Faculty of Computing and Digital Technologies

Staffordshire University

**BACHELOR THESIS**

Christos – Emmanouil Anastasiou

**Efficient simulation of destructible environments in video games**

A project submitted in partial fulfilment of the award of the degree of BSc (Hons) Computer Games programming from Staffordshire University

Supervisor of the bachelor thesis: Bob Hobbs

May 2020

Declaration of caring out this bachelor thesis independently…

Understanding that is being protected under the Copyrights act…

Date and Signature

Title: Efficient Simulation of destructible environments in video games

Author: Christos – Emmanouil Anastasiou

Supervisor: Bob Hobbs

Abstract: Summary of what the thesis covers…

Keywords:

# Acknowledgments

Expressing gratitude...

# Contents

[Chapter 1 1](#_Toc31293320)

[Introduction 1](#_Toc31293321)

[objective 2](#_Toc31293322)

[Chapter 2 3](#_Toc31293323)

[2.1 The importance of project planning 3](#_Toc31293324)

[2.2 Selection of project planning methodology 4](#_Toc31293325)

[Chapter 3 6](#_Toc31293326)

[3.1 Overview of techniques 6](#_Toc31293327)

[3.1.1 Geometry preparation 7](#_Toc31293328)

[Object replacement or deactivation 7](#_Toc31293329)

[Voronoi shattering 8](#_Toc31293330)

[Delaunay tetrahedral decomposition 8](#_Toc31293331)

[Boolean operations 8](#_Toc31293332)

[Convex decomposition 9](#_Toc31293333)

[3.1.2 Real time destruction methods 9](#_Toc31293334)

[Canned Animations 10](#_Toc31293335)

[Real-time Boolean operations 10](#_Toc31293336)

[Finite Element Method (FEM) 10](#_Toc31293337)

[Jointed breakable objects 11](#_Toc31293338)

[3.1.3 Terrain 11](#_Toc31293339)

[Heightmaps 11](#_Toc31293340)

[Voxels 11](#_Toc31293341)

[Marching cubes 11](#_Toc31293342)

[Selection of methodology 13](#_Toc31293343)

[program structure 13](#_Toc31293344)

[Design 13](#_Toc31293345)

# Chapter 1

# Introduction

Computer games have broken new grounds on rendering and animation in the past decade, achieving new levels of realism and believability that contribute to the overall immersion [1] [2]. That illusion, however, can easily be dispelled when the player finds a chain-link fence that is impervious to anything from a machine gun to a tank. Destructible environments refer to the player’s ability to destroy their surroundings, thus knowing that they can use the environment to their advantage [3]. For example, the player can explode a barrel to eliminate enemies that are standing near it as seen in most shooter games or demolish part of a building to gain tactical advantage over other players as seen in Battlefield series. Some early games offered minor destructibility in their levels such as demolishable covers, breakable crates or destructible CCTV cameras such as the game Perfect Dark that also featured specific walls that had “weak” points that could be shot at and destroyed to open paths to subsequent levels.

Breakable objects made their first appearance in Gun Fight (1975) where two players engage in a gun fight while standing opposite to one another and being separated by cactuses that absorb gunshots and disappear partially depending to where the bullet was landed. Not long after, Space Invaders (1978) was released where the player was being protected by simple bricks that were slowly diminishing after each shot. Asteroids was released a year later in 1979 [4] where the player had to shoot flying asteroids to break them into smaller pieces. Rampage (1986) was one of the earliest classic arcade games that featured fully destructible environments where players were taking control of a giant ape with the objective of taking down apartment buildings and cause mayhem in the city.

At the time where games were transitioning from 2D to 3D graphics in the mid-nineties, games such as Magic Carpet (1994) was one of the first 3D games to feature terrain deformation with the use of height maps, the terrain was being modified by altering the height of a given point on the map [5]. Some later games such as Worms (1995) featured fully destructible terrain where the player would fight other worms with a large arsenal of weapons in a 2D setting.

By the year 2000, specialised pieces of hardware or graphics accelerator cards currently known as graphics processing units (GPU) brought games that took destructibility to the next level. The Red Faction series was primarily focused on the destructibility of its environments and buildings, more details on section 3.1.2. The Crysis series was developed on Crytek’s proprietary engine CryEngine [34] and featured destroyable objects using various techniques that are discussed in detail in later sections such as jointed breakable objects, object replacement and Boolean operations. Far Cry 2 (2008) featured a different form of destructibility and that was dry grass that when burned it would spread out giving the player an interesting addition into his tactical options. In the modern age of gaming, large studios such as DICE and Ubisoft Montreal invested valuable time and effort on developing destruction systems into their proprietary engines – Frostbite [7] and AnvilNext [8] respectively - to import into their online, multiplayer shooter games. Current releases include Remedy studio’s Control that is experimenting with realistic small-scale destroyable objects i.e. all furniture and solid material inside a building can be broken down realistically. Also, Noita (2019) currently in early access and developed by Nolla Games is a 2D procedurally generated world where every pixel is physically simulated [35].

Destruction in video games have played its important role in enhancing physics and tactical manoeuvring, two of the five types of game mechanics based on Game Mechanics: Advanced Game Design by Ernest Adams and Joris Dormans. Producing non-scripted destructible environments that are most importantly fun and simulated realistically while also protecting the player from themselves from breaking the game has been a tough challenge for developers and designers in the past. Many games have acknowledged over the years that there is a rise of demand on physically accurate destruction as it can prove to be a good addition to gameplay while going alongside with the game’s context [9].

Current advances in the field, aim to accomplish more realistic; interactable game worlds by innovating and refining several available approaches used for 2D and 3D environments. Although latest graphical hardware components are becoming increasingly more powerful [10], implementing a highly complex physical simulation introduces a development hurdle: the more accurate a simulation is the more impact it has to the system. Taking into consideration the real-time constraints that are required for a computer game that performs sufficiently – achieving the highest and most consistent frame rate possible, the game developers are left with only one choice. To deduct some of the realism by creating simple and compact scenes but also by disregarding inessential aspects of physical simulation.

