Guff's Character Animation System for 3D Games and Applications

Márcio S. Camilo, Aura Conci

Universidade Federal Fluminense, Instituto de Computação, Brasil

Abstract

3D Games frameworks usually have some type of Character Animation System based on a low level layer for animations and a high level layer related with the behavior and intelligence of characters. The low level is responsible for providing a way of obtaining animations and to play them correctly and suitable according to the needs of the high level. This paper presents the implementation of the low level layer of the Guff Framework. This system provides functionalities for obtaining animations based on the Doom3's MD5 models and the possibility of a fine tune post-design configuration stage, which allows game designers to set groups of animations for supplying the high level animation system, animation-to-animation transitions and animations properties such as uniformity, evolution, and timing. Finally, a first approach to a high level animation system, based in a very simple finite state machine, is developed to show how the low level animation system can support the high level animation system.

KeyWords*:* - Computer Graphics, Computer Animation, Character Animation, Games Animation System, Keyframing

Authors' contact:

```
{mcamilo, aconci}@.ic.uff.br
www.ic.uff.br/~{mcamilo, aconci}
```

1. Introduction

Frameworks for 3D Games generally have Character Animation System modules that deal with the modeling, representation and behavior of the characters. Character Animation Systems can be seen in a two layers view. At the high level animation system is where the behavior and intelligence of a character takes place. Meanwhile, the low level animation system treats to obtain and show the animation sequences.

High Level animation system can use an AI based technology to accomplish the intent of giving a "soul" to a game's character. Usually it answers to events happening in the environment that can affect the character [Watt and Policarpo 2001a; Dybsand 2003]. Moreover, it can give a "personality" to the character expanding its "feeling" characteristics in one or more dimensions [Watt and Policarpo 2001a], like good or bad, honest or dishonest and so on. Most of the time the "personality" of a character is described with some type of script [Watt and Policarpo 2001a; Dybsand 2003].

The low level animation system responds to the high level by obtaining the animation sequences and playing them in the correct way[Watt and Policarpo 2001b; Azevedo and Conci 2003; Thalmann and Thalmann 1998]. Low level can be based on three main types of animations: procedural methods, motion capture and keyframing [Thalmann and Thalmann 1998; Hodgins at al. 1999].

Generally, the character animations are accomplished by motion capture and keyframing while procedural methods are often used for objects in particle systems or bands of little animals like flocking birds and swimming fishes [Watt and Policarpo 2001a; Watt and Policarpo 2001b].

Keyframing is a technique usually performed in two stages: *design* and processing. At the design stage models and animation sequences are created. The animation sequences can be or cannot be strictly based on the way the character behaves during the game. Animation sequences are sets of keyframes that shows how the model performs a movement time to time having the initial frame (called baseframe [Making Doom3 Mods 2004]) as a reference.

During the game run-time the processing stage takes place by answering the needs of the high level animation system choosing and playing the animation sequences in a coherent and suitable way. Generally, this stage is also responsible for generating in-between interpolated frames from pairs of keyframes. Figure 1 shows the two stages keyframing approach.

Because of the few complexity and inexpensive costs demanded of the design and processing stage, keyframing is still the most popular way of providing the low level of animations for characters in 3D Games.

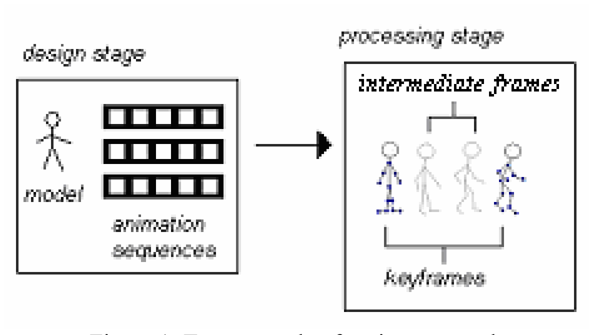

Figure 1: Two stages keyframing approach

1.1 Guff Frameworks System

Guff Framework is a framework for games creation developed in the Universidade Federal Fluminense. Guff Framework is formed by an application layer and a toolkit[Valente 2005].

The application layer determines the architecture of the applications based on the framework. The toolkit gives a set of facilities which make possible games creation. The character animation system is implemented like a character's set of classes, taking part of the Guff Framework's toolkit, hanging low level animation system and high level animation system of the character animation as two layers.

Guff Game Framework's low level animation system uses the keyframing approach to deal with models and animations. The models and animations are based on Doom3's MD5 formats. Besides the two stages keyframing usually contains, Guff Framework uses a third stage called intermediate configuration stage that is responsible for allowing a game designer to do settings that will help to provide the processing stage with the necessary information to do a better work playing the animation sequences of the character.

The high level of the Guff Framework's character animation system was implemented as a simple finite state machine which, in addition, is used to test the low level animation system functionalities.

