The use of Procedural Adaptive Animation to increase Visual Fidelity

Abstract

This project focuses on the problem of animations being created outside of the context of their video game environment, this being a problem as when these animations are used they can conflict with the surroundings in a noticeable way to viewers. It is to this end that this project examined three agents that have had their animations controlled in separate ways. For clarity of movement and easy identification of visual differences the model used for the agents were a spider model named '*Free Low Poly Fantasy Spider'*. One was the 'control' (i.e., use traditional methods of animation) another used methods of procedural animation to move in a similar fashion to the traditional animation and the last contained the previous' procedural animation, albeit utilising it to interact with world around it when planning its movement. Data was collected from these three agents in the form of visual fidelity on a case-by-case basis in comparison with reality. Another metric that was collected involved how they affected computer performance, and in what number of instances do they begin to cause a problem. Unfortunately, the final project differed from the proposition, in that it lacked the proposed Artificial Intelligence functionality and spider behaviours, alongside several errors in the procedural animation itself, which produced unnatural movement. It was found in conclusion, that traditional animation is more cost-effective, less resource intensive and maintained a higher visual fidelity than the final state of the project. However, the project holds potential in that the fixing of errors and optimisation may provide a good long-term investment to the initial problem.

Table of Contents

Table of Figures

1. Introduction

In the Video Game Industry Computer, graphics have developed fast in comparison with the age of the industry. The movement from no graphical image representation all the way to full three-dimensional worlds opened up large amounts of creative space for designers, artists and programmers. However, with every jump in complexity and freedom more time was needed to develop the assets used in these games and subsequent tools that could be used for each discipline (Blow 2004). Due to the branching specificity of the many fields, interaction at the base level is very restricted and potential areas where different disciplines could provide solutions to each other are rarely explored. One such problem is the lack of visual cohesion present in actions that interact with the surrounding world (Tinwell 2014) .

1.1. Motivation

The majority of video game animations are created via the use of motion capture software, which is then manually cleaned up by animators to produce a finished sequence. It has been estimated that roughly 70% of motion capture services are used of owned by the video games industry (Menache 2000). This method has been highly successful in producing high quality animations, with the caveat that these animations look good when viewed out with any context and in perfect situations. This can be mitigated to some degree by interpolating between various slightly different animated variations on the pose dependant on the circumstances.

One company that has utilised excessive motion capture with tweaking minor details to great effect has been *Naughty Dog*. Their process for utilising motion capture contains several stages and multiple softwares to manage and process all of the data into a usable and modifiable format (Damon 2015). However, this is horribly time-consuming, inefficient and reduces the options available to small and middle-sized teams who follow industry trends. A shift in approach to the problem is required in order to maintain efficiency and allow competition in the vein of fluid animation regardless of team size.

Since movement is one of the most performed actions within video games regardless of genre it has been chosen as the animation to be made procedural. One preferred use of procedural animation is merging it with motion capture data to utilise the strengths of both. However, with this method it is only possible to get full motion data for biped humanoids and only partial data for quadrupeds. Even though the use of procedural animation can provide unnatural motion, if refined it can provide a solution to the problem of animating non-bipeds and non-quadrupeds. One of the greater problems with procedural animation mirrors that of premade animation in that the action itself exists within its own frame of motion, in that it contains no knowledge of the animations prior to or later than it. A small attached Artificial Intelligence could solve this problem by manipulating the legs targets and taking into account the surroundings and the path to the goal.

1.2. Problem Statement

The nature of these animations interacting with the surrounding world is what allows them to be automated via programming as the data for surrounding objects can be accessed and utilized in the positioning of the animation. Regardless of the size of an animation team or the time given to them, not every potential position for an animation can be animated. The traditional method for this problem is to have a standard animation for each action, and use them regardless of the context. While this method gives acceptable results while providing within a deadline there is a significant lack of visual fidelity present in the final product as the programmed moment involving interaction with the world is reached it becomes very clear how disconnected the animation is from the world. One of the major disconnects seen is known as 'clipping' where two models intersect which can, at best remain unseen by the player or at worst blatantly remind the player how none of the objects within the game truly carry any weight or how they can be exploited with physics present elsewhere within the game. Automation reduces the chances of clipping with the animation replaced as it can take in the surrounding world as well at the character’s position.

As well as increasing visual fidelity, this project is not a simple exchange of workload from animators to programmers but a reduction as well. It would remove much of the added complexity in large state machines with much of the blending between states becoming obsolete, as the agent in the world’s movement is a product of external stimuli serving as a replacement to what its animation states are derived from. This added reduction in labour is caused by the Artificial Intelligence that is attached being robust enough to require no hard coded responses as it would move away from threats and self-create new goals. It would also be able to be placed into a game world with no pre-loaded knowledge and act on its own, making it a great investment as its modularity allows for usage in future unrelated projects.

This dissertation explores methodologies for kinematic animation in conjunction with procedural animation techniques that currently exist and determine which one is most suitable for the simulation. Methodologies for behavioural AI are then explored and the most suitable is similarly decided upon. A simulation is then presented containing a model utilising procedural animation and the behavioural AI module, as well as a video game 'world' for it to traverse. The Inverse Kinematic solution used is a C# adaption of the Forward and Backwards Reaching Inverse Kinematic (FABRIK) algorithm (Lasenby and Aristidou 2011) and the movement system involves turning the potential target area into voxels to find the most efficient route, then interpolating the movement through a Hermite spline (Kreyszig 2006). The Artificial Intelligence module will involve the use of ant-colony optimisation (Dorigo, Birattari and Stutzle 2006) to seek a series of goals to the end point, while avoiding areas of 'threat' (user created areas to test visual fidelity of sudden changes in movement and plans). In closing, this dissertation will determine if the data gathered shows whether this new method is more or less efficient than the current state of the industry.

2. Literature review.

2.1 Procedural Animation

Procedural techniques have been used in video games since the 1980's, as memory capacity was limited so content was generated at runtime algorithmically (Toy, Wichman and Ken 1980). As memory capacity became less of a constraint procedural techniques moved to provide content that could instead be automated instead of created beforehand, i.e. character's heads moving to face the player in (Quake 3 Arena 1999).

As technology has improved and as space has become less of a concern, many procedural techniques have become obsolete. However, the use of procedural techniques in animation has begun to rise as many companies strive for efficiency and cost effectiveness and this approach can minimize the amount of Manpower required on animation teams.

One such technology used in the industry is euphoria (NaturalMotion 2007), which is used to simulate human self-preservation in the instance of bipedal models in rag doll situations. The technology of euphoria is a very complex, and provides natural and life like simulations that are unique dependent on the surrounding factors.

The game (Overgrowth 2013) utilises an interesting and different approach by not using kinematics at all for any of the major animations (Rosen 2014). It instead uses different forms of interpolation between poses to achieve the desired effect, much like the use of blend shapes when posing facial animation. This leads to animations not existing in the conventional sense, meaning storage space is drastically reduced for every object using this method. Considering the use of interpolation in traditional animation is as management for interruptions, using this method every animation is interpolation the interruptions happened seamlessly.

The system present in (Rain World 2017) much more in line with the goal of the project, albeit in two dimensions instead of three. The gameplay, mechanics and physics are completely independent from the system, which has the sole purpose of visual clarity and fidelity. As it is newly released, the specifics of how it operates are not fully known. However, speculation based on the visuals presented gives the impression of each chain of joints seeking the closest surface in the world to attach to, and when the main body moves out of reach of that point it then seeks a new one.