# objective

There has been a lack of dynamic destruction systems on many games in the past. Canned animations are pre-rendered animations and not actual simulations that are triggered on certain events. Their use has proven to be non-beneficial to gameplay but only as visual effects.

The purpose of this thesis is to closely examine currently available techniques found from research that are used to simulate destructible environments. Additionally, after comparing the techniques, selecting one that can be produced as an artefact that will include a game environment that its main purpose will be to implement the chosen technique but also test its performance in a suitable computer games scenario.

**Thesis Structure**

**Chapter 2: Importance of project planning**

This chapter describes the importance of project planning in software development and selects the appropriate methodology for the requirements of this thesis.

**Chapter 3: Overview of techniques**

This chapter describes and compares various methods used to simulate destructible environments during development and in real-time. One method is selected for implementation using the DirectX11 API.

**Chapter 4: Design and Implementation**

This chapter describes the design and implementation of the method selected from the previous chapter.

# Chapter 2

# 2.1 The importance of project planning

Project planning is often confused with project schedule. That is a list of tasks and dates that specify what needs to be done and when. This tool will certainly come in use but is only a piece of the toolbox.

Project Management Body of Knowledge guide (PMBOK) 2008 edition provides a definition of project management as “application of knowledge, skills, tools and techniques to project activities to meet project requirements. Project management is accomplished through the application and integration of the 42 logically grouped project management processes of initiating, planning, executing, monitoring and controlling, and closing” (*PMBOK guide, Project Management Institute, 2008, p.6*). What the PMBOK guide fails to mention is that there are other activities that are involved in project management such as handling political matters, ensuring that team members communicate with each other effectively and perform at the required level. (*James P. LEWIS, 2011*) [11].

The moment after a new project has come up and been approved, it is tempting to hurry and start carrying out the work immediately as there are people to talk to, things to purchase and a long list of tasks to write. That is the wrong approach to take even how tempting it first seems to be [11]. Before diving into the doing of the project, it is a smart move to take a step back to have a better view of the bigger picture of everything that needs to be produced and the way that they are going to be developed. For this to happen, a plan of the of the upcoming work is necessary. This will surely slow down the start of the actual implementation of the project, it will however save a significant amount of time in the long run due to how convenient it will be to keep track of the project’s progress but also have a clear image of future tasks.

A project plan streamlines the operation and prevents many probable incidences of misunderstanding and confusion for whoever is involved [12]. For instance, an experienced project manager will think ahead and assign the right tasks to the right people for the right dates. As a result, not having to worry if the workers are available for those dates as the PM already arranged for them to be. Another example is that the project manager will have already planned the financial paperwork thus not having to worry about paying the invoices after the job has concluded. Project planning consists of defining the processes that are crucial for determining the project’s objectives and facilitating the structure to enable these processes to come to fruition [13]. The structure represents the various procedures required for a successful project such as budgeting estimation, human resources, quality measures, deliverables etc. The outcome of that process is a project plan. A set of documents that explains what is going to be done and how.

# 2.2 Selection of project planning methodology

Jason Charvat (2003) in his book Project Management Methodologies, he defines project management methodology as follows: “A methodology is a set of guidelines or principles that can tailored and applied to a specific situation. In a project environment, these guidelines might be a list of things to do. A methodology could also be a specific approach, templates, forms, and even checklists used over the project life cycle.” In other words, countless processes guide project managers throughout the project but also provide the steps for carrying out the tasks and running the operation smoothly. There are various project methodologies available that are currently in use but the most popular are Agile; Kanban; Waterfall; Lean; Six Sigma and Project Management Body of Knowledge guide (PMBOK) [14].

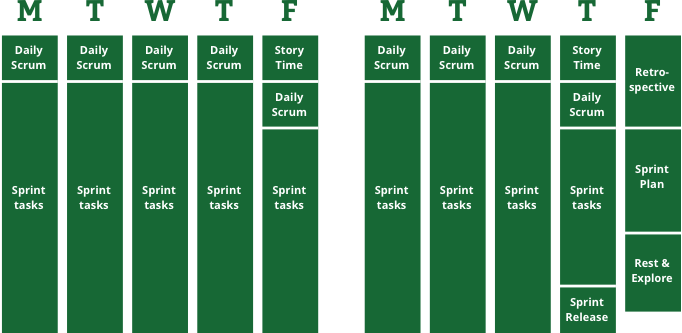
Selecting the appropriate planning methodology for the project is crucial as it determines not only the type of workflow that is going to be structured but also the flexibility and freedom needed for a healthy software development cycle (Charvat, 2003).

When selecting a methodology, it is best to first make a list of the criteria that the methodology needs to meet as this will later assist the searching process by narrowing down the options. First of all, it is worth considering that this is a solo project with research purposes and not with specific deliverables provided by a client. Additionally, it involves the development of software. Therefore, project planning methodologies designed for software development will only be taken into account. The selected approach will work as a framework for managing workload and scheduling tasks while controlling the procedure of development by providing documentation for monitoring progress but most importantly allowing adjustments when required. Tweaking and fine tuning are crucial for improving the prioritization of tasks and ways of approaching unsolved problems resulting to additional planning as the project moves on to later stages of development. For these reasons, an Agile approach appears to be the most suitable approach for the precise nature of this project. However, although there are many Agile approaches available, not all of them fit for this specific circumstance. The application of the Agile methodology that is being adopted is Scrum as it is designed for a small team of people thus making it easier to be modified for one-person operation [15]. Secondly, the benefits of agile methodology are frequent iterations, continuous feedback and overall incremental development that encourages responding to change over following a plan (*Schwaber and Sutherland, 2018)*. Besides, Scrum is lightweight, simple to understand and widely used in the software development industry while also fitted for research purposes according to “The Scrum Guide” [16] that was crafted by Ken Schwaber and Jeff Sutherland, the creators of Scrum.