The remaining of this paper is organized as follows: section 2 presents a background on the keyframing subject. Section 3 explains how a game designer can control the animation sequence based on the number of intermediate frames. Section 4 shows how blending transitions between different animation sequences are accomplished. Section 5 shows the Guff´s high level animation system. Section 6 shows some results accomplished. Section 7 deals with the conclusions and final considerations.

2. Related Work

Early 3D games keyframing techniques were based on the Vertex Animation concept, sometimes called Tweening [Watt and Policarpo 2001b; Anderson 2001]. Lately, 3D games have adopted a more efficient way of representing models known as hierarchical models. Each node of the hierarchical structure is called "joint" and the hierarchical structure itself is mostly called "skeleton" [Watt and Policarpo 2001b; Anderson 2001; Camilo at al. 2003].

These types of animation sequences can hold just the necessary data (as rotations, translations and scales), to indicate how each keyframe is taken from the baseframe. The flexibility of a hierarchical structure allows the processing stage to animate the character by using kinematics (forward kinematics or inverse kinematics).

Doom3's MD5 formats for model [Making Doom3 Mods 2004] and animations are completely up to date using skeleton structure, quaternions for defining the rotations and skinning. Quaternions[Batiste 2003] are a more suitable way of representing the rotations compared with Euler angles and matrices, because they do not fall in gimbal lock problem and permit the use of Slerp (Spherical Linear Interpolation)[Blow 2004] avoiding distortions that result from linear interpolation methods for rotations. Skinning [Lander 1997], that is a specific case of the FDD[Watt and Policarpo 2001b] technique for morphing, it allows vertexes to be associated with more than one joint, solving the common undesirable visual effects caused by the cracking or interpenetration of the rigid parts of the meshes, for instance the upper arm and low arm of a character.

In MD5 model format the skeleton is set in a referential pose called baseframe. In each MD5 animation format there is list of keyframe packages containing the rotations and translations for each joint. Each character may have a set of animation sequences that is associated with the high level animation system of the game through scripts describing the character's behavior [Making Doom3 Mods 2004].

Guff Framework's low level uses the MD5 model and MD5 animation formats to obtain the model and animations of a character. After the model and animations have been created in a 3D modeling tool, it can be converted to the MD5 formats.

Guff Framework's uses an intermediate configuration stage between the design and processing stage to allow game designers easily configure the animation sequences that will represent the behavior of a character. For a preliminary implementation each character has being modeled with a high level animation system based on a simplified finite state machine that can give them rudimentary intelligence.

Some approaches were proposed to help model designers create keyframing sequences. At [Igarashi at al. 2005] they presented a performance-driven approach, proposing a spatial, instead of temporal, interface for creating the animations. In [Terra and Metoyer 2004] is presented a post-creating stage that allows model designers to set timing of the animations in a simpler way than allowed for general 3D modeling tools, using a 2D interface to describe timing by gesture. With Guff Framework's intermediate configuration stage a game designer can set controls of uniformity, evolution (the slow in and slow out effect) and timing which can be dynamically controlled by the processing stage during the game playing. These controls can be accomplished by a varying number of intermediate frames between each keyframe pair (segment) of an animation sequence. These values of intermediate frames will be used as the amount of frames to be interpolated at the processing stage.

Since model designers could feel not comfortable creating blending sequences of animation to each possible pairs of animation sequences that can be played sequentially, it is interesting to have a way of creating suitable blending effects at run-time. In [Watt and Policarpo 2001b] is presented a simple way it can be undertaken by easily interpolating linearly the two animations based on time. Guff Framework's low level animation system uses this approach because it has a low processing cost, and the results are good considering a 3D game demands. However, this method has some drawbacks that have to be carefully taken into account (these drawbacks and the way they can be avoided will be shown later on this work). Guff Framework's intermediate configuration stage has controls that help game designers to avoid the drawbacks when blending between animations.

Figure 2 shows in more details which are the configuration tasks the intermediate configuration stage is responsible for.

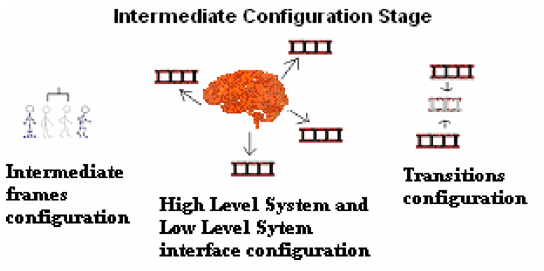
Intermediate Configuration Stage
Intermediate frames configuration
High Level System and Low Level Sytem interface configuration
Transitions configuration

Figure 2: Intermediate Configuration Stage in details

3. Intermediate frames configuration module

The first motive for adopting an intermediate configuration stage between the design stage and the processing stage, is the number of intermediate frames configuration.

Generally, when a model designer create an animation sequence in a 3D modeling tool, at the design stage, all the frames (keyframes or intermediate frames) are converted to key frames when the animation is saved to the MD5 animation format. However there is a problem when using this approach: intermediate frames represented as keyframes demand too much amount of memory because they hang all the information for rotation and translations of each joint, as the real keyframes does. Later, a unique number of intermediate frames are set globally for each pair of keyframes that is used at the processing stage for interpolation of frames. Doom3's low level is based on this concept [Making Doom3 Mods 2004].

