"Character Animation Engines for Interactive Game and Web Applications using hardware assisted functionality"

Tice, S.E. and J. Lander, QuantumWorks Corporation[1], International Workshop on Human Modeling and Animation (HMA) 2000 at Seoul National University (SNU)
Seoul, Korea June 28-29 2000

1 Abstract

Computer graphics based character animation has been a very active topic of research. Much of this research has focused on creating characters that are both the most realistic visually and in motion, regardless of computational cost - both in time and processing power. Creating animated characters for interactive entertainment poses a unique challenge for artists. While characters for cinematic (or cut-scene) sequences can be created by artists using traditional computer animation methods, interactive characters must be rendered at interactive rates. As a result, creating these interactive characters now requires the additional talents of real-time graphics programmer "artists". Further, the processing power used to create these characters is strictly defined.

Traditionally, computer games have used very simple traditional cell animation techniques for creating interactive characters, sometimes known as "sprites" and "sprite animation". Real-time 3D graphics rendering hardware used to be the domain of flight simulators and super-workstations, such as those produced by Evans & Sutherland and Silicon Graphics (SGI). For the production of cut-scenes, film and TV special effects companies have used super-workstations, but the real-time graphics units of these machines and their output is rarely at an acceptable quality. These companies used frame-by-frame rendering tools on these platforms or sent the frames out to their rendering farms to create the suspension of disbelief a.k.a. the high-end output. However, with the increasing availability of 3D rendering hardware on consumer gaming platforms, an increasing number of game productions are using interactive 3D rendering techniques to create character animation in order to squeeze the most realism from the still limited graphics power.

This same increase in availability of graphic rendering hardware and increasing desktop computer power has enabled all computer users, not only the hard-core game players, to experience 3D technology. Web-based entertainment and information provide an excellent opportunity to use this new capability as well.

This paper will examine the problem of creating highly realistic and believable characters which "suspend disbelief," for interactive entertainment on today's new generation hardware. This hardware not only allows basic real-time graphics rendering of characters, but now the hardware allows unique real-time implementations of the traditional frame-by-frame CPU intensive techniques. The hardware includes traditional computing devices such as the PC and Mac, as well as video game consoles such as the Sega Dreamcast, Sony Playstation2, Nintendo 64, yet to be released Nintendo Dolphin and Microsoft X-Box. The emergence of graphics hardware enbled internet appliances and set-top boxes provide further opportunities. We will examine: the authoring pipelines and production techniques; the traditional assets building tools and the game engine specific level/assets creation tools and the run-time engines that play back the characters in the real-time interactive application; and then present solutions using hardware features and techniques in non-standard ways. Specifically, we will examine potential innovations in the three areas that are directly responsible for optimally utilizing the new real-time interactive gaming hardware:

- **Asset construction pipeline** – the general modeling/painting/animation/rendering/compositing suite of tools
- **Game Engine Specific Creator/Editor** – a.k.a the "level editor", includes the game engine and allows specific asset authoring and review
- **Run-time Game Engine** – the core software component part of the distributed game

2 Background and Previous Work

2.1 Definitions

The authors intend this material to be of interest to a wide range of practitioners, from artists, designers, producers to developers, and from diverse fields of study and industries, thus it is necessary to establish a terminology base to aide in further discussion.

- **Interactive character animation** – 30 images or frames per second or more generated in real-time by the game platform system

[1] Contact: Steve E. Tice at stevet@quantumworks.com 818-906-3322 or Jeff Lander at jeffl@quantumworks.com 310-375-6602

- **Software rendered** – images are created through use of software algorithms running on general purpose computing hardware

- **Video game consoles** – closed platform machines such as Sony Playstation (PS-X) 1 and 2, Sega Dreamcast, and Nintendo64

- **Computer game platforms** – Personal Computer or PC-based open hardware platforms of the Intel or equivalent variety or Apple Macintosh

- **On-line gaming** – general category that includes massive multiplayer games involving peer-to-peer or client/server gaming architectures

- **Networked Gaming** – general category for LAN, local area network, based games

- **LOD** – level of detail or various representations in complexity of game scenery, environments, props, actors, special effects that allow the game engine to optimally render the scene specific to the players point of view

- **T&L** – transform and lighting, referred to most often as the 2 sets of mathematics required to place a game's scenes and objects in appropriate positions for application of the appropriate second class of mathematics to calculate the scenes lighting for final rendering

- **Matrix deformation** – using a matrix to deform a model frequently known as “bone animation” or “skinning”

- **Single Mesh 3D Object** - A single set of vertices that make up an object

- **IK** – inverse kinematics

- **Hierarchical 3D Object** - Multiple mesh objects linked in a hierarchy

- **Bone** - Single node in character hierarchy

- **Weighting** - Ratio of influence between bones in a skeletal system

- **MoCap** – motion capture – a technique to capture human motion and to create libraries for game engines to interactively draw upon to create new motions

2.2 Introduction

Interactive character animation has been an important part of games for some time now. However, due to the limited capabilities of the game hardware, the animation was fairly simple. We used cell animation techniques to create 2D animated characters. These characters, while compelling artistically, lacked the ability to dynamically interact with their environment. The move to simple 3D characters enabled a much greater amount of freedom. Characters like Lara Croft in the famous Tomb Raider series, were able to respond to their environment. Her simple hierarchical model enabled the animation system to let her look at objects in the game environment as well as aim her weapons at targets. However, the limited rendering ability of the target game systems meant that the character was fairly simplistic visually.

As game hardware improved, so did the characters. Games began to feature smooth skinned characters and simple facial animation systems. Polygon counts on characters has risen steadily with the increase in this hardware capability. The current games' state-of-the-art consists of highly refined texture map art and polygonal mesh representations of faces, head, hair and bodies. Facial animation is limited to basic expressions and lower face only lip synchronization. The facial animation is being accomplished through texture animation as well as simple vertex morphing and model swapping techniques.

