Real Time Character Animation

Piran Tremethick
c1192971

Real Time Character Animation

Piran Tremethick

1 Abstract

Character animation is concerned with breathing life into a made up character, imparting a sense of meaning to their actions and providing a visible expression of their emotional state. Portraying what a character is thinking and feeling in a believable way encourages an emotional link with the character that can lead to a greater sense of involvement for the audience.

Computer games offer unique opportunities for character animation as the player is typically able to direct the characters, creating a constantly changing backdrop for the expression of individual personalities and traits.

2 Introduction

Character animation in computer games presents a number of unique problems, namely that someone else is going to be in control of the character. This makes it impossible to pre-determine where the character will be and how they will be expected to interact with their environment. Computer games often exhibit visual flaws in character animation, typical problems include feet sliding while walking and feet not placed correctly on slopes and especially stairs. This necessitates a more general approach in character animation systems, where animation is adaptable while retaining it's sense of character and emotion. This paper explores some ideas in how animation can be adapted to allow for a more consistent visual experience. These ideas will be based upon the overall goal of developing a system which allows a walk cycle to adapt to stairs.

3 Related Work

This section presents a brief overview of the techniques relevant to character animation in real time applications that were used as a basis for developing the ideas presented in this paper. It also includes a look at two games that represent the current state of real time animation systems in games, and provided inspiration for the research.

3.1 Skeletal Animation

Skeletal animation is a well established technique, used extensively for character animation in off-line animation packages e.g. Alias's Maya and Discreet's 3D Studio Max. Skeletal systems are now used widely in real time applications, allowing hand animated or motion captured sequences to be played back fluidly.

In a skeletal animation system, a series of connected joints form an underlying skeleton that can be used to deform the geometry of the model [fig 3.a]. Each joint in the hierarchy represents a

transformation from the parent joint, with a single 'root' joint at the start of the hierarchy (this is typically placed at the base of the spine in a human model). Animation can be represented as a series of transformations for each joint in the hierarchy. Each point of the geometry (the vertices in a polygonal model) is deformed by a number of bones – how much each bone effects that point depends on a weighting factor, usually based on the distance from the bone and modified by the artist/animator in the animation package.

Skeletal systems are useful because they allow animators to work with a simplified structure, rather than trying to manually alter the geometry itself. Much less data is required to store an animation, as animations are stored as a sequence of transformations for the joints, with the geometry only needing to be saved in an initial 'rest pose' (or 'bind' pose). This is a big bonus for games where storage space and memory is limited. Skeletal systems also allow several animations to be blended together easily by blending the individual transformations, allowing different animation cycles to be applied to the skeleton at the same time.

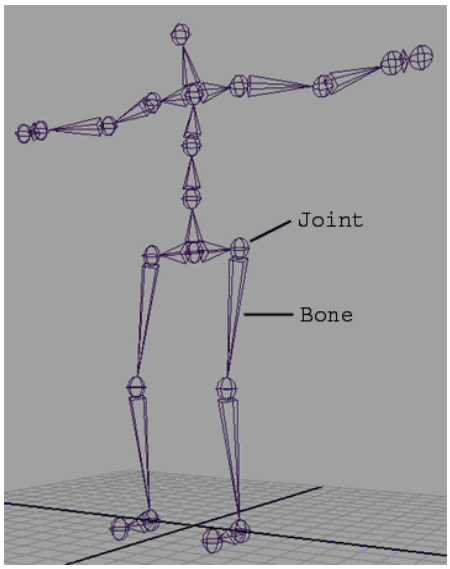

Fig 3.a - A basic skeleton in Alias's Maya 6.0. Each joint represents a transformation from it's parent joint, 'bones' are the connections between parent/child joints.

The skeleton is animated by rotating each bone into position and saving this position as a key frame. The key frames are the major poses that define an action which the animation system interpolates between during playback to produce smooth movement. In off-line animation systems the interpolation is usually controlled by a curve, allowing the animator to fine tune how quickly the system gets from one key frame to the next. This control over the interpolation is not present in real time applications as each animation package has it's own proprietary system, making it very difficult to replicate the curves. Evaluating curves is also computationally expensive, making them unsuitable for real time use. Instead, the animation is 'baked' on export from the animation package. The curves are sampled at regular intervals, and their values used as a key frame. This results in far more key frames, but allows for a much simpler interpolation method to be used in the real time application. The animation data can also be stripped down, with key frames being discarded if there is only a small change from the previous ones. Obviously care has to be taken in choosing a suitable threshold, an acceptable trade off has to be found between the resulting size of the animation data and the quality of the animation.

