Physics and Fluid Simulations in Real Time

When developing *The Barn*, an understanding of physics within real time rendering platforms and the abilities and limitations of physics engines in the present day is essential. The advancement of physics simulations in the past decades has contributed to creating an immersing and realistic environment for real time productions to take place. Physical effects are implemented into games to simulate dynamic cloth, hair and muscle, to imitate Newtonian physics as well as implementing fluid effects and deformable objects using real life properties. With modern physics engines, additional physics mechanics can be implemented to create more advanced physical effects. Physics engines use a form of reverse engineering to develop physical effects, without having to calculate the complete causality. Due to the realism attained by the physics engines using physics as a gameplay mechanic has become common place and almost all modern games make use of them. Whilst physics has become integral to modern games, the use of real time physics simulations is being applied to new and various other fields including medicine, vehicular visualisation and product visualisations.

Physics defines every natural part of life upon earth. Gravity pulls objects toward the planet's centre resulting in weight. Objects collide with one another preventing objects from passing through another object, and liquids dominant the planet surface. Although physics is part of our everyday lives, it has become a monumental struggle to simulate these physical effects within digital entertainment (Macklin, Muller, Chentanez, Kim, 2016).

The first video game to ever use physics was *Tennis for Two* in 1958. Invented by the physicist William Higginbotham, the game featured a horizontal line for the ground, a vertical line for a net and a point for the ball (Tretkoff, 2016). Using only transistors and circuitry, *Tennis for Two* featured limited wind resistance and gravity effects on the ball which could vary based on the player selected planet (Tretkoff, 2016). *Tennis for Two* was mostly forgotten however, and physics in video games did not come to prominence until 1972 with the release of *Pong* (Pong Game, 2016).

Although *Pong* wasn't the first video game to feature physics, it was the most advanced of its time, requiring calculations coded into the circuitry of the system to direct the ball between the player controlled paddles (Niklas, 2016). A collision response would be invoked upon the ball for every instance where the ball hit either the player controlled paddles or border walls. Every collision would cause the ball to be treated as a new object that was given new constant acceleration emulating the physical effect of the ball bouncing. Although *Pong* featured an entirely physics based gameplay, the physics involved were simplified to emulated the effect rather than the cause (Niklas, 2016).

To perform the complex calculations in a real time platform, different physics effects are handled in different ways to increase performance (Firth, 2016). The most common and oldest form of physics handled by a physics engine is collision detection (Árnason, 2016). Collision detection and response are the most basic physics calculations performed by an engine. Collisions involve an algorithm that checks for every game tick, whether an actor is colliding with another actor (Gourlay, 2016). This collision detection can initiate a collision response. A collision response defines the action taken when an actor collides with another actor during gameplay. Results can vary from blocking players from walking through walls, reducing the

health of a player if struck by a bullet, weapon or chain reaction events with other actors (Epic Games, 2016).

Since *Pong*, physics in real time engines has been based upon the principle of reverse engineering a physical effect. (Serrano, 2016). This has led to the development of a multitude of physics engines (Boeing & Bräunl, 2007) which handles the simulation of objects using the principal of Newton's Second Law of Motion, f=ma (Build New Games, 2012). This method increases engine performance and creates stable and reliable results for the user. By using ever increasing efficiencies in the way the physics is handled by the physics engine, highly complex and previously impossible simulations have become increasingly common and implemented (Bongart, 2016). Additionally, the rate in which computational power is increasing immensely aids in the application of real time physics. Currently we see a double in processing power every twelve months resulting in rapid growth of what physics engines are able to simulate (Green, 2016).

The more advanced rigid body physics relies on collision physics to perform. Rigid body physics describes an actor where deformation is ignored when any two given points on the actor will remain the constant distance apart regardless of an external force or collision (NVidia, n.d.). Actors with rigid body physics are defined by convex hulls describing the shape and orientation in addition to position, mass, velocity and objects bounds; these bounds do not change as the actor has force exerted upon it (Gourlay, 2016).

Rigid bodies can be used to simulate simplified gravity effects where the engine determines if the actor is colliding with an actor below it. If not, a new constant acceleration is added to the object and it moves in the new direction until it collides with an actor below it (NVidia, n.d.). Rigid can have forces exerted upon them via character movement or other actor collision invoking a collision response, resulting in a realistic physical response for solid actors such as concrete objects or large metal objects (NVidia, n.d.). However, rigid bodies are unable to deform under force resulting in unrealistic effects when deformable object such as a car simply bounces off a concrete wall at high speeds.

