(**Please Note:** Due to the .pdf format, all the gifs have been compressed into on image and **DO NOT** animate. To see gifs animated, feel free to download the .docx file from: https://github.com/Jack-GHopkins/Masters-Dissertation/blob/main/Masters%20Dissertation%20Write%20Up%20-%20Jack%20Hopkins.docx)

Investigating use of different Procedural Animation techniques in Commercial Video Games and applying them to a Character in Unreal Engine 5.

By Jack Hopkins [180246647]

Project Supervisor: Dr Gary Ushaw

Number of Words: 8429

1 ABSTRACT

With the advancement of computing technology and the growth of the video game market, more and more developers have started to use procedural generation within their games. One more novel use is for animation. This paper will be examining different applications and implementations of it in the industry and in the body of literature surrounding it., This research will be then used to create an animated character in Unreal Engine 5.

2 CONTENTS

3 INTRODUCTION

As the video game industry continues to grow and computing power progresses, the use of procedural generation in video games has become more valuable, not only to expedite the development process, but also to create more dynamic mechanics as a tool for creative expression. Whether it be reactive music [1], inimitable storylines or unique worlds for each playthrough [2]; all these things [3] - as Tanya X. Short et al put it:

> *[Allows] Players [to] have their own personal journey but still have enough common experience to share their tales with others… [4]*

Procedural Content Generation for Games: A Survey [5] gives a very in-depth look into a great swath of procedural generation techniques; from world building to storylines. One of the avenues of procedural generation can be seen in animation. This can range from very simple things like turning head, such as Quake III, to animating entire custom characters which are made in game, as in Spore. [6] [7] As Scottish Game Animator, Jonathan Cooper, puts it:

> *"In video game parlance, 'procedural' animation means anything made by computer algorithms instead of an animator" [8]*

Animation is a time-consuming process and, when done manually, is not adaptive. This can create many situations when the animation breaks the immersion of the player. One of the most common issues is with foot placement on different terrain within walk cycles. [9]

Procedural animation is a large umbrella term that covers many techniques ranging from simple *ragdoll physics*, to *inverse kinematics (IK).* It's rare to see only one used in a video game and instead many use a plethora of methods to get a result the developers are happy with. One example of this would be *Rain World,* where the developer, Joar Jakobsson, chose to use rigid bodies and pathfinding to animate his character. [10]

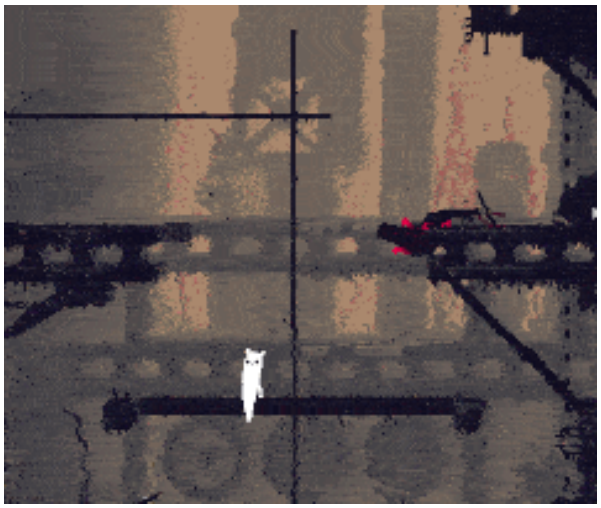


Figure 3.1 Gif of 3 different Rain World characters. All animated with procedural generation.

The project will go beyond existing works by focusing on the compilation of multiple different techniques to create a fully animated character. I will be using as many techniques as I can and explain my decisions to why I'm using all of them in each specific scenario.

This project is motivated by my curiosity in the world of procedural animation, as it is where animation and programming collide and overlap – both disciplines I'm quite interested in. I also want to become more familiar with Unreal Engine 5 because it uses C++ and is one of the most commonly used video game engines in the industry, particularly for large AAA developers.

As I proceed through this dissertation, you will see me apply:

- Multiple active correcting animation techniques like foot and hand adjustment.
- Passive animation techniques as seen with ragdoll.
- The merging of passive and active animation techniques with partial physical animation.

This project will be using technologies such as:

- Unreal Engine 5 for its Control Rig and in-depth online tutorials.
- GitHub for version control.
- Trello for task management.

3.1 Original Aims & Objectives

3.1.1 Objectives

3.1.1.1 Examine and learn different applications of Procedural Animation.

Discuss applications of procedural animation in the current literature and applications of it in commercial video games.

3.1.1.2 Learn and understand Unreal Engine's Control Rig tool.

It is vital I understand how to use the Unity Engine, its preferred language C++ and Blueprints. Develop knowledge and understanding of animation blending, delineation, skeletal animation, ragdoll and more.

3.1.1.3 Implement procedural animation methods.

Use Unreal Engine 5 to create animations for a bipedal character doing multiple procedural animation technique simultaneously.

3.1.1.4 Evaluate results and test prototype.

Compare the prototype to other implementations in commercial video games, analysing its strengths and weakness. Analyse potential uses for it wind

3.2 Project Outline

Introduction

An introduction to the dissertation detailing the motivation, aims and objectives.

Background and Research

Presents the context and background research done for the project.

Implementation

A discussion on what the prototype can do and how I intend to go about implementing the background research into it.

Results and Evaluation

A section on displaying and analysing each procedural animation technique implemented.

Conclusion

A summary on the fulfilment of objectives, summary of the achievements of the project, development of personal skills, and possible future work.

4 Background Technical Material

Within this chapter, I will be discussing the background research that went into my project. Starting with a broad overview of the literature on procedural animation, examine different techniques used in commercial video games and analyse the technique's benefits and detriments.

4.1 Procedural Animation

Character animations are, in most games, premade assets; ones created beforehand by an artist. Whether they are crafted by hand, drawn frame by frame [11], or using motion capture [12], all these animations are predetermined. This means that games may require an extraordinary number of animations for all the different situations a player might find themselves in. This of course depends on the video game, but even in Mario 64, Mario has 209 different animations [13]. And this was a game released in 1996 when the standards and size of animation were not as great as they are now.

While the number and scale of animations can vary greatly upon genre, development team size, etc. it is not common for developers look for ways to make the process of animation creation more time efficient. This is one of the general draws of procedural generation. It can help reduce development time.

However, procedural animation is not just about speeding up the production line. Instead, it can widen the horizon for games developers to exercise their creative authority. Some examples Jonathan Cooper gives are using procedural animation for creating sequences of animations for more natural speaking gestures, or lip-syncing audio and dialogue text. [8] That being said, this doesn't mean that there aren't draw backs. Cooper continues to explain:

> *"[Procedural animation] requires minimal animation, so it can be a great time-saver in terms of asset creation, though it greatly adds to the complexity of character's setup and has yet to produce convincing results on realistic humanoids." [8]*

4.1.1 Passive Animation

In their article in the *2011 IEEE Computer Graphics and Applications*, Hertzmann and Zordan make a distinction between passive and active animation. Passive animation has been a lot more widely used form of procedurally generated animation because:

> *"…they require simulating physics equations but don't require sophisticated control to make them appear natural." [14]*

4.1.1.1 Particle and Visual Effects

One the earliest examples of procedural animation can be seen in particle effects [15] and fluid dynamics. These animations often rely on the physics engine to process many different particles in a short period. These are done procedurally because recreating them with conventional animation techniques can be difficult since they are usually highly chaotic systems based on natural phenomena. Furthermore, these systems are very reactive so having them generated at runtime allows them to be applied to multiple situations.

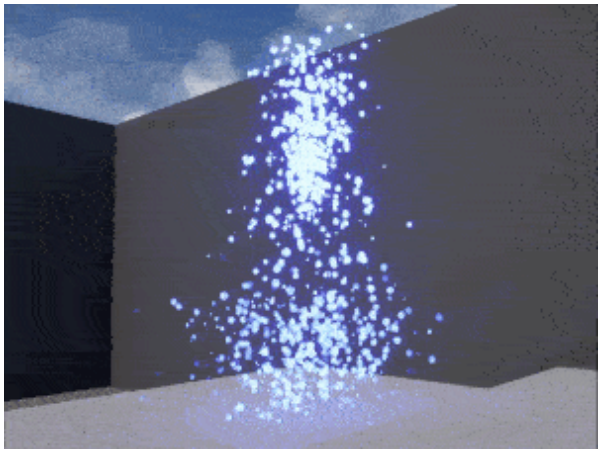


Figure 4.1 Demonstration of particle effect in Unreal Engine 5. [16]

Visual effects, while not generating any physics bodies and traditionally having been drawn by hand, can also be procedurally generated. An example of this is the Niagara VFX System in Unreal Engine 5. [17]

Figure 4.2 Custom VFX tool for creating "Trails" of fast travelling meshes [18]

Figure 4.3 Example of Advanced Niagara Effects in Unreal Engine 5 [19]

4.1.1.2 Ragdoll Physics / Physical Animation

Character animation is often active as:

"It requires descriptions not only for the body's physical but also for behaviour control." [14]

Despite this there are still prominent examples of passive procedural character animation techniques. One example can be seen in *ragdoll physics*. This technique is when the collection of rigid bodies tied to a character are released from any "muscle" control and a procedural physics simulation takes over. However, the rigid bodies will often still apply to any restrictions from the skeletal mesh - often being some sort of inverse kinematics.