Alex Andrews’s article “Scrum Of One: How to Bring Scrum into your One-Person Operation” [17] was used as a reference and guidance for this plan. The author starts off by focusing on the core principles of Scrum rather than the operations that are designed for a team. The first principle states that it is essential to share with other people the state of the product after each iteration as it is notoriously easy to miss the right degree of importance of certain tasks and waste precious time on features that were never needed or wanted. The second principle declares that there should be short term goals that measure the level of productivity at the end of each week. It is important to quantify the progress, so it is easier to track and optimize it. The third principle refers to the ability of self-reflection and review of processes and progress in a regular basis. These principles correspond to the criteria mentioned above.

The chosen methodology operates on several rules comprised of events, tools and roles and the relationship between them. What follows is a trimmed version of traditional Scrum:

**Sprint**: Development is broken down into several iterative cycles also known as sprints. Sprints usually last from two weeks to up to 30 days and for that specified period, the development team focuses on a clear and detailed goal such as adding a new feature or fixing a collection of bugs (Andrews, 2017). A two-week sprint plan has been decided on to allocate enough time for a meaningful set of tasks to complete without having more than enough time to get carried away with insignificant tasks.



**Figure 1.1**: Alex Andrew’s Scrum of one: a two-week sprint plan broken down to reflect each day’s assigned sprint component.

**Sprint plan/Product backlog:** consists of a list of sprint goals that are broken down into smaller sprint tasks that are then considered for the next sprint. A technique Alex used is that each sprint task is associated with an arbitrary number that defines the rough estimated time of completion. For instance, one-point tasks should take on average half as long as two-point tasks. The numbers being used are 1,2,3,5 and 8. The numbers 5 and 8 are used to add a distinct value to the effort needed to complete a bigger task. if a certain task has been employed numerus times before, the chances are it will take less time thus reducing the number of task points regardless of its complexity. Similarly, an unfamiliar task will increase the number of task points even if it is simple. After including all the task points for the upcoming sprint, the next step is to add them all up. Consistency is the utmost importance here as the numbers are used to measure the productivity levels at the end of each sprint and compare the number to previous iterations.

**Daily scrum (5 minutes)**: will take place at the beginning of each working day. The daily scrum will consist of a quick update of yesterday’s progress, obstacles that blocked uncompleted tasks and today’s goals.

**Story time (30 - 45 minutes):** will take place at the end of each week and will allow time for shortly pausing development and having a look at the big-picture tasks. Also known as product backlog. Adding new tasks and reviewing the old tasks is equally as important.

**Task board:** comprises of a board of TODO, DOING and DONE categorised tasks. The tool selected for documenting the task board is Trello, an online tool, described below.

**Weekly sprint review:** Weekly meeting with supervisor that involves weekly update and discussion on obstacles found along the way and how to overcome them.

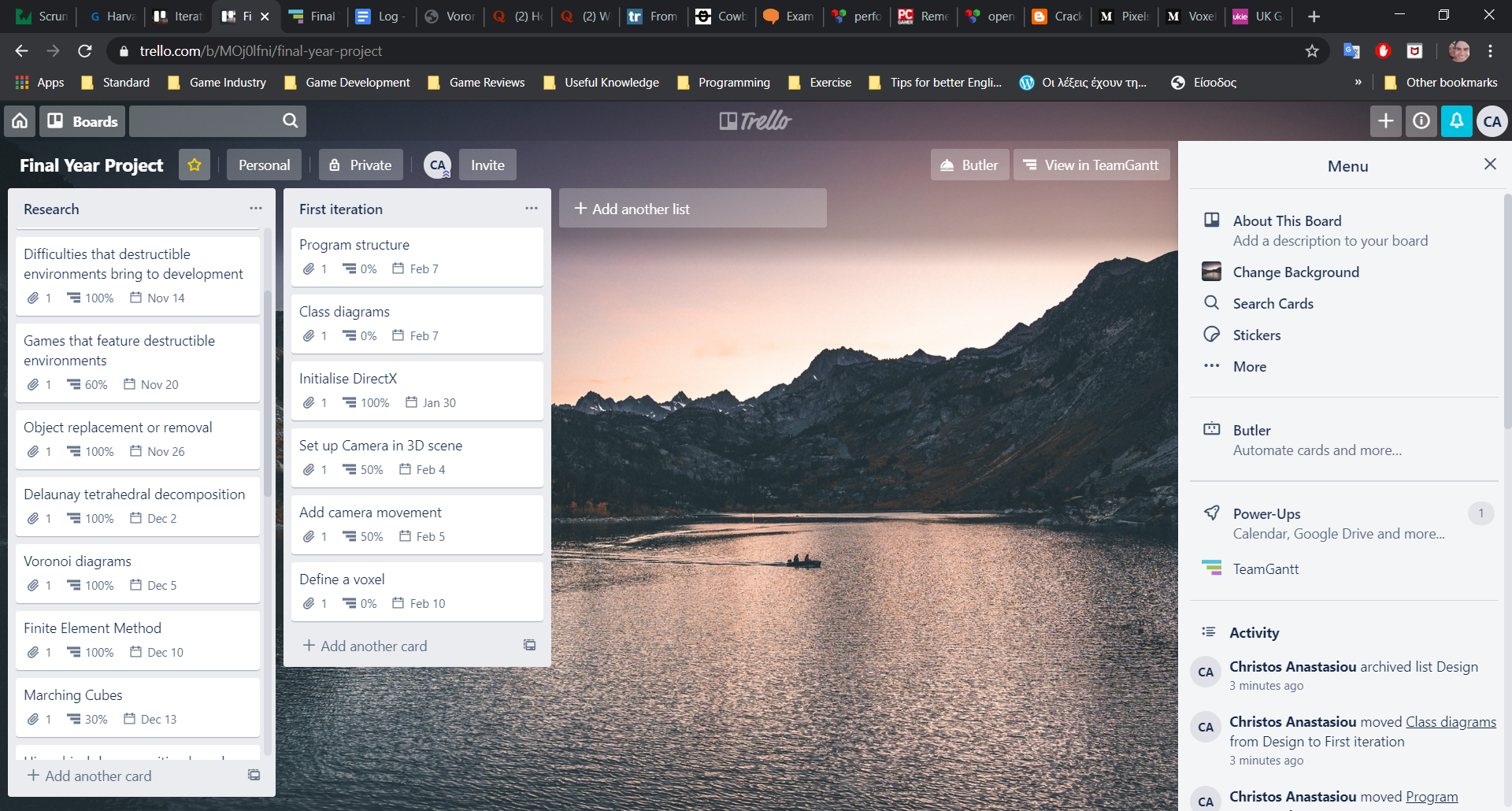
**Retrospective (~2 hours):** At the end of each week, self-reflection takes place on the week that went past by asking questions to detect what worked well and what could have been done better.