Key frames represent transformations, with rotations being the most useful although translation and scaling can also be supported. Translations and scales are stored as values along the x, y and z axes, while rotations may typically be stored in either Euler form (as angles around the x, y and z axes) or as quaternions. There is no consensus on whether an animation system for games should allow scaling and translation of joints as well as rotation. Microsoft's DirectX API provides generic support for rotation, scaling and translation data, while some game engines, such as Valve's Source engine, only allow joints to be rotated, with the exception of the root, which may be translated. Although rotations are the most useful, translation and scaling allow for some special effects.

When playing back an animated sequence, the system interpolates between the two key

frames whose times fall either side of the current time. Typically in real time systems a form of linear interpolation is used:

$$P = P_1 + t(P_2 - P_1)$$

where t is the blend factor. The value of t is how far between the two key frames the current time is, re-mapped to the range 0 to 1:

$$t = (c - k_1) / (k_2 - k_1)$$

where c is the current time through the animation and k_1 and k_2 are the times of the key frames between which the current time lies.

The transformation for each joint is represented as a single 4x4 matrix containing the result of all the transformations applied to that joint. The translation, scaling and rotation transformations are interpolated for a frame in the animation sequence and the resulting transformations concatenated to produce a single matrix. Combining several animations is straight forward with matrices, the resulting transformation of a bone is the sum of the transformations from each animation sequence (where the transformations represent offsets from the rest pose). A blend factor can be specified to 'weight' each animation:

$$M = M_1*B_1 + M_2*B_2 + \ldots + M_n*B_n$$

where M is the resulting matrix, M_1 to M_n are the individual animation transformations and B_1 to B_n are the associated weighting factors.

Rotations, however, are usually expressed using unit quaternions, which do not suffer from the problems associated with Euler and matrix representations. As well as the gimbal lock scenario, problems include the number of operations needed to construct a matrix from Euler angles and the fact that a 3x3 rotation matrix consists of nine values. Quaternions need only four values and interpolation never results in gimbal lock. The down side to quaternions is that while interpolation is very smooth, it is also quite costly, with specialised forms of linear interpolation needed (typically Spherical Linear intERPolation, or SLERP). It is also hard to visualise a rotation specified as a quaternion, given they exist in 4D space [Fig 3.b]. Animations can be blended by interpolating between the different quaternions, however, blending more than two animations can be complicated as SLERP only allows two to be blended at a time. The methods required to blend an arbitrary number of quaternions are an ongoing area of research [Johnson, 2003], and quaternion rotations are usually converted to matrices to be blended. Despite all the problems, the robustness and quality of interpolation and their small

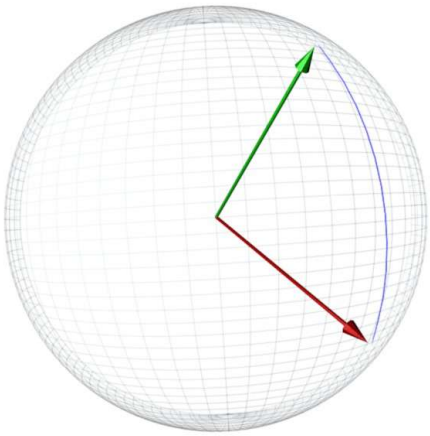

Fig 3.b - Quaternions are usually visualized as describing points on the unit 4D Hypersphere. The arrows represent two different quaternions, the curve follows the arc taken while interpolating between them.

memory requirements make quaternions very useful for animation systems.

For more information on skeletal animation systems see [Pitzel].

3.2 Inverse Kinematics

Inverse Kinematics (IK) is a technique adopted from the field of robotics. Given a jointed limb (such as a robotic arm), animating it so the last joint reaches a certain point in space would typically involve rotating each joint in turn, starting at the root, until the last joint was suitably close to the target. This is called Forward Kinematics (FK), and has a number of problems. A classic example is when a character walks – the foot carrying the weight should remain in a fixed place while the character moves forward. Animating using FK results in the foot 'sliding' about as the entire hierarchy is moving. Inverse Kinematics (IK) is a technique adopted from the field of robotics. Rather than animating the joints directly, the animator can position a 'target' and the IK system attempts to align the joints in the IK 'chain' so the last joint is close enough to it. IK has many advantages, but there are some significant problems that have meant, until fairly recently, IK has not been suitable for real time animation systems. Speed is always an issue for real time applications and IK is a computationally expensive technique. While in robotics or off-line animation systems the most important thing for an IK solver is finding a useful solution, in real time applications it needs to do so faster. Another problem is that there are a potentially several orientations for the joints that provide a solution. In robotics this might not matter as long as the target is reached, and in off-line animation packages additional constraints may be added to allow the animator to 'tweak' the motion. Real time systems need to be robust enough to provide a believable solution, but without the extra cost of evaluating constraints, or an animator to fine tune the results. However, despite the difficulties IK is very much suited to real time systems. The ability to specify a target position and have the IK solver automatically orientate the joints of a limb is very useful for games, where a character's position in the environment can not be pre-determined, and where a character may have to interact with objects in the environment.

