VOXEL RENDERING FOR DESTRUCTIBLE MODELS

by

Aiden Knight

A Dissertation

GDEV60001 Games Development Project

Submitted to the

School of Digital, Technology, Innovation and Business

University of Staffordshire

In Partial Fulfilment of the Requirements

For the Degree of Bachelor of Science

February 2025

SUPERVISOR: Benjamin Williams
SECOND SUPERVISOR: Bradley Davis

Abstract

Destructibility in games can suffer from being less dynamic, as the way a model will be destroyed is often pre-generated. This can be inflexible when aiming to fracture the model and then have subsequent fractures of the broken parts. It may also not fully support breakage from different positions and with varying strengths. To solve this issue voxels were investigated for their ability to support dynamic destruction. Specifically, the ability to re-use existing 3D models and the skills to create them was explored through the ability to voxelise a given model into many voxels. In addition, an artefact was developed using Vulkan and C++ to attempt to optimally render and simulate voxelised models. From a user perspective, this would allow an inputted 3D model to be voxelised at a specified resolution and then have its destruction simulated with dynamic explosions. By investigating the performance of the developed artefact, it was discovered that the performance of simulation is the main bottleneck when it comes to using voxels as a destructibility primitive. This drew the conclusion that voxels are not yet in the place they need to be for use as a primitive for simulating destruction in games even though the naïve rendering performance proved promising.

Nomenclature

API – Application Programming Interface

FPS – Frames per second

GPU – Graphics processing unit

RAM – Random Access Memory

SVO – Sparse Voxel Octree

Voxel – A three-dimensional counterpart to a pixel

2D – Two dimensions or two dimensional

3D – Three dimensions or three dimensional

Table of Contents

Introduction

From simple effects to full simulation, many games have featured destruction to give feedback to the player. Simple effects like an explosion when an enemy is destroyed can be seen as early as Space Invaders (Taito, 1978). However, with the improvement of hardware and increase in demand for games, there are many games that showcase more in-depth destruction, even using destruction as a core mechanic of the game. For example, the game Teardown (Tuxedo Labs, 2022) has fully destructible environments that are crucial to its gameplay, powered by voxels.

Although there are games with fully simulated and rendered destructible models, they often require pre-processing 3D models when placing them in a simulated destructible environment. One of the simplest methods (Hoss and Emma, 2012), is manually creating the destructed version of a model and swapping that version in when destruction occurs. This works as a naïve approach but has the issue of requiring manual creation of the destroyed model.

Various methods for producing destruction models already exist to solve this issue but might fall short of the freedom that voxels give when simulating destruction, as voxels could be used to break models into much smaller parts, allowing players to destroy parts of the model with higher precision. Most approaches that calculate the way a model will destruct ahead of time may not be usable in a real-time fashion to further break apart the model into smaller parts, whereas this would be more trivial when considering voxels. It is also possible that they lack the flexibility to sufficiently simulate destruction, for example when simulating explosions, they may not be able to properly handle the varying position and strength of the explosion.

To give the player the extra freedom that could be gained from rendering and simulating the destruction of models and environments using voxels, it is key that there exist ways to create these models and environments. As there are already many people trained in creation of typical 3D assets, alongside many existing models and environments, a generalised method to convert these into a voxel format, then render and simulate them

optimally, would allow teams looking to create a game with high precision destructible models and environments to focus on other parts of the game's creation.

There is not a wide selection of tools that achieve this effect, and the methods to both render and simulate the result is not necessarily related to the outcome of such a tool. A method to create a program that would allow users to transform their models into a destructible format then test the destruction in an optimised environment backed up by research, which could be directly integrated into their game, would be beneficial to the future development of games that focus on high precision destructibility over graphical fidelity as they would still be able to use their usual pipeline for 3D assets by using voxelisation.

Aims and Objectives

This dissertation aims to firstly review the current state of techniques for destructibility in games, reviewing literature on converting, rendering, and simulating models in a voxel form. Through the review these methods will be compared to determine their suitability for scalable, performant voxel model destructibility in games to determine whether voxels can be used as a destruction primitive. Specifically, the following research questions (RQ) will be answered:

- (RQ1) Which areas are currently underexplored in using voxels as a primitive for real-time destructibility?
- (RQ2) What current methods for converting 3D models into a voxel format exist?
- (RQ3) Is it possible to simultaneously render and simulate voxelised models in real-time using a typical desktop personal computer?

Following a thorough review of related works, an approach will be discussed to measure the performance of rendering and simulating the real-time destructibility of voxelised 3D models to determine the suitability of voxels as a destruction primitive. Real-time will be assessed as an average simulation of at least 60 frames per second (FPS). In particular, the scalability and precision will be evaluated to determine the suitability of voxels as a destruction primitive. Precision being number of voxels a model can be correctly split into, and scalability assessing the number of voxels able to be rendered and simulated in real-time.

Literature Review

Destructibility Methods

Destructibility in 3D has uses across computing, whether for medical carving (Williams, 2009), animated movies (Tollec et al., 2020), simulating fractures in materials, or in video games. For video games more care must be taken when selecting the method used (L'Heureux, 2016) as there are many other systems that must operate around the destructibility systems, and the requirement for real-time simulation means more hurdles to overcome.

When trying to overcome these hurdles it is good to know that given a 3D model there are many ways to simulate destructibility. For the most accurate simulation of model destructibility, offline methods may be preferable, however, longer computation times are required (Workman, 2006), meaning they are limited in the areas they are applicable. Real-time applications like games require solutions that can be simulated at significant speed, with expectations of at least 60 fps expected for modern games.

The work that Workman (2006) conducts also suggests that offline methods used to pre-calculate the way a model will destruct could be used to create patterns that could be later used in a real-time application. These would require less work compared to that of creating a separate destruction mesh that gets put into the original model's place (see Figure 1).

Performance of such pre-calculated methods is still important (Forslund, 2023) as it would speed up the workflow of designers aiming to iteratively design the destructibility of a level, especially in cases where there a multitude of parameters that may need to be fine-tuned.

Figure 1: A crate's mesh (left) and its destruction mesh (right) (Hoss, R. and Emma, T., 2012).

Datasets

A similar alternative to using pre-calculated methods for simulating the destruction of models, are datasets that have been prefilled with a variety of pre-fractured models. However, they are restrictive due to being limited to the models and fractures they offer. Sellán et al. (2022) constructed a dataset of models collected from normal model datasets and applied a fracturing algorithm to them which could be directly used in a real-time application. This would be suitable where realism is not of foremost importance, however, should a more faithful representation of real-world fractures be desired Lamb et al. (2023) present a dataset of 3D scans of objects in both their fractured and whole counterparts. The main limitation of datasets being the potential lack of the relevant model, in which case algorithms for generating the destruction may be desired.

Physically Based Methods

Methods that often require pre-calculation due to their complexity are various physically based modelling methods. These methods are used quite often in simulating destructibility which could be due to their accuracy and the breadth of relevant research

done in material sciences. An advantage to these methods is that they can also be used to simulate deformation (Nealen et al., 2006) alongside fracturing, as shown in Figure 2. They also associate well with physics simulation due to their usage of forces. However, it is commonly the case that the calculation of how a mesh fractures are unfortunately not a real-time solution.

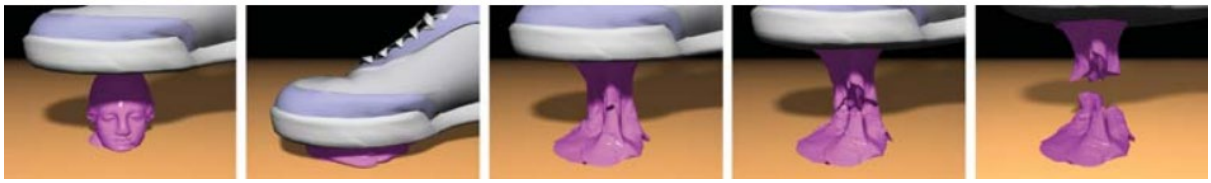

Figure 2: Highly plastic deformations and ductile fracture (Nealen et al., 2006).

Finite Element Methods

One physically based method is the Finite Element Method (FEM) which discretises models as tetrahedral meshes (Morris, 2010), allowing the propagation of strain over neighbouring elements to simulate brittle fractures.

Parker and O'Brien (2009) describe a more dynamic destruction method that also relies on manual artistic input which uses such a method. They achieve a more accurate simulation by fracturing objects when the strain energy exceeds a threshold. The artistic input was required when manually creating the appearance of the broken parts. These broken parts of the object were dubbed splinters and were used to ensure the detail of the object was preserved upon fracturing.

Decoupling the simulation mesh from the crack surface as in the extended finite element method (XFEM) is improved on by Chitalu et al. (2020) to efficiently simulate brittle fracture at a high resolution without having to re-mesh. In a single-threaded and unoptimized implementation of this, the simulation times were recorded as being in the order of seconds. Consequently, the performance of this method would require much more optimisation to be suitable for real-time applications.

For a scenario where less fractures per element are required, Mandal et al. (2023) present a graph-based FEM that can achieve real-time fracture rates for lower resolution meshes. However, the authors do note that their method is limited to fracturing the tetrahedral elements into a maximum of four parts.

To improve the performance of these, or similar, methods (Morris, 2010) when there are no other graphics processing unit (GPU) bottlenecks, the implementations could port their code over to the GPU. This would allow them to take advantage of the parallelism that it provides when performing the same operation to more performantly simulate fracture of destructible models.

Bonded Discrete Element Methods

Such a strategy that leverages the power of the GPU is the Bonded Discrete Element Method (BDEM) as described by Lu et al. (2022). BDEM represents solid materials as a collection of densely packed elements, attempting to copy the micro-structure of the materials by utilising nodes and bonds. Lu et al. (2022) implements it in a more efficient way. They then used various settings to simulate different fragmentation processes, achieved by breaking of bonds between elements. They did find a high computational cost but note that further work, due to the good scaling consistency of BDEM, could overcome this limitation.

In a follow-up study presented by Lu et al. (2023) simulation speeds magnitudes faster than other state-of-the-art methods were achieved. However, the speed of their simulation was still far off milliseconds to compute, suggesting that BDEM still requires further innovation to be computationally fast enough for real-time simulation of destructibility and fracture.

Material Point Methods

Similarly to BDEMs, Material Point Methods (MPM) are another slower numerical way to simulate accurately a variety of materials. MPM models an object not as a mesh but as a continuum body which is an amount of small material points. This leads to more accurate simulations of certain materials, but Fan et al. (2022) used MPM to simulate damage to brittle objects to then simulate fracturing (see Figure 3), but this took hours to fully simulate which they compare as a reasonable time for an MPM.

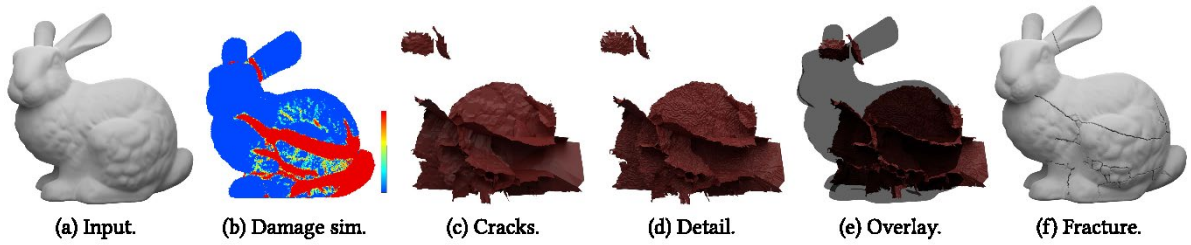

Figure 3: Visual summary of stages for simulating fracture using MPM with the method by Fan et al. (2022).