2.2 Inverse Kinematics

Natural and realistic motions remain a present challenge within the fields of robotics and animation. There are a multitude of methods that have been devised to deal with this problem within the section are several of the more popular methods for solving the problem of inverse kinematics within the industry.

If we let the list of joints in a kinematic chain be represented by $\theta_1...\theta_n$, where *n* is the total number of joints and each θ is the corresponding angle in reference to the plane of rotation (assuming we have knowledge of the rotation axis). Certain joints in the chain are if specified as *end effectors*, which are the joints that are directly aiming for a target. To solve the problem of inverse kinematics each θ value must place the *end effectors* as close to each of their corresponding targets as possible. If there are *k* number of *end effectors* let their positions be denoted as $s_1...s_k$ relative to a fixed origin position. The column vector $(s_1...s_k)^T$ can be written as $\vec{s}$. The target for each *end effector* have their positions defined as $\vec{t} = (t_1...t_k)^T$, where *k* denotes the same number of *end effectors* to correspond to each target. Let $e_i = t_i - s_i$, be the most preferred change of a position of the *end effector* to reach the desired target, where *i* is the *i*th end effector. This equation can be extrapolated to $\vec{e} = \vec{t} - \vec{s}$. The angles of the other joints in the chain are turned into a column vector $\vec{\theta} = (\theta_1...\theta_n)^T$ and the positions of the *end effectors* then become functions of the previous angles in the chain: $\vec{s} = \vec{s}(\vec{\theta})$, or, for $i = 1...k$ $s_i = s_i(\theta_i)$.

This is the mathematics behind *Forward Kinematics* (FK), where the goal is to rotate each joint from the root towards the *end effectors* so that their position matches the target. Whereby in *Inverse Kinematics* (IK) the rotations for each joint are calculated from the *end effector* towards the root. This is represented in the equation $t_i = s_i(\theta_i)$.

Although, there may be instances where the solution is impossible due to the target being out of reach of the chain, or the constraints of the joints prevent the best solution. It then seems to be necessary that any approach to solve the problem would have to be iterative and efficient if accuracy is desired. Most popular approach involves the use of Jacobian matrices to obtain a linear approximation in solving the IK problem.

2.2.1 Jacobian Inverse Methods

Jacobian methods are common in the field of inverse kinematics as one of the original ways in which to handle and compute points within 3D space. They are iterative methods where the functions of s_i are linearly approximated using the Jacobian matrix $J(\vec{\theta}) = \left(\frac{\partial s_i}{\partial \theta_i}\right)$.

Note that J can be viewed as an m × n matrix with scalar entries (with m = 3*k*). Forward dynamics are represented in the following equation that describes the velocities of the end effectors (using dot notation for first order derivatives): $\vec{s} = J(\vec{\theta})\dot{\vec{\theta}}$.

The Jacobian method leads to creating an iterative method for solving the Inverse Kinematics problem ($t_i = s_i(\theta_i)$). Supposing we have the current values for $\vec{\theta}, \vec{s}$ and $\vec{t}$, we can compute the Jacobian $J = J(\vec{\theta})$. We then look to find an update value $\Delta\vec{\theta}$ for the purpose of incrementing the joint angles $\vec{\theta}$ by $\Delta\vec{\theta}$, so $\vec{\theta} := \vec{\theta} + \Delta\vec{\theta}$.

Through use of ($\vec{s} = J(\vec{\theta})\dot{\vec{\theta}}$.) it can be estimated that: $\Delta\vec{s} \approx J\,\Delta\vec{\theta}$, as the change in end effector joint positions is directly impacted by the angles between them. The idea is that the value of $\Delta\vec{\theta}$ should be chosen such that $\Delta\vec{s}$ is approximately equal to $\vec{e}$ (to a specified tolerance for efficiency).

With these facts in mind, one approach to the problem is to solve the equation $\vec{e} = J\,\Delta\vec{\theta}$ for $\Delta\vec{\theta}$. Unfortunately, this cannot be solved for every instance being that J may not be invertible, and in cases where it is the equation $\Delta\vec{\theta} = J^{-1}\,\vec{e}$ is able to be produced, the results may yield incorrectly in the determinant of J is zero.

An alternative method involves the use of the equation J(θ) = $\left(\frac{\partial t_i}{\partial \theta_j}\right)_{i,j}$ where the partial derivative is calculated using the formula for $\left(\frac{\partial \mathrm{s_i}}{\partial \theta_\mathrm{i}}\right)$ with t_i substituted for $\mathrm{s_i}$. The reasoning behind $\left(\frac{\partial t_i}{\partial \theta_j}\right)$ involves altering our perception of the target being a free point, and instead as an attachment to the point of the end effector. The practical change is that with this formulation of the Jacobian, we are trying to move the target positions towards the end effectors, rather than the end effectors towards the target positions. However there is a downside to using this in place of $\vec{\mathrm{e}}$ = J $\Delta\vec{\theta}$, in that the computations of rotations become inconsistent when the target forces the chain backwards into itself.

2.2.1.1 Jacobian Transpose

Jacobian transpose method was first devised by (Wolovich W. A. ,Elliot H. 1985) and the basic idea is simple, where the transpose of J is used instead of the inverse. That is, we set $\Delta\vec{\theta} = \alpha J^T\vec{\mathrm{e}}$ (for some appropriate scalar α). Obviously the transpose is not the same as the inverse, however it is possible to justify the use of the transpose in terms of virtual forces.

The remaining piece of the equation is to determine the value of α, which is a way to reduce the value of the error vector $\vec{\mathrm{e}}$ after iterations. For this, we assume that the change in end effector position will be $\alpha JJ^T\vec{\mathrm{e}}$, and so we choose a value for α that is as close to $\vec{\mathrm{e}}$ as possible. Giving the equation $= \frac{\langle\vec{\mathrm{e}}, JJ^T\vec{\mathrm{e}}\rangle}{\langle JJ^T\vec{\mathrm{e}}, JJ^T\vec{\mathrm{e}}\rangle}$.

2.2.1.2 Jacobian Moore-Penrose

The Moore-Penrose method (also known as pseudoinverse) sets the value of $\Delta\vec{\theta} = J^\dagger\vec{\mathrm{e}}$, where the n × m matrix $J^\dagger$is the *pseudoinverse* of J. The pseudoinverse gives the best possible solution to the equation $J\Delta\vec{\theta}$ = $\vec{\mathrm{e}}$ in the sense of least squares. First, suppose $\vec{\mathrm{e}}$ is in the range (i.e., the column span) of J . In this case, $J\Delta\vec{\theta}$ = $\vec{\mathrm{e}}$; furthermore, Δθ is the unique vector of smallest magnitude satisfying $J\Delta\vec{\theta}$ = $\vec{\mathrm{e}}$. Second, suppose that $\vec{\mathrm{e}}$ is not in the range of J. In this case, $J\Delta\vec{\theta}$ = $\vec{\mathrm{e}}$ is impossible. However, $\Delta\vec{\theta}$ has the property that it minimizes the magnitude of the difference $J\Delta\vec{\theta} - \vec{e}$. Furthermore, $\Delta\vec{\theta}$ is the unique vector of smallest magnitude which minimizes $||J\Delta\vec{\theta} - \vec{e}||$, or equivalently, which minimizes $||J\Delta\vec{\theta} - \vec{e}||^2$.

The psudeoinverse, while a powerful solution, faces problems when dealing with singularities (contexts when a joint lacks degrees of freedom/ the determinant Jacobian matrix is equal to zero). When there is no singularities, the pseudoinverse's computations behave as expected, however when approaching a singularity the Jacobian matrices begin to return angles that overestimate the distance required, even when that distance is negligibly small. When put into practice, rounding errors mean that exact singularities are rare and so potential singularities have to be detected through value checking for near-zero values.