A common use of ragdoll physics is upon death, but it can be used for comedic effect too. [20]

Figure 4.4 The character "Anhur" dying in Smite after leaping.

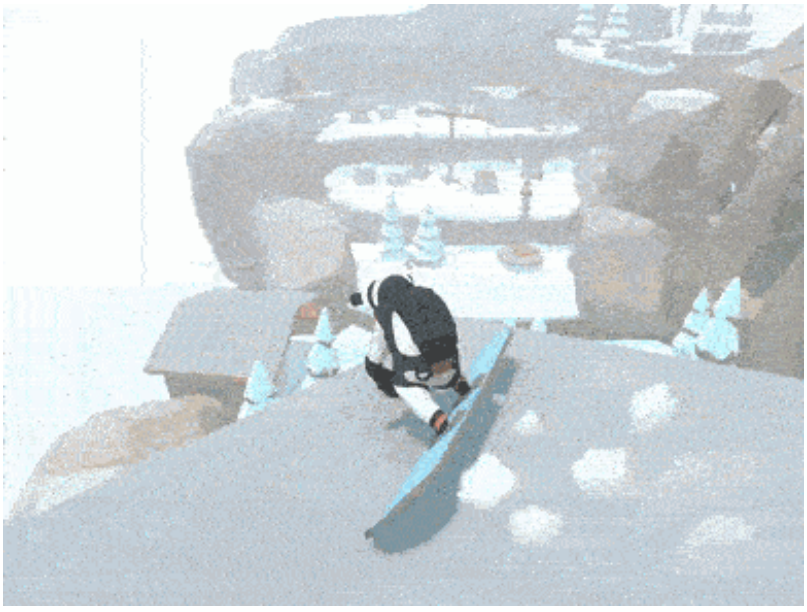
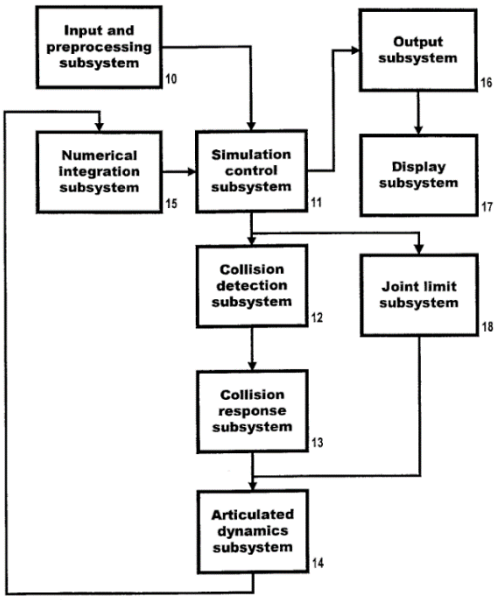


Figure 4.5 Example of Partial Physics based Animation in Human Fall Flat where the play still has active control over the character despite it being very wobbly.

These constrained rigid body approaches are the most common application of ragdoll physics. Some of the algorithms used for implementation are:

- **Featherstone's algorithm** [21] with **spring-damper contacts** [22] – Featherstone is an algorithm used for computing the effects of forces applied to an open kinetic chain (such as an animation skeleton). It is a development on the recursive Newton-Euler Algorithm. Spring-damper contacts is a method for generating realistic collision between animated bodies. It iteratively calculates the body position from previous positions using a simulated contact force between said colliding bodies.



- **"Beads on a wire"** - Constrained particle dynamics that used constraint forces. [23]

Furthermore, there are many "pseudo-ragdoll" techniques that have been used for different games such as *Verlet integration* in *Hitman: Codename 47* [24] and a partial physical body, where only part of the character bodies is using ragdoll physics such as Rain World. [10]

4.1.1.3 Animation Blending

This is the ability to "cross-fade" between different positions of bones to help transitions to be smooth continuous motions. For example, the transition between a walk and run cycle. It enables the bones of a character to not move in a singular predetermined motion (as it would be with just an asset), but rather be moved simultaneously by many motions. A blended motion is created by averaging the weights of each individual motion. These weights can also be changed over time. [9]

This allows for motions to influence one another instead of stopping rigidly. One other application is using two different predetermined motions to synthesis an in-between motion. Rune Skovbo Johansen gives this example:

> *"…a character may have a stored walk cycle for walking straight ahead and another stored walk cycle for turning sharply to the right with a certain turning angle. By blending these two motions, each with a given weights, a new walk cycle can be synthesized that can have a less sharp turning angle." [9]*

4.1.1.4 Segmented Animation Units

Like animation blending, having separated predetermined animations for different sections of a body and allow for these animations to be automatically combined when called for instead of hard-coding every possible animation.

A basic example of this would be segmenting the legs and torso of a bi-pedal character. If you wanted the character to both be able to run in eight directions and also be able to attack with a sword or hold it still; instead of having to animate a total of 16 different animations you can animate eight different run directions for the legs and then play the torso animation of swinging to sword whenever the player calls for it.

Figure 4.6 SMITE's Gilgamesh using animation blending to preserve feeling of "forward momentum" for all 8 directional inputs. [25]

4.1.2 Active Animation

Active animation, by contrast to passive animation, is when a character requires descriptions of the body's physics and uses behaviour controls. [14] While, no fully procedurally generated character has been created (that is realistic) [8], it doesn't mean that there aren't attempts to at partially or fully procedurally animating humanoid characters.

4.1.2.1 Automatic Skeletal Realignment

This technique is when the skeletal mesh reacts to the environment around it and adjusts accordingly. One of the most common examples of this is correcting the placement of limps, such as hands and feet.

This is a common fix for *footskate*, an artifact seen when the feet of an animated character don't seem to stay properly fixed on the ground. [9] This effect can be seen most when the movement speed of a character increases, but the speed of the animation does not. This can also be seen when a character is climbing a set of stairs and their feet are hovering in mid-air.

Figure 4.7 Different examples of the automated foot adjustment from the video game Judgement.

4.1.2.2 Other Heuristic-Based Generation

Having character-based animation has been a much more recent investment in the video game field. This can be seen in games like *Spore*, a progenitor of modern-day real-time animation. [6] [26] Avatars within Spore are created by the player. This customisability made it impossible to pre-make all the animations, thus the Maxis team used procedural animation. One technique to get the characters to walk and run convincingly was to create multiple heuristics to generate movement. An example would create a *walking ratio* from the *number of legs*, *body length* and *bodily symmetry*. From this a gait and walking rhythm could be calculated to result in a more lifelike motion. [26] Within Spore's case their animation doesn't need to be perfectly realistic because these are alien creatures with no real-life counterparts.

Figure 4.8 Weird Alien creature from Spore [27]

This approach can also be seen in Rain World, which also uses the resulting uncanny animation to it's great advantage. It creates a creepy and more anxiety inducing world as these creepy creatures pursue after your small slug cat.

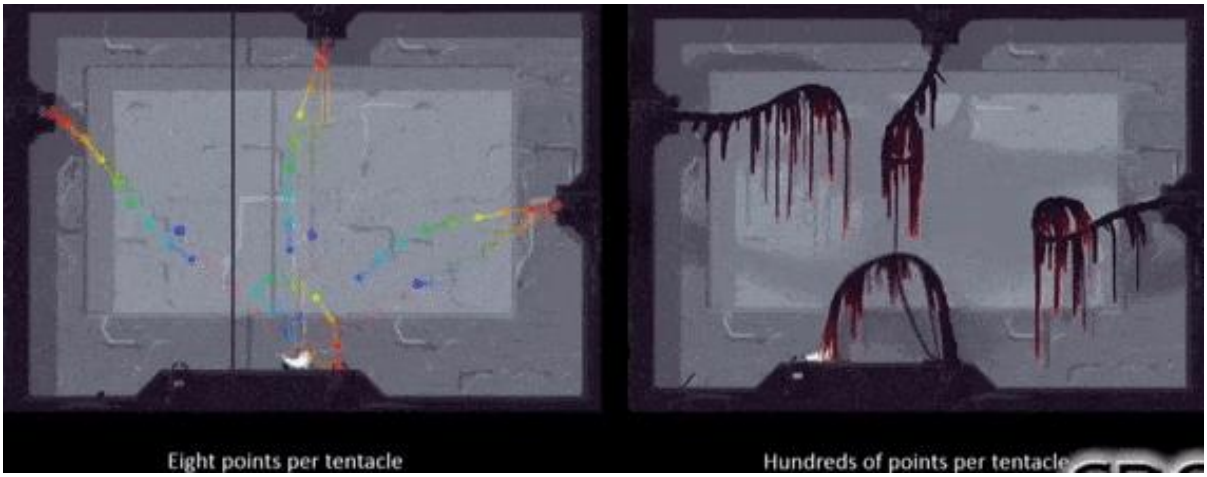


Figure 4.9 Behind the scenes on some of the animation in Rain World at a talk at GDC 2016. [10]

Rain World uses a variation of A* pathfinding to allocate the best location for enemy's place legs, feet and general appendages. The body is controlled by a central hub and reacts on environment, grabbing the most appropriate positions to climb better. However, since finding the perfectly ideal position for each leg every frame is very intensive, bodies will find a "good enough" position and refine it on further frames. This also adds in some unpredictability which aids in making the characters feel more organic. [10]

4.2 Inverse Kinematics

Perhaps one of the most widely used set of constraints used in procedural animation is *Inverse Kinematics* (IK). IK is the process of using kinematic equations to determine a set of appropriate joint configurations for which the end effectors move to desired positions as smoothly, rapidly, and as accurately as possible [28] [29]. Within video game animation, the skeletal mesh provides the chain for which the IK can be applied to. It does this by creating a chain of joints and then placing the joints in succession after the end of the chain – as opposed to forward kinematics.