The research world is pursuing realistic character creation, both interactive and non-interactive, that has generated many opportunities for innovative execution in limited game environments.

In the feature film world, character creation has advanced quite far, but is still quite costly and is certainly not real-time. Feature film companies apply research algorithms and extend them for realistic digital special effects

creation. These are applied to either digital prosthetics (which attempt to be "seamless or unnoticeable effects") and/or creating entire digital human characters, such as digital stunt doubles. Industry standard software renderers and compatible custom shaders and scripting languages, such as Pixar's Renderman, Softimage Mental Ray, Alias|Wavefront's Maya Mel Script, and Discreet Logic/Kinetix 3DSMAXs Maxscript macro language, are dependent on the creation of stunning effects with rendering times anywhere from seconds to many minutes per frame. Any new effect required that is not part of a standard renderer is then created via these compatible custom shaders that address one effect for one object.

To date, seamless digital humans have not taken center stage for close-up performances in any films. This milestone has been attempted for some projects with unique technology [**33**]and we hope to see the first feature film with this level of digital "cloning" within next 2 years or so.

2.3 Related Work

2.3.1 Games which have led the way in hardware implemented methods – real-time interactive milestones

PC

- Activisions' and Nihilistic Softwares' ***Vampire: The Masquerade – Redemption*** - character shadows, lights, shading, body animation,
- ***Tresspasser*** – less then compelling game but notable example of IK to plant character feet on uneven terrain

Sega Dreamcast

- Segas' **Shenmue** - milestone in interactive cinematography and character animation and rendering
- Segas' **Seaman** - milestone in game based character face animation
- Segas' **Virtua Tennis** - character animation, blending of motion captured data
- Midways' **Ready to Rumble** - matrix deformation for animation and muscle bugles

Sony Playstation2

- Capcoms' **Onimusha** - character textures, shading, lighting (rendering), and body animation
- EA Sports – **Madden 2001** - character motion, blending, IK for tackles and catches

2.3.2 Technical papers

There has been considerable work in many aspects of 3D character animation, but one area is especially interesting to our topic - the use of real-time performance animation systems to create 3D character-based computer graphics imaging (CGI) content more efficiently [**1**]. These performance animation systems are also used live to perform a 3D character in front of audiences at various trade show, TV broadcast and entertainment venues. These systems had the luxury of more expensive dedicated graphics hardware such as the super-workstations from manufacturers like SGI and Intergraph. The methods used in these authoring systems are finally applicable to game run-time engines that will be described later in this paper. These types of content authoring systems, inspired by early academic work, are even more applicable today for game character facial and body animation authoring [**2**]. Using only voice and text as an input for creating the initial layers of facial animation has also had some success.[**3**] Combining both performance animation and text to muscle values is next in the evolution of these high performance facial authoring systems.

Controlling LOD and more importantly blending various LODs for all geometry smoothly has seen great advancements in games with the advent of continuous LOD used in many games today [**4**].

Facial animation has been fortunate to have some excellent medical and artistic reference sources to fuel the engineering efforts [**5**]. From the initial work [**6**], to the more recent, greater control and more realistic facial animation run-time methods have been explored [**7**, **8**, **9**, **10**]. In addition, we have employed hardware assisted run-time methods for face/body animation in demo game run-time engines [**11**, **12**, **13**, **14**, **15**]. Two comprehensive works covering these topics are certainly situated on most game programmers' bookshelves [**16**, **17**]

Researchers have explored interesting authoring and potential run-time solutions for real-time methods for hierarchical-based body animation of characters including IK, constraints, and dynamics [**18**, **19**, **20**, **21**, **22**, **23**, **24**, **25**].
Furthermore, gamers have explored real-time procedural animation for its data efficiencies where precise artistic character animation control can be compromised. [**26**, **27**, **28**].

For individual character parts that interact with the game environments, optimized soft body deformable objects implementations are potentially very useful for game productions [**29**, **30**, **20**]. These can be quite useful to simulate character reactions to physical effects from the environment or other characters.

Along similar lines is the application of real-time methods for character hair and clothes animation demonstrated by previous work [**31**].

Getting large sets of efficiently compacted textures to "play" through optimal game engines on platforms with limited memory also has it challenges and some exploration of innovative techniques is becoming more of interest [**32**].

Finally, at the end of the pipeline, is very interesting work done in the area of hardware accelerated rendering [**33**, **34**].

3 State of the Art for Interactive Character Animation in Shipping Games

3.1 Authoring production pipeline to support character construction and animation requirements for today's games

3.1.1 Overview

Let's look at the creative process. Games begin as concept papers, sketches, then storyboards, game story "bibles", character maquettes, scanned game elements, demos and so forth. Depending on game application, human motion and models are scanned and captured. As the game develops iteratively, with successive prototypes, more refined 3D character modeling, texture map creation and character control setups are created. To keep pushing the envelope, required to keep drawing the customers to ones' games, the technical engine developers or programmers usually set the "bar" with new technology demos which require the authoring process to go from the "general" to the "optimized" to run on their new creations.

Figure 1 - Image from Playstation console game Onimusha

So actually two different systems for authoring are required, the asset construction pipeline and the game engine specific creator/editor. We will address the differences and the impact of these on the real-time interactive game engines.

Figure 2 - Sega AM2 developed Shenmue for Dreamcast video game console

Firstly, the asset construction pipeline is a collection of integrated off-the-shelf general purpose modeling/ painting/ animation/ rendering/ compositing tool suites used to create all 2D and 3D assets. This pipeline does include the character animation MoCap subsystems, if appropriate to the game being developed.

Secondly, the game engine specific creator/editor also known as the "level/character editor" is a creation tool that allows the creative team to develop specific levels and other content (including actors, props, effects, etc.) for the game. The real game engine is included for previewing the development. Sometimes these level editors are also released to allow customers to build their own "games."