The pseudoinverse has the further property that the matrix $(I - J^{\dagger}J)$ performs a projection onto the nullspace of J. Therefore, for all vectors ϕ, $J(I - J^{\dagger}J)\phi = 0$. This means that we can set $\Delta\vec{\theta}$ by $\Delta\vec{\theta} = J^{\dagger}\,\vec{e} + (I - J^{\dagger}\,J)\phi$ for any vector ϕ and still obtain a value for $\Delta\theta$ which minimizes the value $J\Delta\vec{\theta} - \vec{e}$. The nullspace method can be found in *Automatic supervisory control of the configuration and behaviour of multibody mechanisms* (Liegeois 1977) where it was used to avoid joint limits. By suitably choosing ϕ, we can implement extra goals that operate in tandem with end effectors tracking the targets. For instance, ϕ might be chosen to try to return the joint angles back to rest positions in an attempt to avoid singular configurations.

With this in mind we can derive an algorithm for the pseudoinverse. Using the previously discussed $\vec{e} = J\Delta\vec{\theta}$ we can arrive at the normal equation $J^TJ\Delta\vec{\theta} = J^T\vec{e}$. For clarity we can let $\vec{z} = J^T\vec{e}$ and solve the equation $(J^TJ)\Delta\,\vec{\theta} = \vec{z}$. In principle, row operations can be used to find the solution to this equation with minimum magnitude; however, in the neighbourhood of singularities, the algorithm is inherently numerically unstable. When all of J's rows are linearly independent, then JJ^T is guaranteed to be invertible. In this case, the minimum magnitude solution $\Delta\vec{\theta}$ to the equation $(J^TJ)\Delta\vec{\theta} = \vec{z}$ can be expressed as $\Delta\theta = J^T\;(JJ^T\;)\,\vec{e}$. To prove this, note that if $\Delta\vec{\theta}$ satisfies this equation, then $\Delta\vec{\theta}$ is in the row span of J and $J\Delta\theta = \vec{e}$. Using these formulae allow us to compute the rotations for several instances of the inverse kinematics problem, and allow us to think 'outside of the box' in terms of traditional methods of computing the angles. However these methods serve better as examples as there is still instability when approaching singularities.

2.2.1.3 Jacobian Damped Least Squares

The damped least squares method avoids many of the pseudoinverse method's problems with singularities and can give a numerically stable method of selecting $\Delta\vec{\theta}$. Rather than just finding the minimum vector $\Delta\vec{\theta}$ that gives a best solution to equation $\vec{e} = J\Delta\vec{\theta}$, we find the value of $\Delta\vec{\theta}$ that minimizes the quantity $||J\Delta\vec{\theta} - \vec{e}||^2 + \lambda^2||\Delta\vec{\theta}||^2$, where $\lambda \in R$ is a non-zero damping constant. This is equivalent to minimizing the quantity $||\binom{J}{\lambda I}\Delta\vec{\theta} - \binom{\vec{e}}{0}||$. The corresponding normal equation is $\binom{J}{\lambda I}^T\binom{J}{\lambda I}\Delta\vec{\theta} = \binom{J}{\lambda I}^T\binom{\vec{e}}{0}$. This can be equivalently rewritten as $(J^TJ + \lambda^2 I)\Delta\vec{\theta} = J^T\vec{e}$.

Under the assumption that $J^TJ + \lambda^2\mathrm{I}$ is non-singular, the damped least squares solution is equal to $\Delta\vec{\theta} = (J^TJ + \lambda^2\mathrm{I})^{-1}J^T\vec{e}$. Now J^TJ is an n × n matrix, where n is the number of degrees of freedom. It is easy to show that $(J^TJ + \lambda^2\mathrm{I})^{-1}J^T = J^T(JJ^T + \lambda^2\mathrm{I})^{-1}$.

Thus, $\Delta\vec{\theta} = J^T(JJ^T + \lambda^2\mathrm{I})^{-1}\vec{e}$. The advantage of this equation over $\Delta\vec{\theta} = (J^TJ + \lambda^2\mathrm{I})^{-1}J^T\vec{e}$ is that the matrix being inverted is only m × m where m = 3k is the dimension of the space of target positions, and m is often much less than n. This equation can be computed without needing to carry out the matrix inversion, instead row operations can find $\vec{f}$ such that $(JJ^T + \lambda^2\mathrm{I})\vec{f} = \vec{e}$ and then $J^T\vec{f}$ becomes the solution.

However, the damping constant depends on the details of the multibody and the target positions and must be chosen carefully to make $\Delta\vec{\theta} = J^T(JJ^T + \lambda^2\mathrm{I})^{-1}\vec{e}$ numerically stable. The damping constant should large enough so that the solutions for $\Delta\vec{\theta}$ are accurate near singularities, but if the chosen constant is too large, then the convergence rate is too slow.

2.2.2 FABRIK

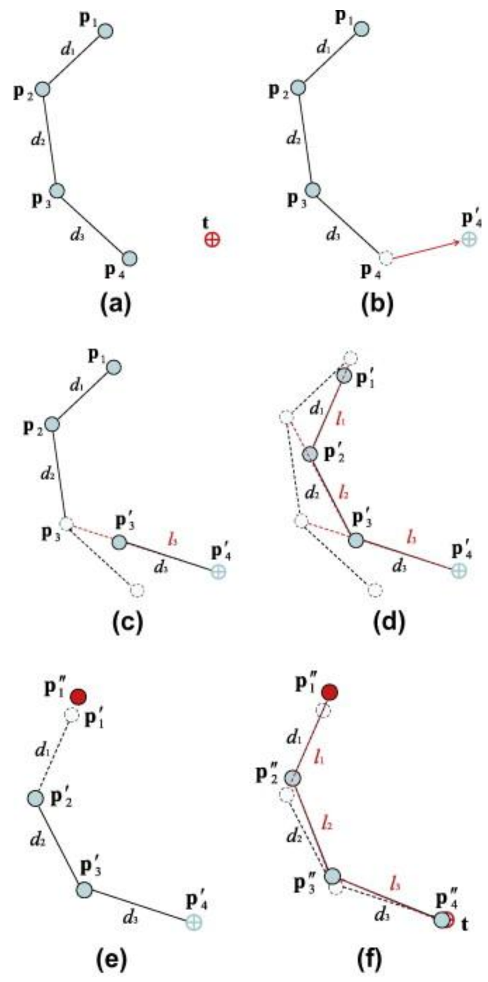

Figure 1 - An example of a full iteration of FABRIK for the case of a single target and four manipulator joints.

(a) The initial position of the manipulator and the target, (b) move the end effector $\mathbf{p}_4$ to the target, (c) find the joint $\mathbf{p}'_3$ which lies on the line L_3 that passes through the points $\mathbf{p}'_4$ and $\mathbf{p}_3$, and has distance d_3 from the joint $\mathbf{p}'_4$, (d) continue the algorithm for the rest of the joints, (e) the second stage of the algorithm: move the root joint $\mathbf{p}'_1$ to its initial position, (f) repeat the same procedure but this time start from the base and move outwards to the end effector. The algorithm is repeated until the position of the end effector reaches the target or gets sufficiently close.

Forward And Backwards Reaching Inverse Kinematics attempts to minimize the system error by adjusting each joint one at a time. This method starts from the last joint of the chain and works forwards, adjusting each joint along the way. It then works backward in the same way, in order to complete one iteration.

This method, instead of using the rotations of joints, treats finding the joint locations as a problem of finding a point on a line – massively reducing the computation required.

