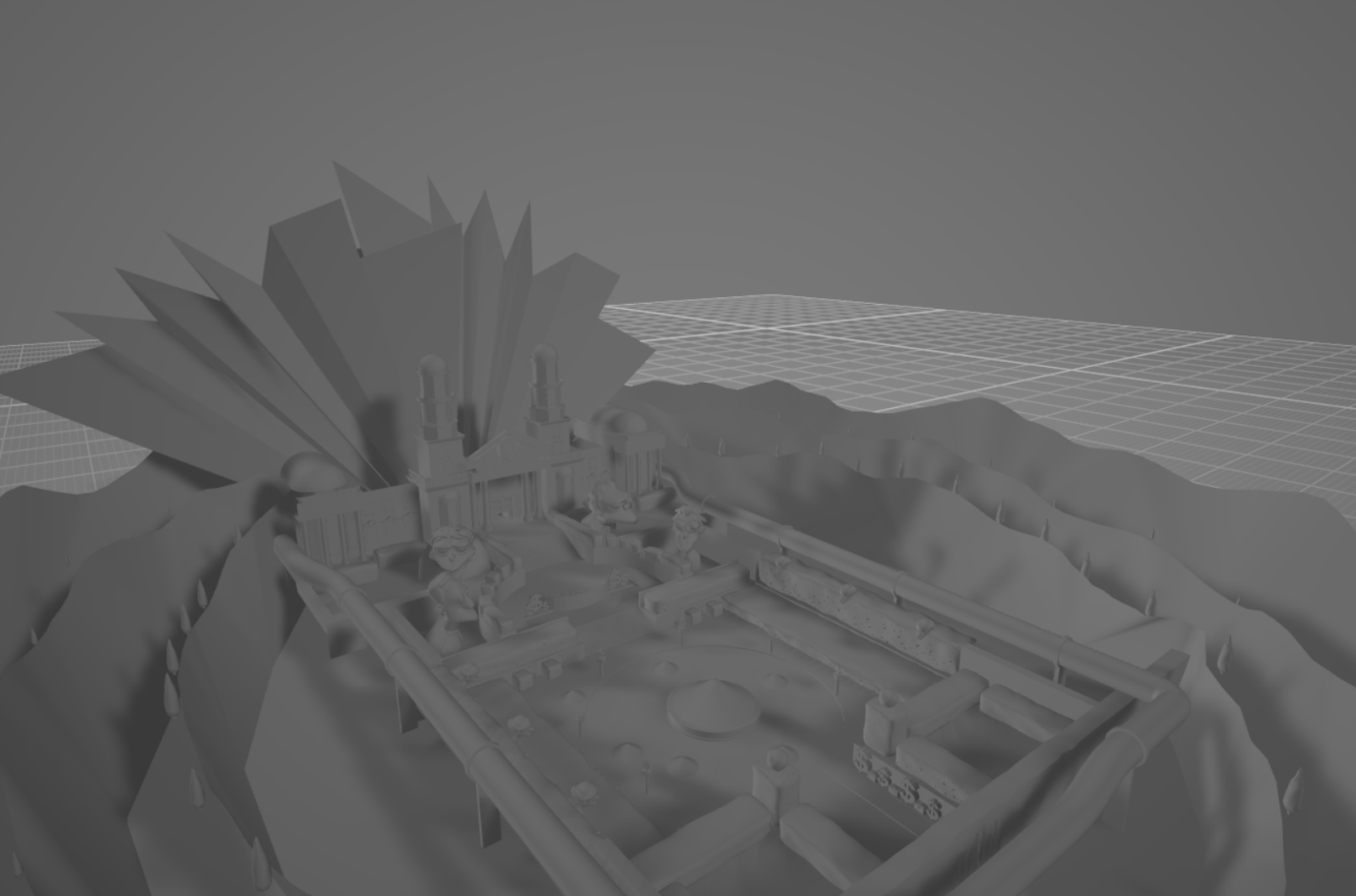
AI-generated content may be incorrect.**