This paper introduces a new approach where just the keyframes will be saved in the design stage. At the intermediate configuration stage, a game designer can set the amount of intermediate frames for each pair of keyframes. The number of intermediate frames for each pair of keyframes is saved in a configuration file as a single integer leaving a larger capacity for memory. The processing stage is not affected, because having a varying (or local chosen) number of intermediate frames between each pair of keyframes does not make difference in the complexity of the animation sequences processing.

A tool was developed, by the authors of this paper, for supporting the intermediate configuration stage, allowing game designers to perform three types of aggregated controls: uniformity, evolution based and timing, over the animation sequences by manipulating the number of intermediate frames.

3.1 Uniformity control

To non-experienced model designers a major concern is the undesirable non-uniformities resulted from the design stage. Uniform control intents to improve the final result of the animation sequences by putting an adequate amount of intermediate frames between keyframes to make an animation sequence more uniform. Varying number of intermediate frames between pairs of keyframes can attenuate the rough effects that a not well balanced animation sequence may present.

An important point in mind when using this type of control is to determine how much a keyframe is "distant" from another. To be more specific what is a distant measure for keyframes? As part of this project it was used a measure based on the difference between two consecutive keyframes. To determine the distance, it was computed the difference between the positions taken for each vertex at each keyframe. This yielded the following result: the bigger the distance between two key frames the more different they will be. The distance measure for keyframes needs to obey the four distance axioms: Positivity, Restrictly Positive, Simetry and Triangle Inequation [Naylor and Sell 1982]. Distance used here is the Euclidian distance for each vertex position taken from the two keyframes. Euclidian distance obeys the four axioms [Naylor and Sell 1982].

For a keyframe the spatial values of a vertex can be called (vx_1, vy_1 , vz_1) being related with the canonic x, y and z axes respectively. For the following keyframe a similar notation for the same vertex was used: (vx_2, vy_2 , vz_2) at the same conditions about canonic axis. Euclidian distance between the vertexes in each keyframe is:

$$dv_{12} = ((vx_1 - vx_2)^2 + (vy_1 - vy_2)^2 + (vz_1 - vz_2)^2)^{1/2} \qquad (1)$$

Dv_{12} is called as the distance between the vertex v at the keyframe 1 and the same vertex v at the keyframe 2. Adding the distances of each vertex yielded the distance between keyframe 1 and keyframe 2. Figure 3 shows the distance between the two keyframes (1 and 2), the blue dots are the character's vertexes.

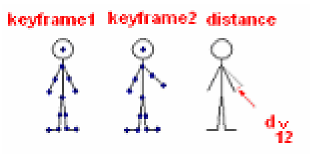

Figure 3: Keyframes distances

To avoid cases such as: when a few vertexes are distant, but the key frames themselves are not different from one another it is best to calculate the sum of all the vertexes' distance. As figure 4 shows, one vertex is not enough to measure the keyframes distance.

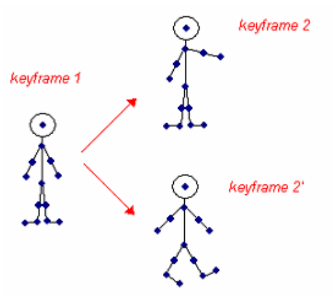

Figure 4: All vertexes contribution to distance computation

If the left hand dot is used as reference then keyframe 2 is the most distant from keyframe 1. But if the whole set of vertexes is taken into account then keyframe 2' becomes the most distant from keyframe 1.

After determining the keyframes distances, they can be normalized and a relation function can be used to set a larger number of intermediate keyframes for greater distances.

3.2 Evolution based control

In this work, evolution is how is called any non-uniformity effect a model or a game designer would want to show in an animation sequence. In this case, as oppose to the last section, the non-uniformity is desired because it can help to show some interesting physical effects like weight, inertia or acceleration and some emotional effect like tired feeling, frightening and run away, etc. A well known example of evolution used to show the physical effect of inertia is called slow in and slow out [Lasseter 1987]. This effect starts as a slow movement then it gets faster and faster until it reaches a culminant point and then starts to slow down until it stops.

Slow in and slow out is shown in the figure 5. The original animation sequence contains five keyframes. Then between each pair of keyframes it was inserted a varying number of frames that makes the animation sequence works slow at the beginning (because of the large amount of intermediate frames) then fast in the middle (few intermediate frames) then slow at end (again a lot of intermediate frames).

In this example, the keyframes are considered to be equally distanced from each other. Slow in and slow out effect can be completely implemented in the intermediate configuration stage since it depends on the amount of intermediate frames. This way, the intermediate configuration stage will work like some time-editing tools that are part of 3D modeling tools.