Boundary Element Methods

Assuming fractures are being pre-calculated, a method that can allow easier iteration is the Boundary Element Method (BEM). Hahn and Wojtan (2016) developed a fast approximation method for BEM brittle fractures, for this purpose of faster iteration times. BEM focuses on the surfaces of 3D objects to avoid the complications from volumetric meshing operations and Hahn and Wojtan's (2016) estimator produces results reasonably close to a full BEM implementation, which could allow better iteration in the development of a pre-calculated fracture for a real-time game or simulation.

Assuming there is sufficient training time available, utilising a machine learning method (Huang and Kanai, 2023) may be more suitable. Once trained they can create close to realistic fractures in seconds rather than minutes. However, these methods would still require extra work to operate as a pre-calculated method for real-time and require testing in more complex game environments where there are dynamic elements involved.

Voronoi

A technique more often used in pre-calculated destructibility methods for real-time applications is Voronoi decomposition, which is a technique that can be used to split up space based on a finite number of points as can be seen in Figure 4. This technique can be used to fracture meshes, with meshes being split along the boundaries of the cells created from Voronoi decomposition. Grönberg (2017) achieved a near real-time approach utilising such an implementation and Müller et al. (2013) managed to implement a real-time and more dynamic approach to destruction, where the impact location affects how a model will fracture. This method still relied on precomputing fracturing but leverages a fracture algorithm to change how a model will fracture. Similarly to other methods that pre-calculate, they found that it would suffer from having to recompute should any change occur to the model. Also, they found that Voronoi methods are not always as physically accurate as other destructibility methods.

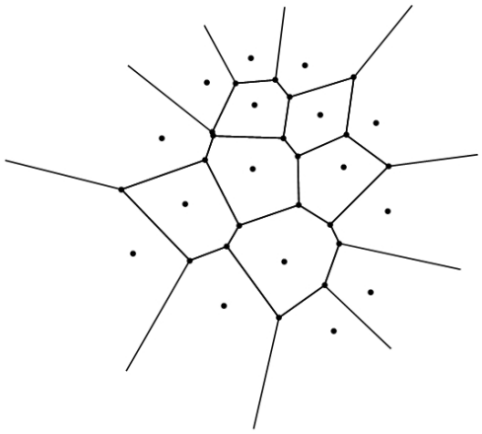

Figure 4: A two-dimensional Voronoi diagram (Grönberg, 2017).

To gain more realism over Voronoi, Sellán et al. (2023) developed a new method to pre-calculate fracture patterns for models that consider the weak points of a model. They use a method to compute a model's fracture modes that can then be used to simulate destructibility based on different impacts in real-time based on where a model is geometrically weak. However, as a pre-computed approach, they still found that they could not simulate fractures past the first due to the computational complexity. This

means their approach would not be as dynamic, especially for larger models which could desire further fracturing.

This lack of dynamism was noticed by the answers to a questionnaire that Thomas and Zhang (2023) received, where the real-time fracturing and thus ability to fracture again after the first fracture felt more realistic to the participants. Some of these realism issues could be resolved by storing more possible fractures to choose from in the pre-fractured case, however, these limitations would need to be properly explored.

McGraw (2024) achieves such a dynamic implementation of destructibility through voxelising a mesh at a low resolution and using 8 position-based dynamics particles to represent the mass, position and velocity of the voxel. The voxels are then utilised as an animation cage for the mesh. Through having face-to-face constraints between the voxels, this allows destruction in a variety of forms like fractured into voxels, peeled into sheets, and shredded into strips.

Voxelisation Methods

An important step to achieve similar results is the method in which an inputted 3D mesh is converted into a voxelised form. Voxel space is a representation of 3D Euclidean space discretised as voxels at fixed intervals, often represented in a binary form (Berg et al., 2021) where a voxel either exists or does not exist. There are also slight variations on how the voxels are stored with some using a metric for how inside an object a voxel is (Karabassi et al., 1999) to allow for smoothing of the object. Other implementations may store data such as surface normals for lighting calculations (Young and Krishnamurthy, 2018), material properties for better rendering (Zhang et al., 2018), or potentially more attributes depending on the requirements of the application. However, it is worth noting that there are extra memory requirements when storing this extra information.

Alongside the method of storage, when selecting a voxelisation algorithm there are a few considerations that must be taken depending on the needs of the application. Simulations (Aleksandrov et al., 2021) may have to consider the separation and connectivity of voxels to determine, for example, where fluid can and cannot flow. The connectivity of a voxel (Huang et al., 1998) is commonly represented with a prefix that communicates what part of the voxels are connected. Therefore 6-adjacent voxels are those sharing a face and 18-adjacent voxels share either face or edge. Finally, 26-adjacent voxels may share a face, edge or corner, forming the full neighbourhood that a voxel could have.

Given this adjacency, the connectivity of a voxelisation is the maximum adjacency held by all voxels. If any voxel on the voxelisation is only connected to another voxel with 18-adjacency, the voxelisation cannot be 6-connected. Then separation is the lack of connectivity of any possible sets of empty voxels that pass through the voxelised model (Cohen-Or and Kaufman, 1995). For example, a voxelisation would be 6-separating if no 6-connected path of empty voxels could separate parts of the voxelisation. However, it could still be separating of a higher order.

Depth Buffer

Potentially one of the conceptually simplest methods for voxelising a 3D mesh is by using the depth buffer. The method has been around for a long time (Karabassi et al., 1999) and still sees some use (Clothier, 2017). The way it works is by rendering the input mesh from three different pairs of perspectives, each pair usually being from opposite sides of an axis. Then the opposing depth values can be used to determine whether a voxel falls inside of the mesh. A significant drawback of this method is the requirement to render the mesh multiple times to obtain the depth values, however, this is not as prudent when considering a tool that pre-creates a voxelised mesh because it would not require the same level of computational efficiency. For such a use, the considerable issue would be

the inability to handle meshes with complex inner geometry, as the pairs of depth values would not be able to capture the internal geometry.

Rasterisation

Some methods for voxelisation (Eisemann and Décoret, 2006) may not provide full coverage and miss voxels, which can lead to issues when features such as accurate collision detection is desired. To combat this some implementations, focus on ensuring every voxel intersecting the input model is correctly detected, often called conservative voxelisation. For example, Zhang et al. (2007) implement such a method based on the rasterisation family of voxelisation techniques, except they calculate a more exact intersection between the rasterised triangle and voxel region by computing a depth range alongside the projected intersection of the triangle, as shown in Figure 5. The depth range is enlarged by a small value to combat floating point errors, together this correctly obtains a full voxelisation coverage. The main downside to their method being that the voxelisation may contain more voxels than necessary.

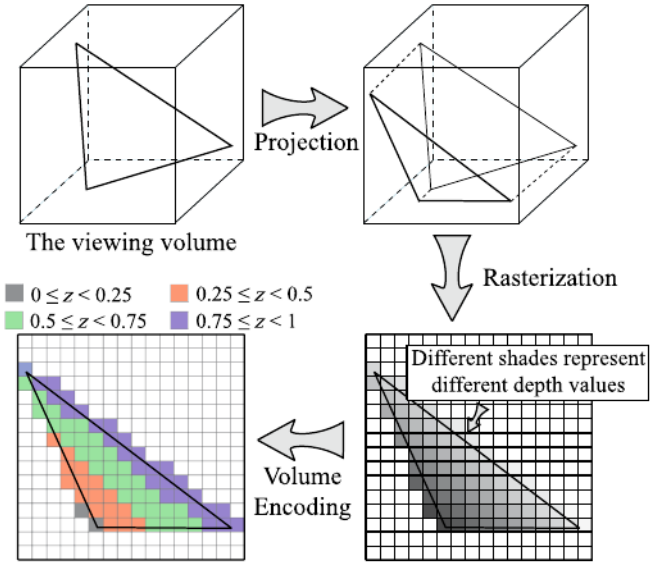

Figure 5: Example of the rasterisation voxelisation pipeline by Zhang et al. (2007)

Young and Krishnamurthy (2018) developed a multi-level voxelisation where they first rasterise slices of the model in a direction and use a stencil buffer method to identify

inside voxels. In this first level they also classify which voxels are boundary voxels using a box-triangle overlap test. Then in the finer pass of their voxelisation, they voxelise the boundary voxels at a finer detail by using a ray-triangle intersection test in a clipped slice, to identify inside voxels at the finer level, with odd number of intersections meaning inside voxels, leveraging their created stencil buffer to ensure the intersection count is accurate. Similar to earlier methods, they found that if they include the boundary voxels, they get an overly conservative voxelisation, however, they also gave the option to only include the inner voxels, which gave them an underestimate of the voxelisation.

There are rasterisation methods that do not focus on conservative voxelisation such as the method implemented by Fei et al. (2012) which tessellates triangles in the geometry shader based on voxel size, then in the rasterisation stage the fragment position converts into the occupancy of the voxels. Something interesting they did in their method was using the displacement map of the mesh to voxelise low poly meshes better. This let them potentially obtain a better voxelisation at higher speeds by voxelising low poly meshes instead of high detailed ones.

The geometry shader is also used by Crassin and Green (2012) to project triangles based on their dominant axis. Then sets up a viewport as a slice of the voxel grid to rasterise a slice of the model so that fragments correspond to voxels based off their position and depth values. Before reaching the fragment shader though, the triangles are enlarged and a bounding box created, meaning in the fragment shader the box can be used to clip the enlarged triangle to get conservative voxelisation through the bounding polygon of the triangle. The advantage of this method is the efficiency of storage, as the voxelisation is stored in a sparse voxel octree (SVO) to ignore the large amount of empty space. The octree is created top-down by flagging nodes based on contents of the voxel-fragment list created in voxelisation, then fills in the leaf nodes of the octree with the relevant voxels before creating a bottom-up mipmap of the voxelisation. This is all done to better handle large and complex objects.

This fragment-parallel approach to voxelisation was found to exhibit poor performance with many small triangles (Rauwendaal, 2012) which led to the consideration of a hybrid

approach to voxelisation involving classifying triangles based on their size to determine whether a rasterisation approach should be used. For large triangles a rasterisation approach like the previous one mentioned would be used, and for smaller triangles a triangle-parallel approach would be used. The approach for triangle-parallel used by Rauwendaal (2012) was the box-triangle overlap test.

Box-Triangle Overlap Test

Usually the box-triangle overlap test involves forming a bounding box around the triangle to first identify all possible voxels the triangle might be able to intersect, then the method's chosen overlap test is performed to identify then set the specific voxels that the triangle intersects (see Figure 6). For example, Faieghi et al. (2018) check for each of these voxels whether the triangle's projection and the voxel's projection onto certain planes overlap, giving a faster version of the separating axis theorem test.

