EXAMPLE BASED SOFT-BODY SIMULATION ON GRAPHICS PROCESSING UNITS

GRAFİK İŞLEMCİ ÜNİTELERİ ÜZERİNDE ÖRNEK TABANLI YUMUŞAK NESNE SİMÜLASYONU

EMİRCAN KOÇ

Assist. Prof. Dr. ADNAN ÖZSOY

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

2019

This work titled **"EXAMPLE BASED SOFT-BODY SIMULATION ON GRAPHICS PROCESSING UNITS"** by **EMİRCAN KOÇ** has been approved as a thesis for the Degree of **Master of Science** in **Computer Engineering** by the Examining Committee Members mentioned below.

Prof. Dr. Suat ÖZDEMİR
Head
[illegible]

Assist. Prof. Dr. Adnan ÖZSOY
Supervisor
[illegible]

Assoc. Prof. Dr. Süleyman TOSUN
Member
[illegible]

Assoc. Prof. Dr. Ahmet Burak CAN
Member
[illegible]

Assist. Prof. Dr. Mehmet KÖSEOĞLU
Member
[illegible]

This thesis has been approved as a thesis for the Degree of **Master of Science** in **Computer Engineering** by Board of Directors of the Institute of Graduate School of Science and Engineering on ...... / ..... / …….

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU
Director of the Institute of
Graduate School of Science and Engineering

ETHICS

In this thesis study, prepared in accordance with spelling rules of Institute of Graduate School of Science and Engineering of Hacettepe University,

I declare that

- all the information on documents have been obtained in the base of the academic rules
- all audio-visual and written information and results have been presented according to the rules of scientific ethics
- in case of using other works, related studies have been cited in accordance with the scientific standards
- all cited studies have been fully referenced
- I did not do any distortion in the data set
- and any part of this thesis has not been presented as another thesis study at this or any other university.

08 / 07 / 2019

[illegible]

Emircan KOÇ

YAYIMLANMA FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kâğıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan "***Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge***" kapsamında tezim aşağıda belirtilen koşullar haricince YÖK Ulusal Tez Merkezi / H. Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

☐ Enstitü / Fakülte yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.

☐ Enstitü / Fakülte yönetim kurulu gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren .... ay ertelenmiştir.

☐ Tezim ile ilgili gizlilik kararı verilmiştir.

08 / 07 / 2019

[illegible]

EMİRCAN KOÇ

ABSTRACT

EXAMPLE BASED SOFT-BODY SIMULATION ON GRAPHICS PROCESSING UNITS

Emircan KOÇ

Master of Science, Computer Engineering Department

Supervisor: Assist. Prof. Dr. Adnan Özsoy

June 2019, 70 pages

Nowadays Soft-body Physics Simulations are used in lots of different areas such as movies, videos and video games. Depending on the application, use of simulation improves realism, fun and teaching factor thus creates more quality in the product. Generally, Soft-body Physics Simulations are very resource consuming operations. In this thesis we proposed a Soft-body Physics Simulation algorithm. Aim of this proposed algorithm is to address three of the big problems of Soft-body Physics Simulations; performance, visual quality and ease of use. Proposed algorithm uses precomputed soft-body physics simulation results and creates its outputs by using precomputed simulation results as examples for a potential solution. Method of using examples simplifies and reduces expensive computations thus improves performance. Visual quality depends on given examples. By simply improving visual quality of examples, visual quality of simulation can be increased. Algorithm only creates results by using examples so there cannot be any unexpected outputs, thus this improves ease of use. Proposed Soft-body Physics Simulation algorithm includes independent tasks and can be implemented fully in parallel. As a result, it can use full potential of graphical processing units, which became exceedingly popular in parallel computation for very promising performance. Proposed algorithm has a time complexity of $O(n)$ where n being the number of vertices. Even in cases that contains three dimensional models that have millions of triangles proposed algorithm computes result in couple of milliseconds. It can easily be used in real-time applications.

Keywords: General purpose graphics processing unit, physics simulation, soft-body physics simulation, NVIDIA CUDA, real time applications

ÖZET

GRAFİK İŞLEMCİ ÜNİTELERİ ÜZERİNDE ÖRNEK TABANLI YUMUŞAK NESNE SİMÜLASYONU

Emircan KOÇ

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Danışmanı: Dr. Öğr. Üyesi Adnan Özsoy

Haziran 2019, 70 sayfa

Yumuşak Nesne Fiziği Simülasyonu günümüzde filmler, videolar, bilgisayar oyunları gibi çeşitli alanlarda kullanılmaktadır. Kullanıldığı alana göre gerçekçiliği, eğlence ve öğreticilik faktörünü geliştirip üretilen ürünün kalitesini artırmakta ve hedefini daha iyi şekilde gerçekleştirmesini sağlamaktadır. Genel olarak Yumuşak Nesne Fiziği Simülasyonu bilgisayarların kaynaklarını kullanma açısından oldukça ağır bir işlemdir. Bu sebep doğrultusunda bu tez kapsamında yeni bir Yumuşak Nesne Simülasyonu algoritması önerilmiştir. Önerdiğimiz algoritma ile Yumuşak Nesne Simülasyonlarının önemli problemlerinden olan performans, görsel kalite ve kolay kullanılabilirlik alanlarında geliştirme hedeflenmiştir. Algoritmamız önceden hesaplanmış Yumuşak Nesne Fiziği Simülasyon sonuçları kullanılarak çalışmaktadır. Bu önceden hesaplanmış sonuçlardan örnek alarak sonuçlar oluşturmaktadır. Çok sayıda hesaplamayı sadeleştirdiği için performans sağlamaktadır. Verilmiş örneklerin kalitesi ne kadar yüksekse bizim çözümümüzün verdiği sonuç da o kadar görsel kaliteye sahip olmaktadır. Önceden hesaplanmış örneklerin dışına çıkmadığı için beklenmedik sonuçlar oluşmamaktadır bu sayede kullanım kolaylığı yüksek ölçüde sağlanmış olmaktadır. Yumuşak Nesne Fiziği Simülasyonu algoritması önerimiz tamamen paralelleştirilebilir bir yapıdadır bu sebeple grafik işlem ünitelerinin tüm gücünü kullanmaya müsait olarak tasarlanmıştır. Aynı zamanda modelde n tane köşe olduğu durumda, zaman karmaşıklığı $O(n)$ olduğu için milyonlarca üçgene sahip üç boyutlu modellerde bile birkaç milisaniyede sonuç verebildiği için gerçek zamanlı uygulamalarda kolaylıkla kullanılabilir.

Anahtar Kelimeler: Genel amaçlı grafik işlem ünitesi, fizik simülasyonu, yumuşak nesne fiziği simülasyonu, NVIDIA CUDA, gerçek zamanlı uygulamalar

ACKNOWLEDGEMENT

I want to thank to my supervisor Assist. Prof. Dr. Adnan Özsoy for his continuous believe in me in the creation of this work. He created time for me every time I needed, even in his most busy times. He gave his energy and will power to me for this work. Without his support, insight and experience this work could not be this successful even this work might not be finished.

I also want to thank to my family, friends and colleagues. They gave ideas, they gave power and energy to me. They supported me until the end.

TABLE OF CONTENTS

FIGURES

TABLES

ABBREVIATIONS

O	Big O notation for algorithm time complexity.
	For example: $O(n)$, $O(n^2)$, $O(n \cdot log(n))$
CPU	Central Processing Unit
GPU	Graphical Processing Unit or Graphics Processing Unit
GPGPU	General Purpose Graphical Processing Unit
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
SMT	Simultaneous Multi-Threading
FPS	Frames Per Second
MS	Milliseconds
BSP	Binary Space Partitioning
BVH	Bounding Volume Hierarchies
CGI	Computer Generated Imagery
MHz	Megahertz
GHz	Gigahertz

1. INTRODUCTION

With the rise of the internet, video sharing platforms, social media, interactive applications and videos visual contents are heavily produced and consumed. Every day humanity spends countless of hours watching videos and consuming social media content. Great proportion of these applications use computer generated imagery to increase realism and immersion. They use computer generated imagery from simple cut, blend effects and filters to realistic three-dimensional animated characters and scenery. These improvements increased qualities of the content that is shared in social media, video sharing platforms and video games.

Physics simulations occupy an important part of the CGI industry. While generating computer generated imagery, physics simulations are heavily used. Those simulations are used for explosions, collisions, water and smoke simulations, breaking and deforming bodies. Also, support CGI effects for various stunts and crashes. So, increase in content creation rate often causes an increase in need for physics simulation solutions. Because of this growing need for physics simulations there is a growing need for improved physics simulations. Three of the important needs of users in physics simulations are visual quality, performance and ease of use.

By looking with a visual quality perspective movie industry is a constantly growing industry where lots of money is invested to improve quality of the images and they increased their usage of computer-generated imagery to perfect the expected results for users. Because of the huge investment in the area and remarkably high interest from developers working on it, the visual quality increased tremendously. Since increasing visual quality will cost more time and more money, it became infeasible for amateur or inexperienced users to create content that offers same amount of visual quality.

Another important subject is performance of the application. Using a simulation that takes long to compute is not affordable for every user. While using an interactive application or rendering a video one of the main concerns is processing time. Most of the interactive application users or video content creators do not have access to a greater computing power. It is both costly and hard to support to use high performance computing systems. So, it is either not affordable to use a high-performance computer or not affordable to wait a home computer to complete the processing for most of the users of physics simulations. Especially for video games industry time constraints for computing a frame shrinking constantly. It has been shrunk to about four milliseconds in some cases and in that four millisecond the application must compute lots of key features, most important ones are animations, artificial intelligence, networking, rendering, game logic and all the physics simulations. Since there are lot of key features to compute in a frame, developers cannot spend much resources just for physics simulations. And every small improvement will help to create resources for other systems and with more resources other areas of the video game can be improved.

Ease of use for a solution is especially important it can propose lots of improvements but if it is harder to use, the contribution may not be valuable. For most of the professional industries there are lots of experienced users for physics simulation software. But for everyday users or inexperienced users, complex and bloated solutions are not easily usable. Most of the time their needs are bounded by a small part of their entire application or video.

Physics simulations can be coarsely divided into certain categories according to their target body type. They are rigid body simulations, smoke and fluid simulations and soft body simulations.

Rigid body simulations often focus on the physics bodies that are not deformable. They focus on computing various physics properties such as velocity, acceleration and momentum. After detecting collisions, they aim to compute resulting velocity and position of the shapes as realistic as possible. They often depend on conservation of the momentum principle, conservation of the energy principle and various kinds of equations and formulas that scientists have found such as static and dynamic friction, inertia computations, aerodynamics and air friction.

Smoke and fluid simulations often aim for simulating behaviors of gasses and fluids. They simulate diffusions and shape adaptations of these gasses and fluids also pressure computations of these are quite different from solids. In smoke and fluid simulations developers use the same principles of rigid body simulations use but in addition they also use other principles of fluids such as surface tension, density, viscosity and different pressure models. Also, for gasses they use properties such as compressibility and expandability, diffusivity and different pressure models for gasses.

Soft body simulations are a blend of these two simulation types. They aim for solid matter but instead of keeping properties of these solids they deform and break those bodies. They focus on material properties of physics bodies such as malleability of metals, viscosity of soft solid materials and breaking points of materials. Also, they focus on breaking pattern and behaviors of physics shapes.

In this thesis, we are proposing a soft-body physics simulation solution algorithm for deforming shapes that shows metallic behavior. At the event of physical collision, they bend and deform according to their given properties. Proposed solution is offering a soft body simulation solution that addresses these three prominent issues; visual quality, performance and ease of use. We define two versions for the algorithm; one works on CPU and other works on GPU hardware. We visualized the results for easier understanding. The proposed solution is aiming for creating a bigger user base by improving visual quality, performance and ease of use. The algorithm uses pre-computed data to look-up and uses basic linear shape matching technique to compute results. The results are promising, offering linear time complexity also offering a linear scaling factor depending on the core count of the hardware. Proposed solution can reach 1 million vertices for simulation in modern CPUs that contain 4 physical and 8 logical cores, and those results are mostly under 3 milliseconds. By using modern graphical processing units, the solution is capable of simulating 10 million vertices under 1 millisecond. Thus, offers a usable solution even in video games that have extremely limited time constraints. These extremely limited time constraints can be as limited as 4 milliseconds to render an application frame. To be able to be used in a real time application as a soft body physics simulation solution component, solution must generate results in sub real times.

2. BACKGROUND

2.1. Representing Three Dimensional Shapes in Computer Generated Imagery with Triangulation Approach

In fields such as movies, videos and video games three dimensional objects are heavily used. To be able to render any kind of object into screen many generalized approaches have been used. One of the used approaches is triangulation approach. In this approach every represented object consists of multiple triangles. Since theoretically there is no bound on how small a triangle can be and how many triangles can be used, any kind of shape can be represented with this approach. There are many kinds of approaches to represent a three-dimensional shape, but they are out of the scope for this work.

2.2. Physics Simulations

Physics simulation solutions are software that supplies ability to solve various physics problems. There are certain fields that physics simulation solutions are addressing, and they offer various features to use.

2.2.1. Collision Detection

Collision detection is one of the most important problems that physics simulations address. It is detecting if two collision shapes are intersecting with each other. Most of the collision detection systems split given complex shapes into certain primitive shapes. These shapes are triangles, planes, spheres, capsules, boxes and convex meshes in three-dimensional geometry. These shapes are called primitives because they can be represented by simple mathematical equations and by using these mathematical equations, algorithms to detect intersection can be developed. Couple of the important general-purpose collision detection algorithms and approaches are Separating Axis Theorem, Gilbert–Johnson–Keerthi distance algorithm and Minkowski Portal Refinement. For specific cases more specific algorithms can be developed such as capsule versus sphere collision.

2.2.1.1. Collision Detection Optimizations

These algorithms are mostly developed with performance in mind but checking every pair of objects with a collision detection algorithm creates algorithms that have time complexity of $O(n^2)$. Space partitioning techniques are used for these situations. Such as Binary Space Partitioning, Quadtrees, Bounding Volume Hierarchies.

To simplify the collision detection algorithms various preliminary collision shapes are used such as axis aligned bounding boxes and bounding spheres. Prior to execute expensive collision detection algorithms these approaches execute a coarse collision detection and offers an early out mechanism.

2.2.2. Collision Response

While detecting collision, these software computes every needed property of collision to handle collision response correctly. The purpose of collision response phase is altering the properties of collided shapes according to detected collision. This altering can mean different in different simulations, for rigid body simulation solutions this altering means only changing the velocity and momentum of the collided shapes. In soft-body simulation solutions velocity and momentum may be changed as in rigid body simulations. Also, shape of the object and mass of the object may be changed, even it can be fractured to smaller pieces.

2.2.2.1. Collision Response for Rigid Bodies

Main areas of research in rigid body collision response field are simulating massive number of objects, numerical stability problems in cases like stacking big number of objects on top of each other, correctly transferring momentum and velocity according to the rules of conservation of energy and conservation of momentum especially for fast moving objects.

2.2.2.2. Collision Response for Soft-Bodies

In the other field of rigid body simulations there is the soft-body simulations. Collision responses for soft-body simulations often focus on changing various properties of physics bodies mostly it is the shape of the object. In soft body simulation solutions, physics bodies often composed by smaller pieces. As mentioned in related work section, various simulations handle bodies as multiple small particles. Other solutions use triangles of the mesh or even some of them divide meshes into tetrahedrons or spheres. Because of the handling bodies as small pieces, the main challenge of the soft body simulation solutions is often not massive number of objects to simulate but massive number of pieces because using a greater number of pieces to simulate the body, allows simulation to be more realistic.