In developing techniques which will run interactively on today's gaming platforms, developers have always struggled with developing efficient authoring pipelines to allow artists to easily test their work on the target platform and to allow constant changes throughout the games development lifecycle.

Current developers have met this challenge with varied success. When game company develops a new game engine, generally this company also develops a set of authoring tools and interfaces to use with off the shelf art creation packages. Because the game itself must be completed within budget and time constraints, development of these latter tools is usually not optimal. Some mature development studios have developed robust authoring pipelines and interfaces to handle any new game engine development with abilities to prototype new uses of the new graphics hardware as it becomes available.

The current state-of-the-art for character development in modeling has been brought about with great hardships. Here are a couple of the latest examples of successes starting to break into the realm of film-like character animation, albeit quite stiff and limited in their emotional communications and appeal. A re-rendered cut scene image of the actual interactive character from the Sony Playstation-2 game Onimusha is shown in Figure 1. An image from the milestone Sega Dreamcast game Shenmue is shown in Figure 2.

Now we will examine each aspect of game character development with respect to the required authoring support subsystems and tools. Facial and Body modeling are sometimes done totally separately. The character set-up artist builds animation control structures for the "skins." Audio tracks are recorded first. The team may choose to use a traditional morph target approach to manually keyframe animation or motion capture data. Alternatively, the team may choose a traditional skeletal character setup with a skinning approach. With this skeletal approach keyframe animation or motion capture can be automatically tracked in an animation system. The artist can also layer or adjust the auto-tracked data. A game engine can be developed specific to target platform(s) and game developers and artists can use the game specific engine to develop and preview each level and all contents – adjusting most game engine parameters.

3.1.2 Character creation for interactive real-time control and response verses creation for cut-scene rendering

Character complexity for real-time games today is currently 200-8000 polygons per model depending on total scene content. When the scene only contains two characters, higher LODs for these characters are possible while maintaining real-time rates (30-60fps). While in the past, the creation of discreet level of detail models was the norm, today continuous LOD game engines have become a prominent part of new games. The artist creates models at the highest LOD in the general asset creation tool suites and then uses plug-ins or separate tools to compute the continuous LOD. Today the control of which polygon collapses and under which viewing conditions) they will appear is available to the artist.

3.1.3 Character Texture

Artists today tend to hand paint textures and/or use a combination of scanning real digital photography and adjusting for proper resolution for the game engine. Textures can be created to show how the character would look due to interactions with environment i.e. snow, rain, strong light sources like sun spotlights, or physical impacts. To display these effects, the textures can be dynamically switched or modified.

3.1.4 Character Animation

For real-time rendering there are two types of CG character animation, pre-rendered and pre-scripted. The asset construction pipeline or the general tools suites used today effectively create content for cut scenes (pre-rendered) or for long sequence of pre-scripted action in response to user or environmental input. Developing the library that the run-time game engine accesses is usually done with MoCap, Performance animation, key framing/ posing and motion blending.

Facial: For facial animation, traditional techniques could be used as indicated above. Alternatively, the technique of motion capturing an actor acting and speaking and manually tracking the CGI character to the captured data is useful under other circumstances. Or one can use an innovation presented by Tackacs, Fromherz, Tice and Metaxas [**35**] which actually puts the animation system and a new type of markerless tracking system in a closed iterative loop to create an entire face synchronization in a automated batch process. The animator then can go into animation system to add layers to adjust the automatically produced channel data developed by the system from the tracking data to derive additional nuance and realism for their characters.

Facial animation has become a topic of great interest in the interactive entertainment arena. As capabilities of hardware increase, polygon budgets used to determine production constraints rise. This gives animators the ability

to explore new areas of expression for their characters. For many animators, the most obvious and easiest method for creating facial animation was through the use of vertex morphing or shape animation (generally called Morph Targets). The animator simply moves the vertices in each frame to create desired facial expressions. As the animator can completely control each vertex directly, creation of specific expressions is quite easy; however, getting these expressions to look right in the specific game engine can be an entirely different matter [**36**]. On the playback side, displaying each morph target is a matter of adjusting the character mesh to match the target vertices. In addition, blending of several morph targets to create in-between expressions is quite easy.

There are downsides to the morph target approach that may lead technology away from vertex morph animation. One complication is that a direct method for applying motion capture and performance animation data to morph-based animation is not obvious. Technology developers such as those from QuantumWorks have effectively used the technique of gesture recognition to provide a bridge between input motion data and morph-based animation [**20**]. Gesture recognition requires the creation of a mapping between specific motion inputs and morph output settings. Once created, this can allow the performance data to be used directly. Another issue is that morph targets require a great deal of memory to store. As vertex counts increase this becomes an area for great concern, particularly on dedicated rendering hardware such as 3D cards and set-top boxes. While compression techniques can alleviate this to some extent, it clearly is an area of concern. Likewise, graphics hardware manufacturers are concerned with bus-bandwidth issues in the transference of large quantities of vertex data. A great deal of data is required for manipulation of morph targets for complex models. The forefront of hardware assisted character animation is for the moment clearly moving away from a vertex morph based approach.

This leaves matrix deformation systems (often called bones-based animation or skinning). In this technique, vertices are not manipulated directly. Individual vertices are related by proportional weights to a one or a series of control vertices (CVs) that are manipulated to animate the object. These control vertices are used as mathematical transformation matrices [**37**]. They can be translated, rotated, and scaled. Any transformation applied to these matrices is applied to the vertices themselves via the weight factors. This allows for a great deal of control with a limited amount of input. However, the creation of these weight factors is difficult. Particularly in the regions of the face where a great deal of local control is required to achieve desired expressions. Matrix deformations are much more widely accepted as a technique for body animation. However, several benefits lead us to believe that matrix deformation will play an important role in facial animation.