**Figure 8 shows the vertices data inside said half a tank**

### .Objs Compatibilities

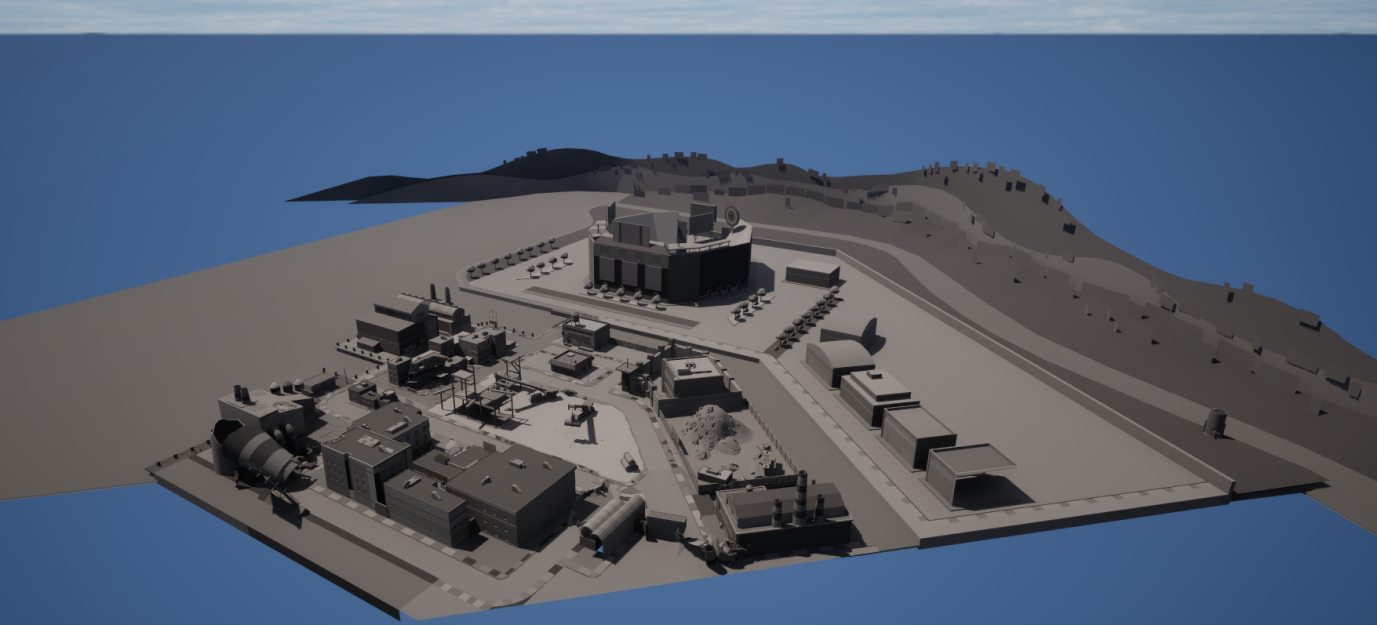
The Final output are a series of .Objs, each .Objs have the correct Vertices in the form of Vs, UVs in the form of Vts and correctly calculated vertex normals in the form of Vns. Each .Objs can be put into .Obj viewer, maya, blender as well as other 3d model rendering software, each vertex, face, edge etc of the geometry can be edited in 3d modelling software.

### Screenshot Proofs



**Figure 9 shows the successful creation of the first stage of the second to last level of the simpsons game, of which were .rws prior**

  
  
**Figure 10 shows a statue prop in the never quest level of the Simpsons game, of which was a .rws model prior**



**Figure 11 shows the entire of the colossal doughnut fight in its entirety, of which were .rws models prior**

## Discussion

The proposed File decoders that were implemented were statistically slower than prior file decoder / openers, this is in part due to the reports artefact performing more computations per CPU cycle when processing the files, for instance the python file opener only looks for the specific renderware geometry header then it begins to process the files contents, whilst the artefacts that were created did far more, such as, reading all renderware headers for future work like texture extracting, .Str opening in conjunction to .rws and .dff converting coupled with decompressing all of the .str files contents.

Furthermore the outputs for each of the .objs contain vertex normals in the from of “vn “ rows, in contrast the .rws converter “turk” created, does not have this functionality due to that persons artefact merely pulling already existing data rather than actually trying to create new data for a more feature rich product, that is to say the artefact ports .dff and .rws to blender, whilst the project artefacts will recalculate the vertex normals for .obj creation and from that they can be put into any 3d software opener with all the functionality intact I.e verts, uvs and normals.