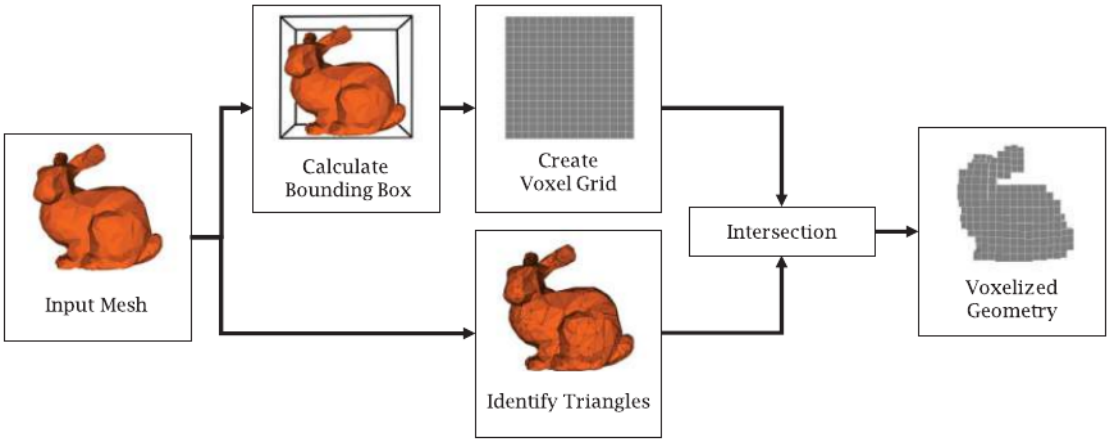

Figure 6: Example of the pipeline to voxelise using Box-Triangle Overlap Test (Faieghi et al., 2018)

Schwarz and Seidel (2010) give a similar approach but give a custom box-triangle overlap test which uses the triangle's plane to test for overlap. This has the benefit of them being able to change different plane offsets in their voxelisation to obtain either a 26-separating conservative voxelisation or a 6-separating thin voxelisation depending on the voxelisation requirements. The other benefit given by this approach is that they describe

a method to also perform a solid voxelisation given the surface voxelisation they previously create. To achieve this, they voxelise the surface then all voxels found past the surface are set as solid with both triangle-parallel approach and tile-based method given to achieve this. However, like the previously mentioned method, they also gave a method to voxelise into a SVO by instead voxelising the surface into the octree, which is slower but fills octree directly. Then a propagation method is used to set the voxels inside of the model in the SVO, notably this means large blocks of set voxels can be stored higher up in the octree to use less memory.

A similar approach to using the box-triangle overlap test is done by Shukla et al. (2022) to be used for moving geometries in fluid dynamics, but once all voxels in the bounding box of the triangle have been found, they instead use the triangle's normal and dot product it with the vector to the voxel's centre to determine whether the voxel is inside or outside of the model. Given this they then provide a flood fill strategy to form a solid voxelisation. The method was later enhanced by Kumar et al. (2024) to add error correction so that it could be used for deformable moving geometries.

Slightly different to the other box-triangle overlap methods, Bergs et al. (2021) first represent the mesh to be voxelised as an R-Tree, then using recursively creates an octree by testing against the R-Tree. Whether the octree subdivides is based on two different tests, the first test is if the number of possible mesh faces in the R-Tree is above a threshold the octree will fully subdivide, then if under this threshold the subdivision is based on a triangle-box overlap test (Akenine-Möller, 2005) against the mesh faces. This continues until a stop depth, or minimum voxel size is reached. Given this surface voxelisation in an octree they also provide a ray casting method to obtain a SVO solid voxelisation by counting intersections as the ray travels.

Ray Casting

This approach for solid voxelisation is quite common, as it is quite simple to use intersection counting to determine whether a voxel is inside or outside of the mesh. Li et al. (2023) focus on internal voxelisation and describe how a surface voxelisation could

be edited to produce a map between surface voxels and triangles to later be used for internal voxelisation. Then the maximum and minimum z values are obtained by from the surface voxels to create a 2D-depth buffer. With these stored values a form of ray cast that traverse the voxels between the two values and counts intersections to flag the internal voxels. This is stored in a SVO like other methods to store the surface in higher fidelity.

More recently, ray casting has been used to directly voxelise (Nourian and Azadi, 2024) by casting multiple rays into the mesh to be voxelised and sampling the boundary of the mesh by counting intersections of the rays. This represents the mesh's boundary as a point cloud which is later voxelised by generating the Morton code of the point's position as a unique index to set voxels.

Scanlines

An approach usually more versatile are methods that utilise scanlines and line voxelisation to voxelise a model. When considering triangle-parallel approaches, scanline methods are usually the preferred choice (Zhang et al., 2018a). Methods that generate a bounding box for each triangle usually perform more unnecessary checks that can be avoided through a scanline approach. The main consideration to be taken when performing scanline voxelisation, is determining the scanline distance to be used on the given triangle.

Zhang et al. (2018a) developed a scanline voxelisation approach that splits up and projects triangles into 2D as they found that no optimal scanline distance exists in 3D. To achieve this, for each triangle they first voxelise each vertex before going on voxelise the edges using a line voxelisation method. Finally, the inside of the triangle inside is voxelised by splitting it up in the Z direction and projecting it into 2D. This allows an optimal scanline distance to be calculated for each segment, so the scanlines can be voxelised with line voxelisation. The line voxelisation used is the choice between real line

voxelisation, super-cover line voxelisation and integer line voxelisation, a line voxelisation used to combat floating point error. The super-cover version can capture more voxels by capturing the voxels that would be missed by any singular points in the line segment, guaranteeing a conservative voxelisation. To fully support the integer line voxelisation they also provide an integer counterpart to their scanline voxelisation which they found to perform similarly but with small error over the floating-point approaches.

Building upon this Delgado Díez et al. (2024) develop a scanline voxelisation technique that ensures voxels are only ever visited once during voxelisation, and that develops a gap detection method so that only a single scanline is required in each 2D step. They achieve this by first sorting the vertices of a triangle along the axis that is both worst aligned with the triangle's normal and has the lowest magnitude in the triangle's normal vector in absolute value, which they call the primary axis. Then the advance direction of the scanline is found by projecting the primary axis onto the triangle's plane, getting a flat advance direction by removing the primary axis component from the advance direction.

Parallel scanlines are then formed equidistantly from the lowest vertex (in terms of primary axis) and progressing along the edge towards the highest vertex. These scanlines are line voxelised with a custom line voxelisation that prefers voxels behind the scanline in terms of flat advance direction. With their gap detection method checking if the neighbour of the voxel in the flat advance direction is close enough to be missed by the next scanline to set it now if needed. Finally, the edges of the triangle are line voxelised as the voxels at the start and end of the scanlines would be missed otherwise. The whole pipeline is outlined in Figure 7.

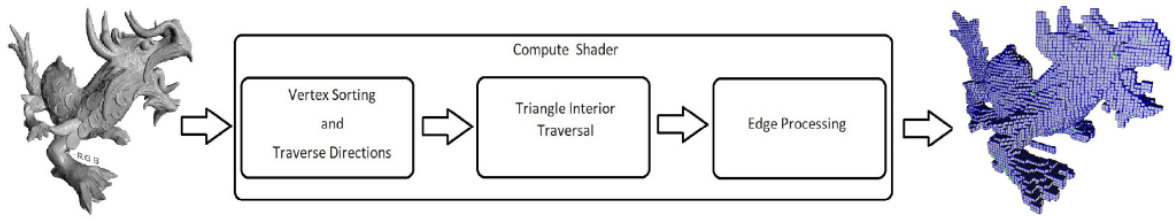

Figure 7: Scanline Voxelisation pipeline in use by Delgado Díez et al. (2024).

Point-in-Tetrahedral Test

A similar approach to scanline is the voxelisation method of Ogayar-Anguita et al. (2020) which uses slicing planes to voxelise tetrahedra, its foundations found in the point-in-tetrahedral test. Like most point-in-tetrahedral test methods, the first step involves a method for generating a tetrahedral mesh, which this method does by ensuring the model is triangulated then creates a tetrahedron from each triangle by treating the fourth point as the centre of the mesh. Then slicing planes get points of intersection with each tetrahedron, rasterising the triangles formed by these points then setting the voxels covered by the triangles. One benefit of this method is that it directly calculates the solid voxelisation of the model, however, it can be filtered by adjacency to empty voxels to still obtain the surface voxelisation.

There are also methods that directly use the point-in-tetrahedral test like the method proposed by Chen et al. (2021). Their method adapts an existing tetrahedral mesh generation method that requires a user defined error tolerance value which they found that, even at higher values than the original method, it does not produce a big difference in the voxelisation. Then for each voxel falling in the bounding box of the tetrahedral they apply a fast point-in-tetrahedral test to determine whether the voxel should be set. The main benefit of their method over others is that they can support non-manifold, non-watertight models due to their adapted tetrahedral mesh generation.

Differentiable

A newer method for voxelisation that also supports non-watertight meshes can be found in the method proposed by Luo et al. (2024). They use the concept of winding number from topology which can be calculated to determine the number of times a surface winds around a point. Winding number was found to not be directly usable due to the sharp edges and irregular triangles found in meshes, so instead the method calculates solid angles to determine the occupancy of points, using a modification of arctan which lets

the occupancy be almost binary for points, with points on the surface of the mesh having occupancy of one and zero otherwise. Alongside supporting non-watertight meshes, there was also proposed a method to support completely open meshes by converting open meshes into almost closed meshes, greatly increasing the number of models the technique can voxelise.

Possible Extensions

Although not full voxelisation methods, there exist a few other methods extending already existing methods to improve them, like the method mentioned earlier (Li et al., 2023) that extends surface voxelisation methods to allow them to achieve solid voxelisation as well.

Similarly targeting SVOs, Baert et al. (2013) proposed a method to voxelise higher triangle models or voxelise at higher resolutions by partitioning the voxelisation, then ensuring the voxelisation is produced in Morton code order. This voxelisation can then be streamed out using their method to construct a SVO out of core, which allows it to combat potential memory limitations. The voxelisation method that they use to showcase their extension is the triangle voxelisation method of Huang et al. (1998).

Another extension to voxelisations was proposed by Zhang et al. (2018b) to combat a couple errors that they found appeared commonly in voxelisations. Specifically, they developed a method to better conserve material aspects of voxelisations by fixing both the thin cover problem and the two-side problem. The thin cover problem arose from thin layers not correctly being identified as the surface when choosing voxel material (see Figure 8), this was fixed by the implementation of a depth test to correctly identify the material a voxel should take on. Then the two-side problem arose from two thin layers at opposite sides of a model, it was fixed by adding side detection and storing both

materials for voxels that are occupied by both sides of the model, with a flag system to allow identification of whether a voxel is two-sided.

Figure 8: An example of the thin cover problem (centre) being solved (right) (Zhang et al., 2018b).

Suitability for Destructibility

In terms of suitability by supporting the greatest number of models, the differentiable method (Luo et al., 2024) and point-in-tetrahedral (Chen et al., 2021) have an advantage over many other methods by supporting non-watertight models. The differentiable method describing a technique to even support open meshes leaves it being possibly the most suitable when number of models is considered. All possible extensions listed should be strongly considered as the ability for adding solid voxelisation (Li et al., 2023), better storage in SVOs (Baert et al., 2013) and solving issues to better retain material properties (Zhang et al., 2018b) are all important features that voxelisations would desire to better replace 3D models.

For complexity and efficiency, more up to date methods like the equidistant scanline with gap detection (Delgado et al., 2024) are very desirable for their ability to voxelise in a single pass of the GPU. However, in terms of a tool that pre-generates the voxelisations, the efficiency is of less importance, and the ability to properly voxelise the models that may require conservative and thin voxelisations is more desirable in methods such as the box-triangle overlap test by Schwarz and Seidel (2010).

Rendering many voxels

An area where efficiency is of great importance is the rendering of the voxels, as higher resolution voxelisations will have many voxels to render, and the ability to render more voxels will allow more models or higher resolutions to be supported. Outside of rendering voxels for games, there has been much work focused on visualisation of volumetric data for scientific visualisation where there is often a large amount of data to display. As there is too much literature to cover fully, for a deeper look into volume rendering the reader is referred to the review on large-scale volume visualisation (Beyer et al., 2015) and a more recent review that focus on techniques beyond structured data (Sarton et al., 2023).