First, the memory problems associated with morph-based animation do not exist. The smaller subset of controls that are actually animated in a matrix deformation system make this an attractive choice. Likewise, the technique is much more in line with innovations in graphics hardware. Accelerated transformation of vertices is becoming a standard feature on consumer graphics hardware and the extension to support full matrix deformation is already in progress. A further benefit to matrix deformation is it is quite possible to have a direct mapping between performance animation data and the animation system. Simply by having a one-to-one correspondence between animation matrices and performance data points, performance data can be used directly in many cases.

On the down side, matrix deformation characters are much more difficult to create. The vertex weighting is complicated and time-consuming though only done once. Visual effects companies often have staff dedicated completely to the issue of character setup, which includes this process. It would be much less time consuming if the animator could simply create the desired morph targets and these matrix deformation setups could be created algorithmically. However, this is an open and difficult problem that we are currently researching.

In practice, a hybrid approach to animation of faces that utilizes both “bones” (a.k.a. clusters, lattices, muscle effectors) based matrix deformation and morphing techniques looks to be the most promising approach.

Body: There has been great success using the methods discussed earlier to acquire the motions and interactive blending to create the final realistic motion. Many moves can be accessed by run-time game engines and blended together on the fly in response to player input to create an entire response sequence.

Interactive control of animation: With facial animation, the same techniques discussed above can be used for the audio/lip sync portion of the CG character performance and facial expressions can now be triggered and blended with the audio-driven character response in real-time. However, as it may not be practical for the animator to completely choreograph the lip-sync for every possible dialog and emotion sequence, extra care must be given to the animation setup. This is particularly true of internet-based entertainment and social applications where the character can engage in entirely unique dialog sequences via text-to-speech or voice transmission technology. In this situation, the animator will not have the chance to adjust the data produced algorithmically by the animation subsystem.

With body animation, a library of many moves can be created and blended together interactively depending on user and environmental inputs. Again, the need for interactivity dictates that the creation of unique animation for the variety of situations a user may encounter is cost prohibitive. Therefore, great care must be taken to insure that the library of actions is very robust and versatile to provide a great palette of base actions that can be blended to handle unique situations. Dynamic simulation of character motion can provide yet another input into the animation stream. However, difficulties in the creation of controllers needed to generate these behaviors have led to their use in only extremely limited situations such as specific athletic activities [**38**]. While this may prove to be a flexible and useful method for generation of unique and physically realistic animation, for the time being, dynamic simulation is of very little use in the creation of believable character animation.

3.1.5 Character related Camera moves

Camera control in most of today's games is only authored for pre-scripted sequences of action. There is limited interactive control in run-time engines in today's released games. The asset construction pipeline tools are quite adequate to preview and author these camera animations.

3.1.6 Character Lighting

There is basic support for simple lighting in today's run-time engines. Since the lights are simply point light sources, authoring is limited to placement of lights within the 3D scene. Some games use real-time colored lights so light properties such as color and falloff can also be defined.

3.2 Run Time Engines - Manage, Create, Playback, Respond, Interactively in Today's Shipping Games

3.2.1 Standard model/texture data implementations

Model databases for run-time access: Models are still mainly created using traditional polygon animation techniques. Each vertex and polygon is in the model is crafted by hand, in the same fashion as real-time vehicle simulation applications. However, the increase in performance of 3D graphics hardware for real-time display has greatly increased the polygon counts possible for animation models. In order to support a variety of target platforms as well as large scale interactive environments, techniques for level of detail management has become a prominent aspect to real-time model creation. Continuous LOD display systems have become a prominent part of run-time game engines. These systems use a mathematic metric to automatically reduce the polygon count in a model into a continuous set of discrete models. This has largely eliminated the need for the creation of discrete level of detail models. However, work must be done to insure that these automatic decimation algorithms can be artistically controlled in order to insure a visually pleasing series of models.

On certain set-top-box platforms, such as the Playstation 2, memory is at a great premium. For this platform, a large polygon count model may be too memory intensive for the system in some circumstances. To combat this problem, higher level model primitives, such as subdivision surfaces, have allowed designers to create a model that can scale to a great deal of detail without the need for a large memory footprint. This is effectively a form of mesh compression that uses an algorithmic method to create a detailed model from a low-detail base mesh.

Texture databases for run-time access: The memory required for detailed texture maps is of great concern for current productions. PC technologies such as AGP, as well as the decreasing cost of video memory, have enabled consumer hardware to gain access to more and more texture storage space. However, the expectations for visual quality have increased even faster. Use of multiple layers of textures and unique textures per surface for effects such as global lighting have caused a great increase in the texture data needed for cutting-edge interactive entertainment. Combine the needs for sophisticated environments and unique and highly detailed characters; the texture memory issue is a major one.

On the console side of things, the memory situation has not improved much at all. Price restrictions have caused console makers to keep memory severely limited even while CPU and raw graphics performance has increased. Particularly on systems such as the Sony Playstation 2 and Sega Dreamcast, where texture memory is at a premium.

To combat this problem, texture compression [**32**] has become a standard feature on emerging consumer graphics hardware. Typically, these hardware compression algorithms yield up to 8 to 1 texture savings. However, for complicated scenes, even this may not be enough. The visual effects industry has long been aware of the expensive nature of texture maps and has had great success with the use of algorithmic routines to generate color data. Procedures for generating textures such as brick, clouds, water, marble, and many other surfaces can be achieved through the creation and manipulation of simple fractal noise, waveforms, and other mathematic

structures. This form of procedural elements was used in the foundation of the programmable shader language, Renderman, which is used a great deal for visual effects production [**39**].

The use of procedural textures seems to be a huge win for both consoles and PCs as detailed textures that would need to be very large could simply be replaced with small algorithms. To some extent this can currently be done in software and has been used in a very limited way in the game *Unreal* for the generation of water and lava. However, to truly benefit from procedural textures, the graphics hardware would need to be capable of generating the procedural texture as needed during the rasterization phase. This implies some form of sophisticated shader language imbedded within the graphics hardware along the lines of Renderman. It is clear that this is a major step for consumer graphics hardware and something we will likely see gradually implemented. Consumer hardware currently in development is taking small steps in this direction and researchers are exploring how these limited capabilities can be increased to achieve Renderman-style quality [**40**].