The results work and are valid due to the conversion setup in the project artefact, as stated prior the geometry type of the .rws and .dff files are triangle strips, said triangle strips cannot be made into .obj unless converted to triangle faces, this is done in both the project artefact and “turks” python artefact, where they differ as stated prior is with the output, but the project artefact converts them to triangle faces then uses those faces to construct face normals then vertex normals.

This is what led to the reliable flat shading as seen in the unreal engine output and the inbuilt object viewer designed by windows. Outputs from the project artefact are therefore correctly lit due to this calculation and application.

The results were also successfully created through proper understanding of file syntax, as shown in the learning tool and methodology, the

# Conclusions

The important take aways from this project are, the successful reverse engineering and preservation of the assets provided as well as the demonstration of the usefulness of the learning diagram. That is to say that the .Str files are decompressed and its subfiles extracted, the .rws and .dff model files were successfully converted to .obj, those .objs can then be put into 3d rendering software successfully. From this the **GAD** has been shown to produce viable file conversions from old proprietary file types to non- proprietary contemporary file types.

As a result, the aim of the project was made successful as the game assets can in fact be put into 3d rendering software like game engines, and from that those games can then be “remastered” using those assets, therefore the diagram acts as a useful tool for the purposes of preserving games and their contents, due to the files being ported successfully.

# Recommendations for future work

For future work having the artefacts include texture reverse engineering and conversion, this would be done as a means of further games preservation as there are a lot of textures that need to be ported over to contemporary formats e.g .dds, png or jpeg. Furthermore, due to time constraints this was not possible, but for the future having this be done via the use of the **GAD** and adding it to the already existing artefacts would be an improvement for games preservations.

More work for the future would include game code decomplication and porting all of the game files preserved were game asset files i.e 3d model files, having code be preserved by virtue of decompiling and recompiling would enable the games to be preserved further as you would effectively have the finished product.

# References

Tito waluyo purboyo, 2017, a review of data compression techniques, international journal of applied

engineering research, 8596 - 8963, (PDF) A review of data compression techniques.

SIDDEQ, mohammed M and Rodrigues, marcos, 2016, Novel 3D compression methods for geometry

connectivity and texture, Sheffield hallam university research archive, 1 - 16, Microsoft Word -

Research\_No\_7\_MR\_final.docx

Mustafa ORAL, Ammar abbas Elmas, 2017, A brief history of 3d mesh compression, Cukorva

university, 1-6, ,

https://www.researchgate.net/publication/327905583\_A\_Brief\_History\_of\_3D\_Mesh\_Compression

Amandeep Singh Sidhu, 2014, Research paper on text data compression algorithm using hybrid approach,

International Journal of Computer Science and Mobile Computing,volume 3 issue 12, 1 - 10,

https://ijcsmc.com/docs/papers/December2014/V3I12201404.pdf.Sjöstrand, M. (2005). *A study in compression algorithms*.

https://www.diva-portal.org/smash/get/diva2:830266/FULLTEXT01.pdf.

Peng, J., Kim, C.-S. and Jay Kuo, C.-C. . (2005). Technologies for 3D mesh compression: A survey.

*Journal of Visual Communication and Image Representation*, 16(6), pp.688–733.

doi:https://doi.org/10.1016/j.jvcir.2005.03.001.

Girard, J.-F. and Koschke, R. (2000). A comparison of abstract data types and objects recovery

techniques. *Science of Computer Programming*, 36(2-3), pp.149–181.

doi:https://doi.org/10.1016/s0167-6423(99)00035-0.

Guttag, J.V., Horowitz, E. and Musser, D.R. (1978). Abstract data types and software validation.

*Communications of the ACM*, 21(12), pp.1048–1064. doi:https://doi.org/10.1145/359657.359666.

Guttag, J. (1977). Abstract data types and the development of data structures. *Communications of the*

*ACM*, 20(6), pp.396–404. doi:https://doi.org/10.1145/359605.359618.

Liskov, B. and Zilles, S. (1974). Programming with abstract data types. *ACM SIGPLAN Notices*, 9(4),

pp.50–59. doi:https://doi.org/10.1145/942572.807045.