One of the most commonly used IK algorithms used in video games is *Forward And Backward Reaching Inverse Kinematics* (FABRIK) as it is smaller in computational cost (which is important for video games since these calculations will run on multiple different actors every frame) and produces realistic poses. It does this by not using rotational angles or matrices and uses a heuristic to find each joint's position by visualising it as a point on a line. Furthermore, constraints can be easily incorporated with FABRIK along with multiple end effectors. This gives it a lot of flexibility and a high number of use cases. [30]

IK isn't exclusive to procedural animation. It's an incredibly useful tool for speeding up the animation process for premade animations as it allows for animators to end elements (e.g. feet or hands) and have the rest of the body react accordingly. [8]

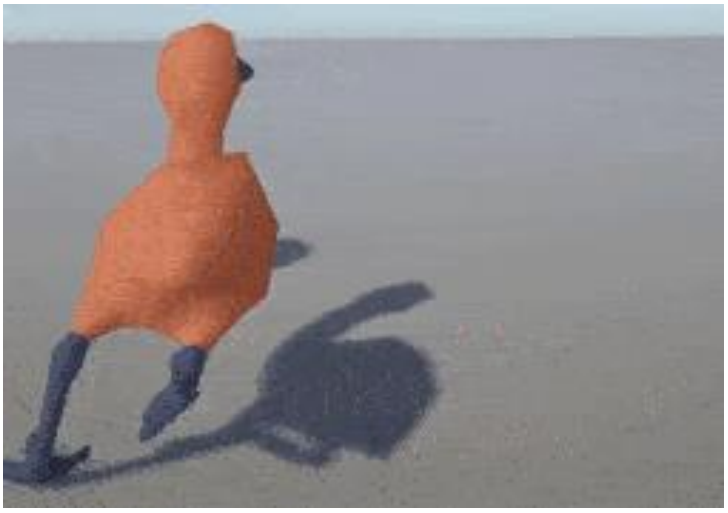


Figure 4.10 Fully walking bird with only rigid bodies and constraints created by Tuatara Game's founder Klemen Lozar. [31]

4.3 A Mixed Reality

In the industry, there is a wide spectrum of animation techniques and its common for video games to incorporate many techniques into individual games. The vast majority of video games, from 3D Triple-A to 2D indie games, will use *passive* procedural animation for capes, particle effects and lip-flaps. Others will use it for their character controls either as a mixture of premade assets and procedural animation or just have 100% procedural animation for the whole game.

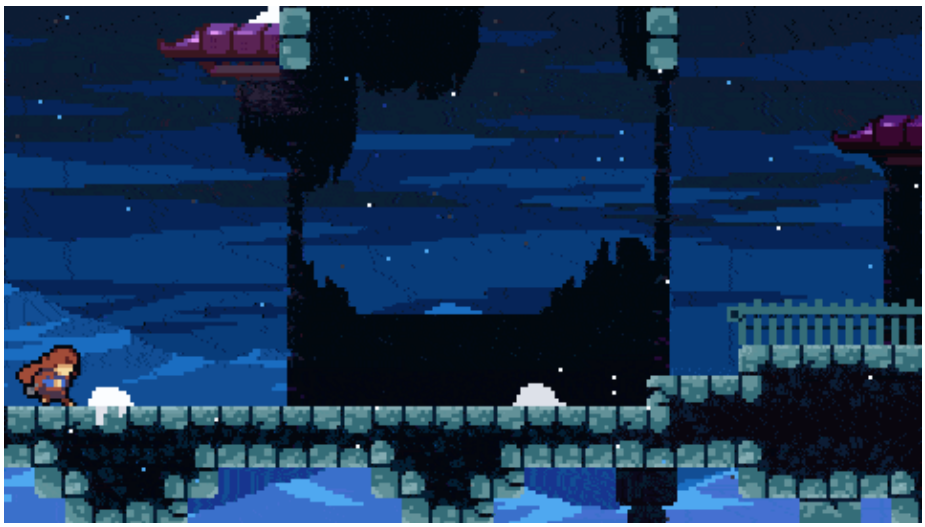


Figure 4.11 Indie game 'Celeste' using procedural animation for playable character's hair. [32]

Figure 4.12 Triple A game 'Elden Ring' using procedural animation for cape movements.

This should be unsurprising. Procedural animation is, as said before, a wide umbrella for many different techniques, and isn't necessarily mutually exclusive with premade asset animation. This allows for game developers to adapt their animation process and pipeline to watch ever suites their game best in terms of time, resources, theming, etc. It's an expansive topic and I aim to explore a wide range of these in my project's implementation.

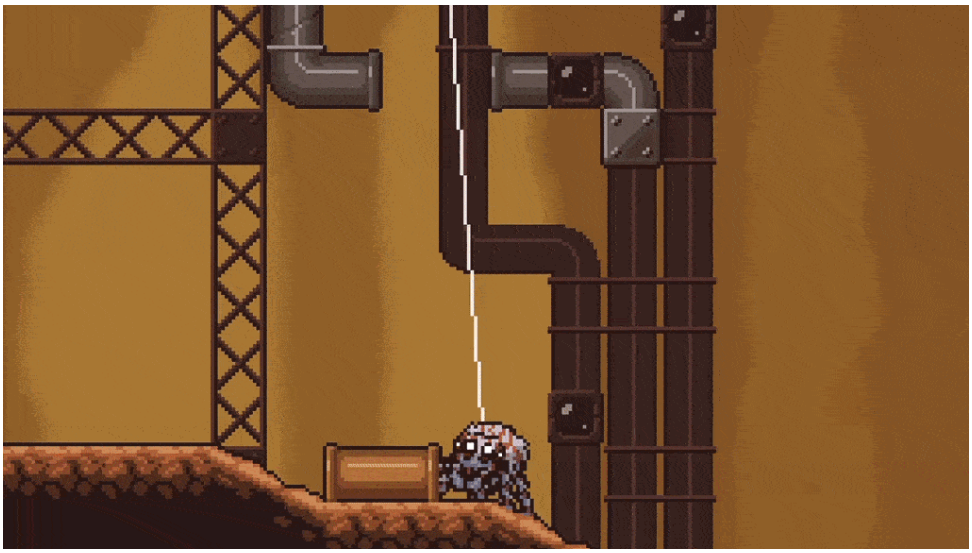


Figure 4.13 Indie game 'Webbed' with procedurally animated webs and spider legs.

4.4 Summary

From this literature review we can take away a couple of details:

- A mixture of techniques within one player control is not unique. Quite the opposite, it is very common to use both premade and procedural animation techniques together. This means my implementation is not emulating a seldom occurrence.
- Passive procedural animation techniques are a lot simpler to implement and make look realistic then active procedural animation techniques.

- Not all procedural animation techniques are used for the purpose of simulating realism. Instead, these techniques can be used for comedy and horror or as a best option for different developers with the resources they have.

5 IMPLEMENTATION

5.1 OVERVIEW

At the start of my implementation, I had to select how I wanted to approach the project:

- either attempt to implement it at a low-level, with technologies such as OpenGL and pure C++.
- or at a higher-level with technologies such as the Unreal Engine.

While it is always tempting to try and deepen my skill in C++ and OpenGL it would be at the cost of the overall project. The main aim of this project is to investigate a wide range of techniques and then implement a few of them within a singular character to further understand their utility. Thus, using an already establish engine such as Unreal Engine 5, made the most sense.

Within the industry, Unreal Engine 5 is receiving a lot of buzz, particularly in the triple-A development scene. It also supports the use of C++, a language I'm more comfortable with.

5.2 PREMADE ANIMATIONS

For this character controller, I used the default Unreal Engine 5 Mannequin and the default premade animations as a foundation to build off all. As stated in A Mixed Reality, many video games use a mixture of techniques and premade and procedural animation are not mutually exclusive.