3.2.2 Standard Character animation implementations

Hardware T&L and Graphics Processing Units: We have recently seen the emergence of transformation and lighting hardware components on consumer level graphics accelerators. This has greatly improved the possibilities for character animation. Until recently, transformation and lighting, as well as the animation techniques such as morphing and skeletal deformation, had to be handled by the general CPU. By relying on this shared resource, only a small percentage of animation effects could be used without effecting overall game performance. When the graphic hardware handles this burden, the animation system has the ability to perform many more effects.

Vertex morphing applied to both face and body animation: Vertex morphing techniques are a very useful animation tool for efficiently achieving fine detailed animation. Vertex morph animations for facial expressions and hand poses are easy to create and provide minimal processing at run-time. Typically, a single vertex morph target involves moving a very small subset of the vertices in a base mesh. The vertex morphing layer can provide the input to the skeletal animation layer, thus providing a very flexible animation system. However, as we discussed in the character animation section above, this flexibility is costly in terms of memory and hardware acceleration compatibility. Therefore, the vertex morphing layer should be used sparingly.

Simple hierarchical Character Body animation: Simple hierarchical animation of characters goes a great ways toward creating believable characters. However, creating the illusion of life necessary for truly realistic characters requires more sophisticated techniques. According to John Lasseter [**41**], one of the chief advantages offered by computer animation is the ability to combine techniques in layers to achieve more realistic and complex results.

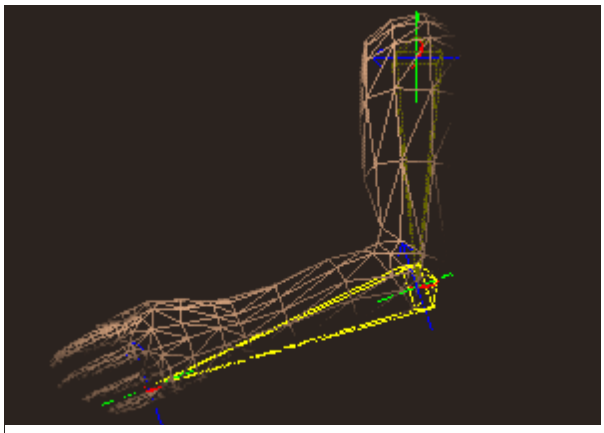

Figure 3 - Single Skin Mesh

Layering: This idea of layering can finally be applied to real-time character animation for the creation of more realistic characters. A skeletal animation system is comprised of a hierarchical structure of bones to provide the base animation layer for the character. The bones are attached to a mesh "skin" [Figure 3] by vertex weight assignments that relate each vertex to one bone or a set of bones in the system. These bones are then kinematically animated to provide the motion. Current computer graphics hardware accelerates this layer through use of the transformation and lighting circuitry (T&L).

Matrix Deformation animation layer: This matrix deformation layer provides a very flexible basis for a character animation system. The underlying structure of the skeleton can be modified on-the-fly to create unique procedural animations such as inverse kinematics. Kinematic animations can also be blended and then applied to the skeleton creating unique derivative animations.

However, fine details in a skeletal deformation system are difficult to achieve. Frame-by-frame animation and rendering systems depend on the artist's ability to create the nuances which breath life into these characters. To create detailed articulation such as for fingers or facial expressions, a great deal of bones must be used. This both increases the production time required for creating these characters, as well as increases the processor time needed to render the character, thus reducing run-time performance. Additionally, relating to control, in order to

create the desired realism this new hardware affords us, the "artist" is replaced algorithmically. The interactively of the game engines then can create the desired realism due to unpredictable nature of user input.

3.2.3 Environment and character interactions

Collision detection and limited IK have been implemented in current games. Lacking these features, characters can tend to look disconnected from the environment. However, characters and their weapons or vehicles modifying their environment and terrain following has seen more prevalent implementations in the latest games. Smarter collision detection methods have found their way into such games. [37]

3.2.4 Camera Control

Camera control in interactive applications is currently very limited. In many action games, a first person viewpoint camera is employed. However in stories where the character interaction is important, this is not an effective method for conveying story; therefore, first-person cameras are generally not used. Many applications have fixed cameras that are positioned by the game creator, such as Capcom's *Resident Evil* or Lucas Art's *Grim Fandango*. While this allows the developers to insure the greatest control over what the player sees, it does not always give the best feeling of immersion. When more immersion and fluid camera control is desired, developers tend to use tethered cameras that are attached to the character at a certain distance. The angle of view and position of the camera around the character can often be controlled by the player. However as the position of the camera is controlled by the player, they will often miss elements that the designers use to convey story.

Many developers are experimenting with hybrid approaches to these techniques. The crucial trade-off is to insure the most control for the player while enabling the designer to be sure that the story is effectively told. This is particularly important in conveying sections of dialog. Often interactive projects steal control during these "cut scene" sequences, as in *Tomb Raider*. However, this loss of control can be frustrating for the player.

3.2.5 Standard rendering implementations for facial skin, facial hair, eyes, body skin, clothes

Color and Texturing: Interactive characters currently are largely rendered with single texture maps on each surface. These maps provide details that do not exist in the model itself. Details such as clothing material, facial hair, wrinkles, and eye color are largely dominated by simple, though often detailed, textures. In some cases where character polygon counts are low, textures can take the place of mouth and eye geometry. However, this is noticeably inferior and should be avoided where possible unless desired for stylistic reasons, such as cartoon rendering.

Hair is typically also represented with simple texturing. In most interactive applications, character hair is modeled as part of the head in a form of colored skull cap. You will often see characters with their hair pulled back or combed flat to allow this limitation to be stylistically consistent. In some more sophisticated examples, hair is modeled loose from the head and rendering using transparency. This clearly provides a much more realistic look.