Musser, D.R. (1980). On proving inductive properties of abstract data types.

doi:https://doi.org/10.1145/567446.567461.

Bertoni, A., Mauri, G. and P. Miglioli (2006). Towards a theory of abstract data types: A discussion on

problems and tools. *Lecture Notes in Computer Science*, [online] pp.44–58.

doi:https://doi.org/10.1007/3-540-09981-6\_4.

A, A. (2021). *Reverse Engineering Research*. [online] doi:https://doi.org/10.13140/RG.2.2.28030.51520.

Mantovani, A., Aonzo, S., Fratantonio, Y., Talos, C. and Balzarotti, D. (n.d.). *RE-Mind: a First Look*

*Inside the Mind of a Reverse Engineer*. [online] Available at:

https://www.usenix.org/system/files/sec22summer\_mantovani.pdf.

Jain, A., Swapnil Soner and Anand Gadwal (2011). Reverse engineering: Journey from code to design.

doi:https://doi.org/10.1109/icectech.2011.5941966.Singh, R. (2013). A Review of Reverse Engineering Theories and Tools. [online] 2, pp.35–38. Available

at: https://www.ijesi.org/papers/Vol(2)1/G213538.pdf.

Cipresso, T. and Stamp, M. (2010). Software Reverse Engineering. *Handbook of Information and*

*Communication Security*, pp.659–696. doi:https://doi.org/10.1007/978-3-642-04117-4\_31.

Müller, H.A., Jahnke, J.H., Smith, D.B., Storey, M.-A., Tilley, S.R. and Wong, K. (2000). Reverse

engineering. *Proceedings of the conference on The future of Software engineering - ICSE ’00*.

doi:https://doi.org/10.1145/336512.336526.

Google Books. (2024). *Reversing*. [online] Available at:

https://books.google.co.uk/books?hl=en&lr=&id=\_78HnPPRU\_oC&oi=fnd&pg=PR23&dq=reverse+engi

neering+software&ots=EP1MLkmRVn&sig=stA7p1aP8xPSkUcFJi6jOCmA\_Is&redir\_esc=y#v=onepage

&q=reverse%20engineering%20software&f=false [Accessed 25 Nov. 2024].

Davis, K.L. and Alken, P.H. (2002). Data reverse engineering: a historical survey.

doi:https://doi.org/10.1109/wcre.2000.891454.

Bachell, A. and Barr, M. (2014). Video Game Preservation in the UK: A Survey of Records Management

Practices. *International Journal of Digital Curation*, 9(2), pp.139–170.

doi:https://doi.org/10.2218/ijdc.v9i2.294.

Davide V (2024). *GTA-Modding.com - Download Area» GTA San Andreas» Tools» RW Analyze*. [online]

Gta-modding.com. Available at: https://www.gta-modding.com/area/file-33-rw-analyze.html [Accessed

25 Nov. 2024].

Winget, M.A. and Murray, C. (2008). Collecting and preserving videogames and their related materials: A

review of current practice, game-related archives and research projects. *Proceedings of the American*

*Society for Information Science and Technology*, 45(1), pp.1–9.

doi:https://doi.org/10.1002/meet.2008.1450450250.

Todd, B. and Hopkins, J. (2019). *Running head: PRESERVING VIDEO GAME SIGNIFICANCE*

*Preserving Video Game Significance: A Practical Guide for Video Game Preservation, Exhibition, and*

*their Significant Properties*. [online] Available at:

https://jscholarship.library.jhu.edu/server/api/core/bitstreams/49d19d02-a439-4a7d-8243-a685429cbcac/c

ontent.Guay-Bélanger, D. (2021). Assembling Auras: Towards a Methodology for the Preservation and Study of

Video Games as Cultural Heritage Artefacts. *Games and Culture*, p.155541202110203.

doi:https://doi.org/10.1177/15554120211020381.

Digra.org. (2024). *View of Before It’s Too Late: Preserving Games across the Industry / Academia divide*.

[online] Available at: https://dl.digra.org/index.php/dl/article/view/468/468 [Accessed 25 Nov. 2024].