It is important to point out that uniformity control and evolution-based control are not opposite ideas. They are really complementary to ones another. Uniformity can be used to fix undesirable non-uniformities presented by a not well-dealt animation sequence. The evolution can be used later to print non-uniformity desirable effects on the sequence animation.

According to [Parent 2001], some types of interpolation functions can be used to model the slow in and slow out effect. Some functions used to find the amount of intermediate frames showing the desirable effect were: senoidals, parabolics and cubic splines [Watt and Watt 1992]. The intervals (between pairs of the two keyframes) were normalized to fit inside the [0, 1] interval.

3.3 Timing adaptative control

Timing is the velocity and acceleration representation of a movement in an animation sequence [Lasseter 1987]. Timing can give weight notion of an object, influence in the perception of the size and scale of scene objects and characters, and helps to define emotional state of a character [Lasseter 1987].

Guff Framework's intermediate configuration stage allows game designers to set a global factor for number of intermediate keyframes that works aggregated with the uniformity and evolution based controls.

Since game designers may not have a real notion of the game application's performance during real-time at the intermediate configuration stage, this work proposes a two-hand control that allow the processing stage to adapt the timing factor according with a

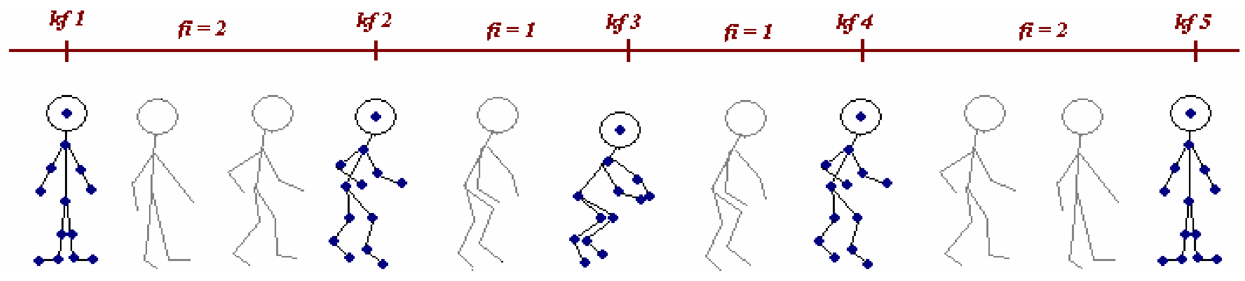

Figure 5: *Slow in* and *Slow out*

performance measure. This way, the computational performance of the game is also taken into consideration when playing animation sequences.

The frames per second rate is used here as a measure for the processing stage timing factor adjustment. When the frame per second rate is too low, the game's application just adjusts the timing factor to a lower value so the animation sequences play faster (with less intermediate frames). Since the frame rate comes to the normal rate then the timing factor can be increased too.

This way, the playability of the game is maintained in spite of the timing quality given by a game designer. It is important to notice that just the timing factor is adjusted. The uniformity and evolution factors keep on working without suffering the adjusting influence, since they are joined just at the moment they are going to be applied during the animation sequence processing.

4. Blending Transitions Module

During the lifetime of a character in a game it will probably have to change its state according with events that happen at the environment and with its own intelligence. As changing states means most of the time changing animation sequences, transitions between animation sequences are an important question in a character animation system.

In this work, two types of change of states the character can present are considered: the deterministic transitions and the non-deterministic transitions.

Deterministic transitions happen when it is known exactly at which frame (being a keyframe or an intermediate frame) the animation sequence will begin to blend to the following one. With non-determinist transitions, in the opposite, it cannot be known at which frame it will begin to blend.

Deterministic transitions occur generally when a character will change its state without any influence of the environment. For instance, it is running (the current animation sequence is running) and slowly stops blending to an idle position (the following animation sequence is idle). Non-deterministic transitions occur when a character suddenly change its state because something that happened in the environment. For example, when a character is shot, it leaves its idle position and begins to run away.

When dealing with animation sequences a system will face two major problems regarding the transitions between animation sequences: positioning and blending. The first problem comes about at the design stage where a model designer will position the model at the beginning of the animation sequence in a place probably completely different from where it should be when this animation sequence begins to be played at the processing stage. The coordinates of each vertex of the model at the first keyframe of the following animation sequence have to be remapped to the point where the model is, in the current animation sequence which is being played.

The blending problem concerns with the smoothness the current animation sequence should transition to the following animation sequence. When nothing is done about blending what is seen is a very rude transition, if the two animations sequences have edge keyframes (final keyframe of the current animation sequence and beginning keyframe of the following animation sequence) very different.

Leaving all the transitions responsibilities for the design stage does not seem to be a good solution, mainly because a model designer could hardly predict all of the possible transitions between sequence animations. Even if they could, it would not be enough to solve the positioning and blending when the transitions are non-deterministic. At Guff Framework's low level animation system the transitions are dealt at the intermediate configuration stage. As will be shown later, at this stage a game designer will set the animation sequences that will be played at each change of state, and the control settings necessary for the transition to be suitable. Here, it was chosen to approach the problems of position and blending in a simple and effective way, because, this way it could make the transitions feasible and at the same time do not overload the processing stage with computations.