Soft body physics answers the shortcomings of rigid body physics by enabling the deformation of objects when forces are exerted upon them (Boeing, Bräunl, 2007).

However, soft body physics, as with rigid body physics cost more in computational power. Soft body physics defines the actor as a deformable mesh surrounded by the actor's bounds (Gourlay, 2016). When force is applied to the object bounds, the vertices are moved to change shape of the deformable actor based on the forces' location on the actor's bounds. Although soft bodies can deform, they retain their connectedness and adjacency of vertices on the body even when forces are applied to the actor (Gourlay, 2016).

Soft body physics are typically used for cloth and hair which are able to be deformed based on global and local forces such as wind and movement of the parent actor (AMD, 2016). However, soft body physics has been steadily rising in other forms of applications such as a relatively cheap fluid substitute, muscle and fatty tissue deformation on characters and environmental destruction (Vlachos, 2016).

As advanced as physics engines have grown within the last decade, fluid simulations have remained an incredibly difficult to accomplish physical effect (Macklin, Muller, Chentanez, Kim, 2016). An early example of a fluid effect took place within the *1956* film *The Ten Commandments* using practical effects (Brosnan, 1974). Water was flooded into a controlled tanks and filmed from multiple angles. These shots were then composited together to create the fluid simulation that would otherwise have been impossible (Brosnan, 1974).

As ground breaking as the practical effect was, simple fluid effects, simulated entirely inside computers in digital entertainment weren't present until the 1998 DreamWorks animated film *Antz* (Guinness World Records, 1998). As computing power began to increase, complex physics simulations using cloth, hair and fluids became commonly used and more realistic in both live action films and animated features (Failes, 2015). However, it has only been twelve years since the release of *Half-Life 2 (2004)* that complex realistic physics simulations have been present in video games (Stelly, 2007).

Fluid Effects are the current hurdle for game engines to overcome (Stam, 2016). Fluids have almost infinite freedom of motion that is completely non-linear with a constantly changing shape and topology (Stam, 2016). Fluids are particularly difficult to simulate accurately as a fluid can assume the shape of any container, are constantly colliding with everything around them as well as colliding within itself

(Gourlay, 2016). This leads to a severe problem when dealing with fluids, if an actor were to collide with just one part of the fluid, the fluid must respond with its entirety (Gourlay, 2016).

There are a two main ways that physics simulations deal with fluids, as a field called an Eulerian view or a group of interacting particles called Lagrangian view. (Müller, Charypar, Gross, 2016). In field based simulations, each point on a grid is assign properties such as velocity, density, temperature and pressure (Gourlay, 2016). The position of the points on the grid never move but describes the flow of fluid inside it (Chentanez, Muller, 2016). The second method is treating the fluid as group of particles that interact. Each particle is described with properties such as position, velocity, density and temperature (Gourlay, 2016). This differs to the field based method by assigning each particle a position rather than the fixed grid of the field based method (Matthias, 2016). Often the two methods are utilised in combination to effectively simulate a fluid (Stam, 2016).

Particle based effects are computationally intensive, taking current software such as *Real Flow* multiple days to simulate a large simulation (Irving, Guendelman, Losasso, Fedkiw, 2016). However, in July 2016, NVidia released *Cataclysm*, a real time, Fluid-Implicit Particle (FLIP) based solver within *Unreal Engine 4* which could simulate the flooding of a city in real time using two million particles (NVidia, 2016). A similar visual effects shot took Digital Domain and Tweak Films, “a long time,” in the 2004 film, *Day After Tomorrow* (Failes, 2015, p.70). This dramatic advancement of physics in game engines has almost now caught up with the near-photorealistic graphics, cinematic quality surround sound and advanced Artificial Intelligence present within modern games (Kinephanos, 2016).

Fluids are used within modern games to immerse the audience in a way that hasn’t been seen since the first simulated fluids were seen in *Antz*. Fluids are generally used for simple water and smoke (Macklin, Muller, Chentanez, Kim, 2016).

Particle physics is very different to other physical effects. Particle effects utilise the physics engine to create thousands of particles from an emitter in real time (Epic Games, 2016). Particles are points within the physics engine, each particle has properties assigned to it such as position, mass, and velocity but no size or shape (Gourlay, 2016). Each particle has a sprite attached which has various user assigned

properties such as materials and light emitters (Epic Games, 2016). Particle systems, such *Cascade* in *Unreal Engine 4* use basic collisions per sprite for collisions in the level as well as being assigned position, mass, velocity and acceleration (Epic Games, 2016). Particle effects are used widely in modern games for visual effects, however, they do not accurately simulate or function as a particle fluid. Due to their limitation, particle effects are generally used for fires, smoke, clouds and steam where the lack of accuracy isn't noticeable to the player (Emperore & Sherry, 2015).