Johansson, C. and Koenitz, H. (n.d.). *Video Game Preservation and Emulation from Three Perspectives:*

*Developers, Archivists and Gamers Video Game Preservation and Emulation from Three Perspectives:*

*Developers, Archivists and Gamers*. [online] Available at:

http://sh.diva-portal.org/smash/get/diva2:1807915/FULLTEXT02.pdf.

Haydock, C. (2018). *Challenges in Preserving Video Games*. [online] Carolina Digital Repository.

Available at: https://cdr.lib.unc.edu/concern/masters\_papers/fn107276t [Accessed 25 Nov. 2024].

Folk, M. and Barkstrom, B.R. (2003). *Attributes of file formats for long-term preservation of scientific*

*and engineering data in digital libraries*. [online] Available at:

https://www.researchgate.net/publication/228726593\_Attributes\_of\_file\_formats\_for\_long-term\_preserva

tion\_of\_scientific\_and\_engineering\_data\_in\_digital\_libraries.

THE DEFINITIVE GUIDE TO EXPLORING FILE FORMATS Mr.Mouse and WATTO. (n.d.). Available

at: https://www.gamedevs.org/uploads/the-definitive-guide-to-exploring-file-formats.pdf [Accessed 25

Nov. 2024].

File Formats for Big Data Storage Systems. (2019). *International Journal of Engineering and Advanced*

*Technology*, 9(1), pp.2906–2912. doi:https://doi.org/10.35940/ijeat.a1196.109119.

Zhenhua, W. (2018). Design and Research of an Image File Format with Rich Information. *Journal of*

*Electrical and Electronic Engineering*, 6(2), p.71. doi:https://doi.org/10.11648/j.jeee.20180602.16.

Ontology of Heterogeneous Image File Formats and their Disparate Applications. (2021). *International*

*Journal of Advanced Trends in Computer Science and Engineering*, 10(6), pp.3138–3143.

doi:https://doi.org/10.30534/ijatcse/2021/091062021.

Dinneen, J.D. and Julien, C.-A. (2021). *The ubiquitous digital file: A review of file management research*.

[online] doi:https://doi.org/10.48550/arXiv.2109.09668.Underwood, W. (2012). Grammar-Based Specification and Parsing of Binary File Formats. *International*

*Journal of Digital Curation*, 7(1), pp.95–106. doi:https://doi.org/10.2218/ijdc.v7i1.217.

Fred brooks, Data definition and file syntax for ISO/TS 14048 data exchange with data storage format

based on ISO/TS 14048, RAUL CARLSON JOHAN TIVANDER, CHALMERS UNIVERSITY OF

TECHNOLOGY, 2001,

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=676dc77da1217c8cdd1b61d76c83cad5

30bc3159

# Bibliography

Tito waluyo purboyo, 2017, a review of data compression technqiques, international journal of applied

engineering research, 8596 - 8963, (PDF) A review of data compression techniques.

SIDDEQ, mohammed M and Rodrigues, marcos, 2016, Novel 3D compression methods for geometry

connectivity and texture, Sheffield hallam university research archive, 1 - 16, Microsoft Word -

Research\_No\_7\_MR\_final.docx

Mustafa ORAL, Ammar abbas Elmas, 2017, A brief history of 3d mesh compression, Cukorva

university, 1-6, ,

https://www.researchgate.net/publication/327905583\_A\_Brief\_History\_of\_3D\_Mesh\_Compression

Amandeep Singh Sidhu, 2014, Research paper on text data compression algorithm using hybrid approach,

International Journal of Computer Science and Mobile Computing,volume 3 issue 12, 1 - 10,

https://ijcsmc.com/docs/papers/December2014/V3I12201404.pdf.

Sjöstrand, M. (2005). *A study in compression algorithms*.

https://www.diva-portal.org/smash/get/diva2:830266/FULLTEXT01.pdf.

Peng, J., Kim, C.-S. and Jay Kuo, C.-C. . (2005). Technologies for 3D mesh compression: A survey.

*Journal of Visual Communication and Image Representation*, 16(6), pp.688–733.

doi:https://doi.org/10.1016/j.jvcir.2005.03.001.

Peng, J., Kim, C.-S. and Jay Kuo, C.-C. . (2005). Technologies for 3D mesh compression: A survey.