We can coarsely divide soft body simulations to two categories based on different main ideas and approaches.

There are realistic soft body simulations that try to implement physical properties as realistic as possible to do simulations as in the real world. And others try to simplify the computations by using approximations, various techniques and precomputed helper data.

2.2.3. Realistic Soft-Body simulations

These simulations are often based on real properties of materials and they use realistic energy transfer models to deform the soft bodies. To be able to be physically correct, these solutions execute computationally heavy calculations. So, they mostly do not fit to real time application constraints.

2.2.4. Approximate Soft-Body simulations

The other main category is approximated solutions. These solutions depend on a simplification idea for the physics simulations. There are solutions that use skeletons like in skeletal animations, there are some that uses springs or various joints to approximate the solution. Also, these simplifications can be used with the data driven solutions.

In data driven solutions there are examples. These examples are results of precomputed collisions. The most important challenges in data driven solutions are searching for the most suitable example, using that example properly and handling the artifacts correctly.

2.3. Data Driven Shape Matching

Data driven idea can be explained very simply. It is using couple of look up tables that happens to be meshes. And matching the initial vertices of the mesh with look up data's vertex positions with given blend weight. Matching process can be detailed but we will only use linear interpolation of two three-dimensional positions as in Equation 2.1.

$$V_{result} = V_{base} \times (1 - Weight) + V_{example} \times (Weight) \tag{2.1}$$

We call our data driven approach as Example Based approach. And we refer precomputed look up tables for precomputed collision result meshes as Examples. Example Based ideas often supply high performance. With other solutions you either do real physics computations or more computationally heavy collision approximations in run time. By using precomputed collision examples cost of simulation in run time can be reduced to faster than real time levels.

Example based physics simulations are mostly easier to implement and manage compared to fully physics simulations. They do not require complicated physics knowledge to properly implement.

Also, one of the crucial reasons to use an example-based solution is that user have more control over collision results. The solution only uses given examples. Results cannot be different than blended versions of given examples. In video games there are a lot of edge cases since video game player has the control. In various cases realistic solutions can result to unwanted deformations such as shown in Figure 2.1. Unwanted results do not necessarily mean unrealistic results. In various cases such as making the video game more fun the video game designers may not want to deform certain objects more than certain threshold. It is particularly challenging to alter the result in an unrealistic way in realistic solutions. And by doing that, user can break the solution in various edge cases and cannot know. It is risky to depend on robustness of unrealistically altered solutions. But if you are using example-based solutions all you must do is arranging given examples according to design needs. You have full control over simulation results.

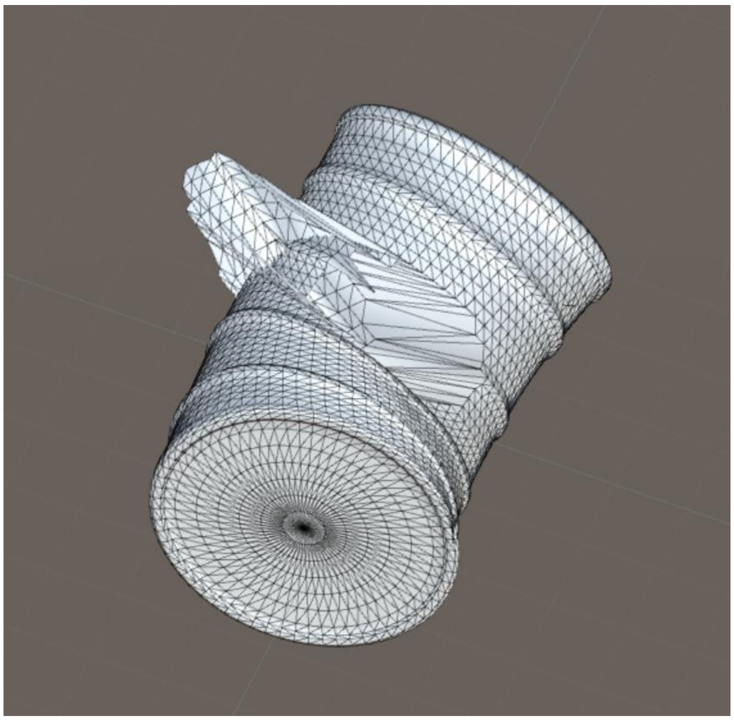


Figure 2.1. Unwanted Deformation Result

Example based shape matching is faster, controllable and less complicated to implement but there are some problems and limitations with it.

For cases that user did not supply a precomputed example collision result, the solution result will not be as plausible or realistic. To generate higher precision and more realistic results user needs to supply more precomputed example collision results. So, memory usage of the solution may increase. And with lots of examples run-time performance of the choosing the example phase will decrease too.

Example Based Shape Matching is suitable for real time applications such as video games, virtual reality applications or any other application that needs real time rendering of soft physics bodies. It supplies faster than real time solution, so user applications will have enough time left for other needed computations.
It is also usable in offline rendered applications such as movies. But since they do not have time constraints as strict as real time applications, they can use more complicated and realistic soft body physics solutions.

2.4. Parallel Programming Models

Parallel programming approaches can be explained in three main topics. Single instruction multiple data, Single instruction multiple threads and Simultaneous multi-threading.

- Single instruction multiple data (SIMD): Small amount of data processed in the same time in parallel. Often in the same thread and same processing core. Often called as vector parallelism.
- Simultaneous Multi-Threading (SMT): Decoupled instructions on different threads on different processing cores run in the same time in parallel. Often called hardware threading.
- Single instruction multiple threads (SIMT): Hybrid approach between vector parallelism and hardware threading.

2.4.1. Comparison between SIMD and SIMT

Both approaches depend on the idea of fetching one instruction and processing multiple data with that instruction. These approaches have their advantages and disadvantages over each other. Since SIMT works on threads it is offering different register sets, multiple control flow paths and can fetch from different addresses of memory freely. But SIMD is limited to same register set, none of the different parallel processing units can divert to different control flow paths and fetching is mostly limited to registers. Addressing to memory in a SIMD instruction is highly limited. SIMD offers lower latency while SIMT offers more flexibility.

2.4.2. Comparison between SIMT and SMT

SMT is the approach of multi-threading used in modern CPUs. Main advantage of the SMT is being able to do different works on different processing cores. And when work needed to be done is not enough and cannot occupy most of the processing unit it still tries to finish the work as fast as possible. In this kind of processing units there are lots of complex features are used such as out of order execution, branch prediction, speculative execution, register renaming, caches and prefetching. Aim of all these techniques and features is maximizing throughput without context switch to another thread. By these techniques and features SMT is supplying full flexibility over different threads without sacrificing much from performance. But these techniques and features makes every processing unit expensive, hard to produce and improve. So, because of their complexity and higher production cost per processing core in processing units created to work with SMT approach cannot have many numbers of processing cores compared to processing units that created to work with SIMT approach. In current generation of CPUs often processing core counts are lower than sixteen. There are other greater CPUs that have about hundred processing cores. But they are not affordable compared to SIMT units that have same count of processing cores [28].

In the other hand SIMT sacrifices flexibility over different threads but massively simplifies the implemented architecture over the processing unit. With this simplification, processing units that created to work with SIMT approach can have vast number of processing units, vast number of register sets. With the vast number of processing units and register count main aim of this approach is the generate gain from having vast number of threads working at the same time. Since this architecture hardware do not have complex techniques and features to deal with, such as handling stalling and increasing cycle per instruction value, these SIMT hardware depend on fast context switches between threads. If there are enough threads to context switch, these SIMT hardware do not lose performance. In cases that do not have enough threads to occupy whole hardware, performance drops greatly. Since SIMT approach executes same instruction on different threads at the same time in the cases that there are lots of control flow divergence and different threads goes into different control flows performance also reduces greatly.

Modern GPUs uses single instruction multiple threads (SIMT) approach. Modern GPU hardware contains many thousands of processing cores [26] with a very affordable costs compared to CPUs that have significantly less processing core counts.

2.5. Effectiveness of Graphical Processing Units

GPUs have lots of processing units. Every one of them can be slower than the average CPUs' processing units but since there are thousands of processing units on them [26] it is possible to process thousands of independent data on different processing units on GPUs at the same time. Modern GPU’s uses single instruction multiple threads (SIMT) approach. For algorithms that are doing same independent computation over a large data set GPUs are best fit. If these computations depend on each other and a synchronization between GPU threads needs to be done, full effectiveness of GPUs cannot be used. There are many framework and platforms to create opportunity for using full effectiveness of GPUs.

2.6. NVIDIA CUDA

NVIDIA CUDA is an application development platform that opens ways to use NVIDIA’s CUDA compatible graphical processing units. It allows developers to use GPUs as general-purpose graphical processing units (GPGPU). CUDA allows developers to create applications not using traditional rendering pipeline and let them access to many features of the GPU such as different memory types on the hardware, customizable block and grid sizes, certain level of synchronization over different threads. In CUDA, CPU and GPU are called as HOST and DEVICE respectively.

By offering these feature CUDA is opening ways use full effectiveness of modern GPUs.

2.7. Compute Shaders

Compute shaders differ from other shaders that are found in various graphics frameworks. Often shaders have certain sets of inputs and outputs and order of the execution is fixed in the rendering pipeline. With the era of the general-purpose GPUs compute shaders have arrived. They offer an interface to GPU and open ways to use many features of GPUs. They are highly customizable, they do not depend on any other part of the rendering pipeline, but they can be integrated into rendering pipeline. These features increase possibilities of GPU usage fields tremendously. In a sense NVIDIA CUDA and compute shaders of various rendering frameworks such as OpenGL and DirectX are remarkably similar in many ways.

2.8. Real-time application constraints

A real-time application can be defined as an application that responses in the time frame that users sense as instant. To generate this feeling generating about thirty frames per second for application is enough, this means that application should do every computation in about thirty-three milliseconds of time. Thirty frames per second creates the feeling of instant only visually, in video games since users also expect their input to be processed instantly, their expected latency is mostly lower. So, sixty frames per second is widely accepted as target frame count to process in a second for video games. It shrinks the time frame for processing single video game frame to be computed only about sixteen milliseconds. With the rise of social media and streaming platforms in recent years professional video gaming have risen. Professional video game players often have more capability visually and physically. So, they want as low latency as possible to be able to be better at the competition. Because of this situation professional gaming hardware products are produced to handle up to two hundred and forty frames per second video gaming experience. To be able to keep up with a hundred and forty-four hertz of frame rate real-time application need to finish its tasks in about seven milliseconds. And at two hundred and forty frames per second, solution only has a little more than four milliseconds.

3. RELATED WORK

Soft body simulation is an active research field. We can divide the research field as realistic soft body simulations and approximate soft body simulations. For approximate soft body simulations literature has good amount of work. Most of the literature for approximate soft body physics simulation solutions, make use of mesh simplification techniques and other ways to simplify the solution to create satisfactory performance results without sacrificing realistic visual quality. Although the prior approach is more exact in terms of showing the real effect of the impact, the latter has the potential to simulate faster since there are less demanding physical calculations at run time. We mostly looked at approximated soft body simulation solutions in this section.

Physics simulations of deformable bodies dates to early days of the computer graphics field. One such work by Terzopoulos et al. [1], in which researchers pushed the limits for approximate physics simulations. Since the nature of soft body simulations are processing power demanding, notable examples of simplifications have created.

Barzel et al. [2] creates base idea for model simplifications and simplification bigger models by using constraints between smaller pieces. This work can be regarded as the revelation for the field. Following this approach many simplification method and approaches have been proposed.

One simplification example, using a skeleton by Capell et al. [3], by Kim et al. [4], by Liu et al. [5] and by Rumman et al. [6]. These approaches achieved plausible simulation results with skeleton based simplifications.

Using a simplified version of the original simulation mesh by creating a tetrahedral mesh used by Alliez et al. [7] and by Wojtan et al. [8]. Tetrahedral mesh simplifications also used for liquid simulations by Ando et al. [9].

Clavet et al. [10] is one of the notable examples on dividing bodies into small components. It connects and disconnects components at run-time to create more fluid simulation look. Also works on rendering part of the fluid.

Example based approaches in another words shape matching is newer in this field. Müller et al. [11] is one of the influential works in example-based solutions. It only depends on the examples and simplifies the solution.

Wang et al. [12] shows that examples can be used as a helper component to whole simulation. In this work we can see examples as a helping component.

Müller et al. [13] uses both, spheres to divide the mesh into smaller components and uses example-based approach

Zhu et al. [14] is one of the inspiring approaches. Keeps examples as divided into tetrahedrons and does the shape matching in reduced tetrahedron space to gain performance.

Using lattice deformers to reduce needed computing power needs are used in by Rivers et al. [15] and by Patterson et al. [16].

Also, there are various methods that are not straight forward in soft body simulations. For example, Molino et al. [17], Irving et al. [18], Teran et al. [19], Budsberg et al. [20], and Saket et al. [21].

Dvorožňák et al. [22] proposes an example-based approach that aims to create two dimensional animations using previously hand drawn example drawings. This work perfectly fits our approach for examples and perfectly solves another problem in another field. But can be regarded as a proof of concept for our proposed solution.

Jones et al. [23] and Koyama et al. [24] are the most inspiring works for our proposed approach. They both create the idea and the baseline for this work.

The difference of our method is we created the solution idea according to SIMT parallelization approach and designed the baseline algorithm to work on GPU's as efficient as possible with SIMT parallelization approach. Our focus is to achieve collision result deformation simulation in faster than real-time. Although there are several earlier works that aims the real-time, our aim is to generate results in a small fraction of the real time budget. Our aim is to use this solution as a part to real time applications. Since real time applications do a lot of other computations, the budget we can get is rarely enough.

Our approach uses precomputed collision examples. At the time of the collision, the system finds the best fit from precomputed collision examples and blend the current state of the mesh to chosen example vertex by vertex. Approach does not use a mesh simplification technique that can create unpredicted results.

Blending vertex by vertex is a straightforward process so the solution can work with models that have millions of vertices. To achieve our faster than real-time goal we got inspired from earlier works and produced a solution that can work fully parallel on GPUs with the SIMT parallelization approach.

4. PROPOSED MODEL

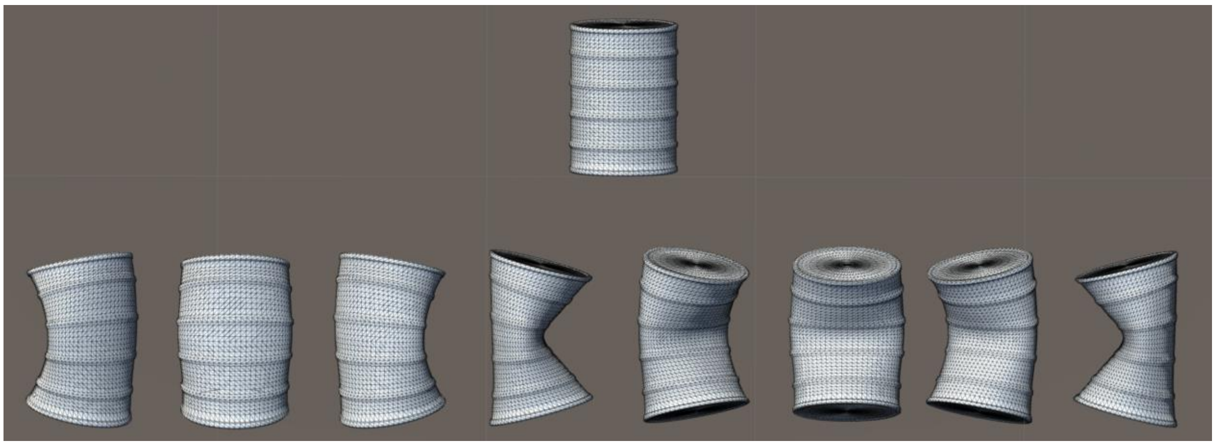


Figure 4.1. Base model and Example Shapes

4.1. Example based soft body physics simulation

As told in the introduction section we are proposing a solution algorithm that uses precomputed collision result meshes that aims visual quality, performance and ease of use. Proposed algorithm works with help of a collision detection tool. While the simulation taking place for each detected collision, proposed solution is getting called and generating results by using base model and examples.