A notable example of volume rendering that has been built open for scientific visualisation is GigaVoxels (Crassin et al., 2009) which loads smaller voxel grids dubbed bricks based off a last recently used method, that uses casted rays to discover what data is missing. The method ensures only the data required for the currently desired level of detail (LOD) is loaded and can easily support filtering through mipmaps and the use of a SVO, or similar, data structure. To store the data on the GPU, a large 3D texture is used which forms the brick pool and another to store the nodes of the used tree. This works very well for real-time rendering of static data on the GPU, especially with a recent enhancement (Richermoz and Neyret, 2024) that uses dynamic parallelism to improve GPU utilisation. Some examples of their approach rendering data that consists of billions of voxels can be seen in Figure 9.

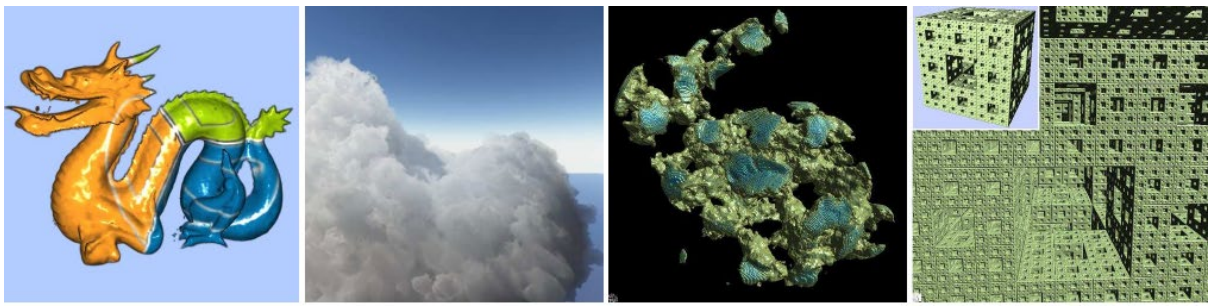

Figure 9: Large volume data rendered by GigaVoxels (Crassin et al., 2009).

On the other hand, the GigaVoxels method cannot be trivially changed to be dynamic, as only the visible voxels are loaded it would not be simple to add simulation to the scene of voxels and, should the voxels be simulated, updating the tree structure could be a costly task. Somewhat targeting this issue, Sarton et al. (2020) developed an interactive volume visualisation pipeline that also allows processing data that is not visible. This could be investigated further for its suitability for a simulation where the voxels in a scene cannot all fit into memory.

Data Format

Memory efficiency is a difficult challenge in rendering pipelines when it comes to rendering voxels, which is regularly combatted by choices of data structures that reduce the memory footprint of storing the voxel data. In the literature VDB (Museth, 2013) was created as a more flexible structure to be used for various volumetric data purposes, then later tweaked to form NanoVDB (Museth, 2021) to make it more viable for GPU usage especially for static sparse voxel grids. Similarly, GVDB (Hoetzlein, 2016) and its recent extension of Depth-Box VDB (Xu et al., 2024), manage to leverage the GPU to achieve real-time rendering of sizable volumetric data by ray casting.

Mostly the data structures mentioned above are similar to SVOs which is suitable for sparse, coherent volumes, however, some consideration should be given to the potential downside of these structures compared to dense, non-coherent volumes as brought to light by Nousiainen (2019) and Hadwiger et al. (2017). Generally, some form of data structure would be preferable to the high memory cost of the raw data. Recently hybrid voxel formats have been investigated (Arbore et al., 2024) to improve efficiency of ray intersections and storage, finding that a combination of raw and distance fields could be a more effective format when attempting to ray trace voxel data.

When tree branches of the data are similar, directed acyclic graphs (DAG) can be used to prevent having to store similar branches more than once, letting a parent node point

to the same child node. A paper (Dado et al., 2016) utilised a custom compression method for the colour data in a scene to still be able to represent the voxels in a DAG when the colour would otherwise prevent similar branches from being compacted.

Although these formats for voxel data have proven performance benefits when rendering voxel data, it can be difficult to perform updates or simulation on the voxels due to the requirement to update the data structure alongside which led Wang et al. (2023) to develop a framework to enable real-time interaction with volumetric data. Similarly, van Wingerden (2015) explored the benefits of unstructured data and its natural benefits when editing the data is desired.

Rendering Methods

Simpler methods for rendering voxels that do not rely on the data to be structured in a particular way exploit the natural ability of the GPU to rasterise triangles, often utilising the geometry shader of the programmable graphics pipeline to create the geometry to be rasterised. Mileff and Dudra (2019) describe a method for rendering smaller sets of voxels, as well Poxels (Miller et al. 2014) triangulates the voxels to render them like a mesh usually would be. Also exploiting rasterisation, Jabłoński and Martyn (2016) rasterise voxels into screen space billboards, but they differ from the other two methods by storing the voxels in a SVO (see Figure 10) with mipmapped textures to allow LOD performance optimisations when rendering.

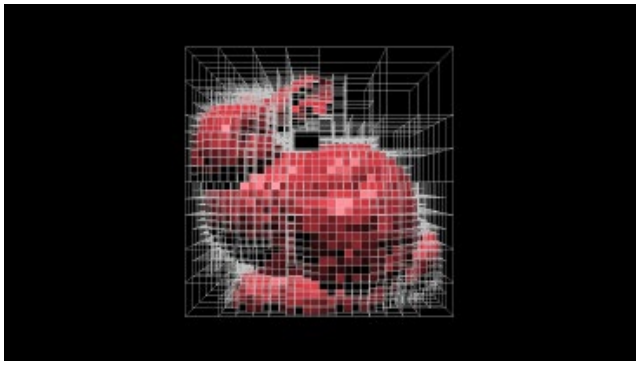

Figure 10: Example of what the SVO looks like Jabłoński and Martyn's (2016) implementation.

Another technique that leverages the sparse voxels (Sun et al., 2024) instead projects voxels into image space then sorts them so that a ray cast can evaluate the colour a pixel should take. However, the method is focused on novel view synthesis so its benefits to rendering voxels for simulation is uncertain. Potentially the most applicable paper that fits the needs for games is by Majercik et al. (2018) which supports the rendering of arbitrarily orientated voxels with no precomputation or spatial data structure. This allows it to support fully dynamic scenes where voxels could change every frame. They achieved real-time rates when ray casting when rendering large voxel scenes including animated ones. Though looking into leveraging some form of spatial data structure, with fast updates when simulating the voxels, could speed up their algorithms to allow more scalability in the scenes that can be rendered, especially if the amount of data does not fit into the GPU memory as is often found in scientific visualisation applications.

Important Considerations

Considerations into data format and structures is important for the rendering speed of voxel data, required in cases where data cannot fit in memory, consideration needed though in the cases of updating these structures when the scene updates. Not previously mentioned but the speed in which these structures can be searched can vary, and a pipeline (Barnes et al., 2024) for accelerating the search of such structures has been investigated.

Often a technique that finds the most benefits and scales well with increasing numbers of voxels is LOD, and for creating worlds that are infinite or scenes with extremely far voxels, out of core techniques can be essential for only loading voxels within a certain distance of the camera. However, as with all optimisations, the performance effects should be measured to determine their suitability. Overall, it seems that the most optimal methods for rendering many voxels are ones involving ray casting.

Key Takeaways

When aiming to simulate realistic destructibility it is common to use physically based methods. However, these methods are often not suitable for real-time simulation, and their use in games would only be suitable in the context of pre-fracturing. When pre-calculating fracture Voronoi methods are commonly used, but the method by Sellán et al. (2023) may be preferred to properly consider the weak points of the model.

One method found to be more flexible in the way it handled destruction used voxelisation and constraints (McGraw, 2024) to simulate destruction of soft bodies. To gain this flexibility the models first had to be in voxel form. Many methods to convert 3D models into voxel form exist, and when aiming for speed an efficient scanline approach like the method by Delgado Díez et al. (2024) may be preferable. As speed of voxelisation is not a concern when rendering the models as voxels, the preferred method for destructibility would be one that can voxelise a higher number of models. A method that stands out for achieving this is differentiable voxelisation (Luo et al., 2024), especially since it has the option to support voxelising models with open surfaces.

Within the context of games many models would likely be rendered at one time. Due to this the ability to render many voxels in real-time could become a large concern. The scientific and medical visualisation fields provide many methods for rendering huge numbers of voxels and can be taken as inspiration when attempting to render voxels for games. Methods specifically targeted towards games may be preferable though, as the voxels still need to be in a format that lets them be simulated. For this, raytracing methods that support dynamic scenes (Majercik et al., 2018) show the most promise for ensuring real-time rendering performance that supports the simulation of destruction.

Research Methodologies

This dissertation aims to determine the suitability of using voxels as a primitive for the rendering and simulation of destructibility of 3D models. A study has been conducted to determine whether it is possible to simultaneously simulate and render voxelised models on a typical desktop machine in real-time.

Artefact Creation

The main aspects that were required when creating the artefact for this study were the voxelisation and subsequent rendering of inputted 3D models. To get the maximum control over the performance of these aspects, the artefact was coded in C++ using GLFW[1] for window handling and Vulkan[2] as the graphics API. GLFW was used as it is lightweight and easy to setup, Vulkan on the other hand was utilised for its finer control over the GPU, and C++ for similar reasons.

Alongside this, a few smaller libraries were used to simplify parts of the implementation; Dear ImGui[3] for immediate mode user interface for editing parameters, GLM[4] for the boiler plate mathematics code, and tiny obj loader[5] for a lightweight model loader. The use of these libraries allowed the artefact creation to be streamlined, so focus could be directed towards the aspects relevant to the research.

During artefact creation, the differentiable voxelisation (Luo et al., 2024) method was chosen for its ability to support a larger number of existing 3D models. During implementation the method was first written for the CPU to ensure its accuracy, then it was ported to a compute shader to utilise the parallel processing capabilities of the GPU. Once voxelisation was implemented, rendering code was implemented with the naïve approach of using the geometry shader to generate a cube from the voxel points. Further

[1] GFLW – Graphics Library Framework [https://www.glfw.org/]
[2] Vulkan – Low-level graphics API [https://www.vulkan.org/]
[3] Dear ImGui – Graphical user interface [https://github.com/ocornut/imgui]
[4] GLM - OpenGL Mathematics [https://github.com/g-truc/glm]
[5] tinyobjloader – single file wavefront obj loader [https://github.com/tinyobjloader/tinyobjloader]

rendering considerations were taken, however, the simulation was identified as a greater bottleneck.

As there was a lack of research around voxel simulation, the artefact simulation featured a simpler implementation on the CPU. This simulation involved simple physics with gravity and axis-aligned bounding box collisions between the voxels. They also had collisions with the bounds of the intended simulation area and constraints to keep the model together. Constraints were formed after voxelisation of the model and updated over the runtime, checking if the difference in velocity surpassed a breaking threshold. If this threshold was not overcome the voxels would try and match velocity with their constraints through averaging both voxel's velocities.

Optimisation in the form of spatial partitioning was required to support the large number of voxels being simulated. Spatial hashing was chosen after comparing the suitability of a variety of methods, however, deeper analysis into which techniques are best is warranted. The option to disable the simulation was also added so that rendering could be evaluated in isolation.

User interface for editing parameters, like the resolution of the voxelisation, was implemented for use during testing. There was also user interface for interacting with the simulation, by generating an explosion at one of the voxels with a given strength and range. Impulse force gets applied to all voxels within the explosion range from the source, with the force using the inverse square law of the distance weighted by the explosion strength. Depending on the breaking threshold this would lead to the model exploding apart due to the differences in velocity.