Advanced features such as bump and environment mapping are largely unused by interactive applications and are reserved for environmental effects. This is the result of these sorts of features being added to a project as an afterthought, in support of a specific hardware feature. While multi-texture techniques are common in environmental situations, they remain unused in character rendering situations.

Occasionally texture switching or modification is used to simulate a variety of effects: facial animation, damage or impacts on characters, or weather effects.

Lighting discussion: Character lighting is generally handled with basic Gouraud shading, where used at all. The shading color is calculated per vertex as a function of the key lighting in the scene (usually one omni-directional light) and the surface normal. The shading value is interpolated across the polygon and is modulated with the character texture to achieve the final pixel color. While at least providing some dynamic lighting properties, this technique clearly has many drawbacks. Gouraud shading is not a very realistic shading model for the display of skin. Skin has a very complicated reflective response and this simple form of shading tends to make the character skin look very plastic. Per-pixel lighting calculations would greatly improve this look.

Simple vertex lighting calculations also cannot handle self-shadowing, where the model casts a shadow on itself such as in the nose shadowing part of the face. In order to achieve self-shadowing, a more complicated lighting technique, such as shadow volumes, would need to be used. This clearly is not done in current interactive applications.

4 New Hardware Capabilities

4.1 New currently available graphics hardware

4.1.1 New Consoles

The game consoles are developing into sophisticated computer graphics platforms. For less then $400 you get graphics computer power greater than the high-end graphics workstations of a few years ago. However, while capable of rendering many polygons, the limitation of graphics memory on the consoles somewhat limits the number of effects that can be achieved.

The Sega Dreamcast is the first of the current generation of game consoles. It has a relatively fast CPU but limited graphic memory and a graphics chip based on PC technology of a couple of years ago. While capable of delivering some very compelling images, it is not a high-end rendering force.

The Sony Playstation 2 on the other hand, is a very powerful multiple processor computer. Through the use of two vector processing units, it is capable of displaying 66 million triangles per second. However, texture memory is very limited. Clearly, developers will make extensive use of high polygon count models in place of large texture rendering tricks.

Microsoft's X-Box is the newest player in the console market. They have designed the most sophisticated console to date. It combines a fast CPU with a graphics chip designed by NVidia that is even more powerful than anything available on the PC today. This system will be capable of rendering millions of multi-textured polygons in a single pass, making use of programmable vertex and pixel shaders. While not available to developers yet, they can begin working with Microsoft's DirectX 8 to get an indication of what will be possible on this new platform.

4.1.2 New Graphics Cards for PCs

Hardware T&L: The current generation of consumer graphics hardware, such as the NVidia GeForce 256, support the full hardware acceleration of model transformation and lighting. This has allowed for greater polygon counts in character models. The addition of hardware acceleration of matrix deformation allows the limited acceleration of model deformation as well. However, in practice, the limitations of this generation of graphics card make this impractical. Fortunately, very flexible matrix deformation can be achieved in software and the final model can then be submitted to the hardware for final transformation and lighting.

Full Anti-aliasing: Several of the current generation of graphics cards support anti-aliasing. This technology automatically attempts to eliminate aliasing problems in rendered images. Anti-aliasing can greatly improve the look of environments, as well as characters, by eliminating the "jaggies" seen at high contrast edges between objects. This comes at a cost of texture fill-rate that in some cases can slow down the rendering speed. However, this is fortunately a run-time decision that the end-user can make allowing the user to decide between better quality or greater speed. As fill-rate on graphics hardware continues to improve, this penalty will become much less of an issue.

Multi-Texture Techniques: Modern consumer graphics hardware is capable of rendering at least two textures per-pixel in a single rendering pass. This has enabled a great variety of new effects such as modulated global illumination such as shadow maps and bump and environment mapping. These techniques have been used greatly for environmental rendering in modern interactive projects. However, they are largely unused in character rendering. For characters, this capability can go a great ways towards achieving more realistic and compelling rendering.

5 New Opportunities with New Implementations now possible

To evaluate the new opportunities which new graphics hardware affords us let us reverse the presentation we earlier made relating to current state-of-the-art. By examining the new implementations first, we can then build up a new set of requirements for the new authoring production pipeline required to support these. While these techniques are possible on currently available consumer graphics hardware, they have not been widely applied in current interactive applications. This may be largely due to the need to generate content for the majority of the available market, as well as the additional artistic and technical resources needed to generate the required assets.

5.1 Run Time Engines - Manage, Create, Playback, Respond, Interactively

It is easy enough to create a cinematic cut scene that follows traditional filmmaking techniques. However, this completely pulls the player out of the interactive experience. Modern 3D game engines can create cinematic

sequences within the game. However, most of the time, these sequences are completely scripted using traditional animation techniques. The sequence fires when the player either enters a location, pulls a lever, or some other trigger mechanism. Once started, the sequence follows a deterministic path. The game designer now has a choice to make. The first option is to control the camera shots to show the drama, freezing the interactivity. Secondly, the player can maintain complete camera control and try to catch the action. This can create a great sense of "What's going on up there?" as you rush to find a viewpoint. The game *Half-Life* from Valve Software used this technique very effectively. However, crucial information cannot be delivered in this manner as the player may miss it by spending too long studying the magnificent architecture.

5.1.1 New Implementations Overview

- A new character rendering pipeline is now possible and major parts of it can be accelerated by consumer graphics hardware. Fine animation of a character mesh can be handled by vertex morphing techniques. The results of this vertex animation are fed into a matrix deformation system. Major parts of this matrix deformation system can be accelerated on consumer graphics hardware that supports accelerated transformations. Fine detail such as skin wrinkles cannot be simulated with mesh deformation techniques. Therefore, texture manipulation should be included as the final step in the pipeline to handle this form of fine detail.