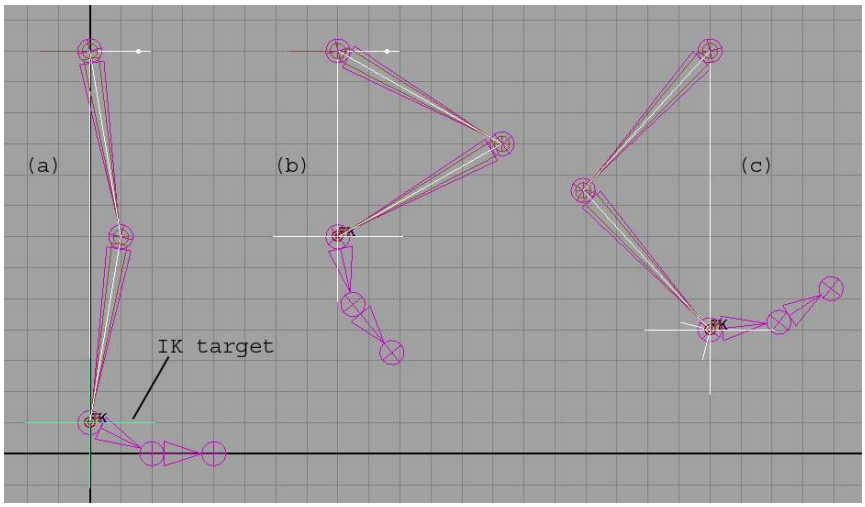

Fig 3.c - A simple IK leg setup in Alias's Maya 6.0. (a) the bones in their rest position, with the IK handle located at the ankle. (b) when the IK handle is moved, the bones re-align to keep the ankle joint as close to the IK handle as possible. (c) this is obviously incorrect for a human leg, but provides a valid IK solution. Note that Maya provides an IK solution with an additional constraint (the white arrow at the top of the leg in (a) and (b)), in order to achieve the position in (c) an alternative IK chain was created without this constraint. (c) is more analogous to a real time IK solver.

3.3 Animation as Visual Feedback

Generally in games, animations are a visual cue to the player as to what the character is doing. If a character moves, they may animate using a walk or run cycle. The actual animation, however, has nothing at all to do with the movement of the character. The motion of the character is not being defined by how their feet contact the ground or the force imparted through their legs, the animation is simply there to make the player *believe* the character is moving realistically. If the movement of the character contradicts what the player knows the character *should* be doing, then the illusion of a realistic game world is broken. Note that the term 'realistic' is used broadly. Animation is often exaggerated for effect (indeed, exaggerated animation is often perceived as *more* believable), but even cartoon style animation follows basic principals of weighting and balance that maintain a sense of *believability*, even if *reality* has long gone out the window. This approach to character animation is limiting in terms of the interaction possible between the character and the environment. It also limits the role of the animator in determining how a character moves. Given the importance of animation in bringing a character to life, it is a shame that animators can not have more involvement once a character is actually in-game.

3.4 Current Examples – *Splinter Cell: Chaos Theory* and *Shadow of the Colossus*

Splinter Cell: Chaos Theory (SCCT) from Ubisoft has a more advanced animation system than most games and is one of the few to use real time IK to procedurally modify a characters motion. This is accomplished using HumanIK, a 3rd party animation system from Alias aimed at game developers. In SCCT, a ray is cast from the characters feet and tested for intersections with the ground. The resulting position is used by the IK system to re-orient the legs, allowing the feet to rest correctly on uneven ground. This positions the feet reasonably well, unfortunately it also results in odd poses – with enemy characters appearing to 'float' and never straighten their legs. How much of this is the result of the animation system, and how much is simply due to the animation cycles themselves, is unclear. At the very least it demonstrates the importance of having good animation before trying to modify it, and the difficulty in integrating animations and a procedural solution. SCCT creates the illusion of a highly interactive environment by allowing the player to perform a reasonably large number of specific actions at specific places within the level. While this mostly works, it does not provide a general solution, with the environment being tailored to fit the abilities of the character. It does not allow the player to explore alternative or original solutions to the game. SCCT is a perfectly enjoyable game, but it often feels constrained.