Artefact Validation

This study primarily employed a performance-based analysis methodology, as good performance is integral to game development. Due to this the data gathered from the

artefact was the amount of time it took to process a frame, considering both the GPU and CPU. To ensure that anomalies have less of an effect, the mean was taken over 60 frames, chosen due to 60 FPS being the desired performance of the program. The input model was reloaded before each test to ensure the integrity of the testing.

Aiming for a duration of 1000 sets of 60 frames, the artefact's mean average frame time was recorded; the duration was lowered when recording 1000 sets would have taken significant time. Several parameters were varied for each test: the resolution, the input model, whether simulation was enabled, and whether the simulation was interacted with. The main independent variable was resolution, which was the cubic root of the number of query points used when voxelising the model. The resolution therefore caused a cubic increase in number of voxels.

The models used for testing were the Stanford bunny and teapot models. These were chosen for testing because they are commonly used 3D test models and had a lower vertex count, which enabled faster voxelisation. As it better correlated with machine hardware the resolution was increased in powers of two, until the performance of the test dropped below real-time rates. Powers of two generally support faster algorithms due to memory and processors being designed in this binary format. The starting resolution was chosen to be 16 as below this resolution the voxelisation was no longer identifiable as the input model.

All the performance values were acquired on a machine with 32 GB of RAM that has a transfer speed of 3200 MT/s, an Intel® Core™ i7-11700 CPU with base speed of 2.50 GHz, and a NVIDIA GeForce RTX 3080 GPU. Performance was also measured with the program rendering at a screen resolution of 1920 by 1080 pixels. For better accuracy data should be gathered on a variety of machines, but this was beyond the scope of this study.

Data Analysis

Once gathered, the data was analysed to determine the precision and scalability of using voxels as a destruction primitive. For simplicity, the precision of the voxelisation was determined to increase with input resolution, as higher resolutions meant more voxels to represent the finer details of the model. Scalability of the voxels was determined through analysing the performance of both rendering and simulation.

To determine the overall suitability as primitive for destructibility, inferences were made to determine whether the given data suggested that the rendering and simulation could be expanded to the larger scale required when creating games. The correlation between performance and voxel count was used to estimate the potential performance that the implementation could have at scale.

Due to the large amount of data gathered, the data was tabulated and put into graphical form, plotting the performance against resolution. Throughout analysis, the bottlenecks were identified to determine the which points are critical when aiming to use voxels for destructibility. Performance was compared with and without simulation enabled to help identify the bottlenecks. As rendering on the GPU runs separately to the simulation before being synchronised, a decrease in performance would suggest that simulation is the bottleneck.

In summary, the performance of the artefact was measured in mean average FPS at varied parameters. Such data was then analysed to determine the overall suitability of voxels as a destructibility primitive by evaluation of whether they can be rendered and simulated at interactive real-time rates.

Results and Findings

During testing, the rendering was tested for two models at 5 different resolutions. The two models chosen for testing were a teapot and the Stanford bunny. The resolutions started at 16 and increased in powers of 2, leading to the resolutions tested in rendering being 16, 32, 64, 128 and 256.

For simulation less resolutions were tested due to the sharper decline in framerate, therefore resolutions of 16, 32 and 64 were tested for simulation. When simulating with explosions only the teapot model was tested, at the same resolutions as simulation. At resolution of 64, the case of simulating the teapot with explosions had a mean average frame rate of 4. Due to this only 100 samples were taken instead of the full 1000, as it would have taken around 250 minutes to fully compute the full amount.

All the other tests had the full 1000 samples computed with each sample being the mean average of the past 60 frames. Each sample in the 1000 samples was taken in succession, meaning that no frames were skipped once sampling began.

The number of voxels for the two chosen models to test with can be seen in Figure 11, where Stanford bunny has just over double the number of voxels in its voxelisation over the teapot model at each resolution. This leads it to require more time for rendering and simulation than the teapot due to the higher numbers of voxels as can be seen in Figure 12. It is worth noting that the Stanford bunny still only occupies roughly 20% of the voxel query points, and teapot occupying even lower with less than 10% so a model that occupies more space would have even more voxels to be rendered and simulated.

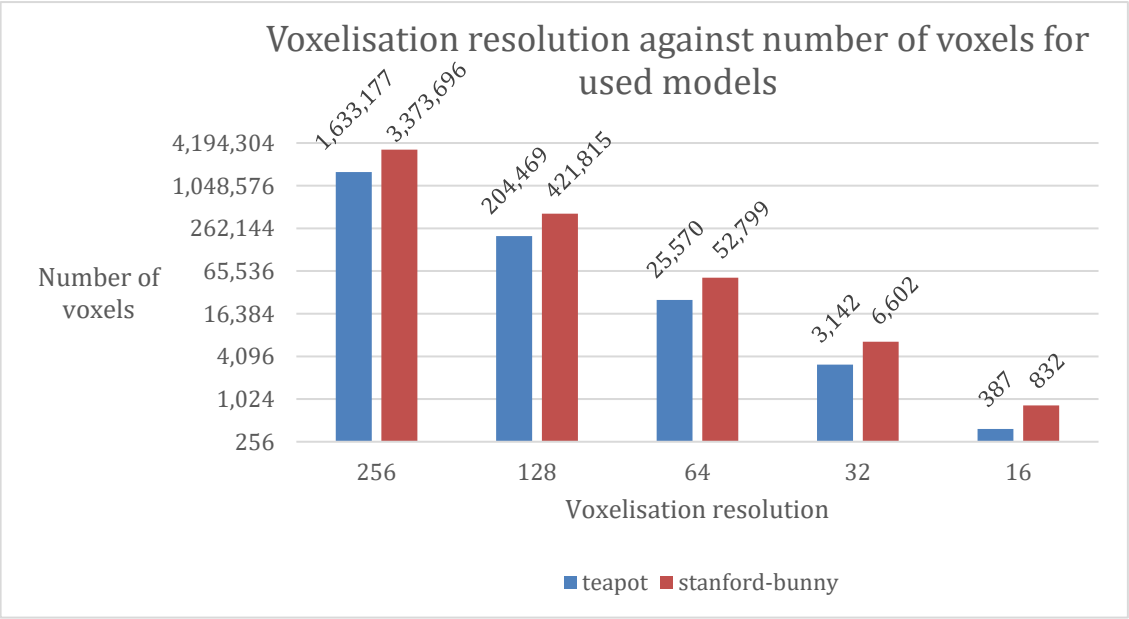

Figure 12: Number of voxels for both used models at different voxelisation resolutions.

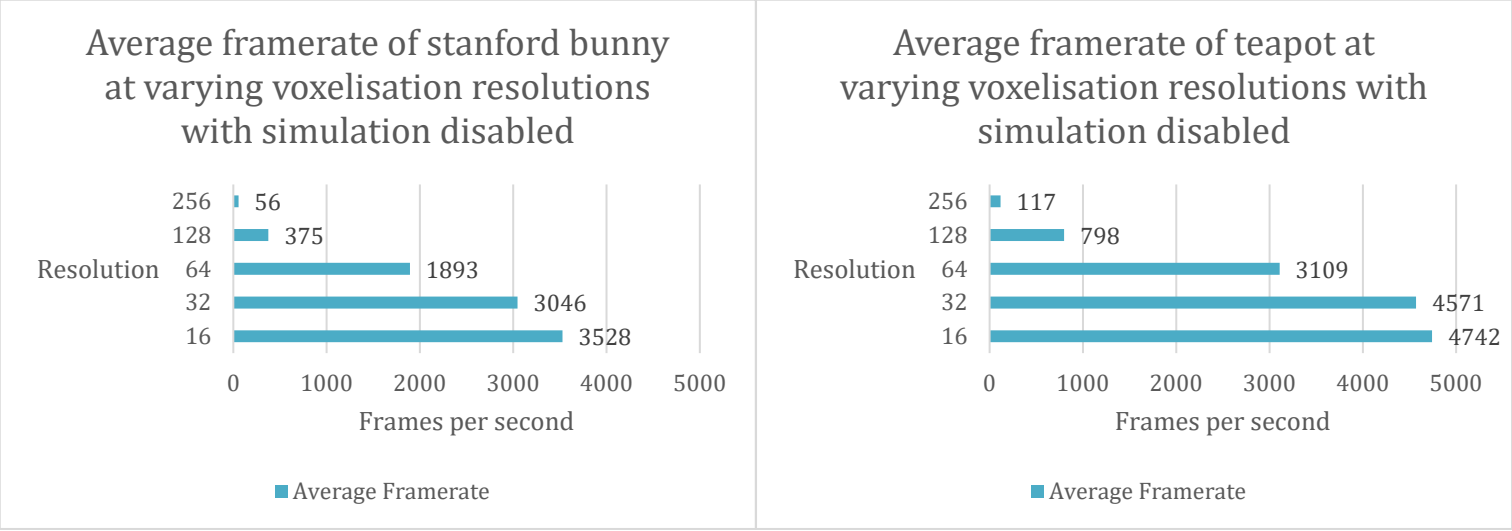

Figure 11: Average frame rate of the rendering the voxelisations at varying resolutions.

When only rendering the framerate stays at real-time rates for resolutions up to 256 for both models, then for the Stanford bunny model the framerate dropped below 60 fps. There is a negative correlation between resolution and framerate with the highest framerates being at the lowest resolution. However, when simulation is enabled, the framerate drops much quicker with the framerate getting well below real-time rates at a resolution of 64. As seen in Figure 13 for every resolution except the Stanford bunny model at resolution of 16, the program is slower when simulation is enabled, becoming significantly slower at resolutions of 32 and above.

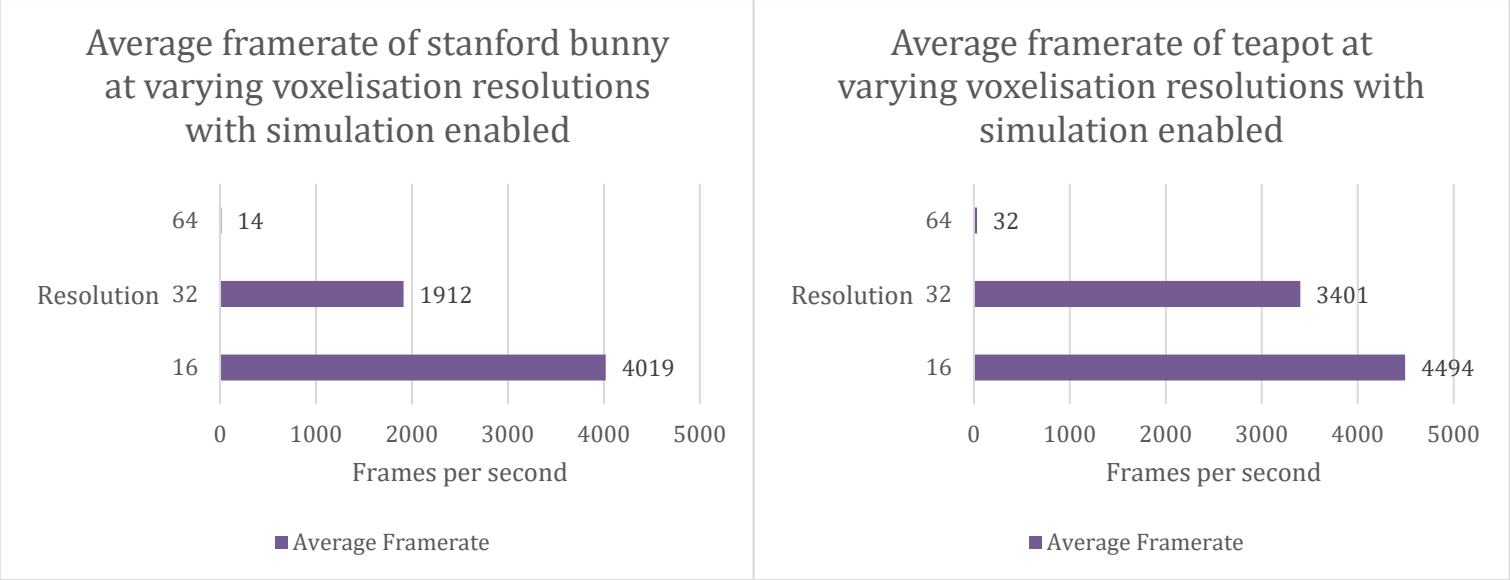

Figure 13: Average framerate for both rendering and simulating the voxelisations at varying resolutions.