5.3 AUTOMATIC FOOT PLACEMENT (AFP)

Early on I found a helpful plugin for Unreal Engine called Control Rig [33], this gives access to the blueprint style interface where you can define how a skeletal mesh should behave. This allows for the implementation of inverse kinematics and the creation of an automated foot placement system. I can then use the main blueprint interface to precisely call when I wanted it. While for this rig, implementing my own IK was not necessary, I wanted to show to range and power of this control rig.

Figure 5.1 AFP on lower ledges.

The AFP implementation can be broken down into 5 steps.

1. The first check is if the IK trace is enabled. After this it then sends out a trace from each foot in every direction (spherical shape). There is a maximum range to this sphere which is the lowest and highest you want your foot to be able to reach – prevents the whole body from moving around violently. From these traces I return just the Z value from the vector (the UP/DOWN) and record them as *Z offset targets*. One for each foot.
2. Second is to take the current *Z offset* values – most likely where the foot is currently at – to the *targeted* values I got in set one. This results in a smoother motion from the legs and feet when automatically adjusting.
3. Third is to set the height of the pelvis in accordance with the lowest foot. The pelvis for the mannequin model (plus many mainstream 3d models) lowest node on the bone hierarchy of the model. This means that when moving, rotating, or scaling a bone it will be done relative to the pelvis. More importantly to this step is that if you move the pelvis, so too will all the bones around it. It's the anchor to the skeleton. Thus, I want to move the pelvis down with the lowest foot for that lower foot to touch the ground, making it look like that leg – which is fully extended – is fully supporting the weight of the body. This step also records if the trace as gone over the maximum limit. This limit is set to lowest height the pelvis can decline without it going further then what should be its natural range of motion. This means, when exceeding the limit, it means the lowest foot will be hovering off the floor [Fig 3.2 LEFT]. I then assign which foot the pelvis should be centred over. If the lowest foot is not touching the ground (for example hanging over a ledge) we then move the pelvis over the higher foot. The helps preserve a feeling of balance to the model. [Fig 3.2 CENTER].

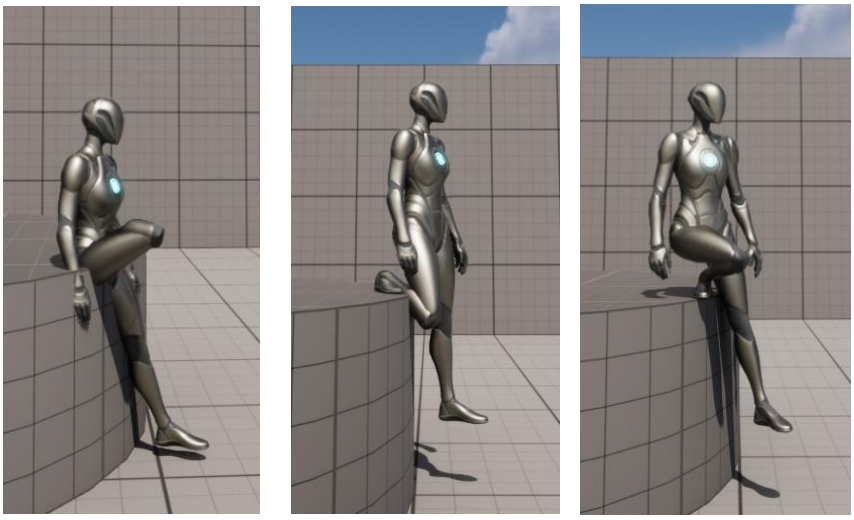


Figure 5.2 ***LEFT****: Figure when no maximum trace limit of pelvis decline is in place.*
CENTER*: When pelvis (centre of mass) is not aligned over highest foot.*
RIGHT*: When both maximum trace limit and pelvis realignment is in place.*

4. Now all the checks are complete I can start to move the bones into their correct positions, interpolating the positions as I go (thanks to step 2).
5. I set what IK I want the legs to follow using *Full Body IK* [34]. This is an Unreal Engine specific function within control rig.

Figure 5.3 AFP on higher ledges.

5.4 Automatic Hand Placement (AHP)

Like AFP, APH is used to place hands correctly in different situations. However, the approach of implementation was a little different. Instead of using the control rig again, I chose to use Unreal Engine's standard Blueprints. This approach could have been used for AFP as well, but for variety's sake I decided not to.

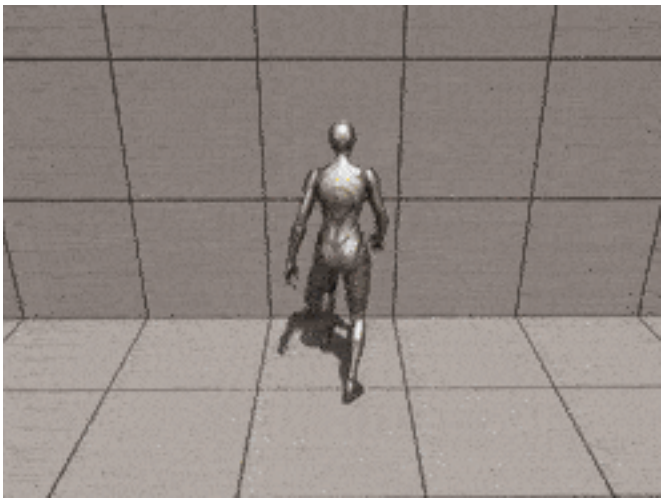


Figure 5.4 AHP

AHP works, in theory, very similar to AFP. It casts a trace from each shoulder and then when you are within the maximum range of the wall it places individual hands up against it. In practise it had a few key differences. Firstly, the trace is a line pointing forward from the playable character, instead of a sphere. This is because I don't want the hand to press up against the wall when its behind you (unlike the foot where it needs to trace above and below it). I create a socket location within our skeletal mesh as a starting point for the trace (shoulder in this case) and then record the position of collision this ray trace has. The hand is then placed on this position; however, it doesn't affect any other bones apart from the arms themselves (no need to move the pelvis or anything like that). With the *Anim Graph* for our Mannequin I then perform the IK. In this case I use a *Two Bone IK* [35]. I then modify the bones after the bone's rotation which is specific for each arm. This is because the rotations are mirrored for opposite sides of the body.

5.5 Automated Head Turning (AHT)

Head turning is an animation designed to have the player character focus on a *point of interest* (POI) that adapts as the player moves around it and only works within a certain range.

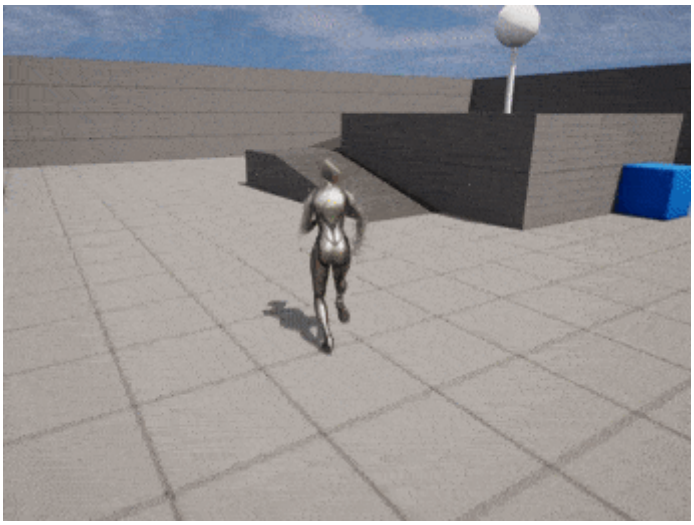


Figure 5.5 Automatic Head turning animation to focus on the large white sphere.

AHT uses control rig once again, this works by using an *aim* function which aligns the rotation of two axis towards a global target. It does this by checking if it is within the radius of the POI [Fig 3.7] and then adding the current target to that object.

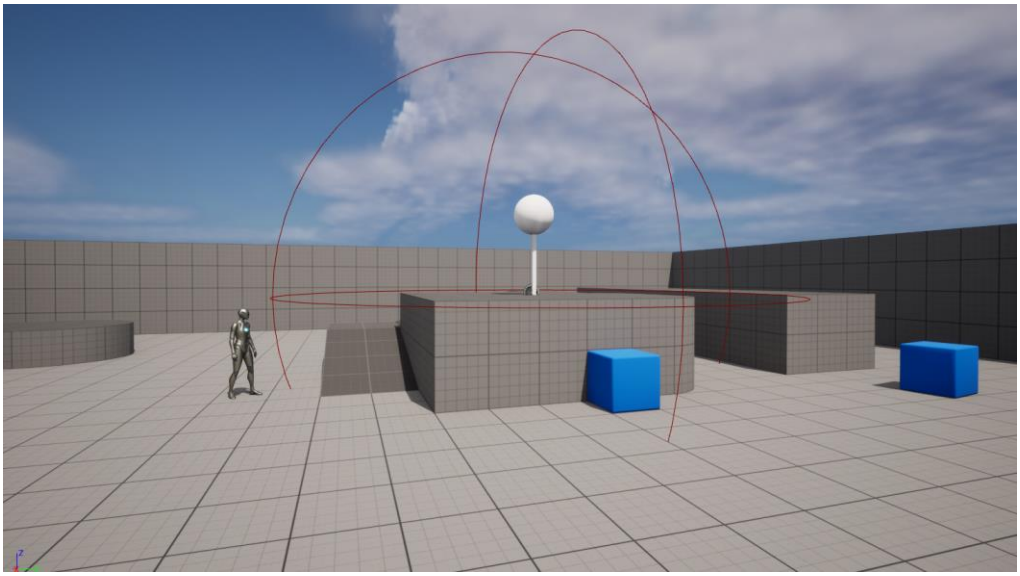


Figure 5.6 Visual of the range of the POI

For this control rig, no direct IK was implemented but instead I called the aim function of each object 3 times (for 3 separate bones). Each transformation of the bones was then weighted in increasing intensity as it got further down the hierarchy. For example, *middle of the spine* was only weighted at 0.25 but the *head* was weighted at 0.75.