In the Figure 4.1. base mesh shown at top and eight different precomputed collision result have shown at bottom.

Using the properties of the collision from collision detection, most similar precomputed collision result mesh is chosen among all of user have provided. After that proposed fully parallel algorithm deforms every single vertex that are affected according to precomputed collision result. Also, to not lose the results of earlier collisions proposed algorithm chooses the correct blending results for already deformed vertices.

We refer pre-computed collision results as examples. We are using them as examples to generate new collision results in the run-time. Supplying more deformation results will increase the simulation's visual correctness. And since proposed algorithm only blends supplied meshes it is not able to create results that cannot be predicted at the beginning. Results can only be in the borders of supplied precomputed collision results. This allows users to feel free while creating precomputed collision result meshes and do not require much tweaking to solve edge case situations. Since proposed algorithm only uses supplied precomputed collision result meshes and only using simple linear blending equation, it is not possible for it to produce unexpected results. This predictability increases maintainability and ease of use. Since every single computation done for each vertex is completely independent to other vertices, algorithm can be implemented fully parallel hence increases performance.

We implemented proposed algorithm both on CPUs using Simultaneous Multi-Threading (SMT) parallelism approach and on GPUs using Single Instruction Multiple Thread (SIMT) approach. In this section we will explain both implementations. Since the algorithm designed for SIMT approach it fits to SMT approach easily, so implementations are mostly similar on both architectures.

4.2. Pre-Solution Stage

This stage is not causally related with the algorithm, but it is about the supplying necessary data to the algorithm.

4.2.1. Creating the example shapes

Proposed algorithm requires examples to work with. A base mesh and multiple collision result mesh need to be provided by user. To create the example shapes users can use various other techniques such as they can create every single example by hand in a 3D modelling software or use a non-approximated soft body physics simulation application to deform base mesh and save results as examples. Creating by hand requires more time and effort but since the user will have full control over results, we encourage users to create their example shapes by hand and tweak them as their liking.

For base mesh and every other example shape, the vertex count, vertex order and vertex hierarchy need to be the same. Proposed algorithm interprets all data according to that assumption.

4.3. Initialization Stage

4.3.1. Pre-Processing example shapes

The straightforward approach to sending the example is sending every vertex of the model. Then blend base model and chosen example with each other. But to support multiple collisions we had to keep already collided vertices unchanged and should not blend them with the new example. At first, to know which vertices to blend we can keep a ledger, a bit array, for each example that shows affected vertices. In the deformation phase we read the bit array ledger to get if the current vertex is affected from the example. But for the SIMT implementation in the GPU this solution increases branching and causes divergency. Threads that assigned to non-deformed vertices were just waiting idle.

As an alternative we can only send deformed vertices of the example shape. This will help us on idle thread problem explained in earlier paragraph. To be able to send only the deformed vertices a basic mapping array will be built before keeping example shape data either in CPU or GPU. This solution will reduce the idle threads heavily. Also, will reduce the memory usage for keeping example shape data and shorten the time needed to move the example shape data to GPU memory.

Since this data is a three-dimensional vector for each vertex that is affected by example and one mapping data. We expect this data to be small.
For example: If we have a model that has a hundred thousand (100000) vertices. And have 100 different examples for simulation. If we assume that every example on average effects 50% of vertices. And we use 3 dimensional vectors of 4-byte floats for every single vertex and mapping index which is 4 bytes. If user is sure that vertex count is lower than biggest representable number by 2 bytes (65536) then mapping index can be 2 bytes. Even it can be 1 byte in low vertex counts.

Our data becomes:

For each vertex:

$$3 \times 4 \text{ bytes} + 1 \times 4 \text{ bytes} \approx 16 \text{ bytes} \tag{4.1}$$

For each example that containing 100000 vertices and 50% average effect

$$100000 \times 0.5 \times (3 \times 4 \text{ bytes} + 1 \times 4 \text{ bytes}) \approx 780 \text{ kilobytes} \tag{4.2}$$

For 100 examples

$$100 \times (100000 \times 0.5 \times (3 \times 4 \text{ bytes} + 1 \times 4 \text{ bytes})) \approx 78 \text{ megabytes} \tag{4.3}$$

For most of the modern systems this memory footprint is very affordable. These steps were the offline part of the solution. We do these steps once on the initialization stage and keep the precomputed data to runtime usage. Also, since examples does not change in any case, they can be shared between multiple instances of same base mesh. Even if the user is deforming multiple models, required memory for example shapes may be constant.

4.4. Runtime Stage

While the simulation is running for each collision detected, proposed solution algorithm does a deformation to collided mesh. We can get the collision result data from any kind of collision detection algorithm. Using the collision result it provided, proposed solution algorithm starts the processing. Runtime stage has three main phases.

- Finding the best matching example for the given collision.
- Sending the chosen example index and blend weight to hardware for applying the deformation.
- Applying the deformation using chosen example and blend weight.

4.4.1. Finding the best matching example for the given collision

At first by using the collision point we search for the closest vertex we use that vertex as collided vertex. Using the applied impulse, impulse apply time and mass of the collided vertex, proposed solution algorithm computes the needed displacement of collided vertex. To be able to do that it uses the Equation 4.4.

$$TargetDisplacement = \frac{Impulse \times DeltaTime}{Mass} \tag{4.4}$$

- Impulse is applied impulse vector as the result of the collision.
- Delta Time is time elapsed between start time and end time of the applied impulse.
- Mass is the mass of the collided vertex.
- Target displacement is the target displacement vector for the collided vertex for proposed algorithm. It will try to choose the best fit precomputed collision result mesh according to this value.

As seen on Figure 4.2. Find best example and blend weight algorithm we use the cosine similarity formula of three-dimensional vectors and we use displacement magnitudes for computing the outBlendWeight. Cosine similarity formula can be seen at Equation 4.5.

$$similarity = \cos\theta = \frac{V_1 \cdot V_2}{|V_1| \cdot |V_2|} \tag{4.5}$$

```
Algorithm: FindBestExampleAndBlendWeight
  Data: ImpulseData /* Contains Applied Force properties:  Point, Direction and
        Magnitude                                                              */
  Result: Chosen Example index and Blend Weight
  /* dispData contains Collided Vertex and TargetDisplacement                  */
  $dispData$ ← GetCollidedVtxDisp($ImpulseData$);
  for $i \leftarrow 0$ to $ExampleCount$ do
      if $CurExampleAffectsVtx(collidedVtx)$ then
          $similarity$ ← GetCosSimilarity($collidedVtx$, $exampleVtx$);
          if $similarity > biggestSimilarity$ then
              $biggestSimilarity \leftarrow similarity$;
              $outExample \leftarrow i$;
              $outBlendWeight \leftarrow targetDisp\ /\ exampleVtxDisp$;
          end
      end
  end
```

Figure 4.2. Algorithm for finding example

To clarify the blend weight computation; If we need a displacement that is 0.8 meters but in example that vertex is displaced 2.0 meters. So, we need to use 0.4 as the blend weight to be able to displace by needed distance.

$$neededDisplacement = exampleDisplacement \times blendWeight \quad (4.6)$$

So, to be able to find the blend weight the Equation 4.6. becomes:

$$blendWeight = \frac{neededDisplacement}{exampleDisplacement} \quad (4.7)$$

4.4.2. Sending the chosen example index and blend weight to hardware for applying the deformation

For the GPU implementation the example data already in the GPU memory we sent the data at the time of initialization phase. Also, for the CPU implementation the data is in the accessible memory. At this stage we just need to send the index of the chosen example and blending weight from current state of the model to given example shape.

Until this step SMT implementation on CPU and SIMT implementation on GPU works similar and all earlier computations are done on CPU in single thread. After this step is done the system executes a parallel for in the CPU for SMT implementation and executes a GPU kernel for SIMT implementation to deform the current model.

4.4.3. Applying the deformation using chosen example and blend weight

Using the results of Find best example and blend weight algorithm seen on Figure 4.2. our system gets the precomputed collision result mesh index. By using these data both implementations blend the current mesh to the example shape using the technique of linear interpolation of two three dimensional positions as in Equation 2.1. shown in section 2.3. Data Driven Shape Matching.

4.5. The CPU implementation

Since the proposed model designed the SIMT approach in mind this implementation uses the full power of the CPU and SMT parallelization approach. None of the threads need to synchronize with each other so no waiting or locking needed.

Deform with example algorithm shown in Figure 4.3. is used for each vertex in this implementation. Deform with example algorithm is called for each vertex in a parallel for loop.

4.6. The GPU implementation

We implemented the GPU version using SIMT approach by using NVIDIA CUDA platform. None of the thread synchronization methods available is used so that full processing power of the GPU can be used. Our choice of NVIDIA CUDA is only arbitrary. Proposed solution algorithm can be implemented by using OpenCL or DirectX’s and OpenGL’s compute shaders or any kind of GPU platform and framework.

Deform with example algorithm shown in Figure 4.3. is used for each vertex in this implementation too. In the deform kernel deform with example algorithm is called for every single thread.

```
Algorithm: DeformWithExample
  Data: ThreadVtxId
  $realVtxId \leftarrow MapRealVtxId(ThreadVtxId)$ ;
  $resultPos \leftarrow Lerp(mainVertices[realVtxId], example[ThreadVtxId], weight)$ ;
  $resultDisp \leftarrow Subtract(resultPos, baseVertices[realVtxId])$ ;
  $curDisp \leftarrow Subtract(mainVertices[realVtxId], baseVertices[realVtxId])$ ;
  if $LengthSq(resultDisp) >= LengthSq(curDisp)$ then
      $mainVertices[realVtxId] \leftarrow resultPos$;
  else
      $mainVertices[realVtxId] \leftarrow Lerp(mainVertices[realVtxId], resultPos, aFineConstant)$ ;
  end
```

Figure 4.3. Deformation algorithm

4.6.1. Usage of the Deformed Result Model

After we finished the deformation of the model usage of it depends on users aim of using this simulation system. If user’s aim is to use an offline system, they can save deformation result model for future use in any format they want.

For real-time usage most user will want to render the result deformed vertices using a GPU. Copying vertex data between CPU and GPU is overly expensive. Profile results are shown at 5. Test and Analysis section, it takes tens of milliseconds for certain vertex counts, as a result memory copy is not a possibility for real time applications. For our CPU implementation there is not an affordable straightforward solution, we must send vertices to GPU by copying the result deformed model data to be able to render. But for our GPU implementation there are fast solutions for quickly rendering result data. Since the result vertices computed by GPU the result data already in the GPU memory, we can use the data by passing data pointers. NVIDIA CUDA supports memory sharing between OpenGL and DirectX. Users can use these frameworks and assign those frameworks’ memory to CUDA as computation result memory. By that after our computations have completed the results become ready to render immediately. Similar cases also possible if proposed solution is implemented OpenGL’s or DirectX’s compute shaders. Those compute shaders can work on same memory with the rendering pipeline so there will not be any need for copying memory.

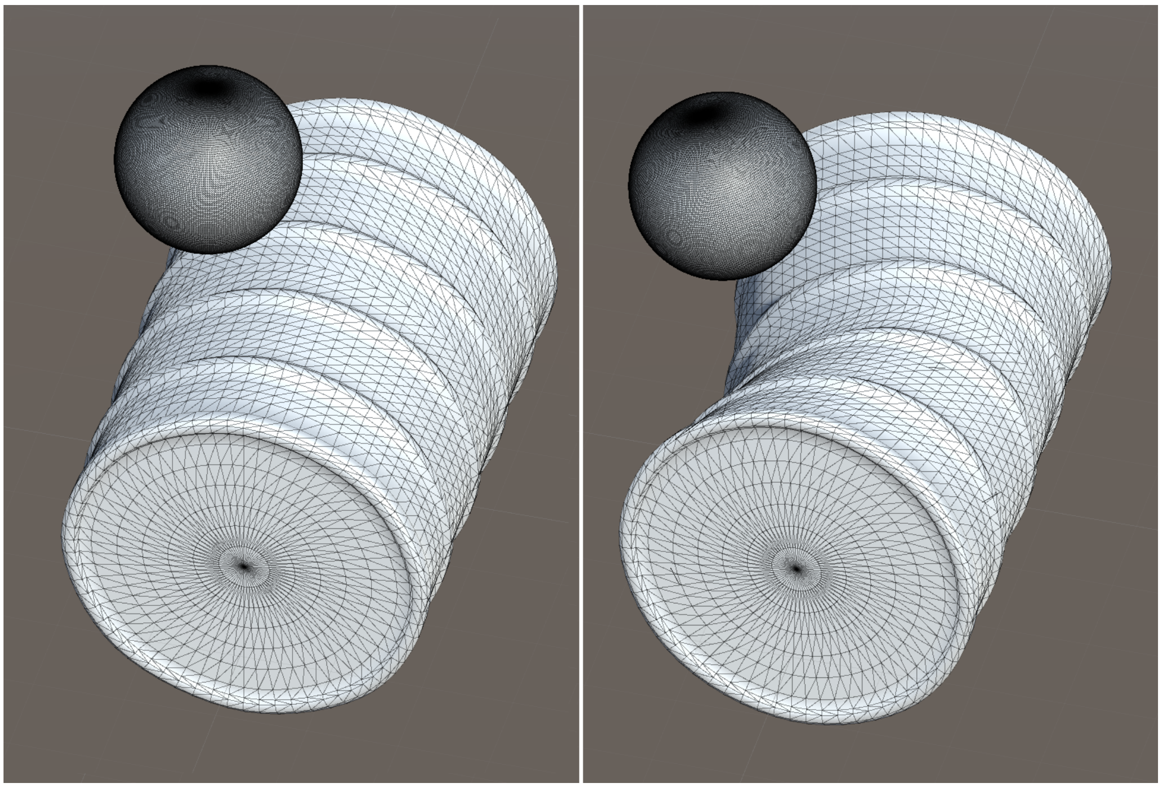


Figure 4.4. Before and After the deformation. Sphere at top hits to cylinder and deforms the cylinder according to collision properties and given examples of the cylinder.

4.7. Challenges

If a vertex needs to be blended but it was already blended to another example because of an earlier collision, we should not blend the vertex directly to the given example. It can cause that vertex's displacement to decrease. So, we had to keep track of the vertices that already blended to something else. For this info we used a bit buffer. One bit for each vertex that tells if given vertex is blended or not. But we needed to write to this bit buffer in parallel in a GPU thread. Such an instruction would require locking or atomic operation because multiple GPU threads can change the different bits in the same byte. If we do not use atomic operations these two threads might affect each other. There were two practical solutions to this. We could either synchronize the threads or use whole bytes for every different vertex to separate the access of the threads. Since memory was not our biggest issue, we have chosen the latter. We could easily use the prior solution in a lower memory situation. But then we realized we do not need extra data to understand if the vertex is displaced before. We do not need the exact knowledge. By computing the current displacement of the vertex comparing to undeformed original mesh we can easily say if the vertex displaced or not.

Since we can properly understand that if a vertex is displaced because of an earlier collision. We were able to produce ideas to solve multiple collision cases. At first, we tried blending these vertices current displacement with newly computed displacement from new example as seen in Figure 4.5. But this blending caused already displaced vertices to artificially lost their already displaced positions. Because of that kind of artifacts if a vertex is already displaced, we compare the new displacement and old displacement and we choose the biggest displacement for every point.