*Journal of Visual Communication and Image Representation*, 16(6), pp.688–733.

doi:https://doi.org/10.1016/j.jvcir.2005.03.001.Girard, J.-F. and Koschke, R. (2000). A comparison of abstract data types and objects recovery

techniques. *Science of Computer Programming*, 36(2-3), pp.149–181.

doi:https://doi.org/10.1016/s0167-6423(99)00035-0.

Guttag, J.V., Horowitz, E. and Musser, D.R. (1978). Abstract data types and software validation.

*Communications of the ACM*, 21(12), pp.1048–1064. doi:https://doi.org/10.1145/359657.359666.

Guttag, J. (1977). Abstract data types and the development of data structures. *Communications of the*

*ACM*, 20(6), pp.396–404. doi:https://doi.org/10.1145/359605.359618.

Liskov, B. and Zilles, S. (1974). Programming with abstract data types. *ACM SIGPLAN Notices*, 9(4),

pp.50–59. doi:https://doi.org/10.1145/942572.807045.

Musser, D.R. (1980). On proving inductive properties of abstract data types.

doi:https://doi.org/10.1145/567446.567461.

Bertoni, A., Mauri, G. and P. Miglioli (2006). Towards a theory of abstract data types: A discussion on

problems and tools. *Lecture Notes in Computer Science*, [online] pp.44–58.

doi:https://doi.org/10.1007/3-540-09981-6\_4.

A, A. (2021). *Reverse Engineering Research*. [online] doi:https://doi.org/10.13140/RG.2.2.28030.51520.

Mantovani, A., Aonzo, S., Fratantonio, Y., Talos, C. and Balzarotti, D. (n.d.). *RE-Mind: a First Look*

*Inside the Mind of a Reverse Engineer*. [online] Available at:

https://www.usenix.org/system/files/sec22summer\_mantovani.pdf.

Jain, A., Swapnil Soner and Anand Gadwal (2011). Reverse engineering: Journey from code to design.

doi:https://doi.org/10.1109/icectech.2011.5941966.

Singh, R. (2013). A Review of Reverse Engineering Theories and Tools. [online] 2, pp.35–38. Available

at: https://www.ijesi.org/papers/Vol(2)1/G213538.pdf.

Cipresso, T. and Stamp, M. (2010). Software Reverse Engineering. *Handbook of Information and*

*Communication Security*, pp.659–696. doi:https://doi.org/10.1007/978-3-642-04117-4\_31.

Müller, H.A., Jahnke, J.H., Smith, D.B., Storey, M.-A., Tilley, S.R. and Wong, K. (2000). Reverse

engineering. *Proceedings of the conference on The future of Software engineering - ICSE ’00*.

doi:https://doi.org/10.1145/336512.336526.Google Books. (2024). *Reversing*. [online] Available at:

https://books.google.co.uk/books?hl=en&lr=&id=\_78HnPPRU\_oC&oi=fnd&pg=PR23&dq=reverse+engi

neering+software&ots=EP1MLkmRVn&sig=stA7p1aP8xPSkUcFJi6jOCmA\_Is&redir\_esc=y#v=onepage

&q=reverse%20engineering%20software&f=false [Accessed 25 Nov. 2024].

Davis, K.L. and Alken, P.H. (2002). Data reverse engineering: a historical survey.

doi:https://doi.org/10.1109/wcre.2000.891454.

Bachell, A. and Barr, M. (2014). Video Game Preservation in the UK: A Survey of Records Management

Practices. *International Journal of Digital Curation*, 9(2), pp.139–170.

doi:https://doi.org/10.2218/ijdc.v9i2.294.

Davide V (2024). *GTA-Modding.com - Download Area» GTA San Andreas» Tools» RW Analyze*. [online]

Gta-modding.com. Available at: https://www.gta-modding.com/area/file-33-rw-analyze.html [Accessed

25 Nov. 2024].

Winget, M.A. and Murray, C. (2008). Collecting and preserving videogames and their related materials: A

review of current practice, game-related archives and research projects. *Proceedings of the American*

*Society for Information Science and Technology*, 45(1), pp.1–9.

doi:https://doi.org/10.1002/meet.2008.1450450250.