To stop the bones from rotating to an unnatural state I create a constrict by finding the angle between the player's and POI's location and comparing that to a maximum rotation variable – which is 85 degrees for this project.

5.6 RAGDOLL

Ragdoll was quite a simple process to implement thanks to Unreal Engine's Physics engine. The mesh could be converted into a multitude of smaller collision bodies by adding a capsule collision around each bone of the skeletal mesh. From this the engine can simulate a pure physics solution (assuming there is no active input). The tricky part was rerigging the ragdoll back to the premade animations somewhat seamlessly. This included 3 steps:

1. Disable inputs from the player controller.
2. Determine if the body is predominately on its front or back. It checked this with a line trace from the pelvis in behind and in front of the character. This function then would add a different animation to a *Montage* [36], which is a tool that allows you to call certain animations within a Blueprint.
3. Called the animation montage to play and then attach the mesh back to the capsule collider. From this I then reset the two to the relative translation and rotation the player had before the ragdoll activation.

Within this player control, I only wanted the ragdoll to be activated when landing from a tall height. For this I created a checked for when the body as just become ground after falling, if the magnitude of the velocity exceeds this threshold before it lands then the ragdoll will be activated.

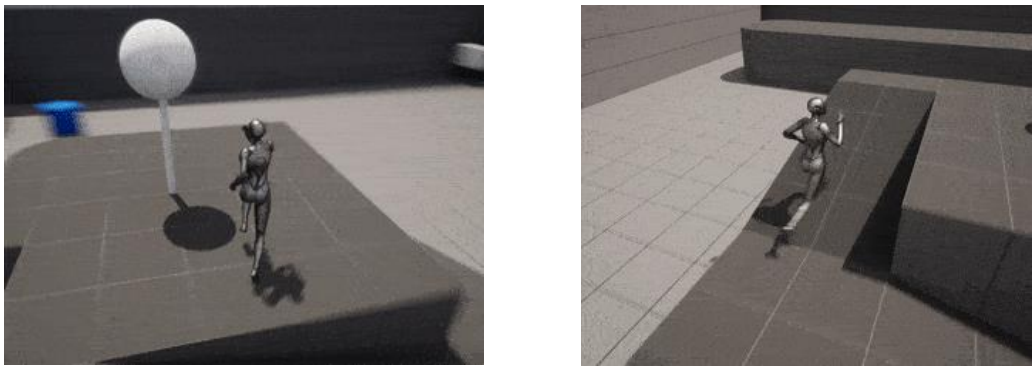


Figure 5.7 Different animation for getting up after enabling Ragdoll.
***LEFT:** Back* ***RIGHT:** Front.*

5.7 PARTIAL PHYSICAL ANIMATION

This form of procedural animation uses a similar method as the ragdoll method mentioned before. However, this time instead of just using ragdoll animation with no active input, I blended it over top of a pre-made animation. The result can be tuned by tweaking a number of factors such as the *orientation* and *angular velocity strength* as they work to constrain the physical animation to the form created by the premade animation.

This physical animation was also restricted to just the upper body for this is example. This made it look more stable and dynamic as opposed to a full body solution.

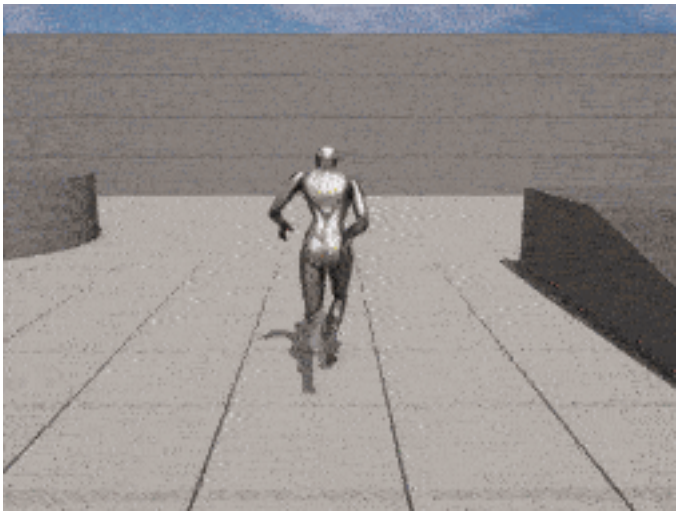


Figure 5.8 Physical base animation layered over the pre-made animation resulting in a more "zombie-like" run.

5.8 Animation Blending

All these animations for different body parts needed to be blended. If not, they would override each other and not be able to happen simultaneously. This is particularly important for the different versions of IK since the feet, hands and head should not affect each other.

This can be handled in Unreal Engine by playing the animations and then caching them after they have been played. Before they are cached, they will go through some specific checks on whether we want the animations playing at this moment. Then we can call *Blend Poses by bool* to then interpolate between or play different animations.

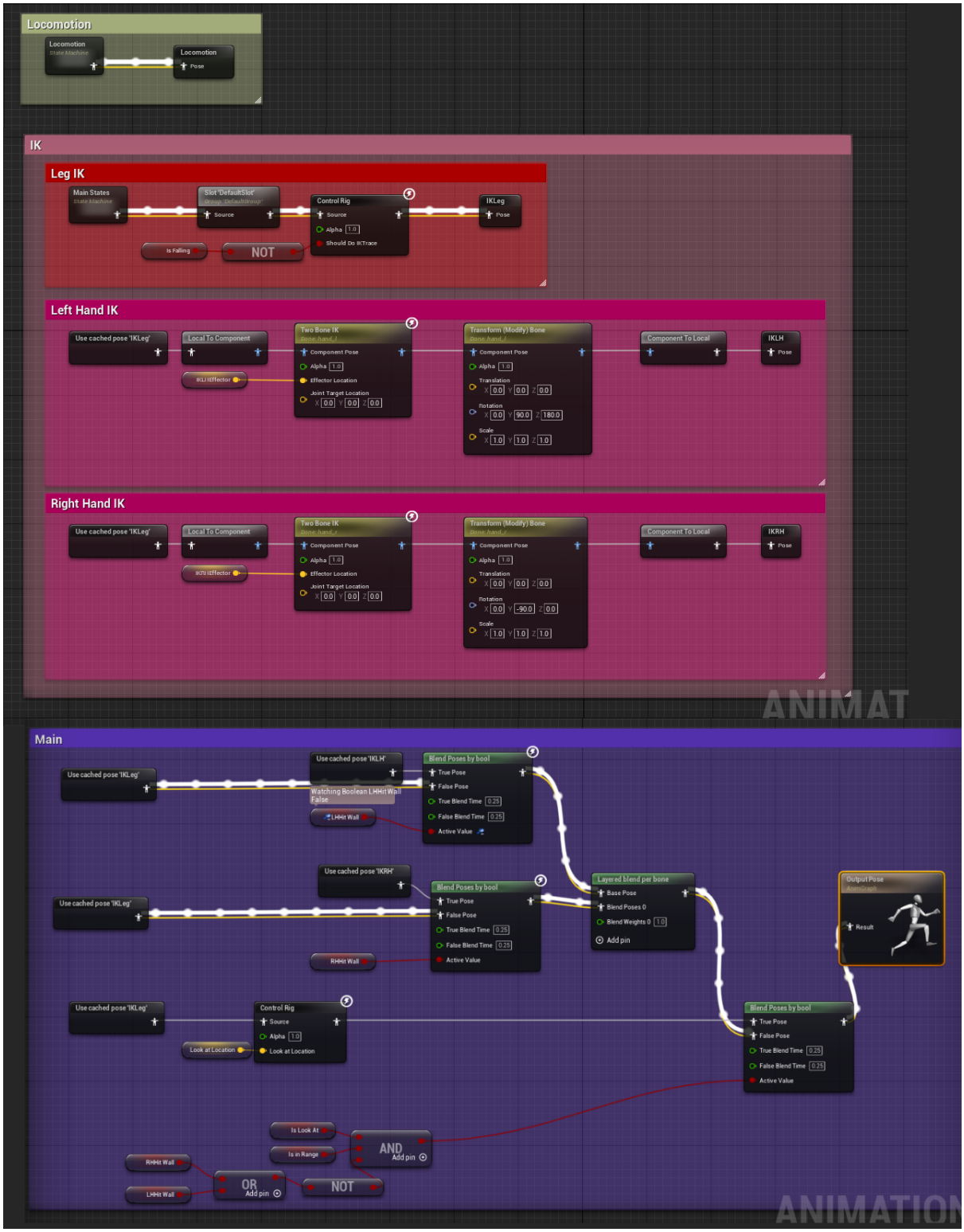


Figure 5.9 The Animation Blueprint for the player character.