**Sprint release**: sharing the current state of the artefact with other people is important for receiving feedback as previously mentioned from one of the three principles. This will take place at the end of each sprint.

# Tools

## Trello

Trello is a free online project management tool designed for creating task boards. The task boards produced can be archived and accessed from any device connected to the internet. Thus, providing a synchronised platform for being able to develop in. A board is where the the management for each project takes place. Each board is made up of a series of lists and each list contains several items that represent each task. These items are called cards and the idea of Trello is that as the project progresses, these cards can dynamically move – starting from the left - to subsequent lists that represent the next stages of the project. Trello is a visual representation of the project’s progress. The free version of Trello allows for a single addon.



**Figure 1.1**: Visual representation of the project using cards and lists in a Trello board.

## Team Gantt

Team Gantt is an online software for producing dynamic Gantt charts and keep track of tasks for a project. It can be synchronised with a Trello board. Hence, breaking down and updating the tasks dynamically but also setting a deadline that can be represented visually.

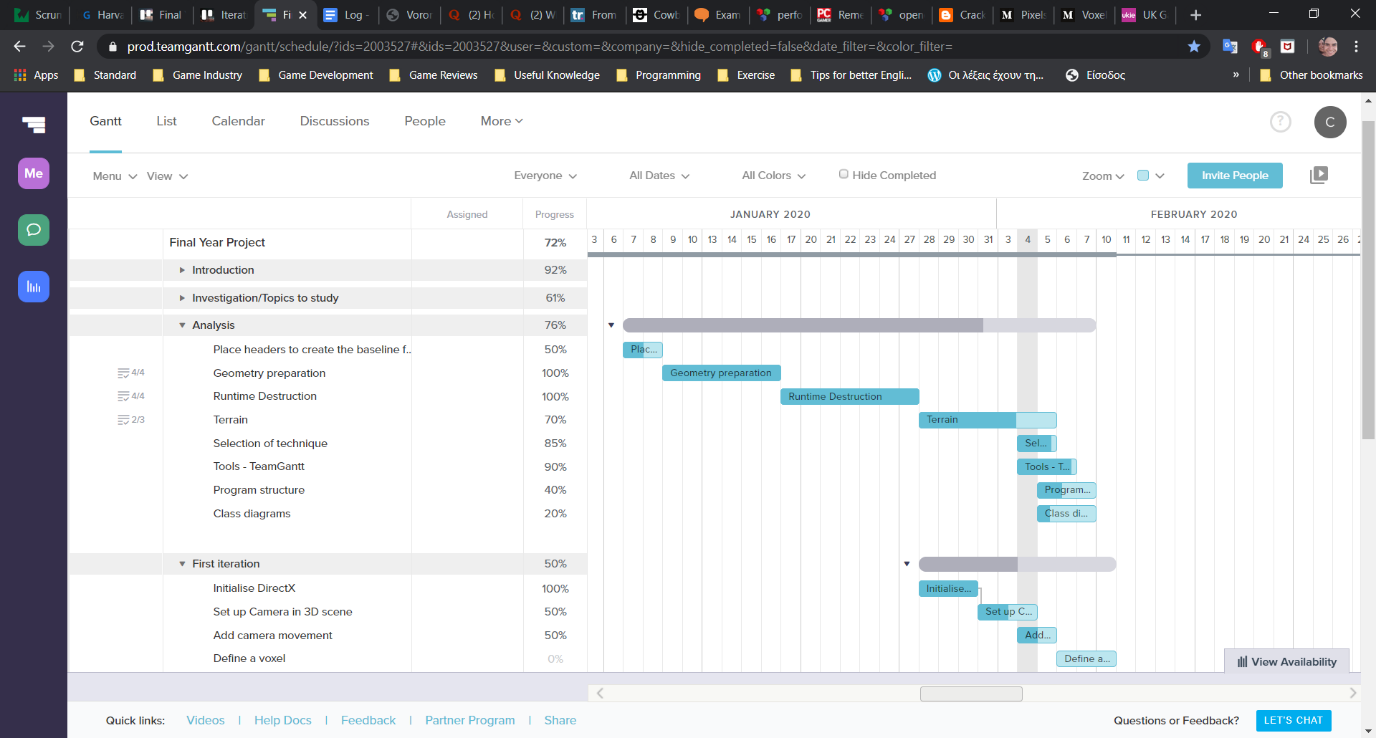


Figure: 1.2: Example of TeamGantt. Milestones are broken down into smaller tasks that are represented visually based on the date that each task is set to be worked on. Some tasks can be dependent of other tasks, this can be represented with the line starting at the end of one task and connecting to the start of the next task thus blocking it from initiating simultaneously.