The positioning problem is solved with a remapping of the coordinate system where the origin is set to be the position of the center of the model at the current frame of the current animation sequence. This way, all of the vertexes of the frames (keyframes or intermediate frames) of the following animation sequence are changed making the following model's position center point the same as the current model's position center point. The calculation for a generic vertex is explained below where v_f is the position of the vertex and $v_{f'}$, the current model's center point is called c_a and the following model's center point is called c_f, is as follows:

$$vf' = vf - cf + ca \qquad (2)$$

The blending solution is also very simple and based on [Watt and Policarpo 2001b]. The key idea is to interpolate linearly two animations making a fade in at the current animation sequence as making a fade out at the following animation sequence.

Calling the current animation sequence as A_a and the following animation sequence as A_f, and the final result for the time t as A, the calculation result is showed as follow:

$$A = (1 - \alpha t) . A_a + (\alpha t) . A_f \qquad (3)$$

It is important to notice that the product t . α varies from 0 to 1 as time passes, α being an interpolating factor. Moreover, the current animation sequence and the following animation sequence are changing too. So at each instant the frames (keyframe or intermediate frame) are different both in the current and in the following animation sequences.

Figure 6 shows the idea of interpolation using colors to represent the interpolating effect between the two animation sequences. During the blending process there is no more difference between keyframes and intermediate frames. They are all frames of one animation sequence being interpolated with its aligned correspondent frame of the other animation sequence.

As stated by [Watt and Policarpo 2001b] this approach for blending transition between animation sequences are not granted to work as a real world transition. However it can give good results since the animation sequences are aligned and similar. Since linear interpolation for transition blending has some drawbacks, these drawbacks are analyzed here and some techniques to solve them are presented.

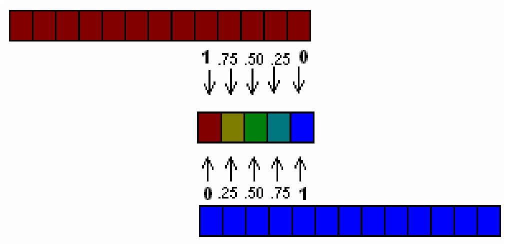

Figure 6: Linear interpolation blending transition

Those techniques became settings at the intermediate configuration stage tool, so a game designer can deal with them to make suitable blending transitions.

In the work on which this paper is based on, it was identified four conditions when the linear interpolation for blending transitions does not work well. They are as follow:

"Rubber" effect: The animation during the transition seems to play by distorting the mesh of the character giving the impression that the character is made out of rubber. Intuitively it happens because of the big difference between the two aligned frames being interpolated. This effect can be seen at figure 6, where in the blending phase, the second resulting frame (yellowish) is not similar to any of the aligned frames that were blended to generate it.

The proposed solution is to choose animations that could play together more smoothly. At the intermediate configuration stage, a game designer is warned whenever it is chosen two animation sequences that have frames aligned that differ more than a pre-set threshold value.

"Goes smoothly then suddenly jump" effect: As the name implies, it happens when a transition goes smoothly then suddenly it jumps to a frame without smoothness. Sometimes it occurs because the current animation ends before the blending phase ends. It happens when a game designer set the current animation not to cycle during the blending phase. For the remaining of the blending phase the following animation goes along giving the non-smoothly impression.

In this case the solution proposed is to choose the current animation as big as possible. Even doing it, at the occurrence of a non-deterministic transition, it still could happen. However, with bigger current animation sequences, a good reduction of this effect can be expected. At deterministic transition, this problem is completely controlled by a game designer at the intermediate configuration stage as it can chosen at which frame of the current animation sequence the blending phase will begin, which in turn will result in making the phase ends before or at the right frame where the current animation sequence ends.

"Goes smoothly then displace then come back" effect: As the name says, this occurs when the animation transition goes smoothly then suddenly the model appears in another point of the world and then come back to the first location. This effect occurs because of the same reason as "goes smoothly then jump" effect, but the result is different because in this case what happen is that a game designer allows the current animation sequence to cycle. When the current animation reinitializes, improperly experience the effect of the wrong positioning calculation; this is why the model appears in a wrong place and then because of the remaining of the interpolation it comes back to the right position in the world.

The idea here is to not allow the current animation sequence to cycle while the blending phase is taking place. As a matter of fact, the following animation sequence can also ends before the blending phase finishes, so it is also recommended that it should not be cycled too. Moreover, the following animation sequence should be chosen so as to never ends before the blending phase finishes. These settings can all be done in the intermediate configuration stage.

"**Hit and back**" **effect**: Occurs when a hit and back impression is seen from the animation transition. To understand why this effect occurs, a deeper view in the relationship between the changes of states, animation sequences and animation transitions, has to be taken.