- New hardware has enabled interactive applications to using algorithms such as IK as a character animation modifier. These modifications correct the results of primary motions to allow for things like correct feet placement and grasping of objects.

- For secondary effects, simulated dynamics can be tied to the character. This is useful for things like character hair and clothing.

- Matrix deformation hardware can be used in non-standard ways to allow the artists more natural animation control. Algorithms such as Free-Form Deformation can be encoded into the animation system to expand the creative pallete for character soft masses.

- While rendering characters, the combination of new vertex shaders and pixel shaders enables new effects. For example, you can now do cartoon-style rendering in a single rendering pass using this hardware. Complex mathematical operations can be encoded within texture maps to enable sophisticated rendered looks. These steps bring us closer to the idea of hardware accelerated real-time Renderman. In every case, the full hardware capability should be used for characters as well as environments. A single character can be rendered with lighting, texture, bump, and specular elements plus environmental reflections when appropriate.

- Camera Control is not specifically enabled with new hardware, however sufficient hardware is available to take cinematography seriously.

5.1.2 Facial Character Animation Implementations

Neither matrix deformation nor vertex morphing alone is sufficient for a flexible and efficient run-time engine. Both systems working together provide the optimal flexibility and control.

Figure 4 - Geppetto 60 fps real-time model using bones for hair deformation and vertex morphing and blending for facial expressions

While matrix deformation will be fully accelerated on the next generation of consumer graphics hardware, the technique is not particularly well suited for facial animation. The number of bones needed to animate a face properly is high, as is the degree of interaction between the bones. Both of these facts make it difficult to implement hardware accelerated facial animation with matrix deformation on these cards. The graphics cards limit the number of matrices that can be manipulated, as well as restrict the number of matrices that can influence a specific vertex. Further, the hardware implementation makes it extremely difficult to have the vertex normals animate with the face if the deformation matrices are translated instead of rotated. Translation of facial bones is a very common method for animation of character faces, so it is an extremely difficult issue to deal with the repairing of vertex normals. For some facial actions, particularly the movement of the jaw, which is governed largely by a rotational bone, matrix deformation works well. However, for detailed facial animation, vertex morphing techniques are still a superior method

on these cards. QuantumWorks has effectively used this technique on its PC-based real-time game engine demos (derived from the Geppetto system) of linear superposition blending of morph-based facial animation using 11 to 20 morph targets depending if a facial asymmetry is desired. [Figure 4] and [**20**]

Interpolated vertex morphing allows for the fine control of individual vertices without the problems of matrix deformation. While it is currently necessary to implement this animation layer in software on the main CPU, extremely efficient data structures can be used to minimize the impact.

However, even vertex morphing techniques are insufficient for extreme facial details such as facial wrinkles and creasing. Even detailed high polygon models are insufficient to handle this task. When facial details such as wrinkles are desired, texture mapping techniques are much more effective.

Figure 5 - Matrix hierarchy driven deformed character

5.1.3 Body Character Animation Implementations

Hardware accelerated matrix deformation is particularly well suited to character body animation [Figure 5]. The number of matrix interactions are naturally limited and the matrices are largely rotated to animate the various body parts. This eliminates the problem of vertex normal recalculation seen in the facial animation case.

Matrices can also be used to create secondary effects such as muscle bulges and chest heaves. By imbedding a matrix within the arm and then animating it by scaling up and down, the vertices can be made to look like the underlying muscles are moving. This effect can be automated by creating an expression, which relates the rotation of a matrix, for instance the lower arm, to the scale of the underlying muscle. Likewise, a periodic scale expression can be used to simulate the chest heaves caused by having the character breath or causing the character to automatically blink periodically. These types of expressions are used commonly in 3D animation packages, however they are largely unused in real-time applications and can greatly increase the realism of the character [Figure 6] .

Figure 6 - Procedural Animation of Eye Blink with periodic function

Having a matrix deformation system within the character allow for interactive animation as well. Instead of relying completely on pre-created animations, these animations can be combined with interactive techniques such as inverse kinematics that allow the character to interact better with her environment.

Another important part of body animation for creating compelling characters is the use of other secondary animations. A secondary animation is an animation that is not used for the current specific action of the character, but is used to convey realism. For example, having the hair of the character swing around as the character turns

her head or the cloth of a character swish while she walks are both examples of secondary animation. These effects can be animated in the same traditional way as the main action or can be handled in an automatic procedural fashion.

Effects such as cloth and hair animation can be handled in a procedural fashion through the use of dynamic simulation [**20**]. The vertices of the model can be connected through a mass and spring system which is allowed to react in a physically modeled manner. Gravity can be applied and collision boundaries can be created to keep the cloth from penetrating the character model. When applied to vertices in the hair, the character's hair will bounce around in a realistic manner when she walks around. The subtle effect can greatly improve character believability.

5.1.4 Free Form Deformations (FFDs) – Animation and Interaction Applications

In their compelling work "The Illusion of Life," Thomas and Johnston outlined the use of squash and stretch, exaggeration, follow through, and overlapping action as key components for character animation [**42**]. While the combination of the two animation layers described above provide a great deal of control, they are not well suited to creating characters and regions of characters that squash and stretch. Animators need a more intuitive parameterization of these properties in order to achieve fluid results.

Figure 7 - 4x4 FFD lattice used to easily animate cloth skirt

Sederberg and Parry introduced the use of Free Form Deformations (FFDs) as an efficient method for animating soft bodies via a structural hyperpatch [**43**]. By abstracting the control surface from the surface of the animated body, the deformation controls can be manipulated without regard to the model itself. This technique has been used to successfully model semi-elastic surfaces.

The FFD technique can be layered on top of our existing animation system [a sample shown in Figure 7]. However, the FFD method is computationally intensive, requiring N^3 evaluations of a parametric function of degree N (typically cubic or 3) at every vertex. This type of computation is not likely to be accelerated in graphics hardware leaving this operation as a CPU task.