Assume $\mathbf{p}_1 \ldots \mathbf{p}_n$ are the joint positions of a specified limb. Also, assume that $\mathbf{p}_1$ is the root joint and $\mathbf{p}_n$ is the end effector, for the simple case where only a single end effector exists. The target is represented as **t** and the initial base position by **b**. FABRIK is shown in a graphical representation of a full iteration with a single target and four joints in Fig 1.

First the distances between each joint must be calculated via $d_i = |\mathbf{p}_{i+1} - \mathbf{p}_i|$, for $i = 1 \ldots n - 1$. It must then be determined whether the target is reachable or not, firstly by finding the distance between the root and the target - let that be named *dist*. If this distance is smaller than the total sum of all the combined joint distances, represented as the equation $dist < \sum_1^{n-1} d_i$, the target is within reach, otherwise, it is unreachable. If the target is within reach, FABRIK can begin reaching the joints towards the target, by using its two-stage iteration. In the first stage, the algorithm estimates each joint position starting from the end effector, $\mathbf{p}_n$, moving inwards to the manipulator base, $\mathbf{p}_1$. So, the new position of the end effector becomes the target position, $\mathrm{p}'_n = \mathbf{t}$. Find the line, L_{n-1}, which passes through the joint positions p_{n-1} and p'_n. The new position of p_{n-1} , p'_{n-1}, lies on that line with distance d_{n-1} from p'_n. Similarly, the new position of the p_{n-2} joint, p'_{n-2}, can be calculated using the line L_{n-1}, which passes through the $\mathrm{p}_{n-2}$and p'_{n-1}, and has distance $d_{n-2}$from p'_{n-1}. The algorithm continues until all new joint positions are calculated, including the root, p'_1.

Having in mind that the new position of the manipulator base, p'_1, should not be different from its initial position, a second stage of the algorithm is needed. A full iteration is completed when the same procedure is repeated but this time starting from the root joint and moving outwards to the end effector. Thus, let the new position for the 1st joint, p^n_1 , be its initial position **b**. Then, using the line L_1 that passes through the points p^n_1 and p'_2, we define the new position of the joint p''_2 as the point on that line with distance d_1 from p^n_1. This procedure is repeated for all the remaining joints, including the end effector. In cases where the root joint has to be translated to a desired position, FABRIK works as described with the difference that in the backward phase of the algorithm, the new position of the root joint, p^n_1, will be the desired and not the initial position.

After one complete iteration, it is almost always the case (observed empirically) that the end effector is closer to the target. The procedure is then repeated, for as many iterations as needed, until the end effector is identical or close enough (to be defined) to the desired target. The unconstrained version of FABRIK converges to any given chains/goal positions, when the total length of serial links is greater than the distance to the target (the target is reachable). However, if the target is not within the reachable area, there is a termination condition which compares the previous and the current position of the end effector, and if this distance is less than an indicated tolerance, FABRIK terminates its operation. Also, in the extreme case where the number of iterations has exceeded an indicated value and the target has not been reached, the algorithm is terminated (however, we have never encountered such a situation).

2.3 Artificial Intelligence

The inspiration for the adaptive aspect for the animation is derived from the paper *Robots that can adapt like animals (Cully et al. 2015)* . Within this paper, a robot is programmed to move towards a point and each limb movement is algorithmically decided with the goal of reaching the point as fast as possible. As this is in effect, the team damage the robots limbs in asymmetric and immobilising ways – and so the algorithm then decides how it can reach the point with fewer limbs using heuristics alongside it. The source code for the paper is freely available; however, it is specifically programmed for the physical robot mentioned within their paper and adaption to a different format would become a dissertation of its own.

2.3.1 Ant-Colony optimisation

One method that was investigated was *Ant-Colony optimisation* (Dorigo, Birattari and Stutzle 2006), this method involves a structure of pathfinding based upon several agents following previous agents who distribute randomly. Each agent creates a time-sensitive 'pheromone' trail behind it for other agents to follow and when it reaches a set 'goal' it returns down its own trail. Due to distribution and random chance, the shortest path (i.e. one where the pheromone trail is consistently refreshed) becomes the most optimal. This method was an interesting possibility, concerning the use of the main project agents as pathfinding agents.

3. Methodology

3.1 FABRIK Implementation

Implementation of FABRIK involves two layers of data structures to improve readability and modularity; these two layers are IKRigJoint and IKChain. IKRigJoint is a simple class that holds data from the points in the chain (position, rotation and constraints) and are collected into IKChain. The IKChain is a class that contains a list of the joints included in a specific chain and their target. It also calculates and stores the distances from point to point and the maximum length of the chain.

Within Inverse Kinematics, there are cases for when the full calculations in FABRIK should not be used, as they can be calculated in other ways more efficiently. One of these cases is in the situation that the chain of joints is less than two joints long. As this does not require any calculations towards reaching the target as it is not a chain and the solution is skipped.

The second case is where the target is outside the total length of the chain of joints. In this case the total distance value from the chain is compared against the direct distance calculated. If it is less then each joint of the chain is placed along a direct vector towards the target by the distance between each point.

The bulk of calculations in FABRIK can be dissected into three parts: The forward reaching phase, backwards reaching phase and the constraint calculation. Before starting the main phase two values need to be stated. The accuracy of the final solution - the minimum distance required from the final joint in the chain and the target before the solution ends – and the maximum amount of iterations before quitting. These two values are in place to prevent the solution from taking up too much system resources. Before any of these calculations however, several variables remain consistent throughout and are initialized for optimisation. Firstly, the amount of current tries must be set to zero, then the initial position of the root of the chain must be taken. An empty variable (tempos) for storing the position during calculations, another empty variable (lambda) for storing the difference between the stored joint distance and actual joint distance for calculations, and finally the initial distance from the end of the chain to the target must be set.

The forward reaching stage sets the position of the final joint in the chain to be equal to the target position. Then it iterates backwards down the chain towards the root, however the iteration ignores the final joint in the chain. We then set lambda to the stored distance between the current point and next one in the chain and then divide it by the actual current distance between points. Tempos is then filled with the output of an equation that determines the current point's new position. The previous position is also stored for use in constraint calculation exceptions. The new position is then run through the constraint calculation that returns either true or false. If true, the current position of the point becomes the new calculated position; if false, the current position remains.

The backwards reaching stage involves working from the root of the chain down towards the end effector; however it ignores the final point in the chain. One major difference in the backwards reaching stage (besides moving along the chain in the opposite direction) is that the point that is altered on each loop is the one next in the chain, calculated from the current point. Lambda is set to the stored distance between the current point and next one in the chain and divided by the actual distance between the current point and next one. Tempos is then filled with the output of an equation that determines the current point’s new position. The position of the next point is stored for constraint calculation exceptions. The constraints are then checked and the next point is assigned its new position based on the constraint check.

The form of constraints used with FABRIK is cone-based constraints, due to the fact that FABRIK is calculated based on position rather than rotation. This is done by projecting a cone from the point previous in the chain in a specified direction and calculating whether the next point will lie within the cone. If it does, the point passes the constraint test and is able to be assigned, if it fails however the point is linearly interpolated to the closest edge point of the cone compared to its out-of-bounds point.

3.2 Model Initialisation

The model itself must be structured in such a way that FABRIK and the Spline interpolator can integrate and manipulate it correctly. When the model itself is being created every joint in each limb chain must be in a parent-child pairing with joints further up in the chain, terminating at the root joint. When the model is being skinned, each joint much contain all of the weight for its surrounding vertices and all vertices further towards the next joint in the chain (until reaching that joint's immediate vertices). Each joint must also have its local rotation axes directed towards the previous joint, lest a secondary layer of rotations be placed on each joint to allow them to remain joined and internally consistent when FABRIK is utilised.