# Chapter 3

# 3.1 Overview of techniques

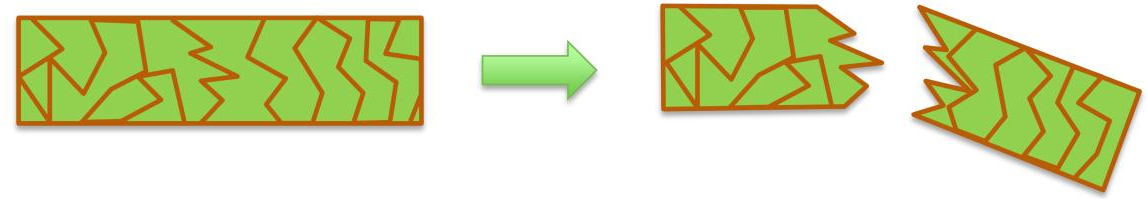
In this chapter, a brief overview of the techniques used in various games developed in the last 40 years that included destructible environments will be studied to assist with the development of a physically accurate and dynamic destroyable environment thus avoiding reinventing the wheel. It is worth noting that most modern 3D video games use the term destructible environment as a reference to destroyable objects such as any building, barrel, crate, tree or any other game object that appear in the game world, excluding terrains, distant geometry such as sky-boxes and game characters as terrain deformation was not featured. In the simplified version of 2D view however, several games included the terrain when referring to destructible environments. As we can see in games such as Scorched Earth (1991) [18], Worms (1995) [19], Broforce (2015) [20] and Noita (2019) [21]. Removal is based on pixels rather than whole objects.

The following comparisons are primarily based upon AMD physics engineer Erwin Coumans’s presentation from SIGGRAPH 2011 [23] where he spoke about current solutions and research on breakable game assets. He broke the destruction process into two categories:

* Geometry preparation during development



* Runtime destruction methods



|  |  |
| --- | --- |
| Geometry Preparation | Runtime Destruction |
| Voronoi shatter, slicing | Canned animation |
| Boolean operations | Real-Time Booleans |
| Convex decomposition | Finite Element Method |
| Delaunay tetrahedral decomposition | Jointed Breakable Objects |

**Table 1.1:** Current methods used for geometry preparation and runtime destruction as described by Erwin.

# 3.1.1 Geometry preparation

“A fracture is the separation of an object or material into two or more pieces under the action of stress.” [30]

Geometry preparation is a process that the artist is responsible for when creating destroyable objects. The artist processes 3D geometry through 3D modelling tools such as 3ds Max, Maya or Blender so that it produces fragments. The assets are fractured before are placed in the game scene or they replace their simpler counterparts during runtime, similarly to object replacement.

## Object replacement or deactivation

Swapping an object with a more damaged version of itself or removing it completely was one of the earliest approaches to simulate breakable objects in games (see Figure 1.1). In fact, this method is still being chosen and widely used to this day due to its simplicity and undemanding implementation. It depends on a set pre-created models so the precise point of collision on the model is not taken into consideration. After the object has taken damage, it is usually followed by a particle effect such as smoke or dust to hide the object replacement (see Figure 1.2). This approach provides the highest degree of artistic control but can be costly in terms of man hours. It can still produce desirable results in spite of its limitations.



**Figure 1.1**: A set of pre-made models get swapped between them based on the amount of damage the door has taken regardless the points that the bullets have landed on. This example can be found in Counter Strike: Global Offensive (Valve,2012). [24],[5]



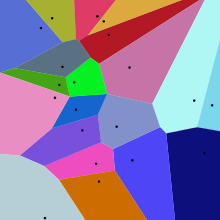
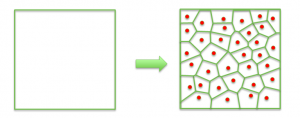
**Figure 1.2:** The crate is removed in a single frame but hidden behind particle effects such as spawned flying wooden pieces and glow. This example can be found in Crash Bandicoot N. Sane Trilogy (Vicarious Visions, 2017).

## Voronoi shattering

Cutting a 3D model into smaller parts using Voronoi slicing is a popular way of simulating solid materials such as concrete, minerals and stone. It is a sweep line method based on Voronoi diagrams. A Voronoi diagram is a method of dividing the space that processes a group of points and generates an equal number of convex regions called cells. Fortune’s algorithm is a known algorithm for generating a Voronoi diagram from a selection of randomly placed points in a plane in O (n log n) worst-case running time and O(n) space [25].

Based Hugo Ledoux’s paper [27], Voronoi shattering is produced the following way:

First the polygonal geometry is sampled to seek for a set of randomly placed points called seeds. Then, for each seed is created a group of planes that result in their matching Voronoi cells. For each of these cells, the mesh is cut causing a hollow that is filled in with newly generated faces. A collection of fragments is the result of this procedure. For this algorithm to work, the parameters need to be a closed mesh and enclosed points.



**Figure 1.3**: Voronoi Shattering, [23], [26]

## Delaunay tetrahedral decomposition

A 3D mesh can be sub-divided into a set of tetrahedral elements following the process of tetrahedralization utilizing Delaunay Triangulation. This is done by given points P that are the mesh’s vertices and a collection of points inside its volume. In order for a tetrahedral mesh DT(P) to be generated it needs to meet the following circumstance: “no point in P is inside the circumscribed sphere of any tetrahedra in DT(P)” [28]