All physics types described above are being utilised in their own ways within modern games. All games today require the use of collision physics for hit boxes, boundary detection and collision response (Boeing, Bräunl, 2007). Rigid body physics are used more broadly with a move onto the mobile platform, with dynamic, rigid body simulations being able to run on mid-range smart phones today, there has been a dramatic rise of games using gravity simulation to drive and enhance the gameplay (NVidia, n.d.).

The use of soft body physics is steadily rising with many games such as *The Witcher 3: The Wild Hunt* integrating NVidia *Hairworks* and *Apex* for dynamic soft body clothing, real time hair and muscle jiggle and skin deformations (Coombes, 2014). However, mobile devices are only just developing the require computational power to access more advanced physics simulation in real-time through soft body physics.

Fluids within games today are becoming more integral to gameplay mechanics as in Valves 2011 game *Portal 2* which utilised speciality shader models called metaballs to developed physics obeying 'sticky' fluids (Vlachos, 2016).

The future applications for physics within game engines are extensive. Collision physics will remain a critical part of game engines determining hits, collisions and collision responses. However, the complexity of the convex hulls defining the bounds of an object will become more accurate defining an actors complex structure as accurately as offline simulations make possible (Epic Games, 2016).

Both rigid body and soft body physics will become more widespread with most games integrating these features into the main production line as has been seen in recent games (NVidia, 2016). With the development of real-time complete

environment destruction players will be introduced to new gameplay mechanics and real time destruction will be used for real life applications (Stelly, 2007).

Fluid physics and subsequently aerodynamic physics will become faster, more efficient and applied to game environments widespread and integrated into gameplay mechanics driving the narrative and player audience involvement (Vlachos, 2016). Additionally, with the increase in accuracy of physics in real time, there will be applications for real-time flood simulations (Bongart, 2016), studies for vehicle performance and design such as rocketry and cars (Tesla Motors, 2016) as well as applications for real-time medical simulation such as blood flow for patient imaging and scientific study of fluid behaviour (Lee, 2011).

Real time physics is being put to use in *The Barn*. Using the NVidia *PhysX* engine contained within *Unreal Engine* itself, I have been able to implement advanced collision response for player interaction with the environment around them. Using collision triggers, the player is able to pick up objects, add them to an inventory and drop them in a new location. These objects will then have rigid body physics applied to them, causing them to act under the influence of gravity.

Non-Player characters in *The Barn* are affected by both rigid body and soft body physics. Soft body physics is implemented in characters clothing and hair using the *Apex* editor and *PhysX* physics engine integrated within *Unreal Engine 4* (Epic Game, 2016). These soft body physical effects contribute to more realistic gameplay and player immersion in the game environment and also serve to advance the aesthetic appeal that is commonly found in large studio games.

The implementation of fluid physics within *The Barn* is unlikely. The challenges of applying a premade solver such as the Cataclysm solver are still significant. The impact real time fluid physics has on performance is still quite large when operating in a game play level. However, alternative for fluid effects such as particle effects and soft body physics will be implemented to mimic fluid physics.

The rapid advancement of physics development within real-time platforms has created unprecedented realism within games and real-time visualisations. With fluid physics on the verge of breaking into next generation platforms we will see a jump in realism and immersion when dealing with games and gameplay mechanics as well as the use of real time simulation for interactive cinema.

Developing an interactive story that involves puzzles and problem solving can now be driven by physics with mass driven trigger and fluid filled objects to solve puzzles and advance through the story through the use of realistic physics in my project *The Barn,* including imitation gravity and resistance, rigid body simulation for player interactivity with objects around the game environment and for problem solving, and potentially the use of soft body physics for deformable metal in a car body, cloth and hair, I will be able to increase player immersion and add a level of sought-after realism.