After the model has been imported, each joint must have the script 'IKRigJoint' attached to it, and each instance of IKRigJoint initialised for that joints position and rotation. Then constraint points must be specified and attached to each joint, and then the constraint position within IKRigJoint must be set to the specific joint's associated constraint. Elsewhere on the model (this project utilises the root parent of all the limbs' roots) the FABRIK scripts must be placed - one for each limb – and then each script populated with the joints in each limb chain from root to end effector.

On an upper level on the models hierarchy, preferably the topmost parent, there should be a container for all of the target points of each limb, for clarity as to where each limb is searching. Each of these targets should have the IKRigJoint script for interfacing with the FABRIK script as well as containing the related Hermite spline scripts, a controller and an interpolator.

Outside of the model there should exist a container that will hold the data for the splines so that the targets may follow them. This container should have child objects relating to each target that are linked to the Hermite controllers, and during runtime each child is populated with the points that will be linked together in a Hermite spline.

3.3 Hermite Spline

The functionality of the Hermite spline requires: assembling a chain of points, adjustment of the interpolation time, interpolation activation calls, and chain resetting.

3.4 Point Collection

The points in the world are collected by creating a grid starting at the root of the chain extending in the direction of the constraint point used in earlier FABRIK constraint calculations. The grid is created in a 'ladder' format with 'rung' being a unit vector of the distance from the root to the constraint. Each rung extends out left and right an equivalent amount of unit vectors as it is from the root. At the end of each unit vector a ray is fired directly down and any point that it collides with is appended to a list. The list is iterated through and the point found closest to the target is chosen as the most suitable point.

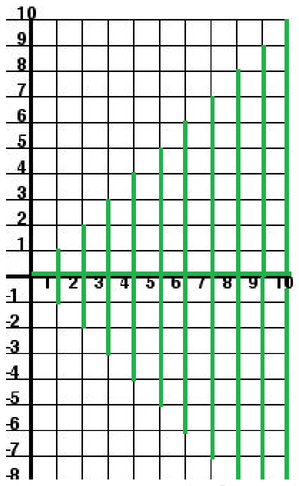

Figure 2 – Numerical increments of cone in point collection

Checking for points is a function call that is accessed whenever a new step is to be taken, and the most suitable point is returned.

4. Results

The results that were collected can be split into two categories: Numerical performance data, and visual output. All of the data was collected from the same University computer for consistency.

4.1 Numerical performance

Data was taken from the FABRIK implementation to determine performance of the method. The FABRIK solution had the target move in a consistent way for each taking of data to increase consistency of the conclusions:

1. The value for the maximum iterations was altered in multiples of 100, until the data structure could not hold the value.

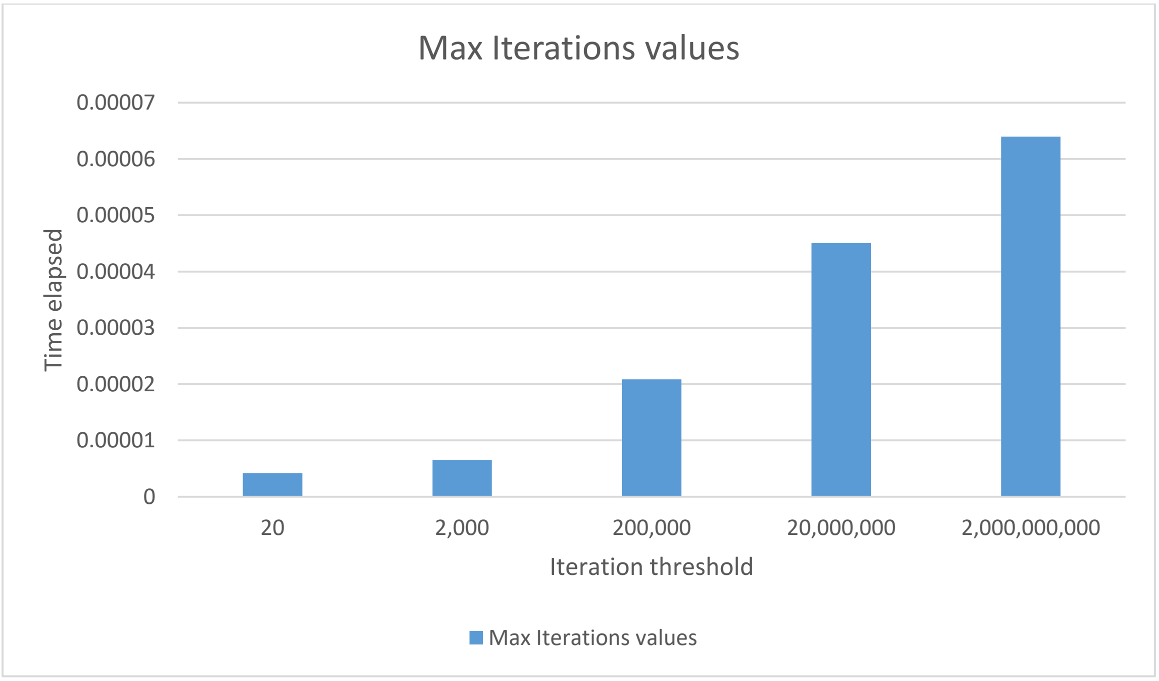

Figure 3 – Maximum iterations allowed against the time taken.

2. The value for the accuracy was altered in multiples of 100, until the data structure could not hold the value.

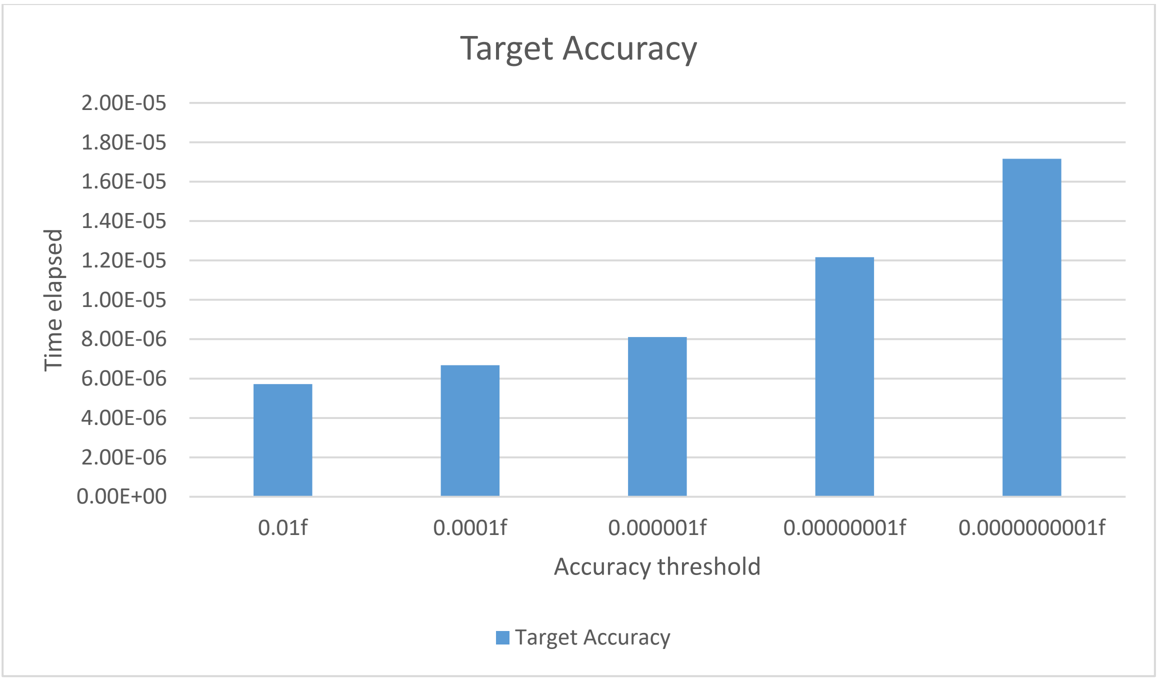

Figure 4 – Required accuracy threshold of solution against time taken.