All the simulations in Figure 13 were not interactive with no explosions occurring in the runtime of the profiling. When some explosions were added to the runtime the performance was generally negatively affected apart from the resolution of 16, with a resolution of 64 slowing down even more drastically as shown in Figure 14.

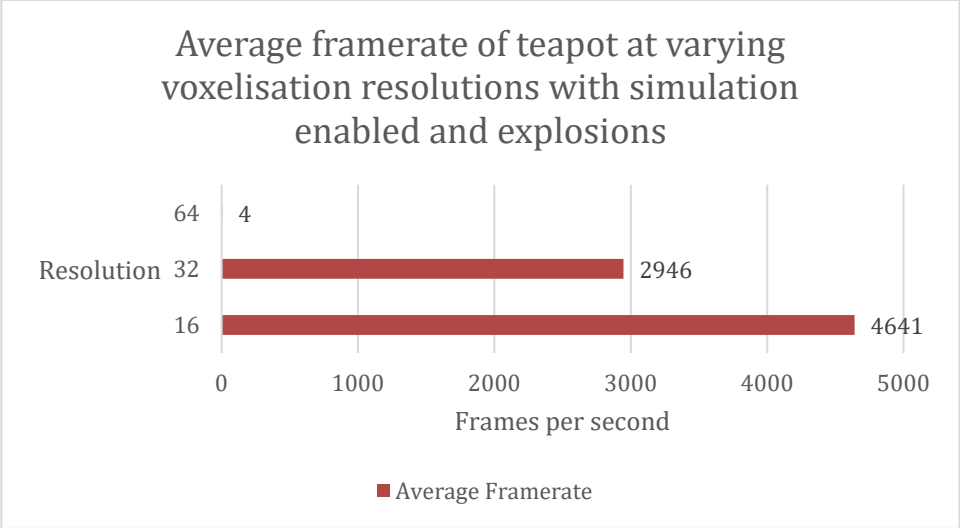

Figure 14: Average framerate for simulating and rendering the voxelised teapot at varying resolutions with interactive explosions occurring.

Figure 15 shows that at resolutions of 16 and 32 the explosion times can be identified as drops in the framerate, when looking at the simulation with explosions at a resolution of 64 the framerate drops and then stays consistently lower for the duration of the program, whereas it only slightly lowers at 32 and fully recovers at resolution of 16. The graphs also show that the slight variance from the explosions do not fully get picked up in the average due to their limited duration regarding the total sample duration.

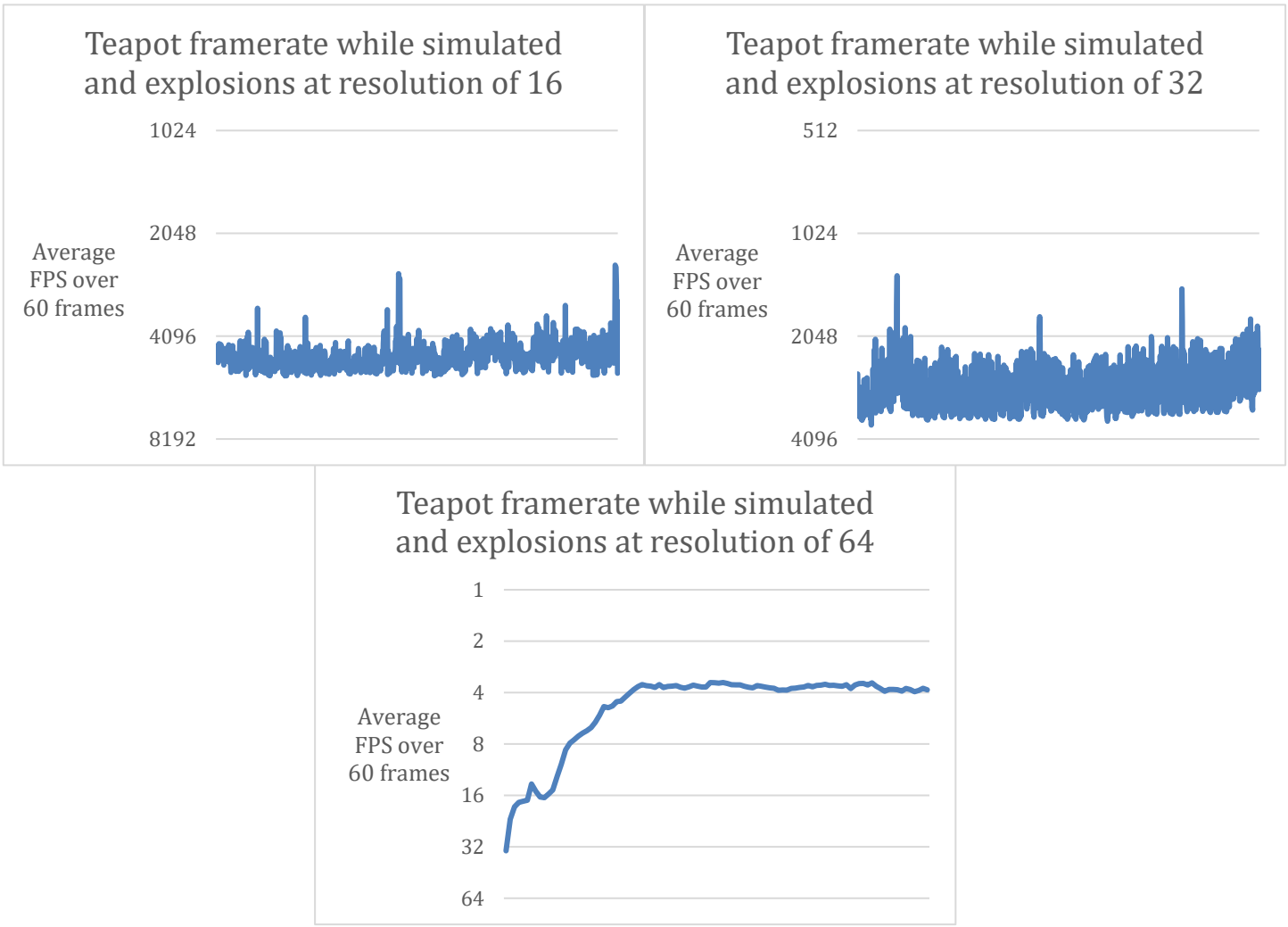

Figure 15: The 60 frame average FPS samples plotted in order that they occur, axis inverted to better show when explosion occurs as the sharp decrease in framerate. The spikes in the graphs show the drops in framerate.

Overall simulation enabled was significantly slower than when purely rendering and resolutions past 32 led to much steeper drops in average framerate for both. Scaling with the cubic increase in number of voxels.

Discussion and Analysis

Visuals

With reference to Figure 16, the teapot can be seen to have significant errors in visual representation when voxelised at a resolution of 16. This is likely due to the lower percentage it occupies of the query points as well as the thinner structures that require voxelising. The Stanford bunny still maintains its fidelity at resolution of 16, however, both models reach an adequate level of fidelity at a resolution of 32. This is assuming realistic visuals are not desired.

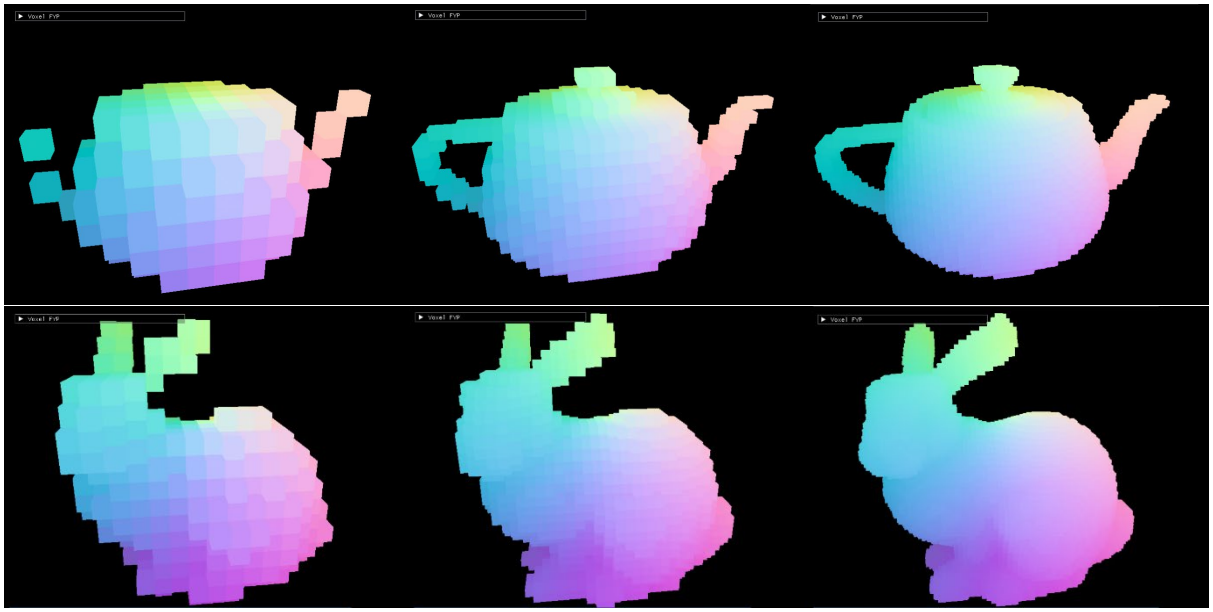

Figure 16: From left to right both used models (teapot above, Stanford bunny below) voxelised at increasing resolutions (16, 32, 64).

These models did not come with textures or materials and currently the implemented voxelisation does not support the maintaining of these visuals as it is outside the scope of the rendering considerations. Each voxel does have an independent colour which could be set based on a texture or material in the future, so the rendering performance should not be affected by these additions later.

Given the evaluated performance, an interesting trade off can be made between the fidelity of the voxelisation and the desired performance of the program. Higher input resolutions significantly reduce the performance but also give more accurate visuals. This would also mean that larger models would likely be less performant when it comes

to simulating their destructibility with voxels, as they would require a larger number of voxels to properly capture their details.

A similar statement can be made on the voxelisation, depending on the level of detail wishing to be captured from the model. For example, high poly models often have many smaller details that would not be captured at lower resolutions when voxelising. On the contrary, low poly stylistic models would not require a higher resolution to capture their details. This suggests that games that aim for performant voxel destructibility might prefer more stylistic art styles, and that certain styles would not be manageable in real-time voxel simulations.

Rendering

Only considering performance of the rendering, the framerates are still largely above real-time, only just dipping below 60 FPS when rendering the bunny at resolution of 256 with 3,373,296 voxels being rendered at once. Assuming the bunny model's percentage occupancy of roughly 20% is close to the mean average for 3D models, then at a resolution of 32 over 500 voxelised models would be renderable at one time. This also assumes that the position of the voxels does not have a significant effect on the performance, so testing would still be required. Extra rendering optimisations like level of detail and frustrum culling could also boost performance when rendering larger scenes containing many objects. Therefore, significantly more than 500 voxelised models could be rendered simultaneously should these optimisations be implemented.