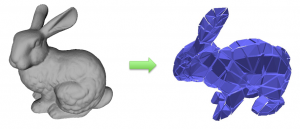
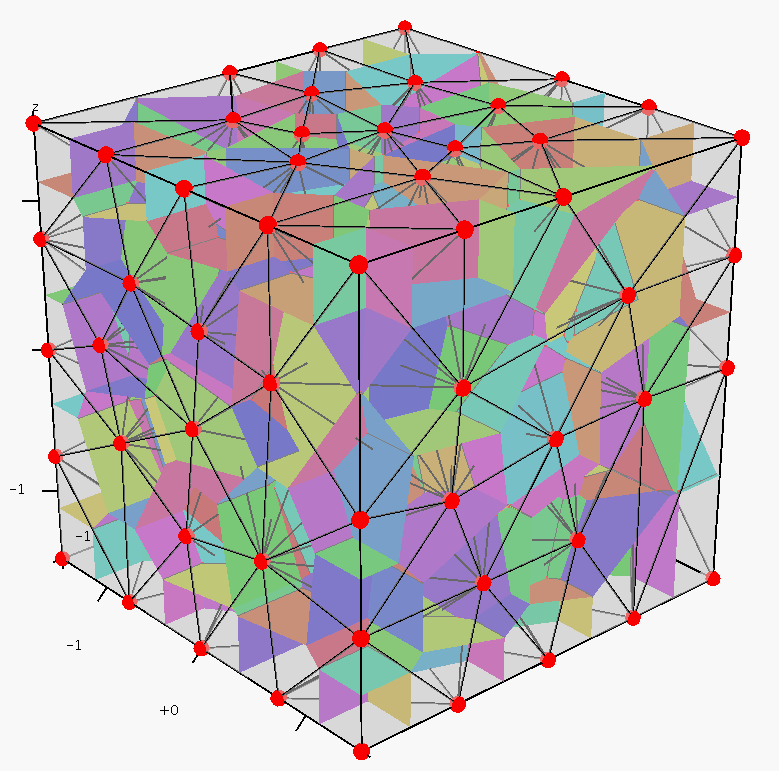
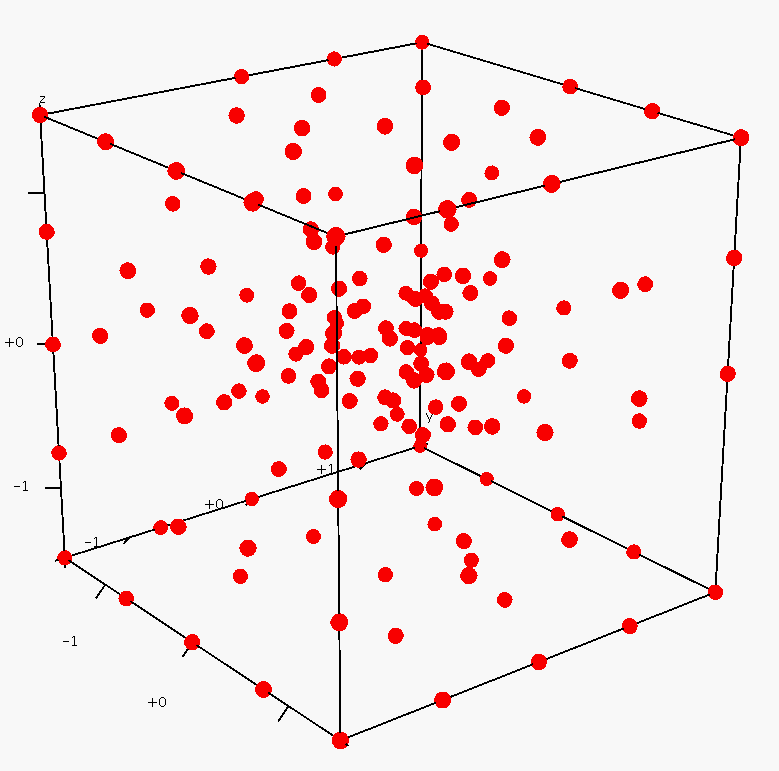


Figure 1.4: Tetrahedralization process. Closely related to Voronoi tessellation. [29][22]

## Boolean operations

Boolean operations are also known as constructive solid geometry (CSG), are used to perform volumetric operations between 3D geometry. For instance, two meshes can be added together to compute differences or intersections for mesh decomposition. Similar to a cookie cutter, it allows us for break a mesh in to smaller parts. This method is not automatic thus requires the work of an artist making it relatively slow but on the other hand provides a high degree of control.

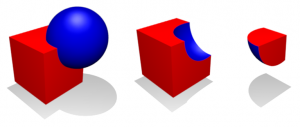


Figure 1.5 (Left) Sphere and cube are intersected (Middle) The cube after calculating the difference (Right) The remained piece.

## Convex decomposition

This method takes in a concave triangle mesh and divides it into smaller convex pieces. An artist can create a convex decomposition manually by using simple convex primitives such as boxes, spheres and capsules. It is possible of creating convex decomposition automatically, but it may involve tweaking of some parameters to get desirable results. Even Though it is an NP-hard problem, an approximate method using top down or bottom up can still be implemented. There is a free implementation made by John Ratcliff. It is a top down approach: Concave meshes are recursively broken down into parts, until each part is convex. Khaled Mammou was inspired by John’s implementation and produced a bottom up approach called Hierarchical Approximate Convex Decomposition (HACD). This approach breaks source geometry into smaller parts in physically expected way, for example, the bunny’s ears come off, instead of breaking at some arbitrary position. The number and shape of those pieces however are determined entirely by the source geometry and does not provide any artistic control.

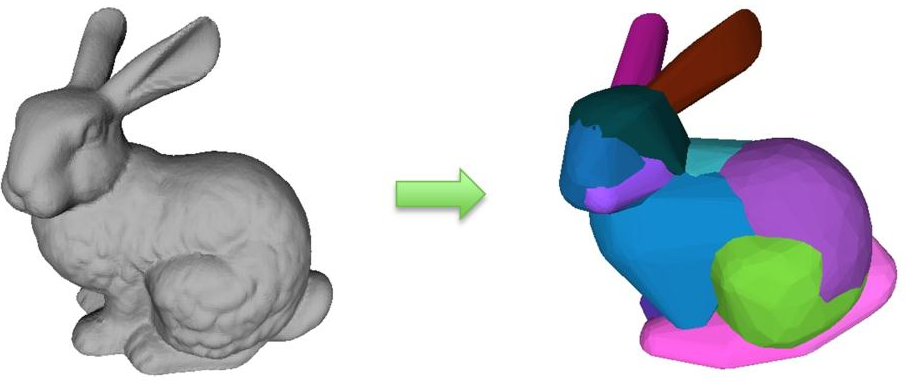


Figure 1.6:

# 3.1.2 Real time destruction methods

Once the generation of pre-fractured assets has completed, runtime breaking methods are used for destroying objects during game execution. The objects are being fractured into separate meshes as part of the simulation and the physics engine also solves the object dynamics and collisions. Current available physics libraries include Havok Physics, Bullet, PhysX, and the Open Dynamics Engine.