3. Both of the values were altered in multiples of 100, until the data structures could not hold the value.

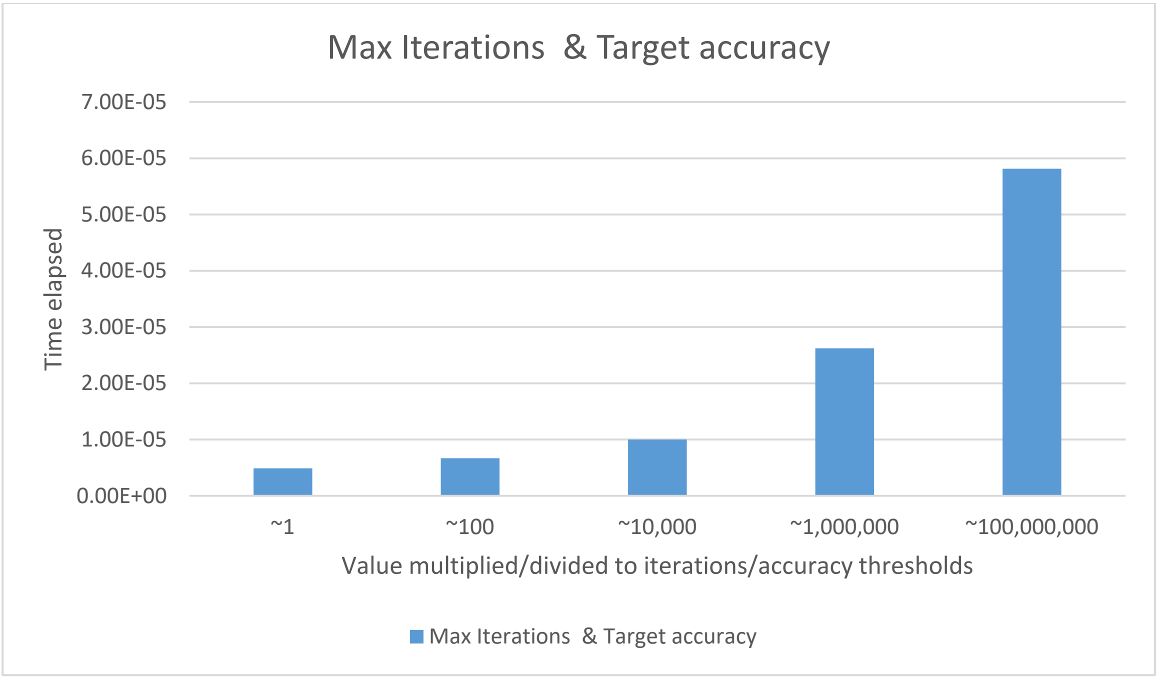

Figure 5 – Combined altered iterations and accuracy threshold against time taken.

Data was also taken that tested at what point the number of instances of each agent caused visible performance issues.

1. The Pre-animated agent -20

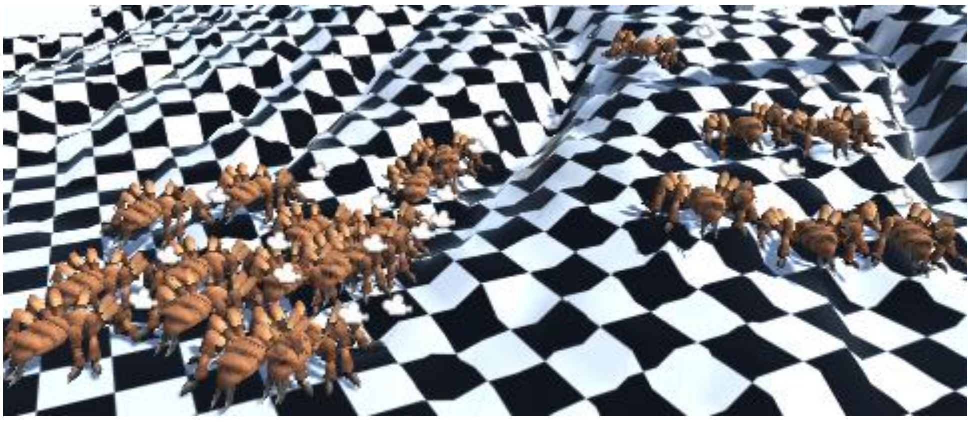

Figure 6 – 20 Pre-animated agents executing code without performance issues.

2. The Procedural agent following pre-defined movements -9

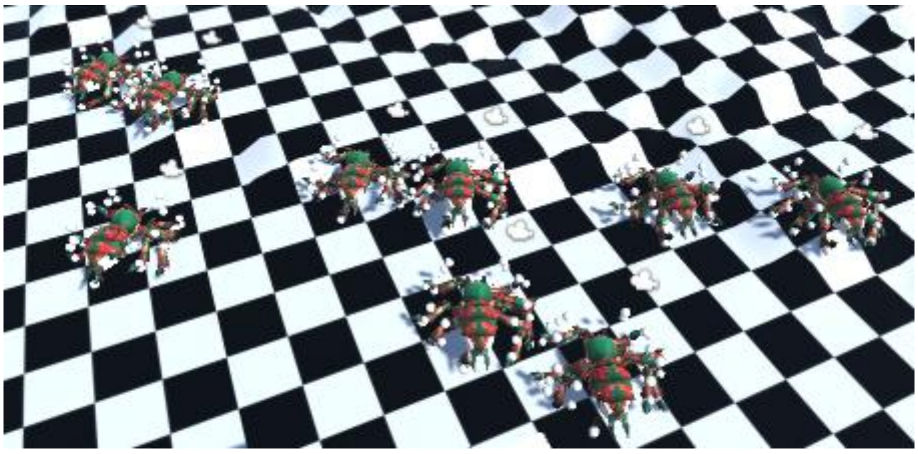

Figure 7 – 9 Procedural agents executing code without performance issues.

3. The Procedural agent with point collection functionality and subsequent Hermite creation - 4

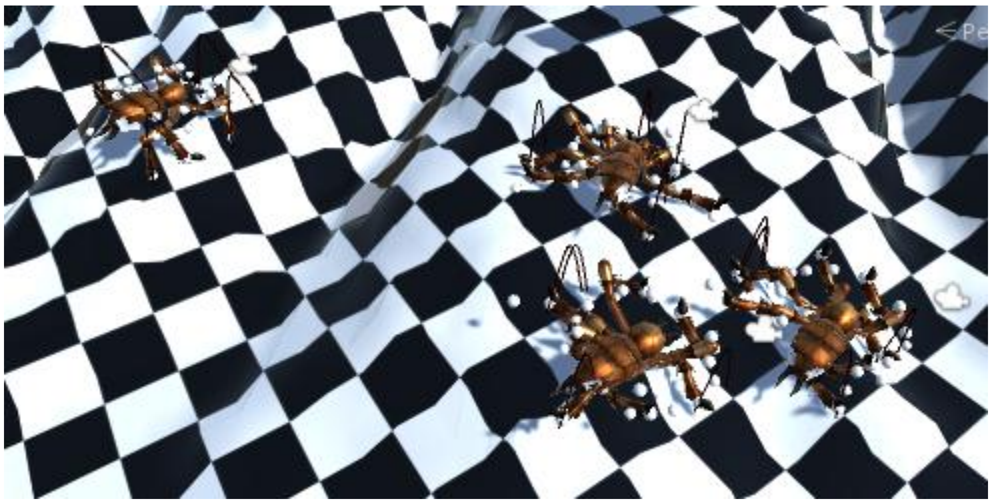

Figure 8 – 4 Procedural agents with point collection functionality and Hermite creation executing code without performance issues.