The further rendering considerations discussed from the literature should also be implemented to evaluate their performance, especially as the number of voxels may increase to a point where they do not fit in the available memory of the GPU. Specifically, the raytracing and alternate data formats that can be used alongside raytracing, could be essential for performance that is more scalable.

Simulation

On the other hand, although rendering was performant at these voxel counts, the performance dropped significantly when simulation was enabled. It is difficult to make inferences when considering the scalability of the simulation as when interacted with the framerate dropped considerably, suggesting that the spatial partitioning did not hold well when handling objects spread out across the scene. This would be a key requirement when considering the larger scale that levels operate on.

Another reason that causes the simulation's scalability to be more difficult to evaluate with the data gathered is the sharp decline in performance between resolutions of 32 and 64. For both models used the performance drops from over 1000 FPS to well below real-time rates, making it harder to evaluate the rough voxel count where the artefact was still suitable for interactive destruction.

At resolution of 32, the simulation still has a significant margin of frame time to work with for interactivity. So, for smaller scale simulations voxels would certainly be suitable, however, the simulation is a large bottleneck when it comes to the wider application of voxels as a destruction primitive. Further investigation and research would be warranted to improve the performance of the simulation. Cases where there are multiple voxelised models interacting should specifically be investigated to better emulate game scenarios.

Overall, the performance behaved as expected, dropping dramatically due to increases in resolution causing large cubic steps in the number of voxels to simulate and render at one time.

Limitations

There were 4 key limitations identified in the approach adopted by this study.

1. One limitation of the study was the lack of testing across different computers with varying hardware. To ensure the suitability for use in games, the test should be run multiple times at hardware specifications at differing qualities. This is needed to truly verify if voxels are suitable to be distributed across all game platforms. For example, mobile devices generally have more restrictive demands when it comes to performance but also has a thriving market for games.

2. The test could also have been impacted by other software currently running at point of testing. This could have been what impacted the performance when rendering the Stanford bunny at a resolution of 16, causing it to perform better when tested with the simulation enabled.

3. One limitation of the research methodology was a proper investigation into the visuals of the program. Although some claims were made based on the clear errors in visuals of the voxelisations, a proper study into the effects of resolution on the visuals should be undertaken. This would allow feedback to be acquired from external sources to determine which resolution would be desired from a visual standpoint. With a desired resolution established, more focus could be directed towards optimisation of the rendering and simulation at that level.

4. Given more time, a larger variety of models from wider sources should be considered to better make judgements on the suitability and scalability. As mentioned, multiple models should also be tested together as this would better represent the dynamism of game worlds. Testing the performance at smaller steps in resolution could also be more beneficial when showcasing the effect that it has, especially in the cases of simulation where the performance drops sharply.

Reflection on Objectives

In the objectives, the main question posed by the methodology was whether it is possible to simultaneously simulate and render voxels for destructibility in real-time. The results of the study suggest that it is possible in smaller projects, but when extended to larger scenarios like games, a lot more work needs to be done to improve the performance of voxels as a destruction primitive.

Due to the under-exploration in the areas of simulating voxels for the use of destructibility the bottleneck in achieving the goal was identified to be the simulation. Inspiration from other simulation research would be desirable to explore what could be done to properly make voxels a viable candidate for a real-time destructibility primitive.

Artefact Reflection

For creating the artefact, the chosen graphics API was Vulkan, though due to scope and bottlenecks in the simulation, was underutilised. Originally the finer control over the GPU was intended to be leveraged as argument for the added complexity of using Vulkan, but the same solutions would likely be more manageable to implement in simpler APIs. This would have allowed more time to be dedicated to the other details of the study, as discussed in the limitations.

The method of differentiable voxelisation (Luo et al., 2024) had effective results for what was required in the implementation, but other voxelisation methods may lead to the same result and be in a better format for rendering and simulation.

Conclusion

Simulating destruction with voxels managed to successfully handle differing strengths and positions of explosions to break apart the test models. This allowed a high level of dynamism when breaking apart a model over some of the more traditional approaches that only give a preset way for objects to break apart. Where the voxels do see a disadvantage is on the realism of breaking apart, as the destruction will suffer from being very blocky in comparison with other destructibility methods.

In the context of providing entertainment to players this is not necessarily an issue as fun can be had outside of the context of realism. Further research would be required to determine whether the use of this less realistic destruction has a significant effect on the enjoyment or immersion of players. Similarly, it should be confirmed that this destruction can provide more enjoyment than other destructibility approaches, as otherwise it may be preferable to use more realistic looking approaches.

It is, however, clear that the destruction correctly uses the benefits of voxelisation to allow models to split up into chunks that may then split again and are not limited to only the initial fracture. The only limit to the destruction is created when the resolution is chosen for voxelising the models. To this end, the performance is obviously crucial to getting the most out of using voxels as a destruction primitive, with better performance allowing more dynamism to be achieved with higher resolutions.

The ability to utilise previously created 3D models in voxel environments would speed up the process of creating games that want to leverage voxels as a destruction primitive. Research around voxelisation is in abundance, and the implemented method proved sufficient for demonstrating destructibility. However, more work needs to be done to fully utilise existing 3D models as none of the data other than the shape of the model is retained.

Games without realistic visuals have still found recent success and were once the standard due to hardware limitations. So, prioritising dynamic environments that let the

player find their own fun within a game, could lead it to success. The results of the study prove that voxels are not far off being easily usable for this.

The building blocks for creating an environment that innately supports destructions through representing the models as many individual voxels has been provided. Optimising the simulation by leveraging the GPU or techniques in other areas of simulation, could push voxels towards the position of being a standard primitive for implementing destructibility in games where realism is not the concern.

When it comes to the required preprocessing steps to using voxelised models for destructibility, unless wanting to upscale the resolution of the voxelisation at a later point, the preprocessing should only need to happen once for other use cases. Adding to this, the method implemented in the performed study was efficient enough that regeneration would not be a long process. What could be a benefit in terms of preprocessing, is a way to extract only the surface voxelisation for models which one would expect to be hollow. For example, the teapot model would likely be expected to not have a solid inside.

The artefact created for the study had a simple physics system with fixed constraints so it could showcase the ability for voxels to be used in destructibility. This all was self-contained outside of where the rendering of the voxels was happening. Due to this the rendering implementation would easily support flexibility in the way the voxels are simulated, and more accurate simulations could be created where desired. Where this becomes even more desired, is when considering the complexity of adding additional gameplay features that interact with the voxels. Having the rendering for the voxels encapsulated would aid in reducing this complexity.

Overall, the performed study has been a sufficient precursory investigation into the possibilities of using voxelisation on existing 3D models as a system for simulating destructibility in games. There is a lack of research into the use of voxels as a primitive in games and more effort should be targeted towards investigating this. The benefits voxels provide can span multiple aspects of game development, as voxelisation can often be

used as an initial step when performing other calculations such as pathfinding and lighting. If further work was done, voxels as a primitive could immediately support these use cases.

Recommendations

Although the literature was well reviewed, there were plenty of gaps in research directed towards the methods for simulating voxels. Accumulating techniques from other parts of simulation research and investigating the techniques specifically suitable for voxel simulation could significantly improve the results from simulating destructibility with voxels.

Specifically, comparing various spatial partitioning methods, especially the combination of them could significantly reduce the effect of simulation on the performance. These considerations should be the larger focus in future studies, as the performance of simulating destructibility is crucial to allow it to be used as an interactive element of a game.

Lots of research was done into methods for rendering large numbers of voxels, however, in the context of interactive games they were not fully considered for their ability to be used alongside simulation of the voxels. Due to the bottleneck provided by the simulation, the performance of the rendering was not further developed and should be investigated more in the context of games. To achieve this, the voxelised models should be placed in an environment more akin to a game's environment. Lighting, shadows and other rendering effects like post processing had no consideration when it came to the rendering of voxels in the performed study but are used often in rendering elsewhere.

The performance of the discussed features should be tested on other hardware, and specifically the current generation of consoles, or even mobiles. These make up large parts of the game market and not considering them could lead to games built with a voxel destructibility system locking themselves out of those markets.

One of the gaps in this study that needs addressing, is the lack of feedback from potential players. Further work may want to determine if voxel destructibility is desired by a large enough group of people. Especially investigations into the specific desires and expectations could streamline future research into this area. Without this, the lack of

graphical fidelity for example, could prevent players and developers from desiring the use of voxels in games.

Similarly, the chosen set of models was very limited, and further work should investigate the use of other 3D model formats, alongside confirming the effectiveness of the chosen voxelisation for its ability to support many 3D models. Whilst looking specifically at visuals, much work should be done to support materials and textures. The normals of the models could also warrant preserving to support lighting systems. In general, the visuals of the voxelisations deserve further work before they can be fully adopted in games.

However, many limitations and recommendations are minor in the context of an initial study into the use of voxels for destructibility in games. The evaluated performance suitably covered the initial requirements and has correctly identified the points of future research required. This is the main purpose of the performed study as the significant gaps in research needed to be identified, and the performance of simultaneously rendering and simulating voxels was measured to properly identify whether voxels could be further used in this way.

References

Akenine-Möller, T. (2005) Fast 3D triangle-box overlap testing. In ACM SIGGRAPH 2005 courses (pp. 8-es).

Aleksandrov, M., Zlatanova, S. and Heslop, D.J. (2021) Voxelisation algorithms and data structures: a review. Sensors, 21(24), p.8241.

Arbore, R., Liu, J., Wefel, A., Gao, S. and Shaffer, E. (2024) Hybrid Voxel Formats for Efficient Ray Tracing. arXiv preprint arXiv:2410.14128.

Baert, J., Lagae, A. and Dutré, P. (2013) Out-of-core construction of sparse voxel octrees. In Proceedings of the 5th high-performance graphics conference (pp. 27-32).

Barnes, A., Shen, F. and Rogers, T.G. (2024). Extending GPU Ray-Tracing Units for Hierarchical Search Acceleration. In 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO) (pp. 1027-1040). IEEE.

Bergs, T., Henrichs, O., Wilms, M., Prümmer, M. and Arntz, K. (2021) Development of a voxelization tool for the calculation of vector-based workpiece representations. Procedia CIRP, 100, pp.7-12.

Beyer, J., Hadwiger, M. and Pfister, H. (2015) State-of-the-art in GPU-based large-scale volume visualization. In Computer Graphics Forum (Vol. 34, No. 8, pp. 13-37).

Chen, J., Tai, K.W., Chen, W.C. and Ouhyoung, M. (2021) Robust Voxelization and Visualization by Improved Tetrahedral Mesh Generation. arXiv preprint arXiv:2106.01326.

Chitalu, F.M., Miao, Q., Subr, K. and Komura, T. (2020) Displacement-Correlated XFEM for Simulating Brittle Fracture. In Computer Graphics Forum (Vol. 39, No. 2, pp. 569-583).

Clothier, M.M. (2017) 3D Voronoi Subdivision for Simulating Destructible and Granular Materials. Ph.D. Thesis. Oregon State University.

Cohen-Or, D. and Kaufman, A. (1995) Fundamentals of surface voxelization. Graphical models and image processing, 57(6), pp.453-461.

Crassin, C. and Green, S. (2012) Octree-based sparse voxelization using the GPU hardware rasterizer. OpenGL Insights, pp.303-318.

Crassin, C., Neyret, F., Lefebvre, S. and Eisemann, E. (2009) Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In Proceedings of the 2009 symposium on Interactive 3D graphics and games (pp. 15-22).