## Canned Animations

Game assets that are non-essential to gameplay and their simulation will not bring substantial value to the game are subjects for implementation through pre-determined animations. These animations are made in an animation tool and will play back when triggered or just play in a loop. Soft-body objects such as a waving flag or flowing water are major examples of this application. Another example is a distant skyscraper being tore down or demolished after a large explosion or earthquake. This approach is computationally cheap and saves a significant amount of processing power as the physically based animation is avoided altogether. The disadvantage of canned animations however is that the object will break the same exact way each time and not correspond to the interactions within the game.

## Real-time Boolean operations

Boolean operations surpass the need of pre-made breakable objects and allow dynamic physical mesh damage to take place during runtime. This is achieved by subtracting a pre-modelled 3D shape from the original object on the exact point of impact. Because this is happening at run-time, the system dynamically re-triangulates the original model – by adding polygons as necessary - in order to maintain the original solid shape with the additional holes or missing pieces. The interior faces take the material and textures from the Boolean shape. Prime example of this approach is the Geo-mod engine. Red Faction (2001) and Red faction 2 (2002) were created using the Geo-Mod (Geometry Modification Technology) game engine developed by Volition [6] . The engine featured the alteration of level geometry such as walls and ground by making holes in surfaces using certain weaponries. This was accomplished by creating a new “empty space” object at the point of collision. This new object would then be subtracted from the surface it just hit in real time – occluded by visual effects such as dust and explosion animations to hide the sudden subtraction of the meshes. As a result, modifying the geometry by adding a hole to it.

On one hand, the approach was unique at the time as it was not replacing an object to its damaged state rather than altering its geometry altogether. On the other hand, the feature arose some game design issues as the game could not protect the player from himself because they could tunnel through everything in the level thus making some areas “unwinnable” if enough damage was done to the surroundings. It made it more difficult for developers to set up various scenarios as it was harder to reason with the player’s unpredictable behaviour. Nevertheless, this was an important attempt to simulate a fully destructible environment in 3D regardless of the real-time constraints. Geo - mod 2 was later developed to create the next game in the Red Faction series, Red Faction: Guerrilla. This time the engine was focusing more on the destruction of buildings with a stress-based collapse model rather than the modification of level border geometry. A collection of smaller objects that are linked together with joints. Besides large or important structures, most buildings in Guerrilla are destructible and react to damage in real-time, eventually collapsing if enough structural support is lost.

## Finite Element Method (FEM)

Finite Element Method as described by Parker and O’Brien [31] is a numerical technique for approximating complex solutions in engineering. It is widely used in the real-world analysis of determining what will happen to a structure when hit by another object or blown up. FEM was folded in the digital molecular matter or DMM library. Its licence software was initially used for real-time destruction in video games. It was then advanced and upgraded for use in motion pictures through computer generated images [32].

It simulates the behaviour of a physical object which can be represented by a mesh made from smaller discrete pieces called finite elements. These elements contain material and structural qualities that determine the object’s reaction to certain physical circumstances. It enables the developer to define the mesh’s physical state by assigning different properties to each element to calculate its stress or temperature. Applications can be found in fluid dynamics; brittle fractures [33]; elasticity; heat transfer and other physical properties to simulate deformation, stress and fracturing.

## Jointed breakable objects

Jointed breakable objects are pre-broken objects that use breakable constraints to hold the pieces together. The joints break when a certain amount of force such as a bullet is applied to it and the pieces start to fall by gravity. This method has a big impact on draw calls, memory consumption and physics calculations. It is also usually followed by a particle effect based on the surface type of the object. A clever way to work with it, is to use as few pieces as possible by spawning more pieces when combined with the particle effect.

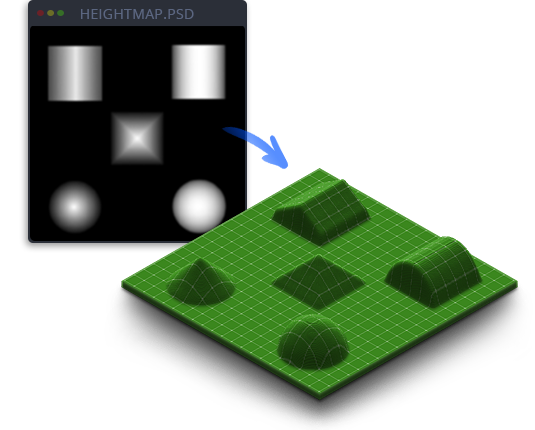
# 3.1.3 Terrain

There are two ways of generating terrain: One is to have a very fine mesh that can store sufficient information that allows the granularity of the resolution to be able to be replicated. However, it is unlikely that large amount of information will be able to be stored in a way that allows it to be processed in an optimal way and that allows time for other processes to be executed as majority of processing time is being spent on rendering the terrain itself. Some techniques can be applied to take a subset of the points and then interpolate between those points to produce features that can correspond to the sorts of ones that are expected to be found in terrains of the specific nature. That then offsets some of the needs to store information with the ability to be able to go through techniques that produce those dynamically as they are required. Hence, mathematical algorithms can be used for generating those as needed. That means that they do not necessarily have to store large amounts of data but store the routines and the production rules that enable to generate those features.

Terrain is important to a lot of games, as a model is very large so creating every point explicitly by hand is not feasible. What follows is an overview of some automated terrain generation techniques.

## Heightmaps

A heightmap or height field is a 2D grayscale raster image used for storing terrain elevation data. A heightmap is a matrix, where each element represents the distance of displacement from the terrain grid. It is used to describe hills and valleys and as a grayscale image, black denotes the smallest height whereas white denotes the largest height and shades of grey signify in-between heights. Only the height of a specified vertex on the triangle grid is provided by the heightmap thus making it impossible to create overhangs, caves or tunnels. [43]



**Figure 1.7**: An example of a heightmap (greyscale image) and its use on flat grid. [42]

Usually a byte of memory is allocated for each heightmap element when stored on disk. Thus, the height value ranges from 0 to 255. When the range of values is not enough when matching the scale of a 3D world, it is possible to scale out of that range by allocating a float for each height element.