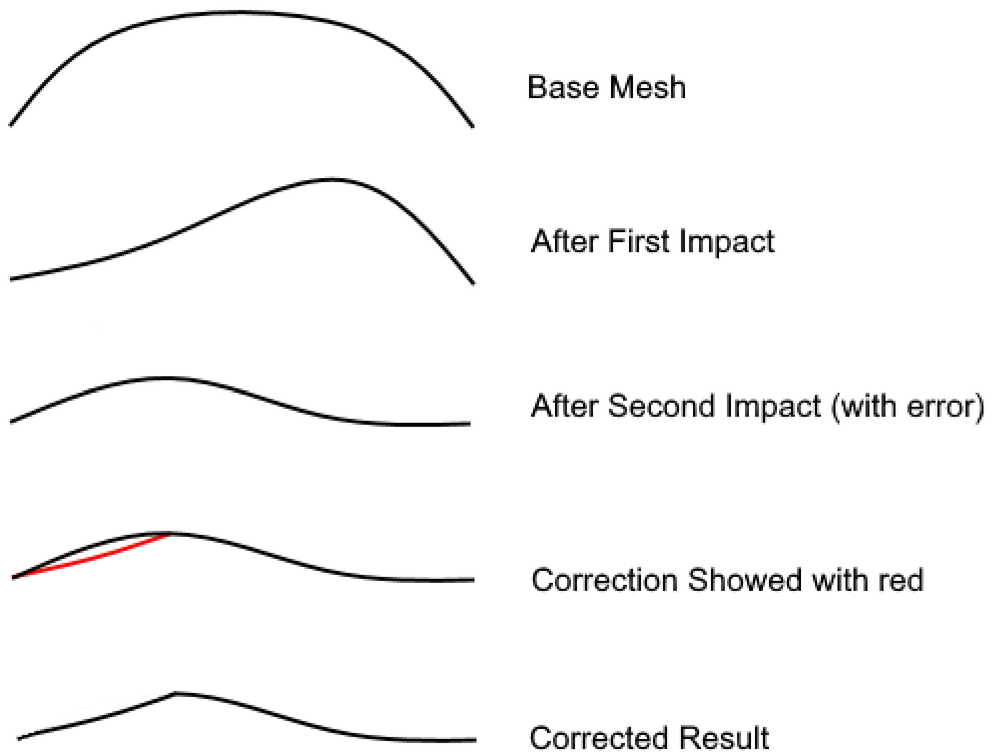


Figure 4.5. Correction for choosing most displaced candidate

But after we try proposed solution with more complex examples. Not blending and directly choosing the most displaced one between earlier displacement and newly created displacement have created different problems. After multiple blending towards different examples these artifacts can be seen in Figure 4.6.

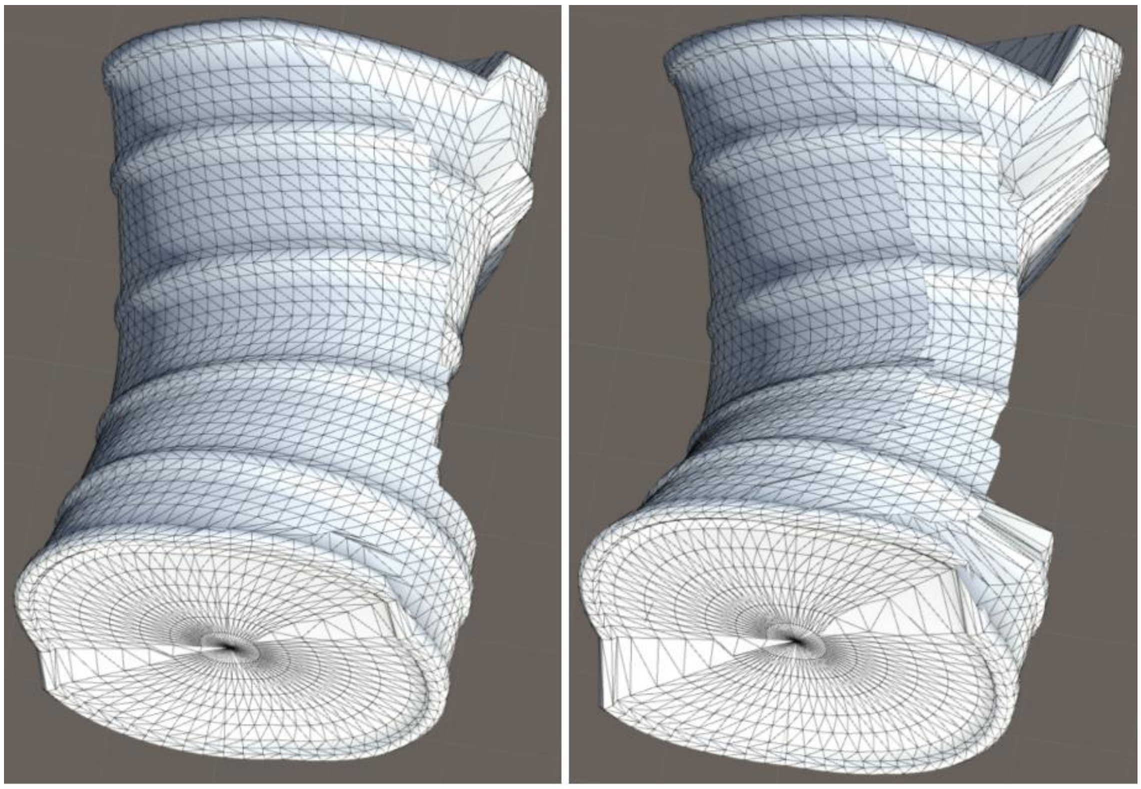


Figure 4.6. Result of blending multiple different examples choosing most displaced candidate

To be able solve this issue we had to fall back to the blending solution we initially tried. It created more stable solutions but created artifacts as the older blending of examples loses their effect.

In the Figure 4.7. we presented image that uses the blending solution. Figure 4.6. and Figure 4.7. are results of same simulations. Only difference is usage of the blending solution.

As a result, it creates more stable solution results, but solution results lose deformation effects of earlier collisions. As we can see at the right images in both Figure 4.6. and Figure 4.7. If we look at the left side of cylinder at both images we can easily see when we activated the blending solution left side of the cylinder lost the displacement.

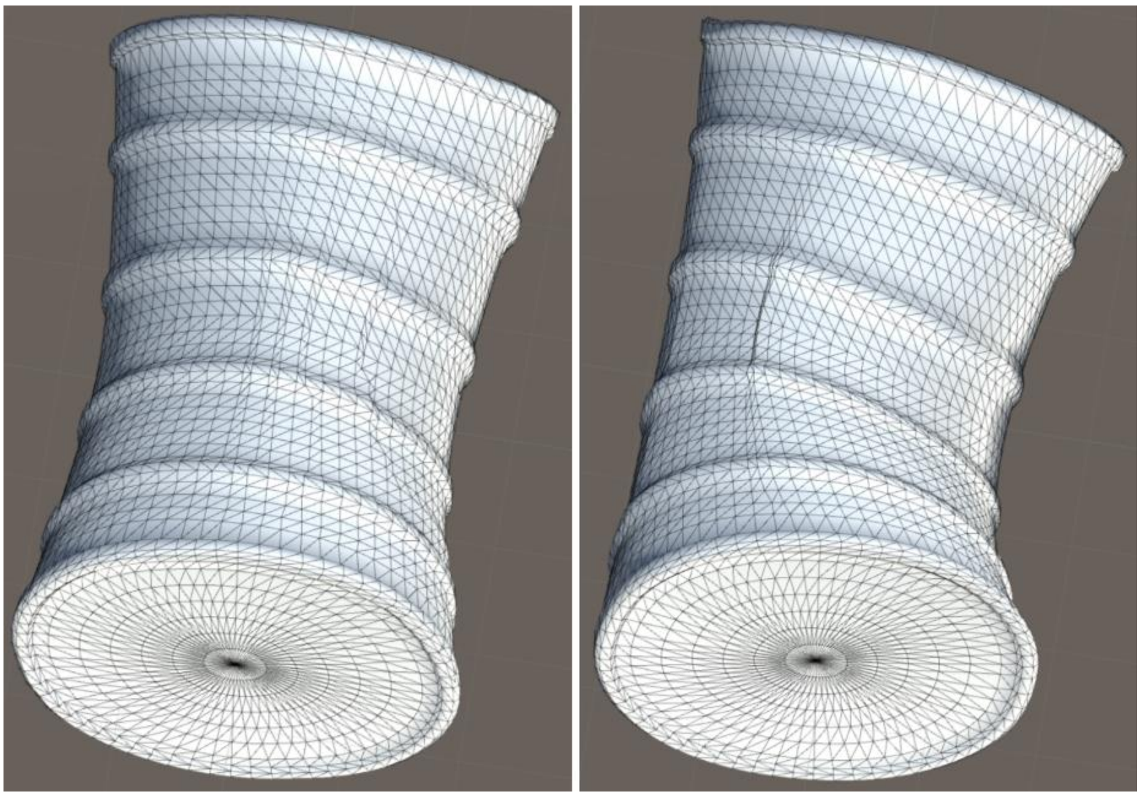


Figure 4.7. Result of blending multiple different examples and blending candidates

5. TESTS AND ANALYSIS

5.1. Test Implementation

We implemented various tests and profiling helpers for testing and analyzing proposed solution. We ran our test implementations on various hardware and gathered profiling results.

Test tool generates random meshes and random examples with various vertex count for them. It starts with 10000 vertex count and by increasing 10000 at each step it goes up to 10 million vertex count for GPU and goes up to 1 million vertex count for CPU. For each step it generates a completely new random mesh and random examples. For each step it does the simulation 75 times and gather the profiling results for them. To stay statistically correct it excludes the outlier values and do not add them to average.

For GPU's we did this analysis for different block sizes and gathered results. We also implemented a multithreaded CPU implementation for the algorithm to be able to compare and prove that the proposed solution can use the power of CPU too.

5.2. Common GPU Results

We did these tests on selected modern hardware. Since we implemented the solution on CUDA platform we had to run only on NVIDIA GPUs.

We used from NVIDIA GeForce, GTX960, GTX1050Ti, GTX1060(6GB version), GTX1660Ti, GTX1080Ti and GTX2080Ti for generating profiling results because about 50% of video game players are using same or similar GPUs [25].

In Figure 5.1. and Figure 5.2. we can see that highest ranked GPU's of last two NVIDIA GPU generation are performing around one millisecond for ten million deformed vertices.

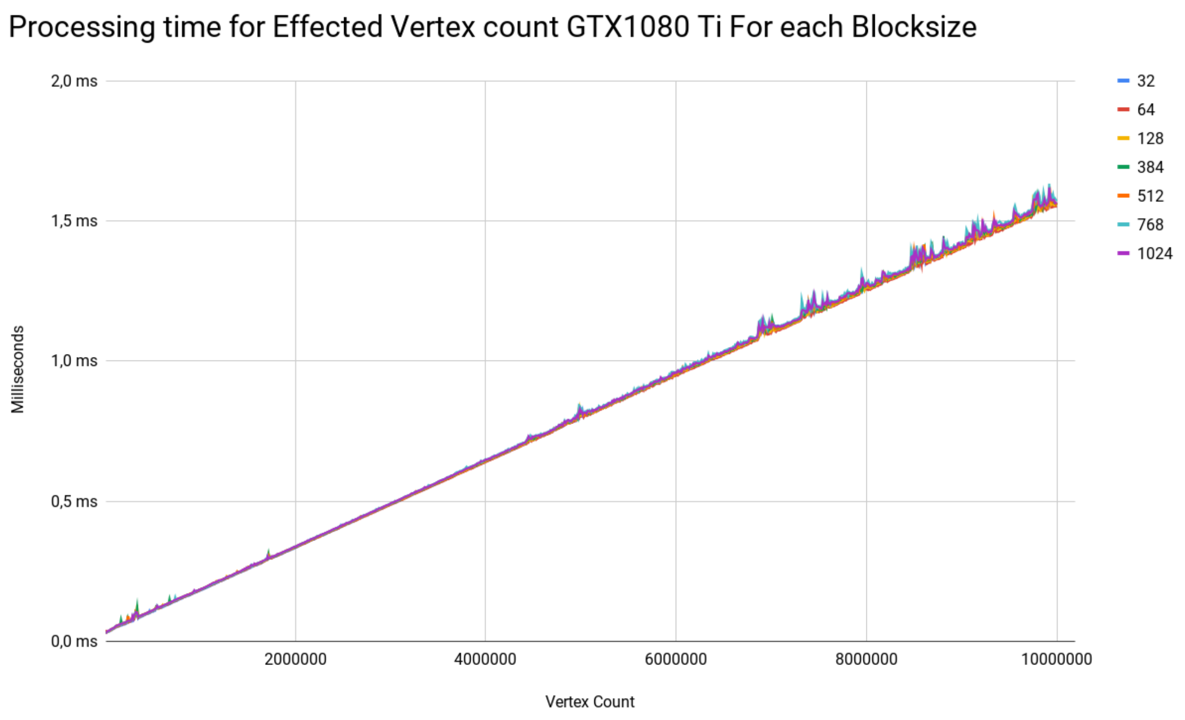


Figure 5.1. On NVIDIA GTX1080Ti GPU proposed solution for 10 million affected vertices takes about 1.57 milliseconds

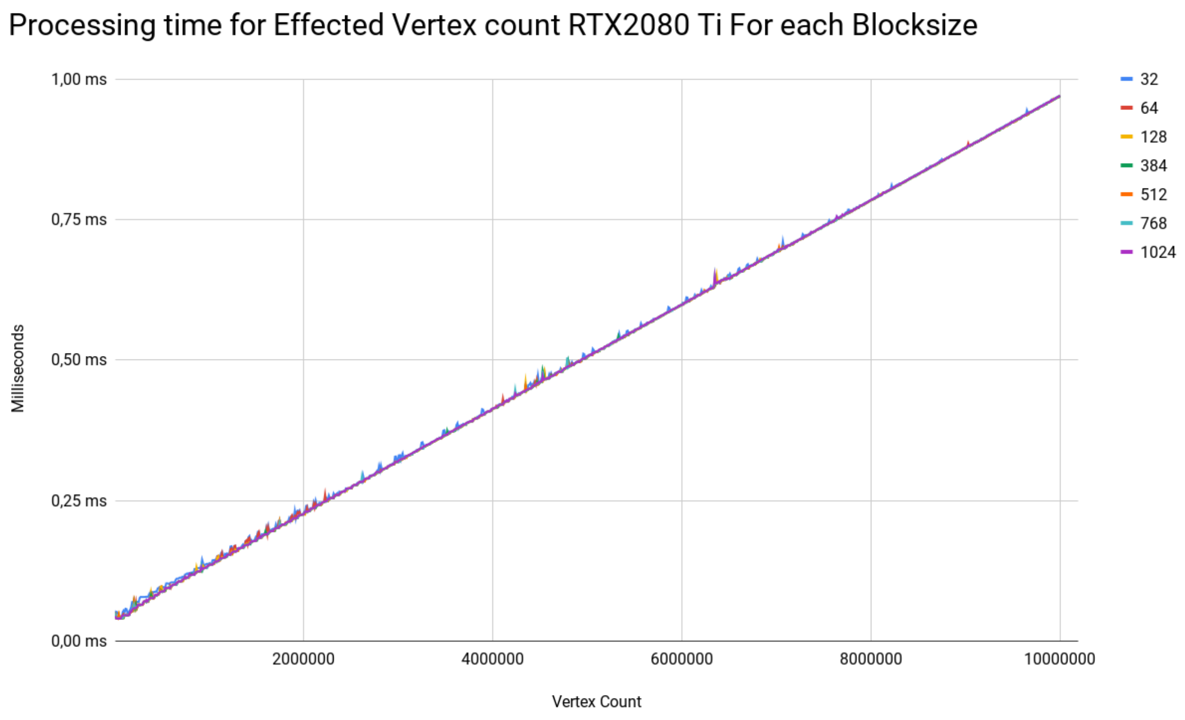


Figure 5.2. On NVIDIA RTX2080Ti GPU proposed solution for 10 million affected vertices takes about 0.97 milliseconds