Dado, B., Kol, T.R., Bauszat, P., Thiery, J.M. and Eisemann, E. (2016) Geometry and attribute compression for voxel scenes. In Computer Graphics Forum (Vol. 35, No. 2, pp. 397-407).

Delgado Díez, S., Cerrada Somolinos, C. and Gómez Palomo, S.R. (2024) Optimizing Surface Voxelization for Triangular Meshes with Equidistant Scanlines and Gap Detection. Computer Graphics Forum, 43: e15195. https://doi.org/10.1111/cgf.15195

Eisemann, E. and Décoret, X. (2006) Fast scene voxelization and applications. In Proceedings of the 2006 symposium on Interactive 3D graphics and games (pp. 71-78).

Faieghi, M., Tutunea-Fatan, O.R. and Eagleson, R. (2018) Fast and cross-vendor OpenCL-based implementation for voxelization of triangular mesh models. Computer-Aided Design and Applications, 15(6), pp.852-862.

Fan L., Chitalu F.M. and Komura T. (2022) Simulating Brittle Fracture with Material Points. ACM Transactions on Graphics (TOG), 41(5), pp.1-20.

Fei, Y., Wang, B. and Chen, J. (2012) Point-tessellated voxelization. In Proceedings of Graphics Interface 2012 (pp. 9-18).

Forslund, E. (2023) Optimising 3D object destruction tools for improved performance and designer efficiency in video game development. Bachelor's Thesis. Blekinge Institute of Technology.

Grönberg, A. (2017) Real-time mesh destruction system for a video game. Bachelor's Thesis. Luleå University of Technology.

Hadwiger, M., Al-Awami, A.K., Beyer, J., Agus, M. and Pfister, H. (2017) SparseLeap: Efficient empty space skipping for large-scale volume rendering. IEEE Transactions on Visualization and Computer Graphics (TVCG), 24(1), pp.974-983.

Hahn, D. and Wojtan, C. (2016) Fast approximations for boundary element based brittle fracture simulation. ACM Transactions on Graphics (TOG), 35(4), pp.1-11.

Hoetzlein, R.K. (2016) GVDB: Raytracing sparse voxel database structures on the GPU. In Proceedings of High-Performance Graphics (pp. 109-117).

Hoss, R. and Emma, T. (2012) Methods of Creating Destructible Assets for Video Games. Computer Games, Multimedia and Allied Technology (CGAT 2012), p.30.

Huang, J., Yagel, R., Filippov, V. and Kurzion, Y. (1998) An accurate method for voxelizing polygon meshes. In Proceedings of the 1998 IEEE symposium on Volume visualization (pp. 119-126).

Huang, Y. and Kanai, T. (2023) DeepFracture: A Generative Approach for Predicting Brittle Fractures. arXiv preprint arXiv:2310.13344.

Jabłoński, S. and Martyn, T. (2016) Real-time voxel rendering algorithm based on screen space billboard voxel buffer with sparse lookup textures. WSCG 2016: The 24th Conference on Computer Graphics, Visualization and Computer Vision.

Karabassi, E.A., Papaioannou, G. and Theoharis, T. (1999) A fast depth-buffer-based voxelization algorithm. Journal of graphics tools, 4(4), pp.5-10.

Kumar, R., Deep, R., Banerjee, D.S. and Arora, N. (2024) Voxelization of Moving Deformable Geometries on GPU. In 2024 23rd International Symposium on Parallel and Distributed Computing (ISPDC) (pp. 1-8). IEEE.

L'Heureux, J. (2016) The Art of Destruction in 'Rainbow Six: Siege', GDC Vault. Available at: https://www.gdcvault.com/play/1023003/The-Art-of-Destruction-in (Accessed: 08 November 2024).

Lamb, N., Palmer, C., Molloy, B., Banerjee, S. and Banerjee, N.K. (2023) Fantastic breaks: A dataset of paired 3d scans of real-world broken objects and their complete counterparts. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4681-4691).

Li, C., Liu, J., Wang, H. and Xiu, P. (2023) Improved internal voxelization algorithm based on depth buffer. In 2023 5th International Academic Exchange Conference on Science and Technology Innovation (IAECST) (pp. 19-22). IEEE.

Lu, J.M., Cao, G.C., Li, C.F. and Hu, S.M. (2023) Variational Bonded Discrete Element Method with Manifold Optimization. arXiv preprint arXiv:2308.10459.

Lu, J.M., Li, C.F., Cao, G.C. and Hu, S.M. (2021) Simulating fractures with bonded discrete element method. IEEE Transactions on Visualization and Computer Graphics, 28(12), pp.4810-4824.

Luo, Y., Wang, Y., Xiang, Z., Xiu, Y., Yang, G. and Yap, C. (2024) Differentiable Voxelization and Mesh Morphing. arXiv preprint arXiv:2407.11272.

Majercik, A., Crassin, C., Shirley, P. and McGuire, M. (2018) A ray-box intersection algorithm and efficient dynamic voxel rendering. Journal of Computer Graphics Techniques Vol, 7(3), pp.66-81.

Mandal, A., Chaudhuri, P. and Chaudhuri, S. (2023) Remeshing-free Graph-based Finite Element Method for Fracture Simulation. In Computer Graphics Forum (Vol. 42, No. 1, pp. 117-134).

McGraw, T. (2024) Mesh Mortal Kombat: Real-time voxelized soft-body destruction. In ACM SIGGRAPH 2024 Real-Time Live! (pp. 1-2).

Mileff, P. and Dudra, J. (2019) Simplified voxel based visualization. Production Systems and Information Engineering, 8, pp.5-18.

Miller, M., Cumming, A., Chalmers, K., Kenwright, B. and Mitchell, K. (2014) Poxels: Polygonal voxel environment rendering. In Proceedings of the 20th ACM symposium on virtual reality software and technology (pp. 235-236).

Morris, D.J. (2010) GPU Techniques: Novel Approaches to Deformation and Fracture within Videogames. Master's Thesis. Coventry University.

Müller, M., Chentanez, N. and Kim, T.Y. (2013) Real time dynamic fracture with volumetric approximate convex decompositions. ACM Transactions on Graphics (TOG), 32(4), pp.1-10.

Museth, K. (2013) VDB: High-resolution sparse volumes with dynamic topology. ACM Transactions on Graphics (TOG), 32(3), pp.1-22.

Museth, K. (2021) NanoVDB: A GPU-friendly and portable VDB data structure for real-time rendering and simulation. In ACM SIGGRAPH 2021 Talks (pp. 1-2).

Nealen, A., Müller, M., Keiser, R., Boxerman, E. and Carlson, M. (2006) Physically based deformable models in computer graphics. In Computer graphics forum (Vol. 25, No. 4, pp. 809-836). Oxford, UK: Blackwell Publishing Ltd.

Nourian, P. and Azadi, S. (2024) Voxel graph operators: Topological voxelization, graph generation, and derivation of discrete differential operators from voxel complexes. Advances in Engineering Software, 196, p.103722.

Nousiainen, O. (2019) Performance comparison on rendering methods for voxel data. Master's Thesis. Aalto University.

Ogayar-Anguita, C.J., Rueda-Ruiz, A.J., Segura-Sánchez, R.J., Díaz-Medina, M. and Garcia-Fernandez, A.L. (2020) A GPU-based framework for generating implicit datasets of voxelized polygonal models for the training of 3D convolutional neural networks. IEEE Access, 8, pp.12675-12687.

Parker, E.G. and O'Brien, J.F. (2009) Real-time deformation and fracture in a game environment. In Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (pp. 165-175).

Rauwendaal, R. (2012) Hybrid computational voxelization using the graphics pipeline. Master's Thesis. Oregon State University.

Richermoz, A. and Neyret, F. (2024) GigaVoxels DP: Starvation-Less Render and Production for Large and Detailed Volumetric Worlds Walkthrough. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 7(3), pp.1-11.

Sarton, J., Courilleau, N., Rémion, Y. and Lucas, L. (2020) Interactive visualization and on-demand processing of large volume data: A fully GPU-based out-of-core approach. IEEE Transactions on Visualization and Computer Graphics (TVCG), 26(10), pp.3008-3021.

Sarton, J., Zellmann, S., Demirci, S., Güdükbay, U., Alexandre-Barff, W., Lucas, L., Dischler, J.M., Wesner, S. and Wald, I. (2023) State-of-the-art in Large-Scale Volume Visualization Beyond Structured Data. In Computer Graphics Forum (Vol. 42, No. 3, pp. 491-515).

Schwarz, M. and Seidel, H.P. (2010) Fast parallel surface and solid voxelization on GPUs. ACM Transactions on Graphics (TOG), 29(6), pp.1-10.

Sellán, S., Chen, Y.C., Wu, Z., Garg, A. and Jacobson, A. (2022) Breaking bad: A dataset for geometric fracture and reassembly. Advances in Neural Information Processing Systems, 35, pp.38885-38898.

Sellán, S., Luong, J., Mattos Da Silva, L., Ramakrishnan, A., Yang, Y. and Jacobson, A. (2023) Breaking good: Fracture modes for realtime destruction. ACM Transactions on Graphics (TOG), 42(1), pp.1-12.

Shukla, R., Arora, N. and Banerjee, D.S. (2022) Voxelization of Moving Geometries on GPU. In 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys) (pp. 904-913). IEEE.

Sun, C., Choe, J., Loop, C., Ma, W.C. and Wang, Y.C.F. (2024) Sparse Voxels Rasterization: Real-time High-fidelity Radiance Field Rendering. arXiv preprint arXiv:2412.04459.

Taito (1978) Space Invaders [Video game]. Taito.

Thomas, R. and Zhang, W. (2023) Real-time fracturing in video games. Multimedia Tools and Applications, 82(3), pp.4709-4734.

Tollec, M., Jenkins, S., Summers, L. and Cunningham-Scott, C. (2020) Deconstructing Destruction: Making and breaking of "Frozen 2"'s Dam. In Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (pp. 1-2).

Tuxedo Labs (2022) Teardown [Video Game]. Tuxedo Labs.

van Wingerden, T.L. (2015) Real-time ray tracing and editing of large voxel scenes. Master's Thesis. Utrecht University.

Wang, J., Xiang, N., Kukreja, N., Yu, L. and Liang, H.N. (2023) LVDIF: a framework for real-time interaction with large volume data. The Visual Computer, 39(8), pp.3373-3386.

Williams, J.A. (2009) Interactive 3D carving using a combined voxel and mesh representation. Library and Archives Canada, Ottawa.

Workman, S. (2006) A Cracking Algorithm for Destructible 3D Objects. Master's thesis. University of Sheffield.

Xu, K., Zhao, J., Tao, Y. and Lin, H. (2024) Depth-Box VDB: Accelerate Sparse Volume Rendering with Depth Maps through Voxel Database. In 2024 IEEE 17th Pacific Visualization Conference (PacificVis) (pp. 272-276). IEEE.

Young, G. and Krishnamurthy, A. (2018) GPU-accelerated generation and rendering of multi-level voxel representations of solid models. Computers & Graphics, 75, pp.11-24.

Zhang, L., Chen, W., Ebert, D.S. and Peng, Q. (2007) Conservative voxelization. The Visual Computer, 23, pp.783-792.

Zhang, Y., Garcia, S., Xu, W., Shao, T. and Yang, Y. (2018a) Efficient voxelization using projected optimal scanline. Graphical Models, 100, pp.61-70.

Zhang, Z., Morishima, S. and Wang, C. (2018b) Thickness-aware voxelization. Computer Animation and Virtual Worlds, 29(3-4), p.e1832.