References:

AMD. (2016). *AMD TressFX Hair*. AMD. Retrieved 10 September 2016, from http://www.amd.com/en-us/innovations/software-technologies/tressfx

Árnason, B. (2016). *Evolution of Physics in Video Games* (1st ed.). Reykjavik: Reykjavik University. Retrieved from http://www.olafurandri.com/nyti/papers2008/Evolution%20of%20Physics%20in%20Video%20Games.pdf

Boeing, A. & Bräunl, T. (2007). *Evaluation of real-time physics simulation systems*. Proceedings of The 5Th International Conference On Computer Graphics and Interactive Techniques in Australia and Southeast Asia - GRAPHITE '07. http://dx.doi.org/10.1145/1321261.1321312

Bongart, R. (2016). *Efficient Simulation of Fluid Dynamics in a 3D Game Engine (1st ed.). Stockholm: Royal Institute of Technology*. Retrieved from https://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2007/rapporter07/bongart_robert_07018.pdf

Brosnan, J. (1974). *Movie magic*. London: Macdonald.

Build New Games. (2012). *How Physics Engines Work.* Retrieved 7 September 2016, from http://buildnewgames.com/gamephysics/

Chentanez, N. & Muller, M. (2016). Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid (1st ed.). NVIDIA PhysX Research. Retrieved from http://matthias-mueller-fischer.ch/publications/tallCells.pdf

Epic Games. (2016). *Collision Overview*. Retrieved 7 September 2016, From https://docs.unrealengine.com/latest/INT/Engine/Physics/Collision/Overview /index.html

Epic Games,. (2016). Cascade Particle Editor Reference. Docs.unrealengine.com. Retrieved 10 September 2016, from https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/Cascade/

Epic Games,. (2016). Visual Effects: Lesson 07A: Using GPU Particle Simulations – Epic Wiki. Wiki.unrealengine.com. Retrieved 10 September 2016, from https://wiki.unrealengine.com/Visual_Effects:_Lesson_07A:_Using_GPU_Particle_Simulations

Failes, I. (2015). *Masters of FX*.

Firth, P. (2016). *Physics engines for dummies* | Wildbunny blog. (2016). Wildbunny.co.uk. Retrieved 7 September 2016, from http://www.wildbunny.co.uk/blog/2011/04/06/physics-engines-for-dummies/

Gourlay, M. (2016). *Fluid Simulation for Video Games Part 1*. Retrieved 7 September

2016, from https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1

Green, S. (2016). *Particle-based Fluid Simulation* (1st ed.). NVidia. Retrieved from http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_ParticleFluids.pdf

Guinness World Records. (1998). *First film with digital water*. Retrieved 9 September 2016, from http://www.guinnessworldrecords.com/world-records/first-film-with-digital-water

Irving, G., Guendelman, E., Losasso, F., & Fedkiw, R. (2016). *Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques* (1st ed.). Stanford: Stanford University. Retrieved from http://physbam.stanford.edu/~fedkiw/papers/stanford2006-01.pdf

Kinephanos. (2016) *Game Engines and Game History*. Kinephanos.ca. Retrieved 7 September 2016, from http://www.kinephanos.ca/2014/game-engines-and-game-history/

Lee, B. (2011). *Computational Fluid Dynamics in Cardiovascular Disease*. Korean Circulation Journal, 41(8), 423. http://dx.doi.org/10.4070/kcj.2011.41.8.423

Matthias, M. (2016). *Real Time Fluids in Games* (1st ed.). Ageia. Retrieved from https://www.cs.ubc.ca/~rbridson/fluidsimulation/GameFluids2007.pdf

Macklin, M., Muller, M., Chentanez, N., & Kim, T. (2016). *Unified Particle Physics for Real-Time Applications* (1st ed.). New York: NVidia. Retrieved from http://mmacklin.com/uppfrta_preprint.pdf

Müller, M., Charypar, D., & Gross, M. (2016). *Particle-Based Fluid Simulation for Interactive Applications* (1st ed.). Zürich: Federal Institute of Technology. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.844&rep=rep1&type=pdf

Niklas, C. (2016). *PONGMECHANIK.* Cyberniklas.de. Retrieved 7 September 2016, from http://www.cyberniklas.de/pongmechanik/

NVidia. (2016). *NVIDIA presents Cataclysm liquid solver for Unreal Engine 4* | PhysXInfo.com – PhysX News. Physxinfo.com. Retrieved 7 September 2016, from http://physxinfo.com/news/12737/nvidia-presents-cataclysm-liquid-solver-for-unreal-engine-4/

NVidia. (2016). *NVIDIA APEX 1.3.1 Documentation*. (2016). docs.nvidia.com. Retrieved 7 September 2016, from http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/apexsdk/index.html