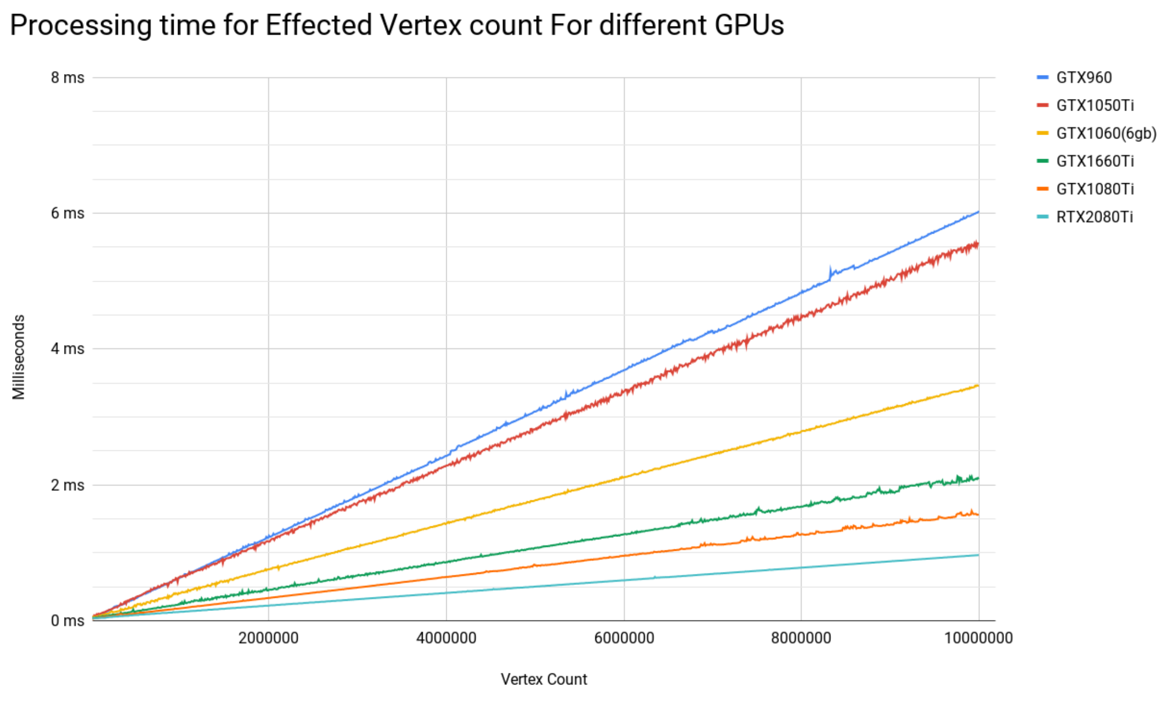


Figure 5.3. Comparison for different GPUs for Block size of 1024 until 10 million vertex count

In Figure 5.3. we can see the profiling result over various NVIDIA GPUs. Detailed profiling results shown at appendix section. Results for every GPU can be found there.

Table 5.1. Relation between core count, core speed and processing time for various modern GPU models

GPU Model	CUDA Core Count	Core Clock Speed (MHz)	Processing Time in MS for 10 million vertices
GTX 960	1024	1178	6.031723
GTX 1050 Ti	768	1290	5.551942
GTX 1060 (6GB)	1280	1506	3.467258
GTX 1660 Ti	1536	1500	2.088206
GTX 1080 Ti	3584	1582	1.566714
RTX 2080 Ti	4352	1350	0.970039

Table 5.1. shows that proposed algorithm scales related to core count and core speed. That means with the developed technology of the feature with GPUs that have more core count and faster cores, this algorithm will still use the near full potential of the GPU and scale linearly depending on new GPUs core count and core speed.

5.3. Common CPU Results

In Figure 5.4. and Figure 5.5. scaling behavior of the algorithm can easily be seen. In Figure 5.4. it is shown that related to the physical core count the slope of the graph changes near linearly. Table 5.2. also supports that proposed algorithm scales in the order of $O(n)$ time complexity depending on the physical core count. And proves that proposed algorithm scales perfectly between multiple CPU cores because we can clearly see a consistent performance increase according to physical core count increase supported by multiple performance test result graphs.

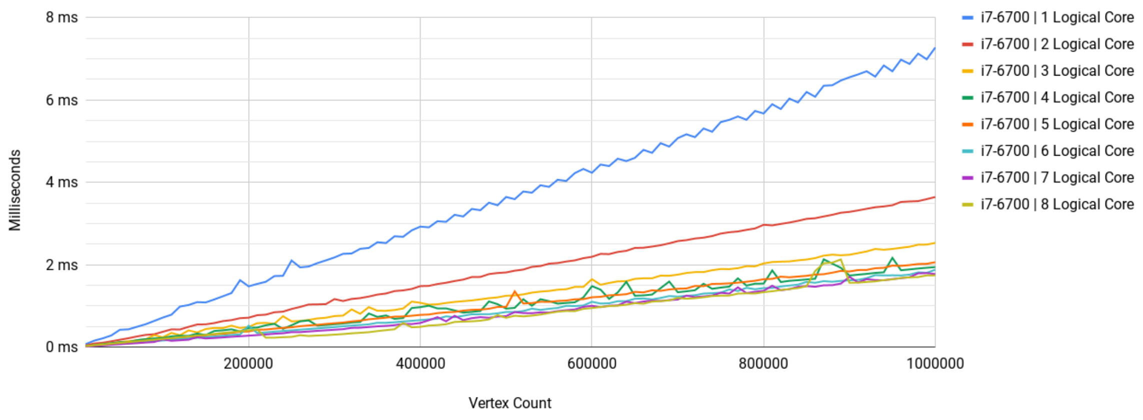


Figure 5.4. Scaling the algorithm over multiple logical cores

Table 5.2. Relation between core count and processing time for Intel® i7 6700 with 4 physical and 8 logical cores

Logical Core Count	Processing Time in MS for 1 million vertices	Time gain multiplier compared to 1 logical core
1	7.275815	x1
2	3.647204	x1.995
3	2.532911	x2.873
4	1.945698	x3.739
5	2.063421	x3.526
6	1.881683	x3.867
7	1.77668	x4.095
8	1.745447	x4.169

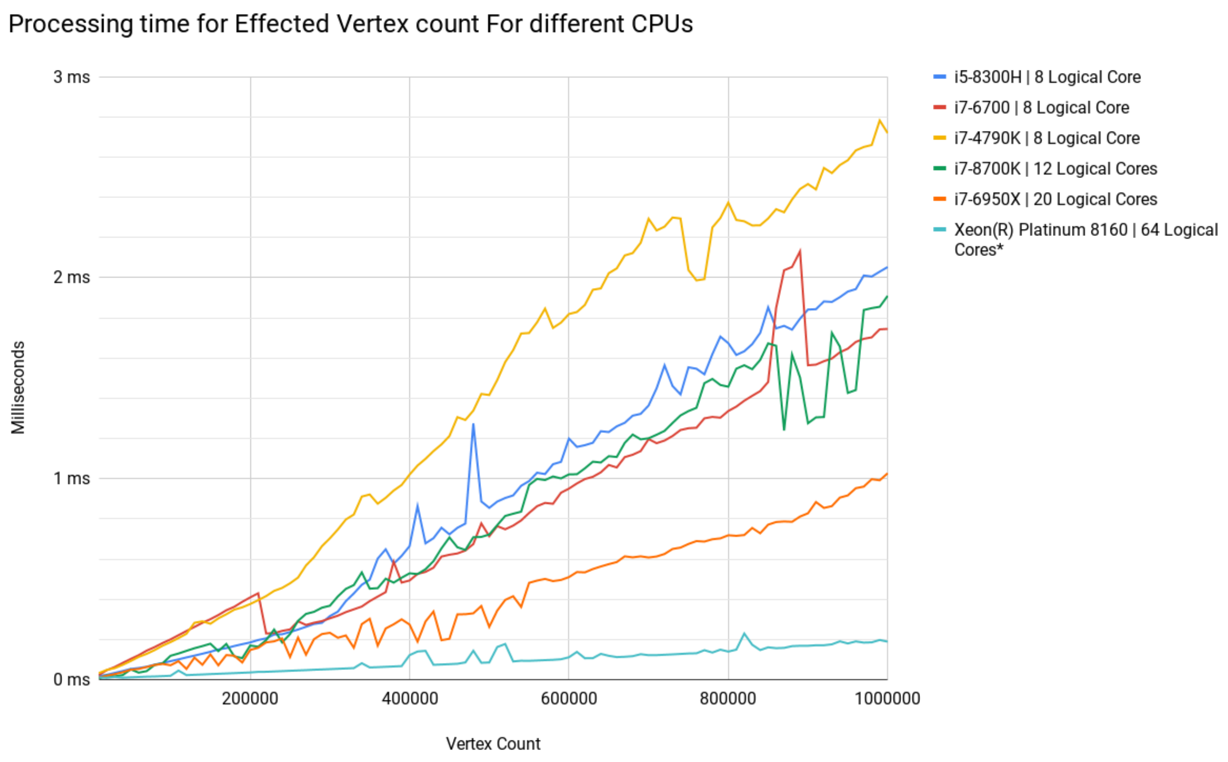


Figure 5.5. Comparison for different CPUs until 1 million vertex count

Table 5.3. Relation between core count, core speed and processing time for various modern CPU models

CPU Model	Physical - Logical Core Count	Core Clock Speed (MHz)	Processing Time in MS for 1 million vertices
i7-4790K	4 - 8	4000	2.718998
i5-8300H	4 - 8	2300	2.053142
i7-6700	4 - 8	3400	1.745447
i7-8700K	6 - 12	3700	1.909862
i7-6950X	10 - 20	3000	1.027708
Xeon Platinum 8160*	32 - 64	2100	0.190181

*Two Xeon Platinum 8160 processors are used. Each of these CPUs have 24 physical and 48 logical cores. That sums up to 48 – 96 however those CPUs were limited to 32 – 64 in total because of reasons we have not control over.

5.4. GPU to CPU Memory Copy

As stated in the 4.6.1. Usage of the Deformed Result Model section copying data from GPU memory to CPU memory takes tens of milliseconds. For real time applications copying memory from GPU to CPU is not a feasible possibility. Solutions should find ways to avoid doing memory copies from device to host memory.

Performance results for copying result from GPU memory to CPU memory can be seen on Figure 5.6.

These tests have done on NVIDIA GeForce GTX 1060(6GB) the have memory type of GDDR5, bus 192 bit and bandwidth of 192.2 GB/s and Intel® Core i7 6700 and Kingston DDR4 memory working at 1200MHz.

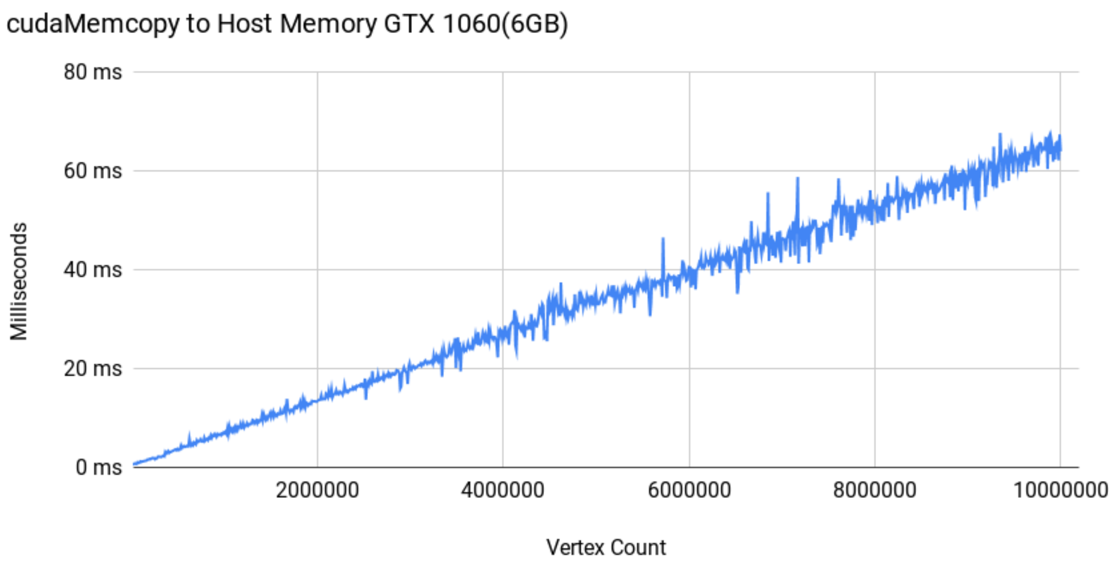


Figure 5.6. On NVIDIA GTX1060(6GB) GPU, copying 10 million vertices from GPU memory to CPU memory takes 64.001808 milliseconds

5.5. GPU and CPU Comparison

As expected mostly GPU implementations performed better than CPU implementations. But for one case CPU was faster than the GPU it is the comparison between an older GPU NVIDIA GTX 960 and modern overly expensive server CPU of Intel. Intel® Xeon® Platinum 8160. Firstly, we need to say that Xeon processor around 11 times expensive than GTX 960 GPU. According to amazon.com prices Xeon processor costs around 4.499.00$ and GTX 960 GPU costs 388.00$. We used two Xeon processes at the same time while measuring performance. Without considering the cost of motherboard and other hardware that is compatible with Xeon processor we can easily say that testing hardware that contains Xeon processor costs at least 22 times more than testing hardware that contains GTX 960. Used Xeon processor is a very high-end processor. Until around 5 million vertex count Xeon processor performed better than the GTX 960. But after 5 million with increased vertex counts GTX 960 dominated the Xeon processor because we started to fully utilize the GTX 960 GPU around 5 million vertex count. Detailed explanation and test results of utilization have shown at the GPU utilization tests section.