When a non-deterministic transition happens, the current animation sequence begins to blend to the following animation sequence. Since the following animation sequence represents a not idle state, soon the character will have a deterministic transition that will bring him back to the idle state. For example, when a character is shot it becomes frightened and starts running and once it feels safe it stops running and starts patrolling.

For deterministic change of state a game designer has to choose a frame where it will begin the transition to an animation sequence that represents an idle state. If the game designer chooses a frame that is very near to the beginning of the animation sequence it can occur that this frame is taking part of a non-deterministic transition.

Because the blending phase does not allow the following animation plays like a current animation of another animation transition, the second transition cannot be performed. So the former following animation sequence that now is the current one, have to cycle once more to initiate the deterministic transition. That second cycle causes the hit and back effect in the animation transition.

To avoid this type of effect a game designer has to set the animation sequences, for deterministic change of state, big enough to support the transitions where it is the following animation sequence, a fragment of the sequence without transition and finally the transition where it plays the role of the current animation sequence.

5. High Level Animation System

As a first trying for creating a high level animation system for the Guff Framework's character animation system it was implemented a simple finite state machine that can make the character interact with the player and the environment.

This first approach also serves as a test for the low level animation system as it can ask for animations previously set, and the low animation level can supply it with the animations suitably. Figure 7 shows the finite state machine diagram where the states can be explained by their own names.

The "Detect Enemy", "Shot", and "Death" events are non-deterministic change of states. The remaining events are deterministic change of state.

For the "Detect Enemy" event it was used an oriented bounding box – sphere collision detection algorithm based on [Möller and Haines 1999]. For the "Shot" event it was used an oriented bounding box – ray collision detection algorithm also based on [Möller and Haines 1999]. For the "Death" event, it is set an amount of life for the character. Whenever he is shot its amount of life is decremented. When it comes to zero the character begins to die.

For all the events a game designer is allowed to set animations sequences and frames of beginning of deterministic change of state transition animations at the intermediate configuration stage.

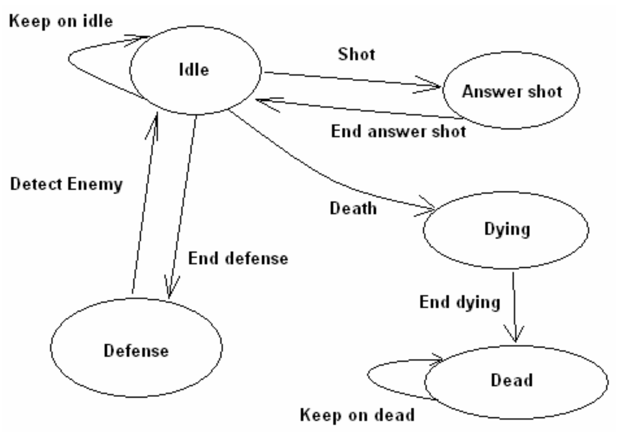

Figure 7: Finite State Machine of Guff Framework's high level animation system

6. Results

Some tests were run to verify the Guff Framework's character animation system and its facilities [Camilo and Conci 2006]. Tests were made using real characters' animation sequences taken from Doom3. It was used animations from Archvile, Guardian, and Cherub characters.

At first it was tested the character animation system using a simple finite state machine to act as a high level animation system. It worked as expected showing that the low level layer does well in supplying the high level with the animation sequences demanded. Figure 8 shows the Archvile character defending itself using the "attack" animation sequence. The lines around the character are its axis aligned bounding box represented.

Then some tests over the controls of number of intermediate keyframes were done. These tests showed that the three controls work fine in their intents and that they can work together to make animation sequences suitable for games. The uniformity control was applied to an animation already created with non-uniformity behavior. The control did well in giving uniformity to animation sequences. Figure 9 shows the amounts of intermediate frames should be created to uniformize the animation sequence "attack" of the character archvile.

For the evolution based control, it was identified that the best approach for the slow in and slow out evolution effect was the cubic spline curve. With this curve the slow in and slow out effect was played well giving to the animation more smoothness than the other curves as showed at the figure 10, where the three types of evolution approaches were applied over the uniformity control.

The timing control and its behavior during the execution of the application were also tested. A 60 frames per second rate was set as a threshold for adjusting timing control. Behind this value the timing factor was adjusted according with the frame rate variations, making the animation sequences playing speed vary coherently with the frame per second rate. Figures 11 and 12 show the variation of the timing factor according with the frames per second variation, for 7 and 10 characters being animated.

The intermediate configuration state for settings of the blending transitions was also tested. It succeed well in allowing setting all of animation sequences transitions of characters avoiding the undesirable effects for animation transitions cited before.