5.9 / ASSETS USED

5.9.1 Trello

For project management I used Trello since it is something I am comfortable with, and it allows me to work to the *Agile* standard with a Canban-Sprint style of workflow (fortnightly sprints). There is a

point system plugin I used to help record and monitor my *velocity* during each sprint. This allowed me to keep track of my rate of work and project progress.

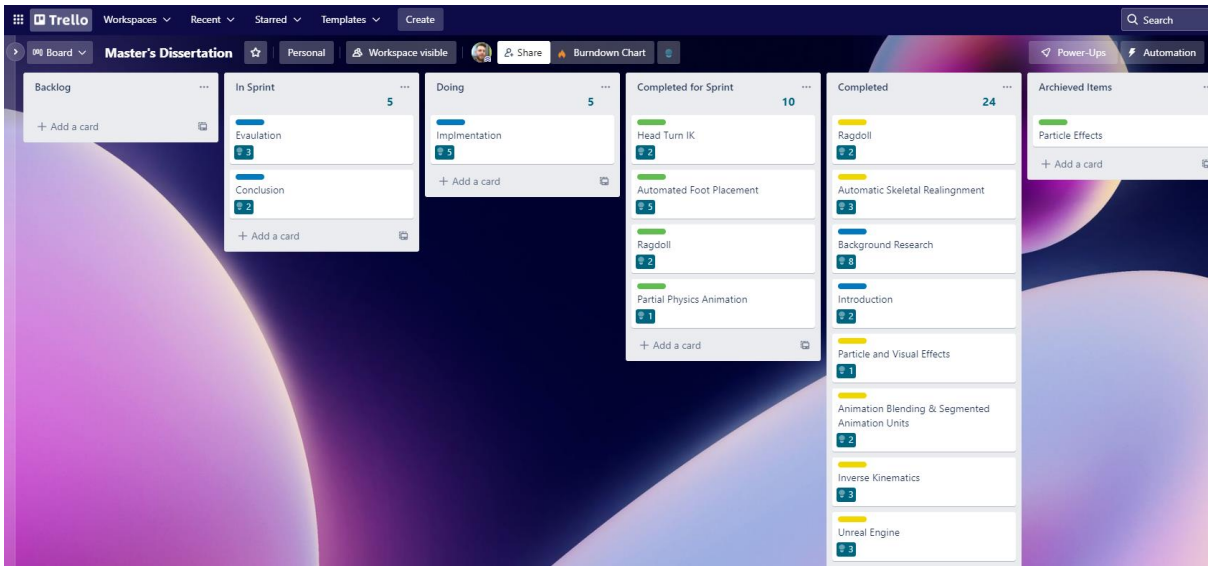


Figure 5.10 Picture of my Trello board

5.9.2 GitHub

During the research and writing of my dissertation, I worked from multiple locations including my flat and campus buildings such as the Urban Science and Fredrick Douglas Buildings. GitHub and GitHub desktop allowed me to seamlessly update my directories and work on multiple parts of the project at once using different branches and pull requests. Having some sort of version control was very helpful for collecting pictures from my past iterations to show where I have improved.

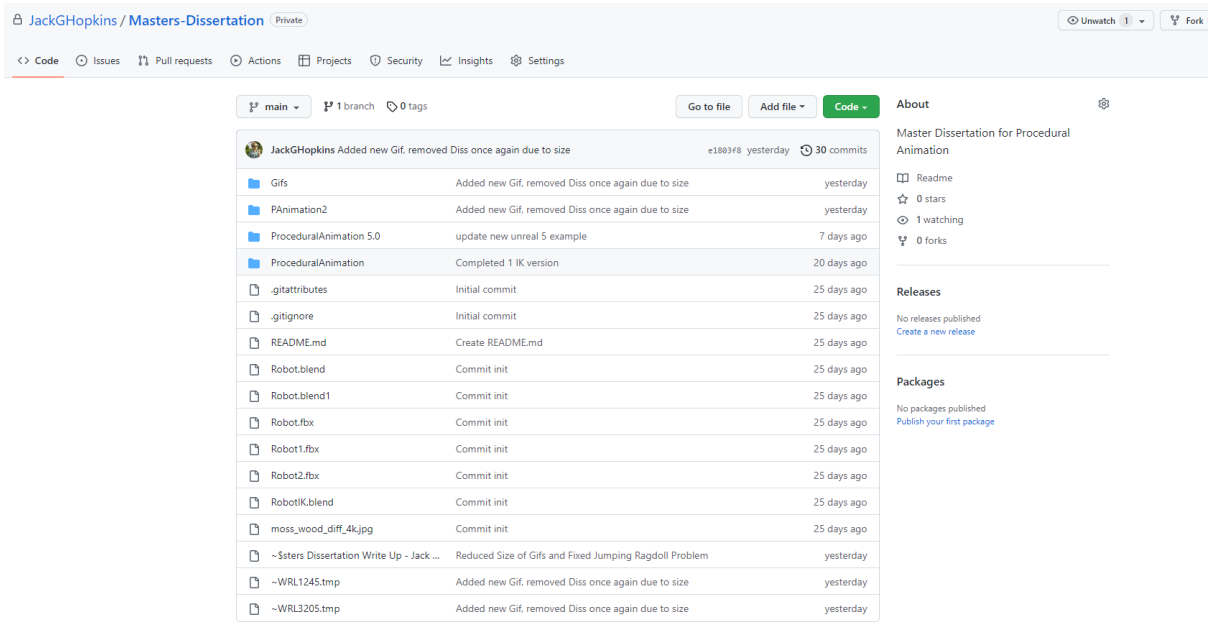


Figure 5.11 GitHub repository

6 RESULTS AND EVALUATION

Within this section, I will be analysing my implementation in a qualitative manner. Thus, the decision of quality will be from my subjective lens rather than, for example, running the numbers and then compiling them into graphs.

That being said, my analysis will try to cover 3 areas for each technique:

- What was the aim of implementing the technique? This is for both the technique in isolation and as part of the whole.
- Does it fulfil this aim? / How well does it fulfil this aim?
- What situations other than this aim could it have been used in, and would it be a better fix?

6.1 AFP

The aim of AFP was to improve a sense of realism with the game. While it is a small detail and video game players are generally satisfied with some foot skating and floating, it is still a neat detail that often will be subconsciously noticed.

The AFP implementation is somewhat lightweight and not overly complex. Because of this, I find there to be little reason not to try and implement it in some regard for many 3rd person games, particularly those that want to immerse the player in realism. For games that are competitive multiplayer games I would say this feature is not really a necessary focus. Often these games' focus is on other aspects aside from procedural animation. Furthermore, having consistent and precise hurt/hit boxes is much more crucial for a competitive environment then subtle animation techniques.

The AFP itself is consistently accurate for the lower steps. The only issue with the AFP is that sometimes the mesh passes through the steps to get to its position. This is less likely to happen with smaller steps, however it is not hard to recreate if you try. That being said, within general gameplay it is not very noticeable unless you are looking for it. If the average consumer doesn't notice it or care greatly, then I see it as a successful technique.

Figure 6.1 Small AFP issues on shorter steps

Taller steps, specifically those that result in the foot not touching the floor, have more apparent visual glitches. This is the result of the pelvis being realigned over the higher foot. This causes the lower foot to clip through the ground beneath them in an unnatural way.

Figure 6.2 Visual glitch of taller steps with the AFP

This could be resolved through more checks but that could over complicate the implementation process which would probably warrant a re-evaluation as to whether or not it would be worth it for your specific project. For future projects it might be worth spending time on this specific issue.

As for other issues that could arise, I was surprised that I didn't have much of an issue with the foot being unable to decide which step to place the foot on. Looking closer at Figure 4.7 you can see in some instances the character's ankle wiggle as it is adapting to its environment in real time. This makes it look less purposeful and natural. It should be noted that Judgement's IK controller is much more sophisticated and complex than mine as the jitter is probably caused by the character having two ray traces to fix the issue I mentioned earlier of parts of the foot clipping through the floor. However, it would be up to the developer to decide which one they prefer.

One other method I used was the FABRIK node (see theory explain here: Inverse Kinematics) instead of Full Body IK. This resulted in very comparable results when adjusting for lower steps. However, when it moves to higher steps the differences become more apparent.