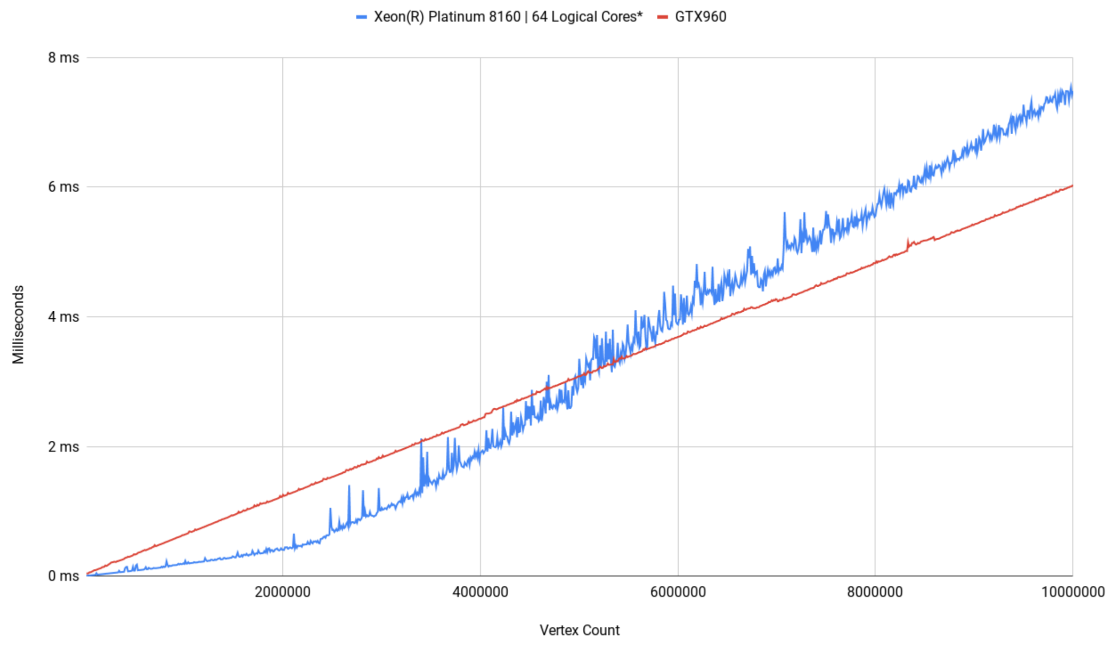


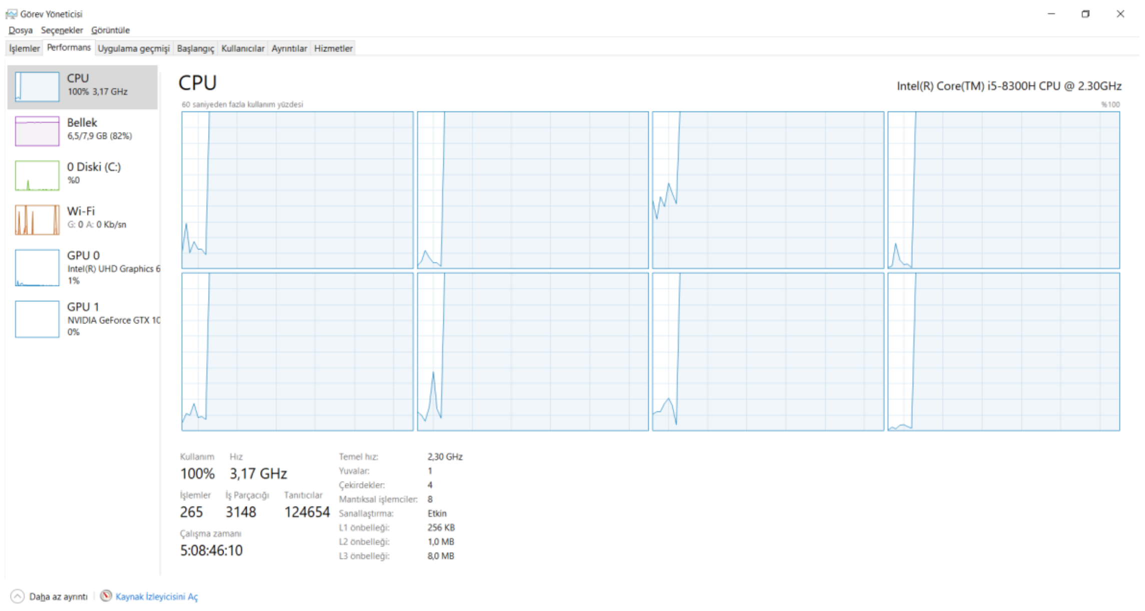
Figure 5.7. Comparison between best CPU and worst GPU of our tests

5.6. Hardware Utilization Tests

Utilizing the hardware is especially important. If an algorithm is using half of the potential of hardware it is working on, it is not completely correct to compare different hardware while testing the performance of the solution.

5.6.1. CPU Utilization Test

According to CPU utilization tests proposed solution utilizes all the logical cores on CPU. Even in Xeon processor that has 64 logical cores utilization was 100% it was rarely dropping down to 99% for some logical cores. This shows that our implementation uses the full potential of CPUs at least up to 64 logical cores. Since proposed solution was implemented according to SIMT parallelization approach there is not any dependency between threads, so it was expected to reach near perfect utilization on hardware that is produced for SMT parallelization approach. Utilization of 100% can be seen on Figure 5.8. for Intel® Core i5-8300H 4 physical 8 logical core processor.


Görev Yöneticisi
Dosya Seçenekler Görüntüle
İşlemler Performans Uygulama geçmişi Başlangıç Kullanıcılar Ayrıntılar Hizmetler
CPU 100% 3,17 GHz
Bellek 6,5/7,9 GB (82%)
0 Diski (C:) %0
Wi-Fi G: 0 A: 0 Kb/sn
GPU 0 Intel(R) UHD Graphics 6 1%
GPU 1 NVIDIA GeForce GTX 10 0%
CPU
60 saniyeden fazla kullanım yüzdesi
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
%100
Daha az ayrıntı
Kaynak İzleyicisini Aç

Kullanım 100%
Hız 3,17 GHz
İşlemler 265
İş Parçacığı 3148
Tanıtıcılar 124654
Çalışma zamanı 5:08:46:10

Temel hız:	2,30 GHz
Yuvalar:	1
Çekirdekler:	4
Mantıksal işlemciler:	8
Sanallaştırma:	Etkin
L1 önbelleği:	256 KB
L2 önbelleği:	1,0 MB
L3 önbelleği:	8,0 MB

Figure 5.8. All eight logical cores on i5-8300H spiked to 100% utilization when proposed solution algorithm started

5.6.2. GPU Utilization Tests

In hardware that are working with the SIMT parallelization approach utilization depends on having enough thread to utilize the system, having less control flow diversion and better memory access pattern and memory usage. All of these is for utilizing every core in the GPU constantly. If we analyze the proposed solution according to this data, its control flow diversion is low because we only have one control flow diversion point (if-else block). Since proposed solution does not access big memories latency because of memory accesses can be hidden when there are enough number of threads. After our GPU utilization tests, we found out that with the increased vertex count and increased thread count utilization of proposed solution on GPU increases.

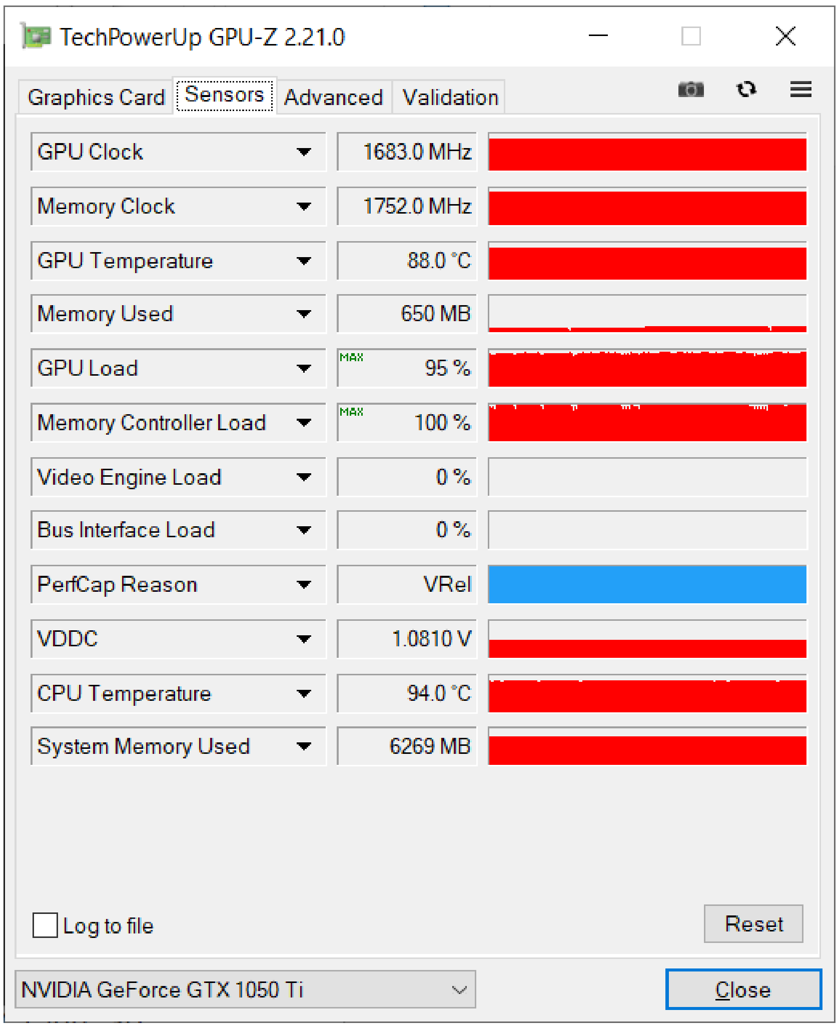
Table 5.4. Utilization compared to vertex count on NVIDIA GTX 1050 Ti GPU

Vertex Count	GPU Load
1 million	56%
2 million	69%
3 million	77%
4 million	88%
5 million	89%
6 million	92%
7 million	92%
8 million	93%
9 million	94%
10 million	95%

As shown in Table 5.4. proposed solution can use the potential of GPU in big vertex counts until 95%. Around 4.5 million vertices count the memory controller load of GPU hits 100% percent for NVIDIA GTX 1050 Ti for proposed solution. Because of this increase in GPU load cannot hit to 100%. With our memory access pattern in NVIDIA GTX 1050 Ti it is not possible to use more than 95% percent of potential computing power of the GPU. We only used the global memory to keep example shapes. We are accessing the global memory with an aligned and coalesced pattern, but we are not using any of the shared memory or constant memory. Without the power of different memory types our test environment can utilize the GPU at 95% of full potential. This situation is not bad for testing performance over different GPU models. Results will not differ greatly if we hit to 100% utilization. But it is good to use full potential of the hardware. By using shared and constant memories users can implement versions of our proposed solution algorithm that uses 100% potential of GPUs.

5.6.2.1. GPU Utilization Test software

All the utilization tests have done using the 2.21.0 version of the GPU-Z Video card Information Utility by the TechPowerUp.


TechPowerUp GPU-Z 2.21.0

Graphics Card | Sensors | Advanced | Validation

GPU Clock	1683.0 MHz
Memory Clock	1752.0 MHz
GPU Temperature	88.0 °C
Memory Used	650 MB
GPU Load	MAX 95 %
Memory Controller Load	MAX 100 %
Video Engine Load	0 %
Bus Interface Load	0 %
PerfCap Reason	VRel
VDDC	1.0810 V
CPU Temperature	94.0 °C
System Memory Used	6269 MB

Log to file

Reset

NVIDIA GeForce GTX 1050 Ti

Close

Figure 5.9. GPU Utilization test tool

5.7. Visual Tests

We implemented a testing tool to see the results of the cases that considerable number of collisions have happened. Our tool chooses random example and random weight for each test. Testing tool did 75 randomly created collisions to generate each result. These results can be seen in the Figure 5.10. and Figure 5.11.