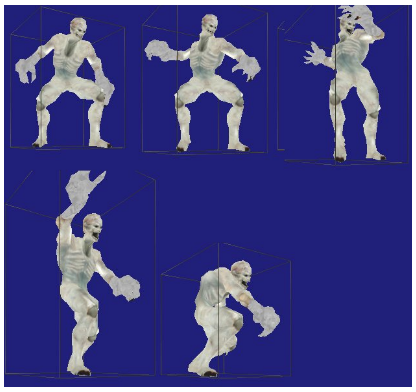

Figure 8: The "Attack" animation sequence of the Archvile character defending himself

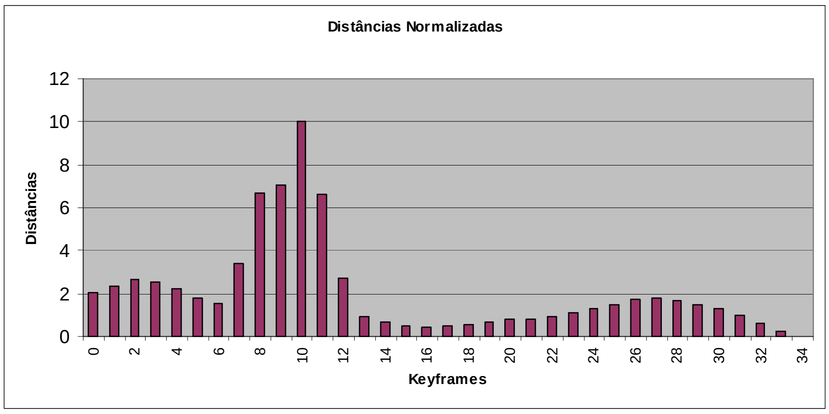

Figure 9: Segments distant between pairs of near keyframes show how much interpolated frames should be created to uniform the animation sequence

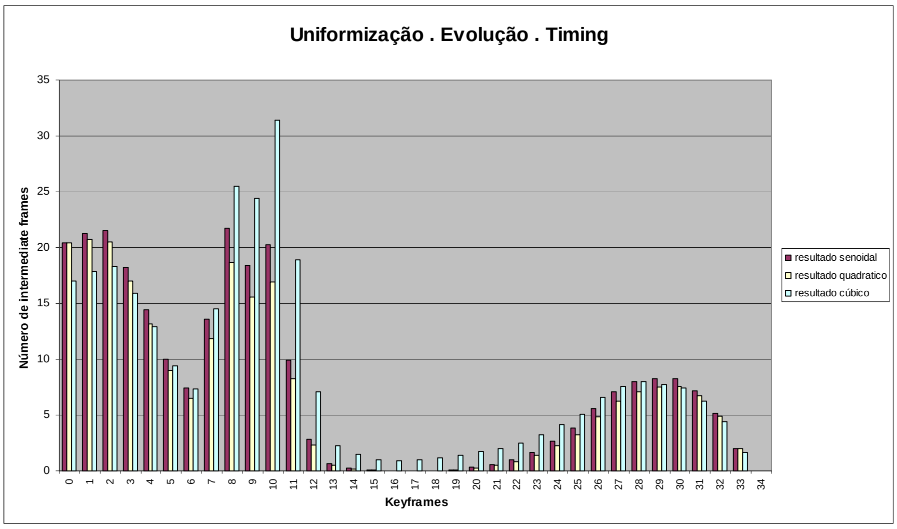

Figure 10: Uniformity and Evolution combined

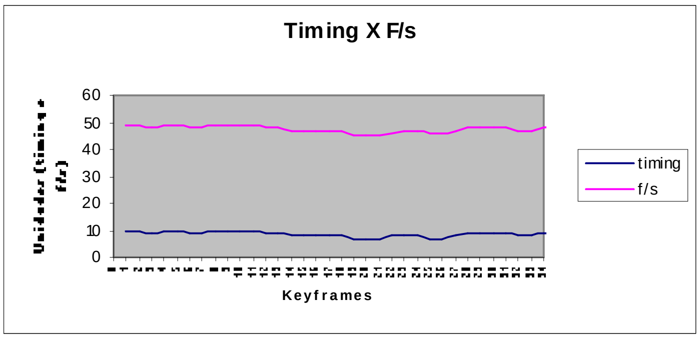

Figure 11: Frames per second and timing variation for 7 characters being animated and (b) 10 characters being animated

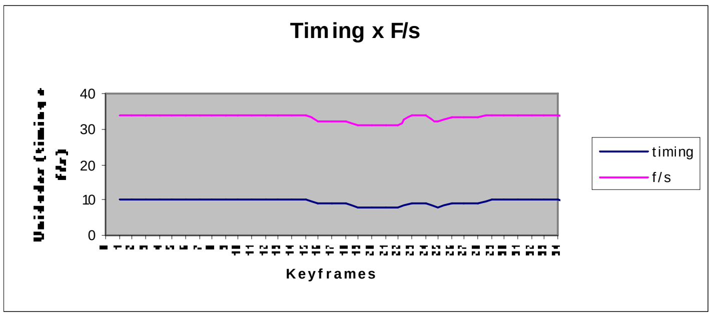

Figure 12: Frames per second and timing variation for 7 characters being animated and (b) 10 characters being animated

Finally is presented here a comparison between Guff's Framework Animation System and others current well known game engines. Table 1 shows this comparison.