Shadow of the Colossus (SOTC) from Sony presents an innovative use of IK. Throughout the game the player must control the lead character as he climbs up the various colossi looking for a way to defeat them. All the while the colossi attempt to shake the player loose, resulting in some stunning sequences as the players character clings to a constantly moving surface. The animations are fluid and characterful, with procedural motion mixing seamlessly with pre-animated cycles. As the game was released towards the end of this project, little information is available regarding the techniques used [SOTC, 2005]. However, an IK system is used to position the hands and feet of all the characters (including the players horse). The character can cling to surfaces and ledges at quite extreme angles to the direction he is facing (usually while jumping), with the hands reaching a

realistic position to grab hold. This combines with a 'pendulum' physics system to produce the fluid swinging movements as the character is flung about. The most impressive thing about SOTC is the fluidity of the animation system, motions link together without any indication as to what is pre-animated and what is procedural. In fact, the procedural elements only appear procedural simply because of their variety and adaptability to a constantly changing environment. Whether hanging onto a ledge with feet firmly planted, or swinging from the arm of a colossus, one arm and feet dangling free, the movements are fluid and believable. This is achieved by the innovative way animation and procedural elements are layered. The movements begin with a hand animated element which is then modified by the IK and physics systems to produce the final pose. This allows the motion to adapt to the environment, but the hand animated base retains the animators work in expressing the character. The result is stunning, both visually and in terms of emotional depth, and represents a step forward in character animation in games.

4 Approach and Implementation

Re-cap: The focus of this paper is on developing ideas about how real time animation systems can adapt to the requirements of the environment, by combining IK and other procedural elements with existing pre-animated cycles. Due to the size of the project it was decided to focus on the specific case of adapting a walk cycle to cope with stairs, a situation common in games where the animation visibly breaks down.

4.1 Combining Animation with a Procedural Layer

The dynamic way in which a character can move in a real time environment suggests a procedural 'layer' is needed to enhance animation. However, it is the pre-animated sequences that provide the sense of character so any procedural system needs to complement the existing animations as opposed to simply over-riding or replacing them.

The system presented in this paper uses a layered approach. The basic animation system produces a pose from the pre-animated cycle. This basis is then corrected by the IK solver to allow for changes in the environment, placing the feet in more believable positions. This is blended with the initial animation to retain the sense of character. Finally, this pose is then used by the foot locking system to drive the motion of the character through the environment, with offsets to the feet being transferred back to the root. This blending of multiple elements, starting with actual pre-animated data, allows the animators work in creating character to show through. The procedural elements compliment the animators work without over powering it.

One of the key ideas to arise from this research is that a possible approach is to actually drive the characters motion using the animations. Rather than animation cycles simply being 'played back' to provide visual feedback, they are responsible for the motion of the character. This basis will be expanded upon later, with a discussion of a technique for generating an offset for the root.

While there are several papers detailing the usefulness of IK systems and their benefits to real time character animation, few provide any details on implementation and there seems to be little in the way of examples. Actually incorporating an IK solver into a real time animation system seems to be an undocumented area. One of the best descriptions of IK and implementing it in a real

time application is from Game Developer magazine [Lander 1998] and [Elias]. These are the methods used in the animation system presented in this paper.

4.1 Skeletal animation with Quaternions

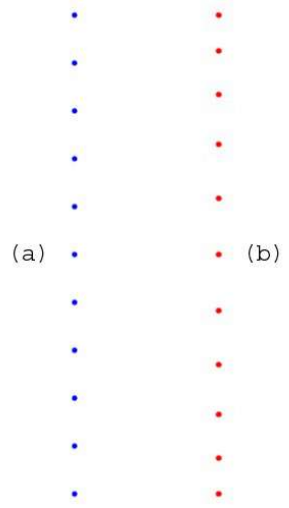

Fig 4.a – (a) the spacing of interpolated points when using SLERP (b) the spacing of points when using NLERP. There is a slight 'bunching' of points at either end of the interpolation with NLERP, but both follow the same arc.

Quaternions are now the standard representation used for rotations in an animation system. Interpolation between rotations is often achieved using Spherical Linear intERPolation, which provides an even spacing between interpolated points as well as a constant speed of interpolation. The SLERP algorithm is quite slow however, which is a major disadvantage to a real time system. The system described here uses Normalized Linear intERPolation (NLERP), a faster although slightly less accurate method of interpolation [Blow, 2004]. The arc followed by the interpolated rotations is the same as with SLERP, however the speed of interpolation is not constant, exhibiting a slight 'ease in, ease out' effect [Fig 4.a]. This is not noticeable, and is an acceptable trade off for the performance gain. The algorithm for interpolating two quaternions simply becomes a standard linear interpolation between them, with a normalization of the result:

$$Q' = \frac{Q_1 + t\,(Q_2 - Q_1)}{|Q_1 + t\,(Q_2 - Q_1)|}$$

4.2 Exporting the Animation Data

The sample application presented here uses data exported from the open source 3D animation package Blender [Blender 2.41] using a custom exporter and XML file format. Blender was chosen because of it's flexible and simple scripting API (which uses the Python programming language [Python]). This allowed a focus on what data was exported rather than how, as accessing most aspects of Blender's internal systems is straight forward. A skeleton can be modelled and animated using the existing tool set, with the exporter extracting the relevant data as needed. This approach is an important one, as the animator should be free to work with the tools without additional restrictions. Unfortunately, due to the time-scale of this project, it was not possible to 'polish' the exporter, resulting in some rather crude hacks – however the unobtrusive nature of the exporter is certainly something to build upon.

A custom XML (eXtensible Mark-up Language) file format was chosen as it allows the resulting files to be easily read and edited by the user. XML is an extension to HTML (Hyper Text Mark-up Language) used for making web pages, which means that the exported XML file can be viewed in any modern web browser. This is very useful when it comes to de-bugging errors in the

exporter as most web browsers format the XML file automatically, making it neat and easy to read. In order to import the XML file, the system uses a third party XML parser, TinyXML [TinyXML]. This small C++ library handles the loading of the file and presents a convenient structure for navigating the document.

The exported file is broken down into two main sections, each of which consists of a number of sub-sections. The first section contains all the data relating to the skeleton (or 'armature' in Blender) - the matrices of the bones in their rest pose, the connectivity of the hierarchy and also the IK constraints. The second section contains a number of animation sequences (or 'actions' in Blender), each containing a list of key frames for each bone in the skeleton. Actions and key frames also have information on timing, specified in milliseconds.

The overall structure of the file is flexible, allowing bones, constraints and animations to be added as needed. This makes it straight forward to accommodate unusual characters.

4.3 Inverse Kinematics

There are many different methods for finding an IK solution, but all of them suffer the same problems when applied to a real time system. Demonstrations of real time IK techniques usually involve simple 'chains' of joints, where there are less constraints on the final orientations of the joints. For a character, it is important that the solution is believable, simply finding a solution is not enough, and this often requires additional tools to achieve in an animation package.

[Lander, 1998] describes a technique known as Cyclic Coordinate Descent. This is the best documented technique, but no actual 3D implementation could be found. This produced a solution, but exhibited a very distracting jitter. The solution was mostly unusable due to the lack of constraints, it is necessary to constrain the knee to point forwards but this is not straight forward to implement.

The technique described in [Elias] does not produce an immediate solution, instead the effector gradually approaches the target over several frames. This is not necessarily a problem but still produced the same problems as with the CCD method – mostly unusable results due to the lack of a knee constraint and the difficulty in incorporating such a constraint into the system.

The result was an alternative IK solver, based on a stiff spring system, that provided quite interesting results. Spring systems are a well established method for physical simulations, especially in real time applications[]. By modelling a spring for each bone, the IK solver ran through a number of iterations, gradually converging on a solution [Fig 4.b]. The end of the first spring was fixed at the root of the IK chain and the end of the last spring was fixed at the target position. This left the ends of the springs shared at the knee free to move about. Each iteration the springs would push or pull the knee into a position that maintained their length. While far from accurate, the gradual convergence meant the results where quite acceptable for real time use. The constant iteration also meant that any solution was gradually improved each time the IK solver ran, effectively reducing the error over time until it was unnoticeable.