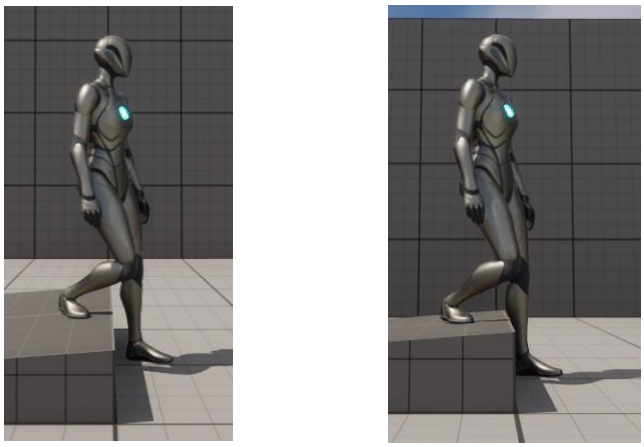


Figure 6.3 ***LEFT****: FABRIK* ***RIGHT****: FULL BODY IK*

Figure 6.4 FABRIK when standing on higher steps.

When standing on higher steps [Figure 6.4] instead of constraining the foot the FABRIK elects to stretch the bones to fit within that area. This is different to Full Body IK where the bones are just fully extended.

Now, this is possible to be constrained within FABRIK, but I did not for this example because there could be a possible use for this result. Within many video game animations, squash and stretch are important principles used to help create smooth and cathartic animations [8] [37]. Furthermore, if you have a character that has an item that stretches and need to grapple on to a place automatically, this shows that procedural animation here is an option.

Figure 6.5 Elongating arms is a power Sonic gains in his "Werehog" sequences in Sonic Unleashed.

6.2 AHP

Much like AFP, AHP's aim is much more focused on realism. Its application is arguably more useful in more types of video games because it is not uncommon for hands to be visible in first person games. However, the application for this technique is probably more varied since our hands are used for a wider array of actions.

In my implementation, I find my AHP to be an overall success even though it has more noticeable issues. There are more instances of weird interactions resulting in odd animations. The first is that you can run along the wall with your arm full extended against it. This is a fix that seems somewhat simple as the range of detection against the wall could be lessened.

The second issue is one that is more complex. When a hand is being released from the wall it will animate in a natural motion causing it to fly off the wall.

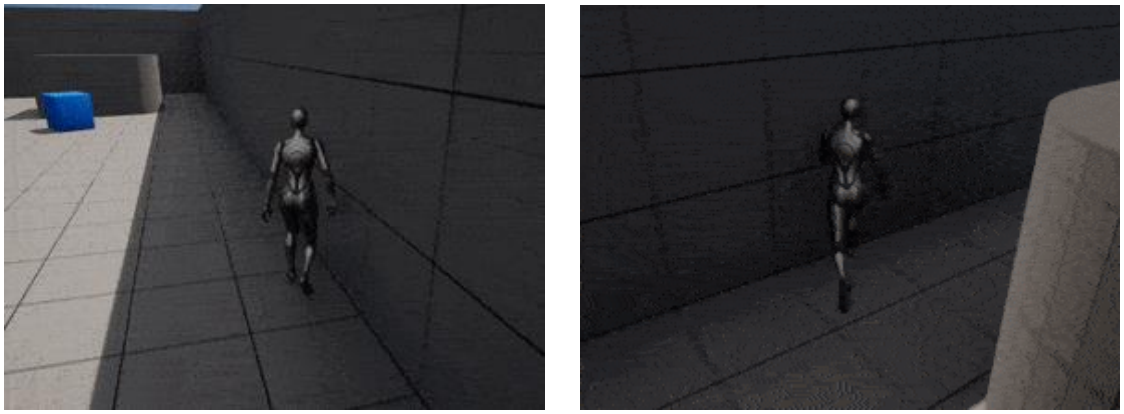


Figure 6.6 ***LEFT:*** *Running with your hand stuck against the wall is possible.*
RIGHT: *In some instances, releasing your hand from the wall looks a little odd.*

Ultimately, while this feature is a nice one in theory and works in most situations, it would be understandable if developers decided to ignore it for a few reasons:

- Arms and hands are very useful for many different situations, especially comparatively to the legs and feet. This means that any implementation would have to be significantly more complex to cover these many situations. This could be dealt with by a mix of procedural animation and premade animations.
- This is not even taking into consideration fingers and hit fidelity motions which procedural animation has yet to be shown to be very successful with.

6.3 AHT

AHT is a very simple movement that is not nearly as subtle as AFP. Its uses aren't just for realism but can also be used to guide a player to a POI in a more engaging and intuitive manner for both cartoony and realistic games. It is very expandable and can be used on multiple locations. Furthermore, I had very few, perhaps even no issues at all with the head turning from an aesthetic perspective.

This feature would see more milage in 3rd person adventure games and is one I would highly recommend for developers to consider. It is simple to implement and is a neat piece of animation that would be effective.

6.4 RAGDOLL

The ragdoll's aim was mostly for comedic purposes, and for that alone I'd say it was very much a success. There is good reason it is such a popular technique for less serious games as something about it does feel intrinsically goofy. As a developer, it elicited the most chuckles from me on this project.

One unintended result I came across was how the actor's weight changed upon activating the ragdoll state. This led to some funny results:

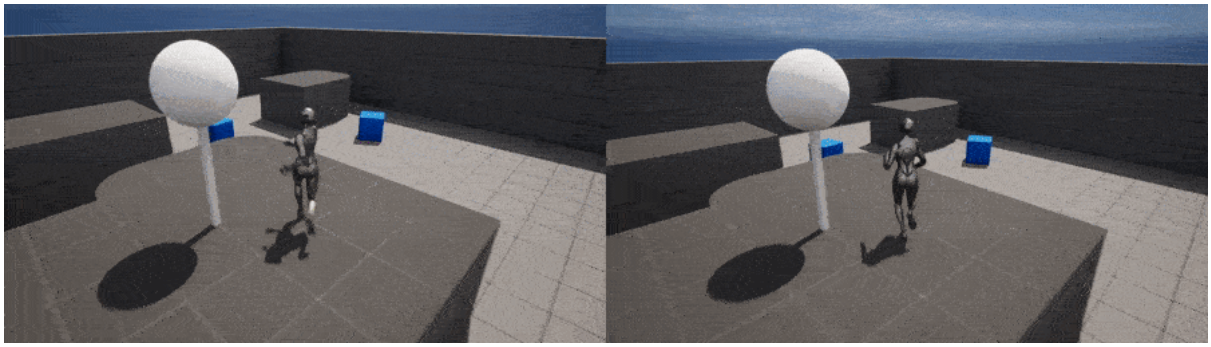


Figure 6.7 ***LEFT:*** *Normal Body*
RIGHT: *Ragdoll*
The differenced in masses meant you could jump a lot higher after becoming a ragdoll.

While this was an unintended side effect and a difference that more developers may want to fix, I still see much potential in this difference, both comedic and otherwise. One use could be for a game mechanic where accessing certain platforms would require precise activation of the ragdoll to reach that height making for some unique game design.

In conclusion, I think the use of the ragdoll, while not incredibly varied, is very effective, delivering upon its aims. Plus, it has some possible interesting potential for the future.

6.5 Partial Physical Animation

In some ways I see this style of animation being the evolution from fully physical animation (a.k.a Ragdoll) as we are seeing it's use grow in frequency. Furthermore, when it is part of a game it's more likely to be a staple / focus of the gameplay itself because it can provide similar comedic flavouring while still allowing the player to play the game (unlike ragdoll which restricts any active input). It can also be merged with premade animations and tweaked to give a desired result.

This technique, while being more complex to implement, was worth the time investment. Especially if this is going to be a fundamental part of your game's animation direction, which has been shown to be viable by many successful games.

Figure 6.8 One of break out successes of 2020 was Fall Guys. It made great use of partial physical animation.

As for the implementation within my project, I'd say it was a success. I created a flexible control which produced the wanted results without issue. It overlapped well with the premade animations creating a unique feeling that could be used for a more comedic game or a creepy uncanny enemy that could be used to scare the player.

The initial aim of this technique was for comedic purposes like the ragdoll before it, however, my implementation made a good example of a zombie running cycle that could be used for more eerie, less light-hearted games.

6.6 Animation Blending

The aim of animation was to integrate different techniques together seamlessly. This was overall a success as blending the AHP and AFP was perfect as well as the transitioning in and out of AHT. However, what I did not do was try to implement multiple different techniques for the same bones. This would have made the blending significantly more complex and could have caused many more complications in the future.

6.7 Unreal Engine

Unreal Engine has been an incredibly powerful tool in this development process – in some cases it has multiple ways of implementing the same technique through typical blueprints and the control rig system. It makes the task of implementing these techniques easy enough that the engine's ability should not be a greatly significant factor if developers wanted to attempt this style. However, this one example of implementation doesn't find a limit to the system or stress test it's abilities in any meaningful way. With this in mind, the ease of implementation can only be regarded for the techniques expressed in this paper.