Todd, B. and Hopkins, J. (2019). *Running head: PRESERVING VIDEO GAME SIGNIFICANCE*

*Preserving Video Game Significance: A Practical Guide for Video Game Preservation, Exhibition, and*

*their Significant Properties*. [online] Available at:

https://jscholarship.library.jhu.edu/server/api/core/bitstreams/49d19d02-a439-4a7d-8243-a685429cbcac/c

ontent.

Guay-Bélanger, D. (2021). Assembling Auras: Towards a Methodology for the Preservation and Study of

Video Games as Cultural Heritage Artefacts. *Games and Culture*, p.155541202110203.

doi:https://doi.org/10.1177/15554120211020381.

Digra.org. (2024). *View of Before It’s Too Late: Preserving Games across the Industry / Academia divide*.

[online] Available at: https://dl.digra.org/index.php/dl/article/view/468/468 [Accessed 25 Nov. 2024].

Johansson, C. and Koenitz, H. (n.d.). *Video Game Preservation and Emulation from Three Perspectives:*

*Developers, Archivists and Gamers Video Game Preservation and Emulation from Three Perspectives:Developers, Archivists and Gamers*. [online] Available at:

http://sh.diva-portal.org/smash/get/diva2:1807915/FULLTEXT02.pdf.

Haydock, C. (2018). *Challenges in Preserving Video Games*. [online] Carolina Digital Repository.

Available at: https://cdr.lib.unc.edu/concern/masters\_papers/fn107276t [Accessed 25 Nov. 2024].

Folk, M. and Barkstrom, B.R. (2003). *Attributes of file formats for long-term preservation of scientific*

*and engineering data in digital libraries*. [online] Available at:

https://www.researchgate.net/publication/228726593\_Attributes\_of\_file\_formats\_for\_long-term\_preserva

tion\_of\_scientific\_and\_engineering\_data\_in\_digital\_libraries.

THE DEFINITIVE GUIDE TO EXPLORING FILE FORMATS Mr.Mouse and WATTO. (n.d.). Available

at: https://www.gamedevs.org/uploads/the-definitive-guide-to-exploring-file-formats.pdf [Accessed 25

Nov. 2024].

File Formats for Big Data Storage Systems. (2019). *International Journal of Engineering and Advanced*

*Technology*, 9(1), pp.2906–2912. doi:https://doi.org/10.35940/ijeat.a1196.109119.

Zhenhua, W. (2018). Design and Research of an Image File Format with Rich Information. *Journal of*

*Electrical and Electronic Engineering*, 6(2), p.71. doi:https://doi.org/10.11648/j.jeee.20180602.16.

Ontology of Heterogeneous Image File Formats and their Disparate Applications. (2021). *International*

*Journal of Advanced Trends in Computer Science and Engineering*, 10(6), pp.3138–3143.

doi:https://doi.org/10.30534/ijatcse/2021/091062021.

Dinneen, J.D. and Julien, C.-A. (2021). *The ubiquitous digital file: A review of file management research*.

[online] doi:https://doi.org/10.48550/arXiv.2109.09668.

Underwood, W. (2012). Grammar-Based Specification and Parsing of Binary File Formats. *International*

*Journal of Digital Curation*, 7(1), pp.95–106. doi:https://doi.org/10.2218/ijdc.v7i1.217.

Fred brooks, Data definition and file syntax for ISO/TS 14048 data exchange with data storage format

based on ISO/TS 14048, RAUL CARLSON JOHAN TIVANDER, CHALMERS UNIVERSITY OF

TECHNOLOGY, 2001,

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=676dc77da1217c8cdd1b61d76c83cad5

30bc3159Greavesy1899 (2023). *Resource Types*. [online] GitHub. Available at:

https://github.com/Greavesy1899/VisceralToolkit/wiki/Resource-Types [Accessed 25 Nov. 2024].

Greavesy1899 (2024). *Releases · Greavesy1899/VisceralToolkit*. [online] GitHub. Available at:

https://github.com/Greavesy1899/VisceralToolkit/releases/ [Accessed 25 Nov. 2024].

# Appendices

A black and pink background with lines

AI-generated content may be incorrect.