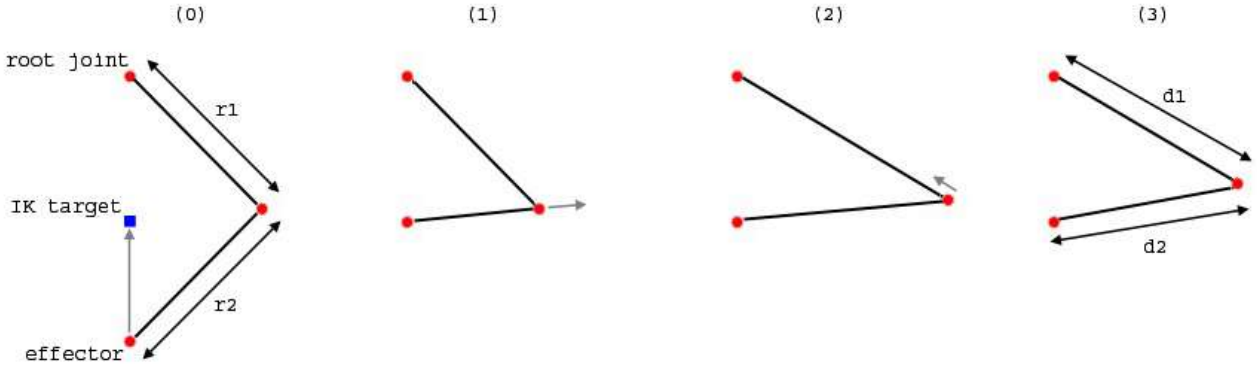

Fig 4.b – Spring based IK. Springs are modelled between the joints of the leg. The end of the spring at the root is fixed, the end of the spring at the effector is fixed to the IK target, while the shared ends at the knee joint are allowed to move freely. The springs attempt to maintain their length by moving the knee joint, over a number of iterations this converges on a solution that satisfies both springs simultaneously. The solver stops when d1 and d2 are suitably close to r1 and r2 respectively (where r1 and r2 are the original lengths of the springs), or when a maximum number of iterations has been reached.

The main advantage of this system was that an additional 'aim' constraint was very easily added to ensure the knee always bent the correct way. This constraint was simply another spring, with one end at the knee and the other at a point somewhere in front of the character. If the IK solver was pushing the knee into an unnatural position, this extra spring was evaluated to pull the knee forwards into a more believable pose. This was achieved by evaluating the dot product of two vectors formed from the second spring in the IK chain and the spring forming the knee aim constraint. Once normalized, the dot product between these two was used to determine if they faced in the same general direction, i.e. the knee was pointing at the aim constraint. If it was, then the constraint needn't be evaluated. It was necessary to disable this constraint when not needed as it significantly distorted the spring system. Typically however, it would only need to be active during the first few iterations, after that the system would converge on an acceptable solution without further modification.

4.4 Generating Offsets to the Root

A key idea explored throughout this project is that the animation sequences should drive the motion of the character. In order to achieve this a procedural technique was developed to allow the forward motion of the character to be determined from the movement of the legs in a walk cycle. The foot carrying the weight is the one imparting a force to the body to drive it forward, so initially a system involving foot contact with the ground was explored.

When a foot collided with the ground plane it was considered locked and currently carrying the characters weight. The position of the collision was stored. For subsequent frames, the new position for the foot was generated from the animation cycle, but before being applied to the skeleton a vector between the foot's new position and the contact position was created. This determined how far the foot would have travelled since the last frame, by offsetting the root bone by the inverse of this vector the foot would remain in the same position and the character move forward. Once the other foot came into contact with the ground, it would be considered to be carrying the weight and would be locked, while the original leg would be unlocked.

The problem with this approach is that it places certain restrictions on the animator. The feet

have to collide with the ground very precisely. If one collides later than the other then the locked foot may not be unlocked when it should, instead generating a negative offset. This has the effect of pushing the character backwards momentarily until the other foot collides. This exhibited a very noticeable jarring effect when tested due to a slight variation between the feet in the walk cycle. The restrictions placed on the animator by this method are too tight. Additionally, there is no guarantee that the feet will collide with the ground when they should. If a foot collided with, for example, a raised bump, then it would become locked and start offsetting the character, regardless of whether it actually should be generating the force or not. An alternative approach was needed that did not rely on collisions.

The technique used in this system determines the offset to the root depending on the direction of motion of the feet. The idea stemmed from the basic principal that in order to impart any motion an opposite force was required. This is a fundamental principal in animation as well, with actions preceded by a build up of energy and force. The leg moving in the opposite direction to the direction the character should move in is the one providing the force for the movement. This is true regardless of whether the feet are in collision with the ground or not, as far as the animation cycle is concerned. The animator can specify a direction vector that describes the general direction the animation cycle should move the character in. This is relative to the direction the character is facing. Each frame a vector is formed from the new position of the foot to it's position in the last frame. If the dot product of this vector and the specified direction vector for the animation is positive (when both have been normalized), then the vectors are pointing in the same general direction [Fig 4.c]. This suggests the foot has moved in the opposite direction to the direction of movement of the character, therefore the character should be offset by the vector formed between the foot and it's previous position. This works very well, without the need to keep track of which foot is locked and without restricting the animator.