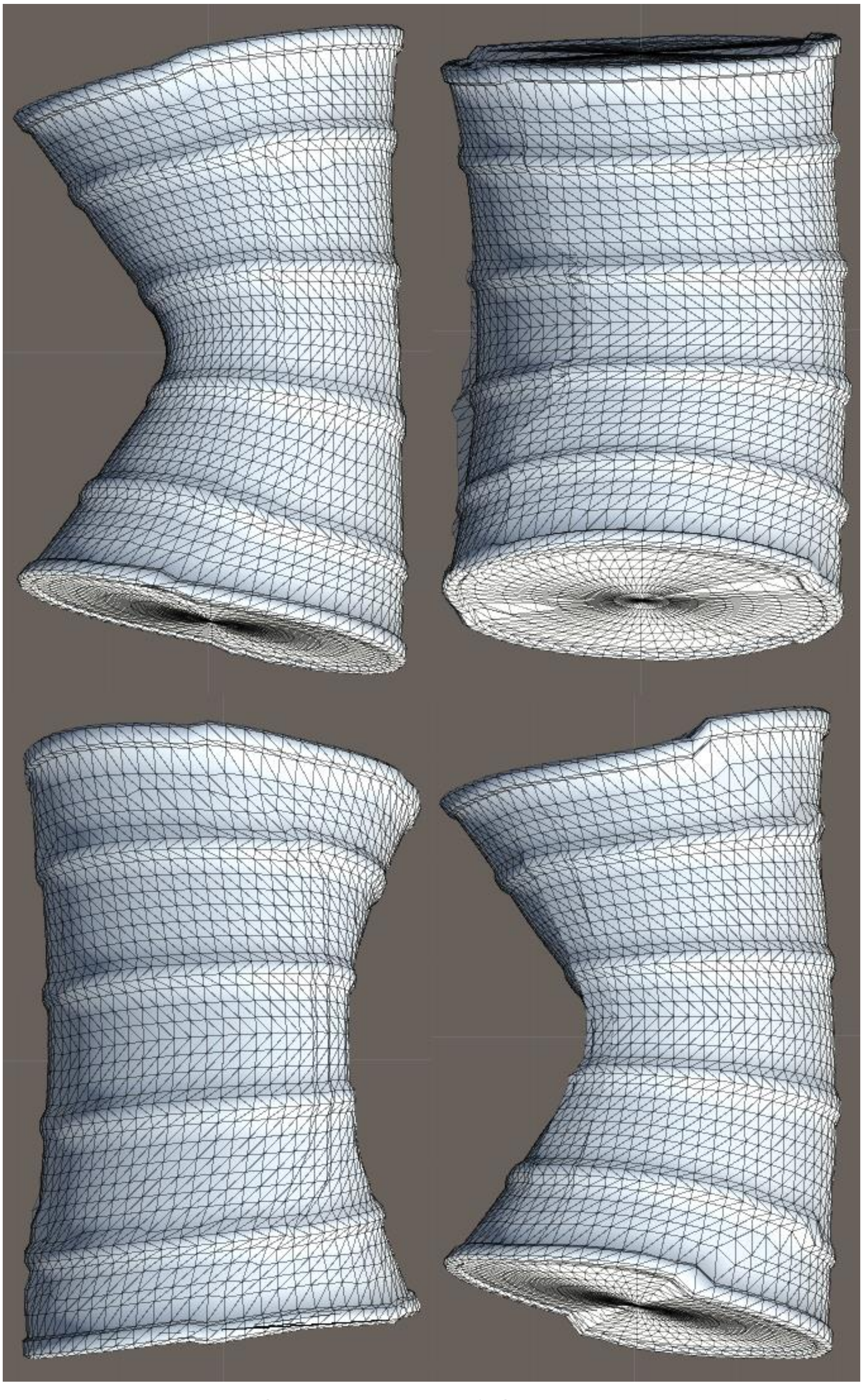


Figure 5.10. Results of Visual Tests

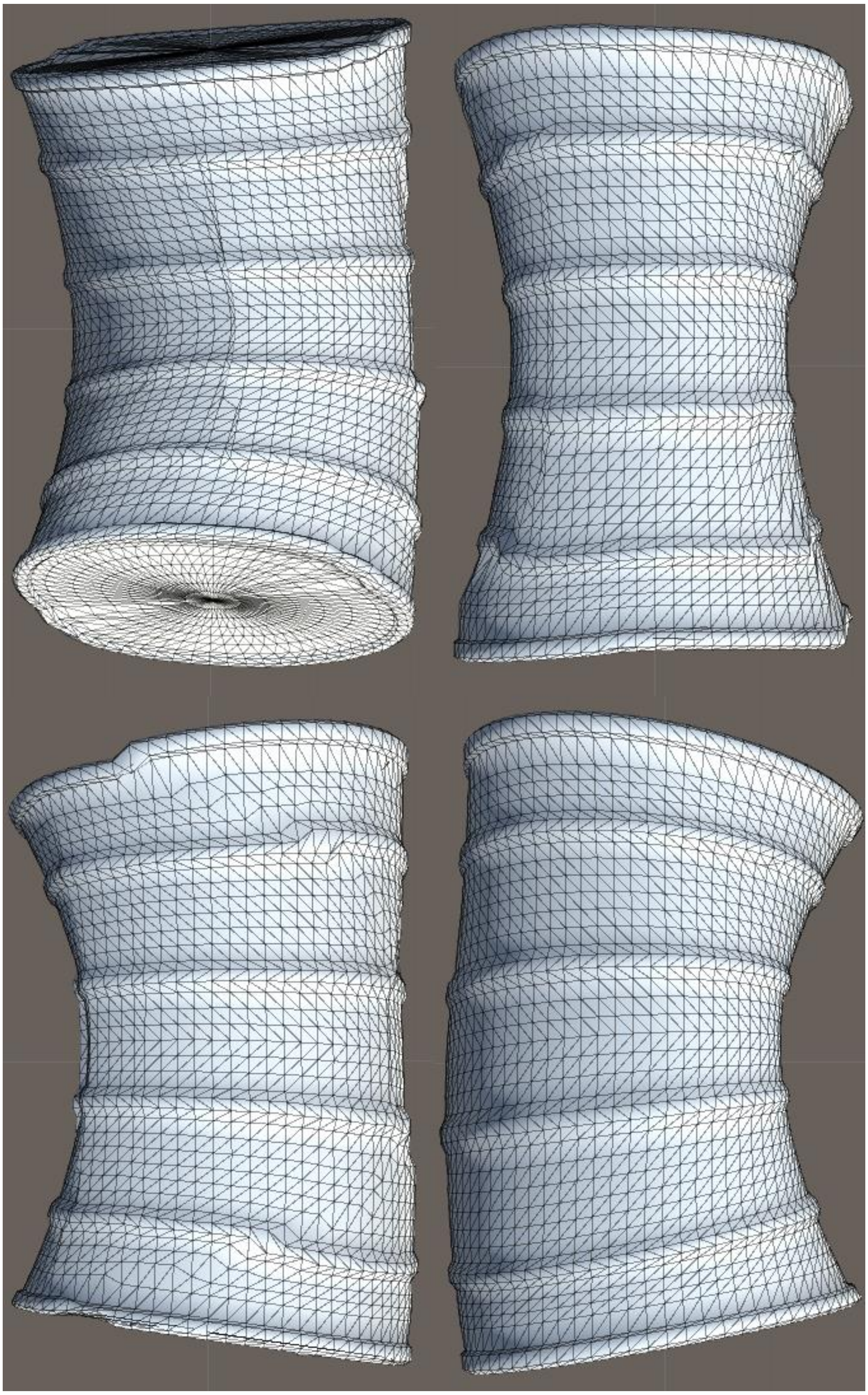


Figure 5.11. Results of Visual Tests

6. CONCLUSION

Proposed solution algorithm offers real time, visually pleasing and easy to use solution. With all implementation details shared it is easy to implement and simple to maintain. It is easily usable and can easily be implemented to an existing software as extension. Since it is producing predictable results and generating results fast enough for any kind of application, iterating and tweaking results going to be fast and painless process. This will allow users to spend more time for other important topics of their content and contribute to creation of more quality content.

In current generation of real time applications such as video games it is only feasible to render couple of million vertices in real time. According to results from GPU implementation proposed solution can deform 10 million vertices under one millisecond. Depending on the example if algorithm is deforming 10 million vertices full model can have much more vertices in various cases, more than 100 million. Having models that contain 100 million vertices in real time is not feasible in current generation of video gaming, but still proposed solution is able to deform them under 1 millisecond making solution feasible for using in time constrained video games. In current generation of video games single model mostly contains couple of hundred thousand vertex maybe 1 million vertices at most at best level of detail option. Of course, there are exceptions, it depends on the context of the video game and complexity of the scene. In the cases that model have 1 million vertices and given collision is deforming 50% of the vertices, 500 thousand vertices will be deformed. If we analyze the profile result for 500 thousand deformed vertex case from Table 6.1. We conclude that proposed solution perfectly fits to real time application time constraints. Even in older NVIDIA GTX 960 GPU solution converges in about 300 microseconds. To be able to spend whole millisecond in NVIDIA GTX 960 GPU we need to deform about 1.7 million vertices according to our profiling data. But in the newer NVIDIA RTX 2080Ti GPU we were not able to spend 1 millisecond even with 10 million deformed vertex count.

Table 6.1. Processing time for 500 thousand deformed vertices in various modern GPU models

GPU Model	CUDA Core Count	Core Clock Speed (MHz)	Processing Time in MS for 500 thousand vertices
GTX 960	1024	1178	0.333733
GTX 1050 Ti	768	1290	0.35219
GTX 1060 (6GB)	1280	1506	0.213597
GTX 1660 Ti	1536	1500	0.140712
GTX 1080 Ti	3584	1582	0.110823
RTX 2080 Ti	4352	1350	0.08893

According to data gathered in profiling and tests, proposed solution is usable in real time applications. But those results do not mean the solution only aims for real time applications. It is also usable for offline rendered and very demanding applications such as movies. Since they need long rendering hours, they use big computing power farms. These farms cost a lot and rendering process takes lots of time. With this algorithm computation time can be reduced. This will result to decreased costs and in case of any mistake it will be much easier to correct the mistake and re-render the whole movie scene.

Our CPU implementation is slower compared to GPU implementation. Since proposed solution scales according to physical core count and perfectly fits to the SIMT parallelization approach it was expected at the beginning. But in various cases user might want to implement their solution on CPU. To create an early idea about performance of the CPU version we did tests and supplied the results in this work. The only difference of result between GPU and CPU version is performance. There is not any visual difference between created results of GPU and CPU. It is perfectly safe to expect same result from other hardware. This may help users in various cases for example if they do not have access to GPU or they only have access to a fast GPU and slow CPU, and they are not able to run the proposed solution on both hardware.

This algorithm is usable for many industries ranging from real time video game industry to highly demanding offline rendered professional movie industry. For every industry it supplies an improvement.

REFERENCES

[1] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. SIGGRAPH Comput. Graph. 21, 4 (August 1987), 205-214.

[2] Ronen Barzel and Alan H. Barr. 1988. A modeling system based on dynamic constraints. IGGRAPH Comput. Graph. 22, 4 (June 1988), 179-188.

[3] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. 2002. Interactive skeleton-driven dynamic deformations. In Proceedings of the 29th annual conference on Computer graphics and interactive techniques (SIGGRAPH '02). ACM, New York, NY, USA, 586-593.

[4] Junggon Kim and Nancy S. Pollard. 2011. Fast simulation of skeleton-driven deformable body characters. ACM Trans. Graph. 30, 5, Article 121 (October 2011), 19 pages.

[5] Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. 2013. Simulation and control of skeleton-driven soft body characters. ACM Trans. Graph. 32, 6, Article 215 (November 2013), 8 pages.

[6] Nadine Abu Rumman and Marco Fratarcangeli. 2014. Position based skinning of skeleton-driven deformable characters. In Proceedings of the 30th Spring Conference on Computer Graphics (SCCG '14). ACM, New York, NY, USA, 83-90.

[7] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. 2005. Variational tetrahedral meshing. ACM Trans. Graph. 24, 3 (July 2005), 617-625.

[8] Chris Wojtan and Greg Turk. 2008. Fast viscoelastic behavior with thin features. In ACM SIGGRAPH 2008 papers (SIGGRAPH '08). ACM, New York, NY, USA, Article 47, 8 pages.

[9] Ryoichi Ando, Nils Thürey, and Chris Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. ACM Trans. Graph. 32, 4, Article 103 (July 2013), 10 pages.

[10] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. 2005. Particle-based viscoelastic fluid simulation. In Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '05). ACM, New York, NY, USA, 219-228.

[11] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. 2005. Meshless deformations based on shape matching. In ACM SIGGRAPH 2005 Papers (SIGGRAPH '05), Markus Gross (Ed.). ACM, New York, NY, USA, 471-478.

[12] Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James F. O'Brien. 2010. Example-based wrinkle synthesis for clothing animation. In ACM SIGGRAPH 2010 papers (SIGGRAPH '10), Hugues Hoppe (Ed.). ACM, New York, NY, USA, Article 107, 8 pages.

[13] Matthias Müller and Nuttapong Chentanez. 2011. Solid simulation with oriented particles. In ACM SIGGRAPH 2011 papers (SIGGRAPH '11), Hugues Hoppe (Ed.). ACM, New York, NY, USA, Article 92, 10 pages.

[14] Fei Zhu, Sheng Li, and Guoping Wang. 2015. Example-Based Materials in Laplace-Beltrami Shape Space. Comput. Graph. Forum 34, 1 (February 2015), 36-46.

[15] Alec R. Rivers and Doug L. James. 2007. FastLSM: fast lattice shape matching for robust real-time deformation. In ACM SIGGRAPH 2007 papers (SIGGRAPH '07). ACM, New York, NY, USA, Article 82.

[16] Taylor Patterson, Nathan Mitchell, and Eftychios Sifakis. 2012. Simulation of complex nonlinear elastic bodies using lattice deformers. ACM Trans. Graph. 31, 6, Article 197 (November 2012), 10 pages.

[17] Neil Molino, Zhaosheng Bao, and Ron Fedkiw. 2004. A virtual node algorithm for changing mesh topology during simulation. ACM Trans. Graph. 23, 3 (August 2004), 385-392.

[18] G. Irving, J. Teran, and R. Fedkiw. 2004. Invertible finite elements for robust simulation of large deformation. In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '04). Eurographics Association, Goslar Germany, Germany, 131-140.

[19] Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. 2005. Robust quasistatic finite elements and flesh simulation. In Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '05). ACM, New York, NY, USA, 181-190.

[20] Jeff Budsberg, Nafees Bin Zafar, and Mihai Aldén. 2014. Elastic and plastic deformations with rigid body dynamics. In ACM SIGGRAPH 2014 Talks (SIGGRAPH '14). ACM, New York, NY, USA, Article 52, 1 page.

[21] Saket Patkar, Mridul Aanjaneya, Aric Bartle, Minjae Lee, and Ronald Fedkiw. 2015. Efficient denting and bending of rigid bodies. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '14). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 87-96.

[22] Marek Dvorožňák, Pierre Bénard, Pascal Barla, Oliver Wang, and Daniel Sýkora. 2017. Example-based expressive animation of 2D rigid bodies. ACM Trans. Graph. 36, 4, Article 127 (July 2017), 10 pages.

[23] Ben Jones, Nils Thuerey, Tamar Shinar, and Adam W. Bargteil. 2016. Example-based plastic deformation of rigid bodies. ACM Trans. Graph. 35, 4, Article 34 (July 2016), 11 pages.

[24] Yuki Koyama, Kenshi Takayama, Nobuyuki Umetani, and Takeo Igarashi. 2012. Real-time example-based elastic deformation. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '12). Eurographics Association, Goslar Germany, Germany, 19-24.

[25] Steam Hardware & Software Survey,

https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam (Retrieved 6 July 2019)

[26] NVIDIA GPU specifications, https://www.nvidia.com/en-us/geforce/ (Retrieved 6 July 2019)

[27] NVIDIA CUDA documentations, https://docs.nvidia.com/cuda/ (Retrieved 6 July 2019)

[28] Intel® Core™ Processor specifications,

https://ark.intel.com/content/www/us/en/ark.html (Retrieved 6 July 2019)

APPENDIX

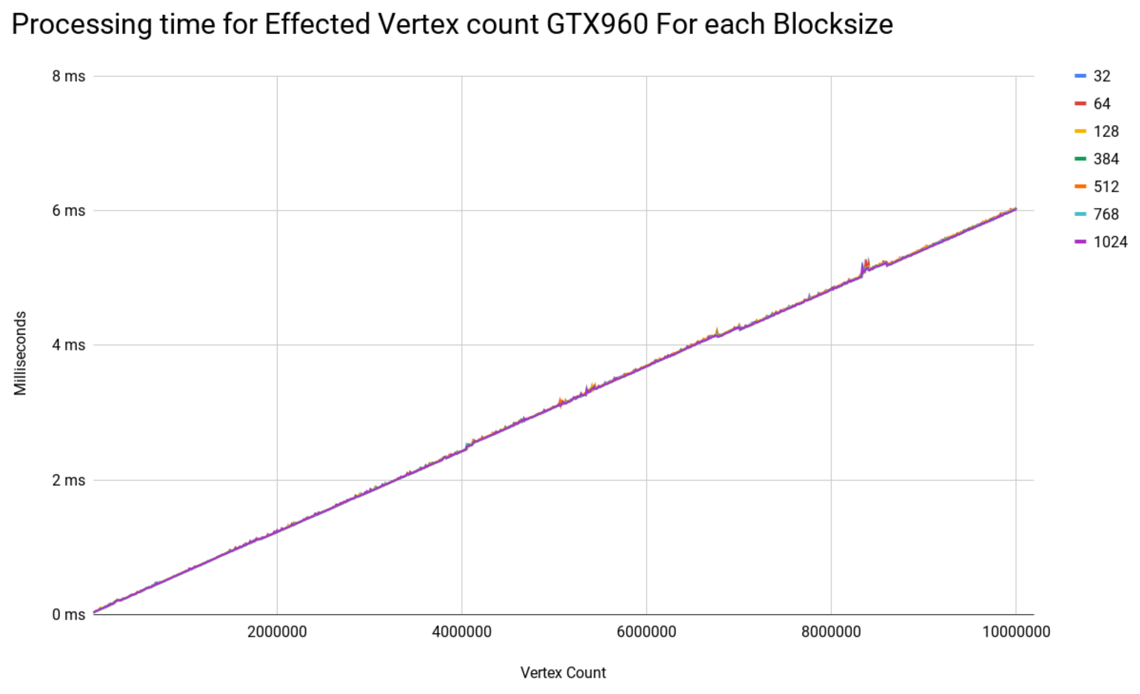


Figure 0.1. On NVIDIA GTX960 GPU proposed solution for 10 million affected vertices takes about 6.031723 milliseconds

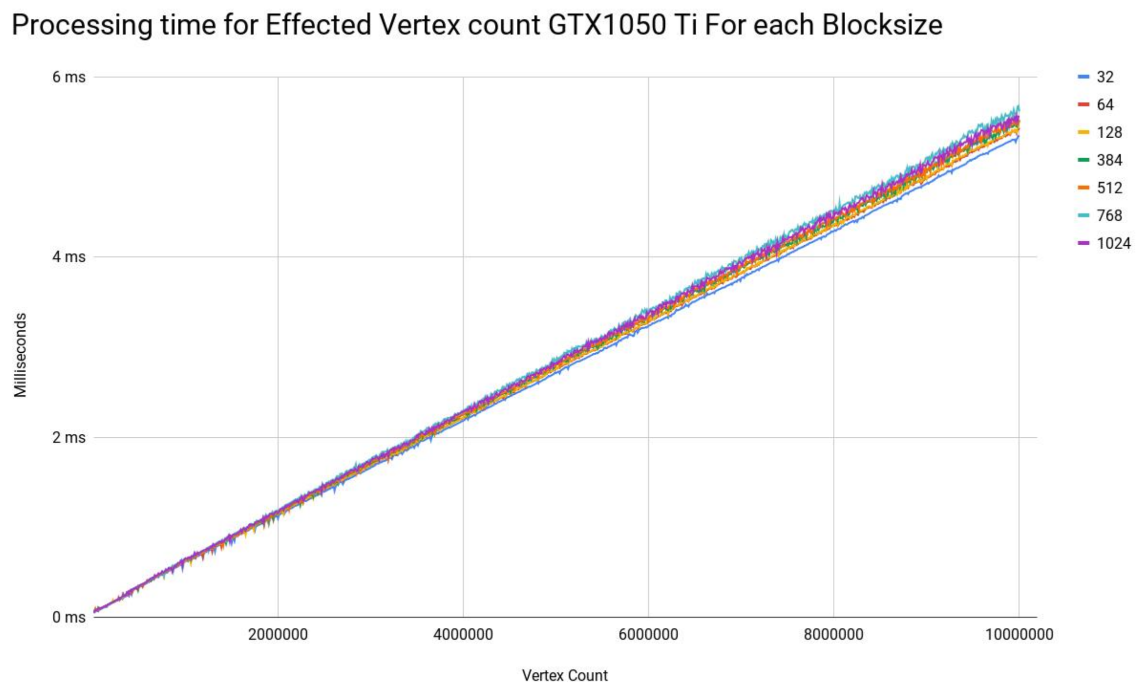


Figure 0.2. On NVIDIA GTX1050Ti GPU proposed solution for 10 million affected vertices takes about 5.551942 milliseconds