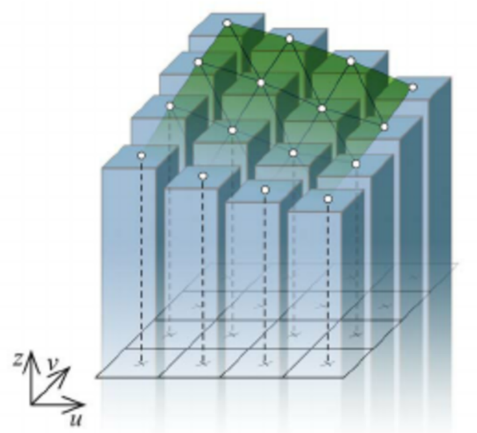


Figure 1.7: Representation of terrain as a heightmap from Holz et al [41]. In his paper, he analyses his implementation of height fields that can be updated dynamically.

## Voxels

The word voxel derives from the combinations of the words “volume” and “element” and is the 3D equivalent of a picture element or pixel. Voxels are constructed from 3D regular hexahedrons or cubes that represent a single data point on a three-dimensional regular grid. A voxel is the smallest volume when dividing 3D space into discrete uniform regions [44]. Each volumetric pixel is defined by a position in 3D coordinates, the colour at that location or density as used medically for CT and RMI scan data images. This enables for the generation of complex terrain geometry that contains features not possible with heightmaps such as caves and overhangs [36].

## Marching cubes

The marching cubes algorithm was the result of the research that William E. Lorensen and Harvey E. Cline conducted for finding an efficient way of visualising data from CT and MRI scans for General Electric [37].

An implicit function in the form of f (x, y, z) = 0 takes in a point in space and gives out a value. The function then samples points in regular intervals inside a specified region. Some points represent empty space and some other points may be positioned inside or on the surface of some shape. The goal of the marching cubes algorithm is to construct the surface of that shape from triangles so it can be displayed as a mesh. In computer games is used to smooth out terrain generation made up of voxels by converting to polygons before rendering.

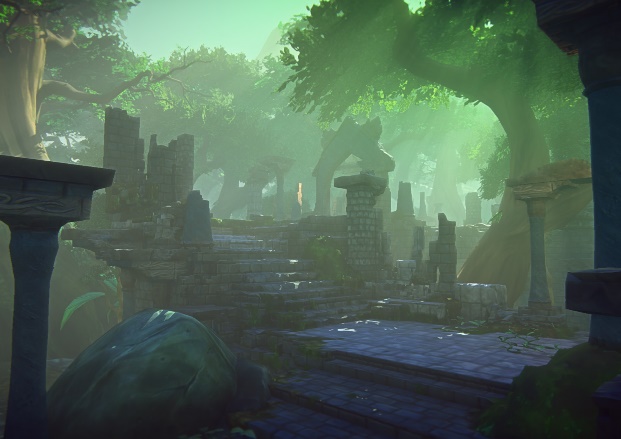


Figure 1.7: One game example that manage to combine voxels and marching cubes for the terrain and polygon meshes for the characters was Ever Quest Landmark.[40]

Ben Anderson [38] described it as follows:

Provided an object, in order to test if an arbitrary point is within or on the surface of the object or bounds within the area which the object exists:

1. Division of the space within the region of interest into an arbitrary number of cubes.
2. Testing of the vertices of every cube for whether they are positioned inside the object. For each cube where some vertices are inside and some vertices are outside the object, the surface must pass through that voxel, intersecting the edges of the voxel in between vertices of opposite division.
3. Drawing a face inside each voxel connecting these intersections.
4. Source object is drawn.

To simplify the problem, we can break it down to a single cube. There are 8 vertices, and each can be inside and outside of the shape which results to 28 = 256 different configurations (see figure 1.8).

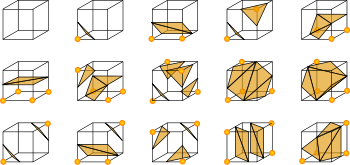


Figure 1.8: There are only 14 unique cases, the rest are just symmetries of those. []



1. (b)

Figure 1.9: For example, one point that is within the shape, a single triangle gets generated as seen in (a). If the point on its left is also inside the shape, a rectangle comprised of two triangles is also generated as seen in (b).

# Selection of technique

Taking into consideration the investigation conducted on destructible environments and the available time allocated for implementing a student project, one approach has been selected to be employed and tested in a game environment. The desirable objective is to generate a destructible terrain by producing a simple C++ game engine built upon a voxel-based design.

Several challenges arise with this approach as voxel objects take O(n3) memory due to their cubic nature. In order to achieve efficient manipulation of voxels, solutions to problems involving effective use of cache memory, minimisation of polygon count and efficient communication between the CPU and the GPU need to be addressed.

The phased implementation is broken down into discrete iterations. At the end of each iteration there will be a test plan, test result and iteration conclusion included:

## Outline of iterations

Iteration 1 – Program structure and design:

In this iteration a suitable application programmable interface (API) is chosen, a product back log is generated subsequently prioritising the desirable elements using the MoSCoW technique. Class diagrams will the be used to assist on directing the program’s structure.

Iteration 2 –

Iteration 3 –

Iteration 4 –

Iteration 5 –

Iteration 6 –

# Chapter 4

# implementation

# Iteration 1 – Program structure and design

Description of the iteration

## Selection of Application Programmable Interface (API)

There are many APIs available but due to author’s past experience the one that is going to be used for this project is DirectX, version 11. According to Frank D. Luna, Direct 3D is a low – level rendering library used for writing high performance 3D graphics applications. Its API closely models the underlying graphics hardware it controls.

Product backlog, MoSCoW

Class diagrams

Test plan

Test results

Iteration conclusion

* Review
* Retrospective
* Trello board
* Team Gantt