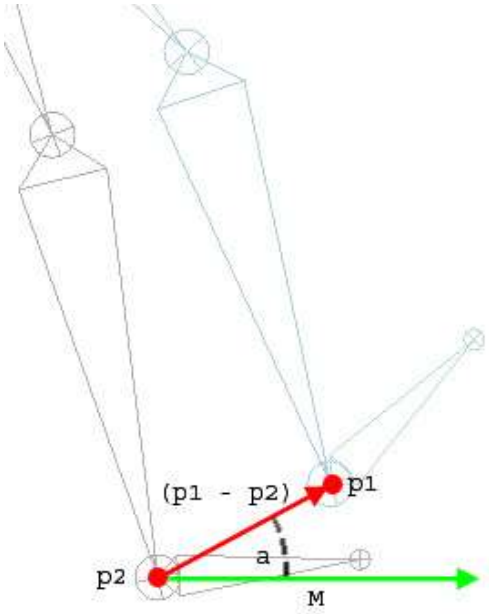

Fig 4.c – 'p1' is the joints position in the previous frame, 'p2' is the position this frame and '(p1 – p2)' the vector between them. 'M' is a direction vector specified by the animator. The dot product of 'M' and '(p1-p2)' is the cosine of 'a' (when both vectors are normalized). If this cosine is > 0 then this joint is powering the motion, and '(p1-p2)' is applied to the root, effectively keeping this joint in the same location..

4.5 Walking Up Stairs

For believable results it is necessary for the character to appear to react to the stairs before reaching them. Typically in the animation package the animator will specify a number of key frames that define the major poses of the action. These may be several frames apart depending on the speed of the action. However, in a real time system the same animation may have key frames that were sampled at every frame, due to the less sophisticated interpolation methods used. The system presented here keeps track of which key frames correspond to *actual* key frames in the animation package, as opposed to the key frames generated when exporting the sequence. To avoid

confusion, the key frames that were set by the animator will be referred to as *key poses*.

Every time a key pose is reached, the next key pose should be evaluated. If the foot positions in this pose are invalid (they intersect the stairs, represented by simple bounding boxes), then they can be corrected by offsetting the foot in the global Y axis until it rests outside of the stairs bounding volume. This position can then be used as a target for the IK solver, which re-orientates the leg bones to align with the target. This then provides a new key pose. The animation system can continue to interpolate the animation as normal, however, this will then be gradually blended with the modified key pose. When the animation reaches the key frame used for the corrected pose, the corrected pose will have been blended in fully so the affected bones will be using the corrected pose rather than the pose generated from the animation. However, because this corrected pose is based on the animation, it should retain the characteristics of that animation and provide a seamless transition between animated and procedural elements. At this point the next key pose is analysed and the process starts over. Additionally, the offset to the foot in the Y axis from the *previous* key pose can gradually be applied to the root of the skeleton, while interpolating into the next. This provides the offset required to raise the character up the stairs. As the root will be constantly moving forward, it may also be necessary to check the interpolated pose against the stairs each frame to allow the IK solver to provide minor adjustments to the legs.

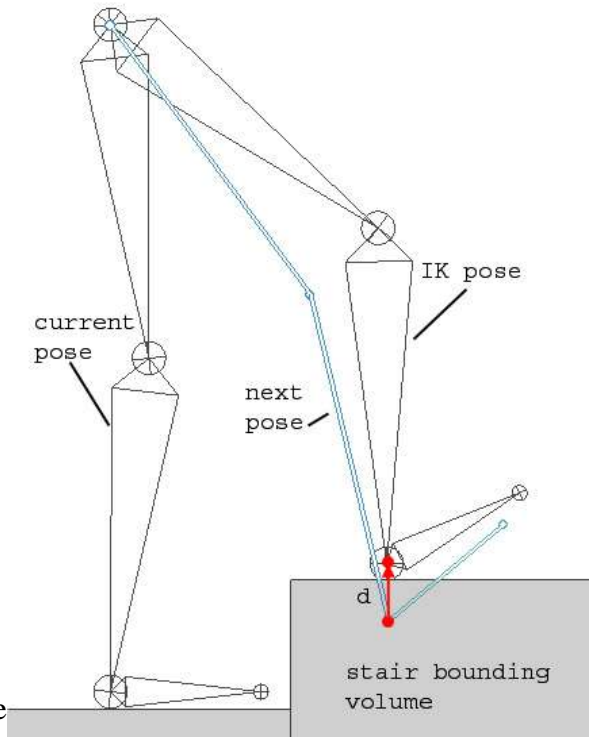

Fig 4.d – If the next key pose causes the foot to intersect the stairs, the foot is offset in the global Y axis. This position is used by the IK solver to evaluate a new pose for the leg. The animation system continues to interpolate the walk cycle as before, except now the final pose is blended into the one generated by the IK system. Correcting the key poses (as opposed to the exported key frames) should make the character appear to 'know' where to place their feet, rather than simply correcting a pose once it has been reached.

Unfortunately it has not been possible at this time to implement this system, and there will most likely need to be many optimizations and 'hacks' needed to get acceptable real time performance. Additionally, travelling down stairs will most likely require extra work as the leg that needs to be modified will be the one carrying the weight. However, it should not be too much of a problem to expand these ideas to allow walking up and down stairs smoothly and with correctly placed feet. Notice that the feet are still allowed to intersect the stairs while moving. Until implemented it is unclear how much intersection there will be and how acceptable this inaccuracy may be. It is worth noting that in animation it is the key poses that are important, so as long as these are believable any 'roughness' in the bits in between are often overlooked.