NVidia. (n.d.). *Rigid Body Dynamics*. Retrieved September 07, 2016, from

http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/RigidBodyDynamics.html

Pong Game. (2016). Ponggame.org. Retrieved 7 September 2016, from http://www.ponggame.org/

Serrano, H. (2016). *How does a Physics Engine work? An Overview*. (2016). Harold Serrano. Retrieved 7 September 2016, from http://www.haroldserrano.com/blog/how-a-physics-engine-works-an-overview

Stam, J. (2016). *Real-Time Fluid Dynamics for Games* (1st ed.). Toronto: Alias | wavefront. Retrieved from http://www.intpowertechcorp.com/GDC03.pdf

Stelly, J. (2007). Physical Gameplay in Half-Life 2 (1st ed.). Valve. Retrieved from http://www.valvesoftware.com/publications/2006/GDC2006_PhysicalGameplayInHL2.pdf

Tesla Motors. (2016). *Get a job: Tesla Motors is hiring an Unreal Engine Artist*. (2016). Gamasutra.com. Retrieved 8 September 2016, from http://www.gamasutra.com/view/news/258343/Get_a_job_Tesla_Motors_is_hiring_an_Unreal_Engine_Artist.php

Tretkoff, E. (2016). *This Month in Physics History - October 1958: Physicist Invents First Video Game*. Aps.org. Retrieved 9 September 2016, from https://www.aps.org/publications/apsnews/200810/physicshistory.cfm

Vlachos, A. (2016). Water Flow in Portal 2 (1st ed.). Los Angeles: Valve. Retrieved from http://www.valvesoftware.com/publications/2010/siggraph2010_vlachos_waterflow.pdf

Bibliography:

BeamNG. (2016). *BeamNG.drive.* BeamNG. Retrieved 8 September 2016, from http://www.beamng.com/

Behavior Tutorials - Havok. (2016). Havok. Retrieved 7 September 2016, from http://www.havok.com/behavior-tutorials/

Boury, F. (2016). *Geometry shader – Metaballs* (1st ed.). Fries Boury. Retrieved from http://friesboury.be/geometry-shader/

Coombes, D. (2014). *The Witcher 3: Wild Hunt with NVIDIA HairWorks shown at Gamescom*. Developer.nvidia.com. Retrieved 9 September 2016, from https://developer.nvidia.com/content/witcher-3-wild-hunt-physx-and-nvidia-hairworks-shown-gamescom

Emperore, K. & Sherry, D. (2015). *Unreal Engine Physics Essentials*. Packt Publishing Ltd.

Epic Games. (n.d.). *Physics Sub-Stepping*. Retrieved September 07, 2016, from https://docs.unrealengine.com/latest/INT/Engine/Physics/Substepping/index.html

Gaffer on Games | *Game Physics*. (2016). Gafferongames.com. Retrieved 7 September 2016, from http://gafferongames.com/game-physics/physics-in-3d/

Gamasutra - *Practical Fluid Dynamics: Part 1*. (2016). Gamasutra.com. Retrieved 7 September 2016, from http://www.gamasutra.com/view/feature/1549/practical_fluid_dynamics_part_1.php?print=1

Infusion Studios. (2016). *Real-Time Visualization R&D with Unreal Engine 4.5*. Infusionstudios3d.com. Retrieved 8 September 2016, from http://www.infusionstudios3d.com/index.php/real-time-visualization-rd-with-unreal-engine-4-5/

Loi,. (2016). News | The University of Sydney. Sydney.edu.au. Retrieved 10 September 2016, from http://sydney.edu.au/news/84.html?newsstoryid=15052

Restuccio, D. (2016). *The Day After Tomorrow's Photoreal Effects*. Postmagazine.com. Retrieved 10 September 2016, from http://www.postmagazine.com/Publications/Post-Magazine/2004/June-1-2004/THE-DAY-AFTER-TOMORROWS-PHOTOREAL-EFFECTS.aspx

Space Engineers on Steam. (2016). Store.steampowered.com. Retrieved 8 September 2016, from http://store.steampowered.com/app/244850/

The History of Game Engines. (2016). prezi.com. Retrieved 7 September 2016, from https://prezi.com/w56f8xawwcyg/the-history-of-game-engines/

Yeh, T., Faloutsos, P., & Reinman, G. (2016). *Enabling Real-Time Physics Simulation in Future Interactive Entertainment* (1st ed.). Los Angeles: University of California. Retrieved from http://web.cs.ucla.edu/~reinman/mars/papers/SANDBOX-06-Physics.pdf