Data for the point collection in Hermite interpolation was also recorded to improve understanding of potential areas requiring optimisation.

Average time to run point collection	0.0008296967s
Combined average when computing 4 limbs	0.002632141s

4.2 Visual output

Each agent has its own performance on level surfaces, as detailed here:

1. The Pre-

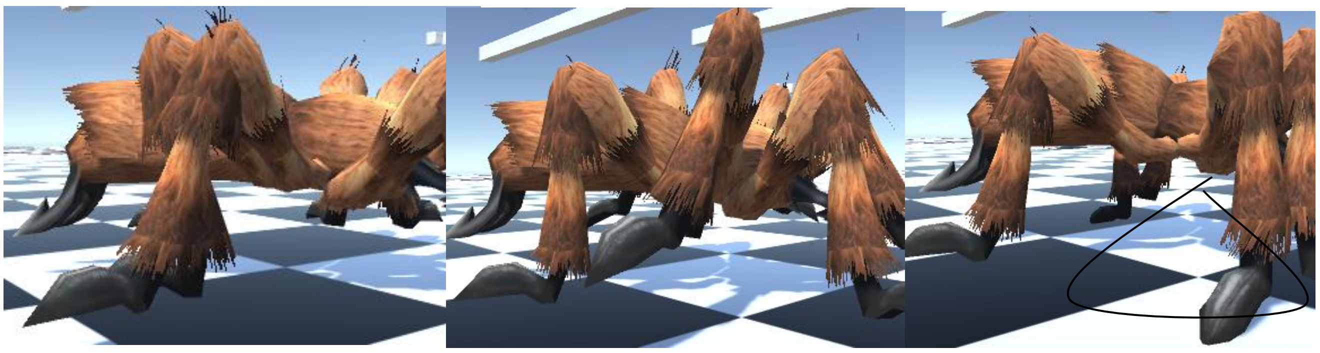
Figure 9 - Foot arc of Pre-animated agent.

animated agent

2. The Procedural agent following pre-defined movements

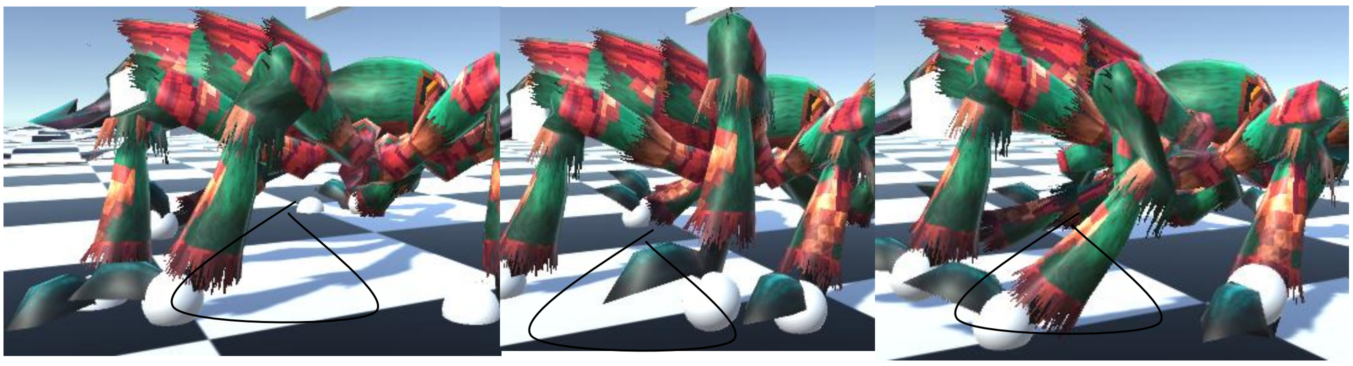
Figure 10 – Foot arc of procedural agent.

3. The Procedural agent with point collection functionality and subsequent Hermite creation

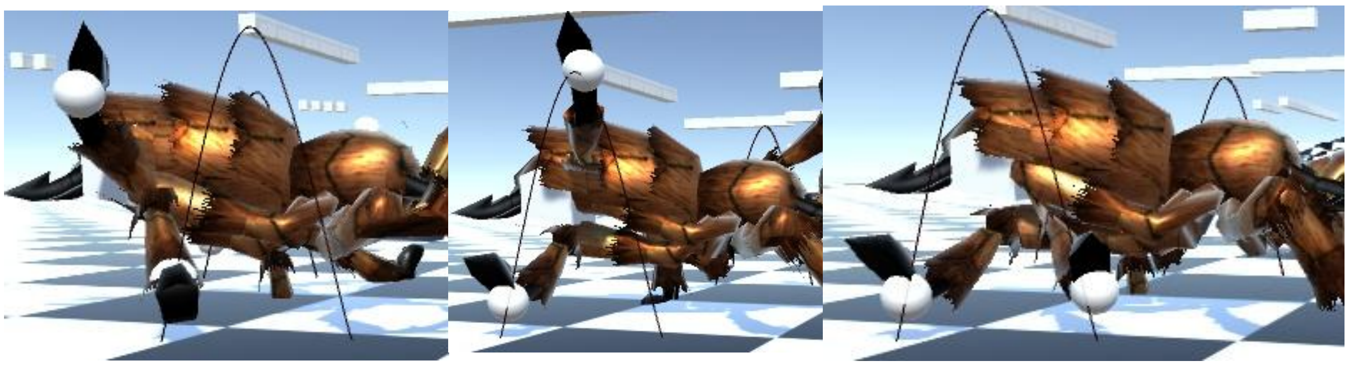

Figure 11 – Generated foot arc of Procedural agent with point collection functionality and Hermite creation.

Comparison performances can be found for each of the agents on varied terrain:

1. The Pre-animated agent

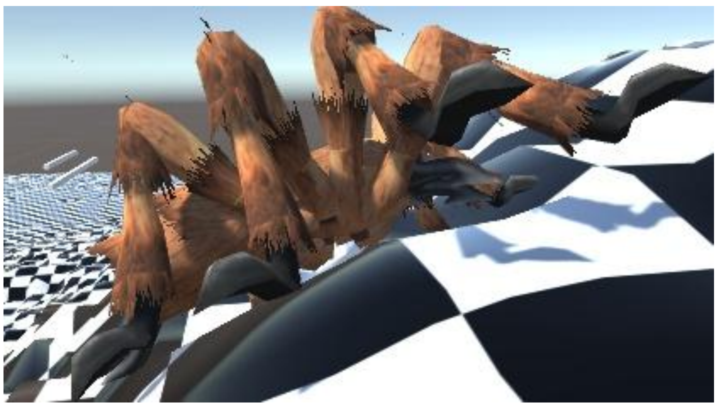

Figure 12 – Foot placement of Pre-animated agent on varied terrain.