5 Conclusion

Generating the motion from the animation cycle works very well, the resulting movement is smooth and free from 'sliding'. Choosing to base the system on the animation rather than collisions between the feet and ground makes it much more flexible. This approach effectively puts the control over the character's motion in the hands of the animator, allowing the animator to express character through movement without being constrained by the requirements of the animation system. The only additional requirement is that the animator specifies a vector for each animation cycle to describe the general direction the character should move in. Although untested, it should be possible to analyse the first few key frames of an animation cycle and determine the direction vector automatically. This could possibly be done by comparing the height of the feet in the first key frame, with the lowest foot considered to be carrying the weight. By finding the difference between the positions for that foot in the first and second frames a vector can be formed describing a probable direction of motion. This could at least provide an initial 'guess' that the animator may adjust if needed.

Although it seems a relatively small part of an animation system, developing ideas about how the animation should drive the character became one of the largest areas of investigation on this project. Once the character is moving according to how they are animated, many other aspects of the system can build upon this. The IK system in particular can cooperate with such a system, allowing the foot positions to be dynamically altered by the IK solver and then using this new animation data to drive the character's motion. This was to be the approach taken in the implementation for this paper, unfortunately it became clear that more extensive research into IK solvers was required.

The initial IK solver implementation was based on [Lander, 1998] and proved to be very unstable. While not expecting a 'plug and play' solution, the instability and general unsuitability of the results was very discouraging. It became obvious at this point that the ideas developed on modifying a walk cycle to cater for stairs, would have to stay as just theory within the time scale of the project. After implementing a second IK solver based on [Elias], with similarly disappointing results, it was decided to focus the rest of the project time on the theory and developing ideas about IK. The resulting spring based IK solver seems to be worth investigating further. Whether this presents a genuinely usable solution will require further work and optimization, however the ease with which additional constraints were added was encouraging.

The principal ideas to come out of this project are that animation systems in real time applications need to become more than simple 'playback' systems, and that any procedural elements added to a system should use animated cycles as a basis.

In order for a character to interact in a more believable way with their environment, the animation system needs to take a greater role in determining how the character moves. This will allow obvious errors, such as feet sliding, to be minimised, and provide animators with much more control over how their animations are used in a real time environment. This animator control and involvement is very important – it is the animator, after all, that is creating the illusion of life within the character. That essence of life is such an important aspect of an animation. It is something that cannot be captured using purely procedural methods, hence the need to use an animation as a basis

for a procedural system. The emphasis should be on modification of existing animation, rather than the generation of motion from scratch. Although the scope of this project has allowed only a small area of this expansive topic to be investigated, it is obvious that current techniques are becoming rapidly unacceptable in terms of believability and emotional range.

This project has generated many ideas that it would be interesting to implement. The system presented with this paper is the earliest foundation, upon which future work will definitely be built. It is a shame that Sony's Shadow of the Colossus was released towards the end of the project, as it provided much inspiration. It demonstrated beautifully just how important animation is in generating a greater emotional depth, integrated as part of an artistic whole, and also served as a pointer as to what can be done. Until now, more expansive animation systems have been computationally too expensive – that is changing and hopefully this is an area that far more research will be conducted in.

The sample program "AnimDemo" demonstrates the basic animation system with procedural root offsets. The controls are as follows:

Mouse movement + left click – rotate the camera
W – walk forwards
A – turn left
D – turn right
T – change bone display type
C – change the walk cycle used

Note that SDL (Simple Direct Media layer) must be installed to run the sample program.

6 References

[Blender 2.41] http://www.blender.org/cms/Home.2.0.html

[Blow, 2004] Jonathon Blow, "Understanding Slerp, Then Not Using It" - http://number-none.com/product/Understanding%20Slerp,%20Then%20Not%20Using%20It/index.html

[Elias] Hugo Elias, "Inverse Kinematics", http://freespace.virgin.net/hugo.elias/models/m_ik.htm

[Johnson, 2003] Michael Johnson, "Exploiting Quaternions to Support Expressive Interactive Character Motion" - MIT, 2003.

[Lander, 1998] Jeff Lander, "Making Kine More Flexible", Game Developer Magazine November issue, 1998.

[Pitzel] Steve Pitzel, "Character Animation: Skeletons and Inverse Kinematics", Intel.

[Python] http://www.python.org/

[SCCT, 2005] http://www.splintercell.com/uk/

[SOTC, 2006] http://www.dyingduck.com/sotc/making_of_sotc.html.