When it was not possible to infere, based on the documentation provided by the framework or game engine distributors, if the characteristic is presented or not, a interrogation mark is showed.

Framework or game engine	Skeletal animation and skinning	Intermediate configuration stage	Transition control	*High level layer*
OGRE [Ogre 3D 2006]	yes	?	yes	*no*
Cal3D [Cal3D 2006]	yes	no	yes	*no*
Doom3 engine [Making Doom3 Mods 2004]	yes	yes	?	*yes*
Guff	*yes*	*yes*	*yes*	*yes*

Table 1: Frameworks and game engines comparison

7. Conclusion

This paper presents the Guff Framework's character animation system. This character animation system is based on two layers: low level animation layer that deals with the animation sequences, and a high level animation sequence where the character's intelligence takes place.

The low level animation is based in keyframing techniques. It was presented here an approach of inserting an intermediate configuration stage between the design stage and the processing stage of the keyframing process. The intermediate configuration stage is also useful for setting the animation sequences set that will supply the high-level animation system and the blending transitions between animation sequences.

Future work development might uncover a better non-deterministic approach for the high-level animation system. Also a more effective approach for evolution control during the intermediate configuration stage, like parameterized spline curves [Azevedo and Conci 2003], will be studied to accomplish more suitable results.

References

Anderson, E. F., 2001. Real-Time Character Animation for Computer Games. Bournemouth University. Available from: http://ncca.bournemouth.ac.uk/newhome/alumni/docs/CharacterAnimation.pdf [Accessed 02/2006].

Azevedo, E., Conci, A., 2003. Computação Gráfica. Teoria e Prática. Rio de Janeiro. Editora Campus.

Batiste, S., 2003. Squeezing the Animation. Game Developer. The H. W. Wilson Company.

Blow, J., 2004. Understanding Slerp, Then Not Using It. Game Developer. The H. W. Wilson Company.

Camilo, M., Conci, A., 2006. Um Estágio Intermediário de Configuração de Animações para jogos e aplicações de simulação 3D. Rio de Janeiro. In SPOLM 2006.

Camilo, M., Martins, R., Hodge, B., Sztajnberg, A., 2003. Considerações sobre Técnicas para Implementação de Skeletal Animation em Jogos 3D. Rio de Janeiro. Cadernos do IME - UERJ. Available from: http://www.ic.uff.br/~mcamilo/gprogramming/skeletalanimcarcara.pdf [Accessed 02/2006].

Dybsand, E., 2003. AI Middleware: Getting Into Character. Game Developer. The H. W. Wilson Company.

Hodgins, J., O'Brien, J. F., Bodenheimer, R. E., 1999. Computer Animation. Atlanta. Georgia Institute of Technology.

Igarashi, T., Moscovich, T., Hughes J., 2005. Spatial Keyframing for Performance-driven Animation. Eurographics / ACM SIGGRAPH.

Lander, J., 1997. "On Creating Cool Real-Time 3D". GamaSutra. 1997. Available from http://www.gamasutra.com [Accessed 07/2005].

Lasseter, J., 1987. Principles of Traditional Animation Applied to 3D Computer Animation. Pixar. San Rafael, California. ACM.

Making Doom3 Mods: Introduction, 2004. Available from: http://www.iddevnet.com/doom3/ [Accessed 01/2006].

Möller, T., Haines, E., 1999. Real Time Rendering. Massachussets. A. K. Peters.

Naylor, A. W., Sell, G. R., 1982. Linear Operator Theory in Engineering and Science. Holt, Einenart and Winston. New York.

Parent, R., 2001. Computer Animation Algorithms and Techniques. San Francisco. Morgan-Kaufmann.

Thalmann, M. N., Thalmann, D., 1998. Computer Animation in Future Technologies. Switzerland. Miralab.

Terra, S., Metoyer, R., 2004. Performance Timing for Keyframe Animation. Eurographics / ACM SIGGRAPH.

Valente, L., 2005. GUFF: Um framework para desenvolvimento de jogos. Niterói. UFF. Available from: http://www.ic.uff.br/~lvalente/en/projects.html [Accessed 02/2006].

Watt, A., Policarpo, F., 2001. 3D Games: Real-Time Rendering and Software Technology. Vol.1. New York. Addison-Wesley.

Watt, A., Policarpo, F., 2001. 3D Games: Real-Time Rendering and Software Technology. Vol.2. New York. Addison-Wesley.

Watt, A., Watt M., 1992. Advanced Animation and Rendering Techniques: Theory and Practice. New York. Acm Press. Addison Wesley.

Ogre 3D. Available from: http://www.ogre3d.org [Accessed 02/2006].

Cal3D. Available from: http://download.gna.org/cal3d/documentation/api/html/index.html [Accessed 02/2006].