While this is speculative, I believe that more advanced techniques (particularly those that are IK focused) are perfectly possible and practical in Unreal Engine 5.0. Furthermore, with Unreal Engine 5.0 being the standard engine for most AAA video games currently and in the foreseeable future, the ability of these tools will be ever expanding, and it will see much developer support.

7 Conclusion

The area of procedural animation is incredibly wide and is finding itself increasing in use as technology progresses and more techniques are being refined and discovered. Unreal Engine 5.0 is a powerful tool that allows for many different techniques to be implemented in a multitude of ways. Control Rigs is a particularly impressive tool and I look forward to seeing Epic Games expand upon it the future.

The objectives given at the start of this dissertation were all met as I was able to implement many techniques in Unreal Engine 5 to a competent degree, showing I had both researched these techniques thoroughly and learnt how to implement them in Unreal Engine 5.

The implementation of the techniques discussed in the background research were largely a success, however, some of them, such as the automatic foot placement, could use refinement. The most successful technique implemented was the automatic head turning as it had issues with it, however the most useful techniques for developers, I'd argue, would be partial physical animation as it is a possible core technique that can be used for a variety of characters and themed games.

Jonathan Cooper argues that no satisfyingly realistic wholly procedurally animated controller has been implemented in a commercial video game [8]. And from my research, I would agree. However, there is a plethora of video games across a vast number of genres, created by a diverse set of studios that have used some procedural animation to help with the development time of a game, or instead,

set the tone of their video game. Video games are not just simulations of the real world, but artist and entertainment pieces where these tools are used when and where developers decide they are more effective.

Within my implementation, there are some avenues that I could have explored but choose not to for the sake of brevity and scope. The most glaring omission was the lack of particle effects. While this uses a completely different system within Unreal Engine, I didn't include it because it did not make use of a skeletal rig. In further research, it may be worth looking into this area more closely.

Another possibility for further work would be to spend more time investigating a modern game example that relies heavily on procedural animation. Video games like Spore and Rain World have been written on extensively but it would be worth looking into a larger more recent studio that is pushing the technology in new places never seen before.

8 FIGURE LIST

9 WORKS CITED

[1] D. Plans and D. Morelli, "Experience-driven procedural music generation for games," *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 4, no. 3, pp. 192-198, 2012.

[2] J. Freiknecht and W. Effelsberg, "A Survey on the Procedural Generation of Virtual Worlds," *Multimodal Technologies and Interactio,* vol. 1, no. 4, 2017.

[3] M. Brown, *How the Nemesis System Creates Stories,* Game Maker's Toolkit, 2021.

[4] T. Short and T. Adams, Procedural generation in game design, CRC Press, 2017, p. 1.

[5] M. Hendrik, S. Meijer, J. Van Der Velden and A. Iosup, "Procedural content generation for games: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM),* vol. 9, no. 1, pp. 1-22, 2013.

[6] Spore, Electronic Arts, 30 May 2008. [Online]. Available: https://www.youtube.com/watch?v=6dilwN8mN6s.

[7] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard and K. van Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," *ACM Transactions on Graphics (TOG),* vol. 27, pp. 1-11, 2008.

[8] J. Cooper, "Advanced Game Animation Concepts," in *Game anim: Video game animation Explained*, CRC Press, 2021, pp. 75-77.

[9] R. S. Johansen, "Automated semi-procedural animation for character locomotion," *Aarhus Universitet, Institut for Informations Medievidenskab,* 2009.

[10] J. Jakobsson and J. Therrien, "The Rain World Animation Process," 16 July 2017. [Online]. Available: https://www.youtube.com/watch?v=sVntwsrjNe4.

[11] C. Jake, "Cuphead's Animation Process and Philosophy," GDC, 17 August 2017. [Online]. Available: https://www.youtube.com/watch?v=RmGb-jU3uVQ.

[12] S. David, "The Motion Capture Pipeline of The Last of Us," GDC, 18 December 2016. [Online]. Available: https://www.youtube.com/watch?v=2GoDlM1Z7BU.

[13] The Cutting Room Floor, “Super Mario 64 (Nintendo 64),” April 16 2022. [Online]. Available: https://tcrf.net/Super_Mario_64_(Nintendo_64)#:~:text=Mario%20has%20a%20total%20of,indexed%20from%200%20to%20208.. [Accessed 22 April 2022].

[14] A. Hertzmann and V. Zordan, “Physics-based characters,” *IEEE Computer Graphics and Applications,* vol. 31, no. 4, pp. 20-21, 2011.

[15] W. T. Reeves, “Particle Systems - A Technique for Modeling a Class of Fuzzy Objects,” *ACM Transactions on Graphics (TOG),* vol. 2, no. 2, p. Particle SystemsmA Technique for Modeling a Class of Fuzzy Objects, 1983.

[16] Epic Games, “Unreal Engine 5.0 Documentation - Particale Lights,” Epic Games, [Online]. Available: https://docs.unrealengine.com/5.0/en-US/how-to-create-particle-effects-that-emit-light-in-niagara-for-unreal-engine/.

[17] Epic Games, “Unreal Engine 5.0 Documentation - Creating Visual Effects,” Epic Games, [Online]. Available: https://docs.unrealengine.com/5.0/en-US/creating-visual-effects-in-niagara-for-unreal-engine/. [Accessed 25 April 2022].

[18] DillonGoo, “Mesh Trails Add-on,” 15 December 2021. [Online]. Available: https://www.dillongoo.com/products/mesh-trails.

[19] J. Lindquist and V. Brodin, “Advanced Niagara Effects | Inside Unreal,” Epic Games, 6 November 2020. [Online]. Available: https://youtu.be/31GXFW-MgQk?t=85. [Accessed 18 May 2022].

[20] J. Švelch, “Comedy of contingency: making physical humor in video game spaces,” *International Journal of Communication,* vol. 8, p. 23, 2014.

[21] R. a. O. D. Featherstone, *Robot dynamics: equations and algorithms,* vol. 1, IEEE, 2000, pp. 826--834.

[22] M. a. s. f. g. r. c. i. g. simulations, *Method and system for generating realistic collisions in graphical simulations,* Google Patents, 2000.

[23] A. Witkin, “Physically based modeling: principles and practice constrained dynamics,” *Computer graphics,* vol. 9, p. 27, 1997.

[24] T. Jakobsen, “Advanced character physics,” in *Game developers conference*, 2001.

[25] A. J. (. Walker, D. (. Cooper and I. Turner, “SMITE - Update Show VOD - King of Uruk,” Titan Forge, [Online]. Available: https://youtu.be/FdNSnxBuejE?t=7857.

[26] D. Kushner, “Engineering Spore,” *IEEE Spectrum,* vol. 9, no. 45, pp. 36-40, 2008.

[27] Fatbajuronzorz, “Weird Spore Creature WTF,” 29 July 2008. [Online]. Available: https://www.youtube.com/watch?v=U6--7Uk_YeA.

[28] A. a. L. J. Aristidou, “Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver,” 2009.

[29] MathWorks, “Inverse kinematics (IK) algorithm design with MATLAB and Simulink,” MathWorks, [Online]. Available: https://www.mathworks.com/discovery/inverse-kinematics.html#:~:text=Inverse%20kinematics%20is%20the%20use,between%20bins%20and%20manufacturing%20machines.. [Accessed 18 May 2022].

[30] A. Aristidou and J. Lasenby, “FABRIK: A fast, iterative solver for the Inverse Kinematics problem,” *Graphical Models,* vol. 73, no. 5, pp. 243--260, 2011.

[31] K. Lozar, “Twitter,” 19 Jan 2022. [Online]. Available: https://twitter.com/klemen_lozar/status/1483638638986424321?cxt=HHwWgsCjveje-JYpAAAA. [Accessed 22 May 2022].

[32] N. (. Berry, “How is the hair in celeste made?,” 24 August 2018. [Online]. Available: https://www.reddit.com/r/gamedev/comments/9a0cfr/comment/e4rvrg2/?utm_source=share&utm_medium=web2x&context=3. [Accessed 22 May 2022].

[33] Epic Games, “Unreal Engine 5.0 Documentation - Control Rig,” Unreal Engine, [Online]. Available: https://docs.unrealengine.com/5.0/en-US/control-rig-in-unreal-engine/. [Accessed 2022 05 25].

[34] Epic Games, “Unreal Engine 5.0 Documentation - Full Body IK,” [Online]. Available: https://docs.unrealengine.com/5.0/en-US/control-rig-full-body-ik-in-unreal-engine/.

[35] Epic Games, “Two Bone IK,” Epic Games, [Online]. Available: https://docs.unrealengine.com/5.0/en-US/animation-blueprint-two-bone-ik-in-unreal-engine/.

[36] Epic Games, “Unreal Engine 5.0 Documentation - Animation Montage,” [Online]. Available: https://docs.unrealengine.com/5.0/en-US/animation-montage-in-unreal-engine/.

[37] F. Thomas, O. Johnston and F. Thomas, The illusion of life: Disney animation, Hyperion New York, 1995.