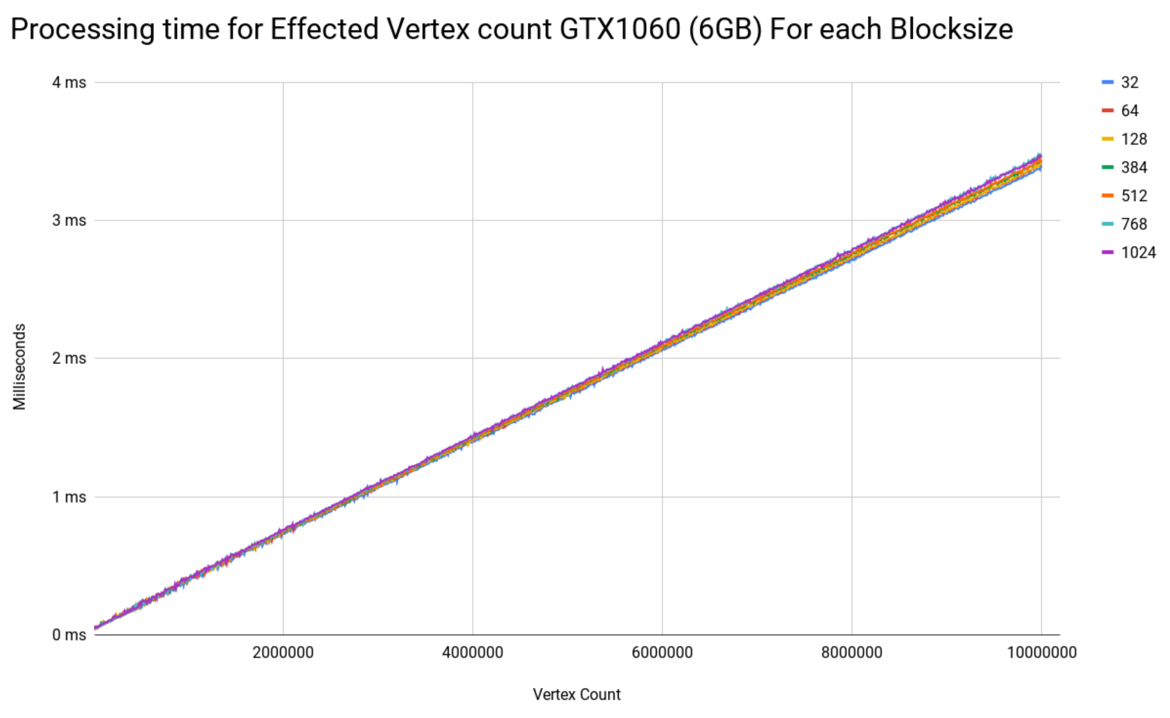


Figure 0.3. On NVIDIA GTX1060 (6GB) GPU proposed solution for 10 million affected vertices takes about 3.467258 milliseconds

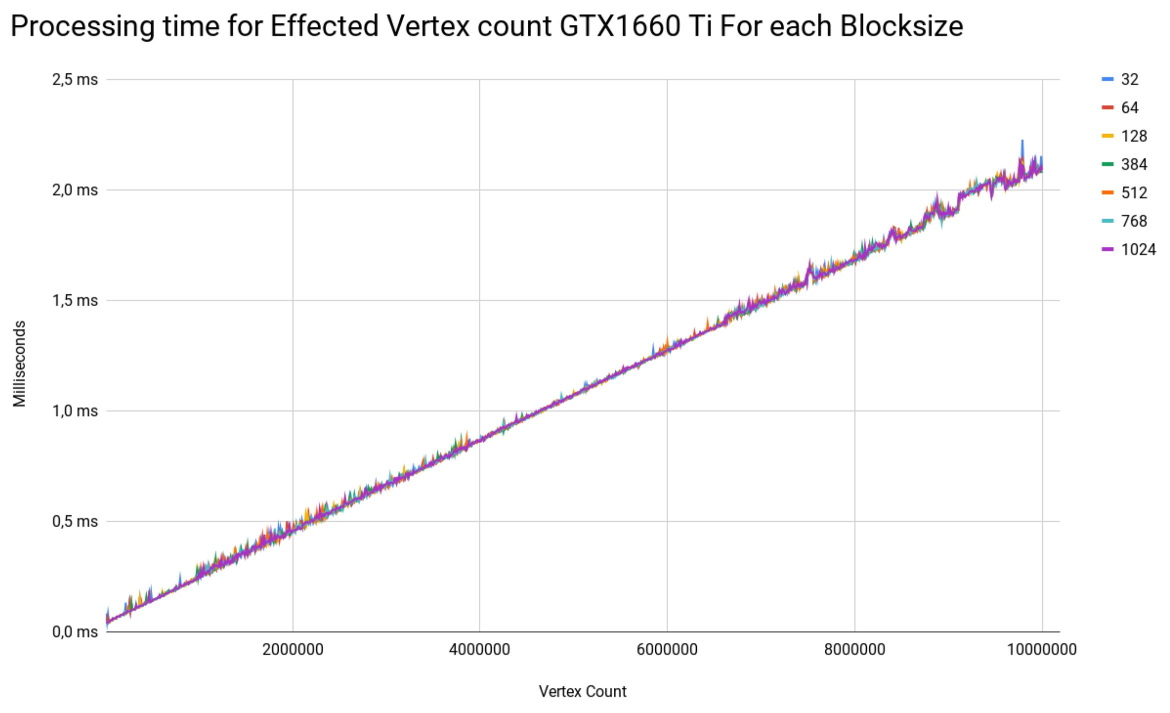


Figure 0.4. On NVIDIA GTX1660Ti GPU proposed solution for 10 million affected vertices takes about 2.088206 milliseconds

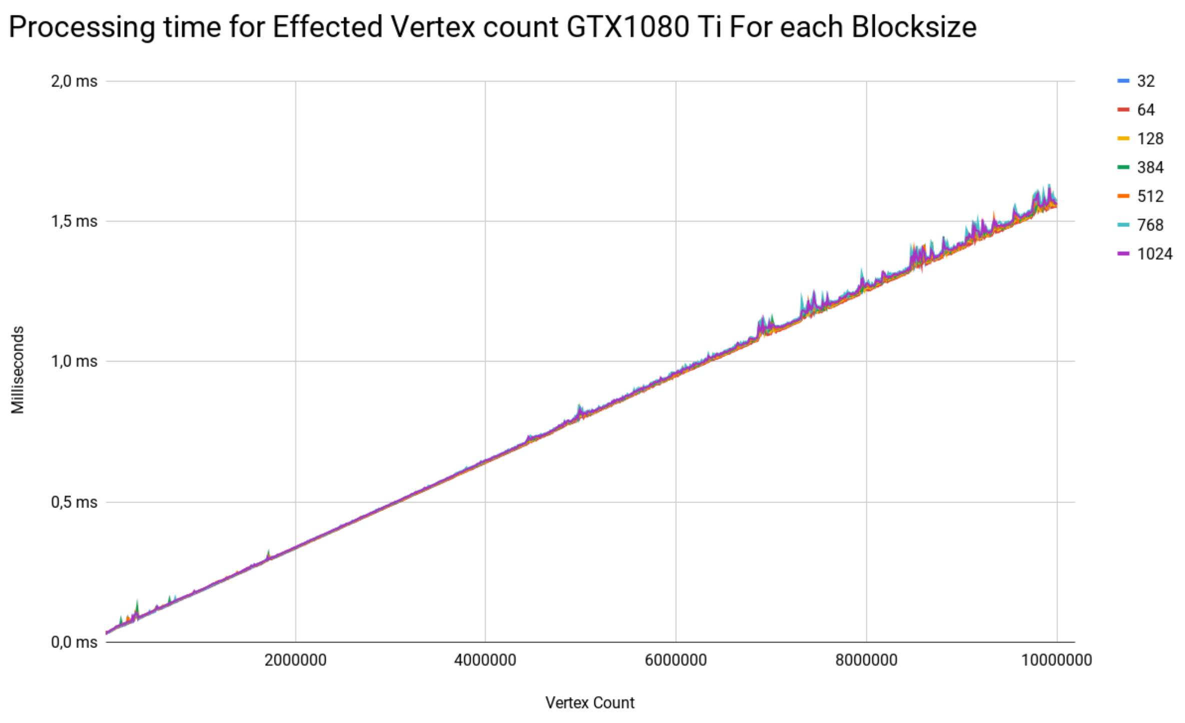


Figure 0.5. On NVIDIA GTX1080Ti GPU proposed solution for 10 million affected vertices takes about 1.566714 milliseconds

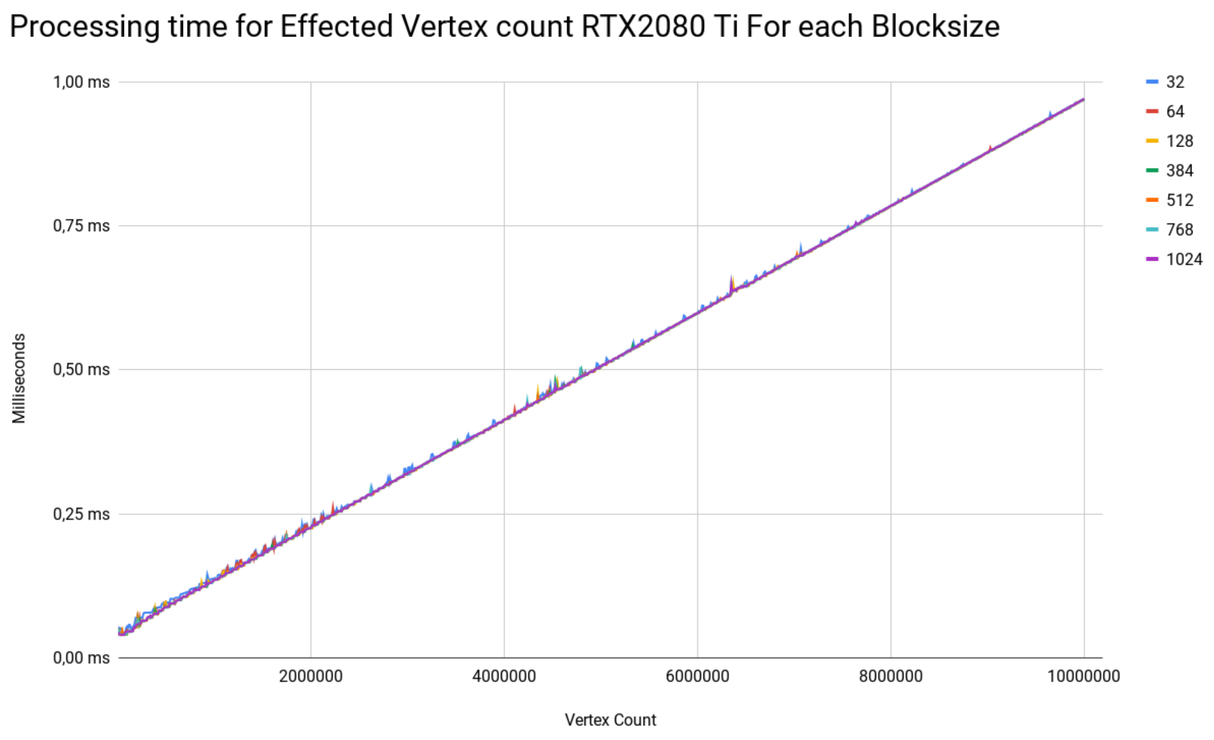


Figure 0.6. On NVIDIA RTX2080Ti GPU proposed solution for 10 million affected vertices takes about 0.970039 milliseconds

HACETTEPE UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
THESIS/DISSERTATION ORIGINALITY REPORT

HACETTEPE UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
TO THE DEPARTMENT OF COMPUTER ENGINEERING

Date: 08/07/2019

Thesis Title / Topic: EXAMPLE BASED SOFT-BODY SIMULATION ON GRAPHICS PROCESSING UNITS

According to the originality report obtained by my thesis advisor by using the *Turnitin* plagiarism detection software and by applying the filtering options stated below on 08/07/2019 for the total of 54 pages including the a) Title Page, b) Introduction, c) Main Chapters, d) Conclusion sections of my thesis entitled as above, the similarity index of my thesis is 1%.

Filtering options applied:

1. Bibliography/Works Cited excluded
2. Quotes excluded / ~~included~~
3. Match size up to 5 words excluded

I declare that I have carefully read Hacettepe University Graduate School of Science and Engineering Guidelines for Obtaining and Using Thesis Originality Reports; that according to the maximum similarity index values specified in the Guidelines, my thesis does not include any form of plagiarism; that in any future detection of possible infringement of the regulations I accept all legal responsibility; and that all the information I have provided is correct to the best of my knowledge.

I respectfully submit this for approval.

Date and Signature
08/07/2019
[illegible]

Name Surname:	Emircan KOÇ
Student No:	N15224437
Department:	Computer Engineering
Program:	Computer Engineering MSc. with Thesis
Status:	[✓ Masters] ☐ Ph.D. ☐ Integrated Ph.D.

ADVISOR APPROVAL

APPROVED.

[illegible]

Assist. Prof. Dr. Adnan Özsoy
(Title, Name Surname, Signature)

CURRICULUM VITAE

Name Surname	: Emircan KOÇ
Place of Birth	: Bursa
Date of Birth	: 02/11/1992
Marital Status	: Single
Correspondence Address	: Çiğdem Mah. 1550/1. Cd. No:17/7 Çankaya/Ankara
Phone	: 0 506 220 18 96
E-mail Address	: koc.emircan@gmail.com

EDUCATIONAL BACKGROUND

Master's Degree :	
Hacettepe University, Ankara, Turkey,	2015 - 2019
M. Sc. in Computer Engineering	CGPA: 3.78 / 4.00
Bachelor's Degree :	
Hacettepe University, Ankara, Turkey,	2011 - 2015
B. Sc. in Computer Engineering	CGPA: 3.11 / 4.00

WORK EXPERIENCE

TaleWorlds Entertainment	2015 - Present
Software Engineer	
TaleWorlds Entertainment	2013 - 2015
Part-Time Software Engineer	
TaleWorlds Entertainment	2013 - 2013
Software Engineering Intern	

PUBLICATIONS

E. Koc and A. Ozsoy, “Approximate Data Driven Parallel Shape Matching for Soft Body Physics Simulations”, International Conference on Artificial Intelligence and Data Processing, 2019.

E. Koc and A. Ozsoy, “Example Based Soft-Body Simulation on Graphics Processing Units”, The Visual Computer, International Journal of Computer Graphics (Submitted).