2. The Procedural agent following pre-defined movements

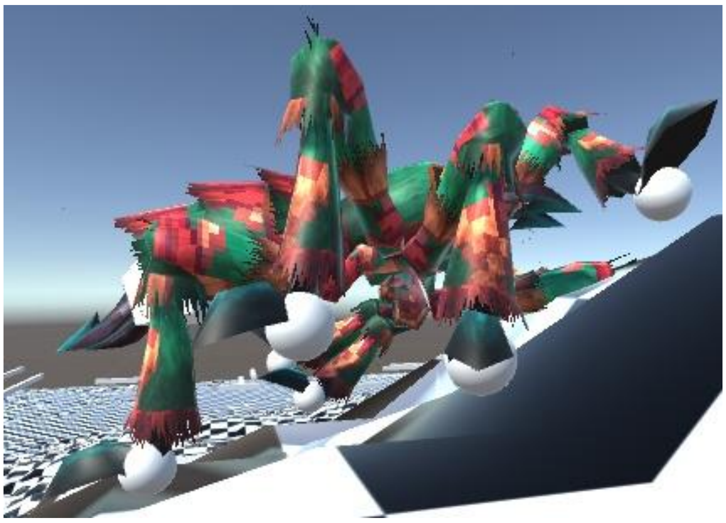

Figure 13 – Foot placement of Procedural agent on varied terrain.

3. The Procedural agent with point collection functionality and subsequent Hermite creation

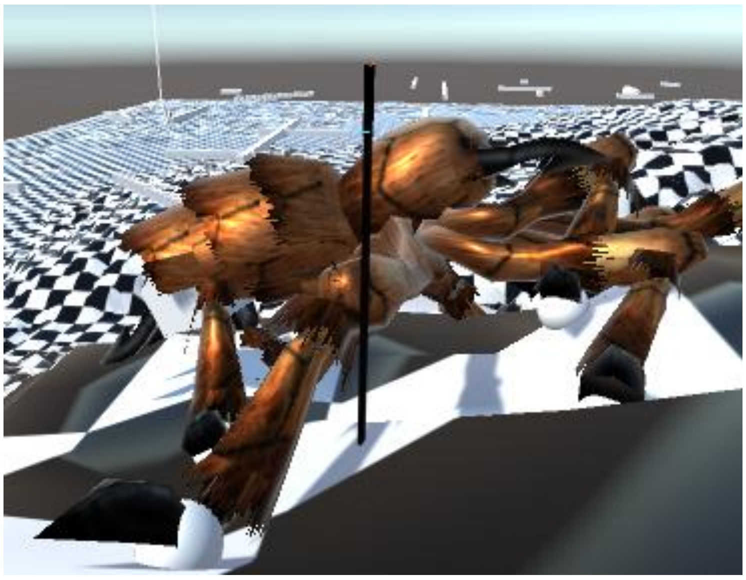

Figure 14 – Foot placement generated attached to varied terrain from Procedural agent with point collection functionality and Hermite creation.

5. Discussion

With the quantitative data, we can extrapolate that the FABRIK implementation maintains solid efficiency with negligible performance issues, as increasing the number of iterations is comparable to increasing the number of FABRIK instances. However, the more significant performance issue arises from increasing the maximum number of iterations over the accuracy threshold, suggesting that the accuracy threshold is rarely the terminating argument, most likely due to floating point imprecisions.

As anticipated, the Pre-animated agent affected the system the least, allowing for 20 instances before noticeable problems. The Procedural agent following pre-defined movements began causing performance issues at nine instances, suggesting the computational complexity in that agent is roughly twice as much as the pre-animated agent. The Procedural agent with point collection functionality and subsequent Hermite creation performed the worst in terms of system efficiency, with only four instances before instability. Suggesting further that the complexity was over double that of the Procedural animated agent.

One potential reason for increased latency involves the metrics acquired for the point collection, in that it takes roughly three milliseconds to assign points for four limbs (four specified as the agent can be set to walk with steps in two sets of four limbs) which becomes a cumulative problem with every agent added.

Concerning overall visual fidelity, the three different animation techniques can be displayed in descending order of fidelity. The handmade animation is the most visually appealing, due to each movement being completely fluid and minute alterations to any faults already fixed. Foot-placed procedural animation is next, as it gives visual cohesion with its surroundings however tweening into and out from each footstep lacks fluidity. The repeating procedural animation contains the downsides of both methods as the procedural movement looks stilted with the repeating nature exacerbated because of it.

6. Conclusion

Several methods are used within the industry to animate actors involving combinations of motion capture and procedural methods. However no such methods exist for actors where motion capture data does not exist, and all animation of their movement is hand-animated. In this dissertation an alternate method is presented where all of the movement in a multi-legged actor is procedural. The outcome of the project when compared to traditional animation ruled in the favour of traditional animation in general efficiency and visual fidelity, however the project contains areas that could be optimised and defective areas of procedural functionality that could be fixed.

Taking the length of the project and the level of experience of the project creator into consideration, unless any game or future project contains procedural animation as a major feature the investment into the technology is more suited to long-term projects or pure research.

6.1. Future work

The potential for future work lies within four areas. Firstly, improvements on the FABRIK system of constraints are required, or potentially complete recreation of the entire handling of that system. Secondly, a model created with the intent of including FABRIK and structured to prevent confusion and importing errors. Thirdly, a more dynamic method of foot path construction that the Hermite interpolation is required for the agent in order to allow for concurrent world dynamics to be able to affect movement. Finally, time must be given to allow development and implementation of an artificial intelligence system that allows damage to be taken to limbs and subsequent adaptations to movement because of that.

7. References

Reference list

Blow, J. 2004. Game development: Harder than you think. *Game Development: Harder than You Think.* 1(10): pp.28.

Rain World. 2017. [computer game].Sony PlayStation 4, Windows. Adult Swim Games

Overgrowth. 2013. [computer game].Windows, macOS, Linux. Wolfire Games

Cully, A. et al. 2015. Robots that can adapt like animals. *Nature.* 521(7553).

Damon, S. 2015. Capturing the last of us: Motion capture pipeline. In: *Gdc 2015,* 2nd-6th March 2015.

Dorigo, M., Birattari, M. and Stutzle, T., 2006. Ant colony optimization. *Ant Colony Optimization,* 1(4): pp. 28-39.

Quake 3 arena. 1999.[computer game] Windows.Activision Blizzard.

Kreyszig, E. 2006. Cubic hermite spline. In: Anon. *Advanced engineering mathematics.* 9th ed. Wiley. pp.816.

Lasenby, J. and Aristidou, A. 2011. **FABRIK: A fast, iterative solver for the inverse kinematics problem**. *FABRIK: A Fast, Iterative Solver for the Inverse Kinematics Problem.* 73(5): pp.243-260.

Liegeois, A. 1977. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man and Cybernetics.* 7(12): pp.868-871.

Menache, A. 2000. *Understanding motion capture for computer animation and video games.* Morgan kaufmann.

NaturalMotion2007. *Euphoria.* NaturalMotion.

Rosen, D. 2014. Animation bootcamp: An indie approach to procedural animation. In: *Gdc 2014*, 17th-21st March 2014.

Tinwell, A. 2014. *The uncanny valley in games and animation.* 1st ed. CRC Press.

Toy, M., Wichman, G. and Ken, A. 1980. *Rogue: Exploring the Dungeons of Doom.* Epyx.

Wolovich W. A. ,Elliot H. 1985. A computational technique for inverse kinematics. In: *23rd IEEE Conference on Decision and Control* , 1984 1985. IEEE Xplore, pp.1359-1364.

7.1. Appendix

Free Low Poly Fantasy Spider. 2008. [Model]. Made by Kalamona. Available from: http://www.turbosquid.com/3d-models/free-spider-animations-3d-model/398764.