It is possible to formulate the FFD technique in a way that could take advantage of graphic hardware. In an FFD lattice, a series of equations, called the Bezier basis functions, relate the vertices in the base mesh with the FFD control structure.
For a degree 3 FFD, these functions take the form:

$B_0(u) = (1 - u)^3$
$B_1(u) = 3u(1 - u)^2$
$B_2(u) = 3u^2(1-u)$
$B_3(u) = u^3$

The basis functions serve the same role as the vertex weights in a skeletal animation system. We can think of the control points in the FFD lattice as a set of bones. The above functions are then used to create vertex weights relating each of these control points (bones) to the vertices in the base mesh. In this way, the vertex weights implicitly contain the Bezier basis function. Once these weights are computed and the control points positioned, the standard skeletal animation system is used to process the results. This technique is now fully hardware accelerated in the same way as the rest of the skeletal deformation system. By strategic positioning of an FFD lattice, it can be applied, for example, to an upper arm such that the muscles can bulge. Additionally, this technique can be used to accelerate effects that would normally require per vertex dynamic calculations such as cloth and hair animation.

5.1.5 Environment and character interactions

While character animation has achieved outstanding levels of realism in today's game titles, these characters are largely ignorant of the environment around them. Even the latest current state-of-art *Onimusha* game on Sony PS-2 shows beautiful characters and blended body animation moves; however, they are uncoupled to the ground plane and this breaks the realism. Characters have very believable walk animations that work quite effectively on flat terrain, however, on uneven terrain, the limitations of these animations is evident.

The advances in computer graphics hardware, such as accelerated transformation and lighting, have freed a great deal of processor time to deal with these issues. Real-time inverse kinematics can be used to guide foot placement over somewhat uneven terrain. It can be also used to allow a character to reach for an object without needing to resort to a fixed animation, as seen in games like *Tomb Raider*. Likewise, inverse kinematics can be used to allow two characters to interact, for actions like shaking hands, without resorting to very staged animations. We have found these types of situations ideally suited for blending inverse kinematics with artistic animation. Using inverse kinematic algorithms to drive animations directly is much less compelling. The animations generated are stiff and often not very realistic. However, when an artistic animation is blended with an IK solution to fix-up the final position, the results are quite realistic.

5.1.6 Rendering implementations for facial skin, facial hair

While the use of multiple textures to create more realistic environments is quite common in current interactive titles, these same technologies have been largely unused for characters. Consumer graphics cards commonly support simultaneous rendering of two textures and an iterated color. The latest graphics card from ATI, the Radeon, will support three simultaneous textures. This effectively allows for the blending of a texture, bump map, and specular texture in a single rendering pass. These extra texture passes can be used to greatly improve the look of the characters [**44**].

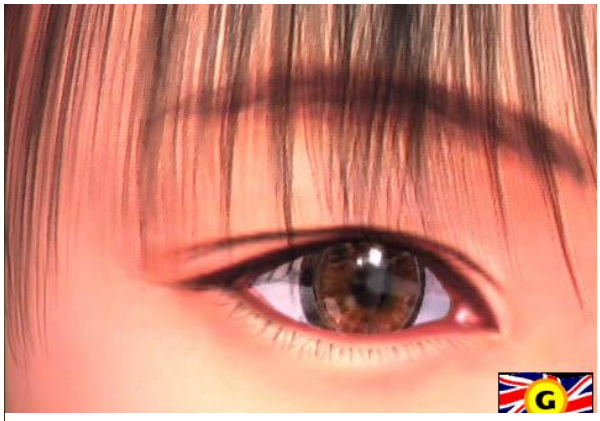

Figure 8 - Reflection Map on Shenmue Character Eye

For example, bump maps are used for many environmental effects in current games such as water ripples and rough surfaces. However, this same effect can be used to create wrinkles, facial creases, skin pores, clothing texture, and facial stubble. Bump maps for characters are widely used in visual effects work and they certainly could greatly increase the realism of real-time characters.

The use of specular highlights is another useful technique for rendering shiny surfaces [Figure 8]. Facial skin has a specular component as a result of the reflection of moisture in the skin. When a model is lit with a simple Gouraud shading model, the face can look plastic and unrealistic. True specular highlights require the interpolation of the surface normal across the triangle. This is not possible on consumer graphics hardware. However, the current leading graphics cards such as the NVidia GeForce 256 allow an artist to create a normal map that encodes the surface normals in a texture. These normal maps must be generated from the model data in an offline tool. This map is then used to compute a per-pixel shading value by taking the dot product of this interpolated “normal” and the light position. This can give reasonably accurate specular highlights.

These same normal maps can be used with a “reflection map” which represents the world around the object. The graphics card can use the surface normal to calculate where the view vector would reflect into the map. Reflection maps like this are very useful for things such as character eyes or shiny clothing that would have some reflection, such as armor for a knight.

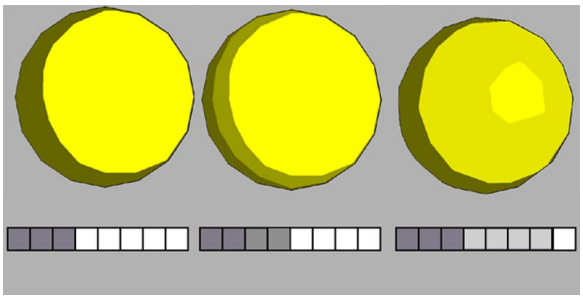

Figure 9 - Cartoon Shade look-up Table

This same multi-texture hardware can be used to create non-realistic real-time renderings such as cartoon shading. In the development of story concepts, game project presentations and other applications where our intent is to not distract the audience with the “realistic look” of the presentation, toon-style or artist sketch-drawing-style 3D rendered output is quite useful [**45**].
This is accomplished through novel use of texture map hardware. Textures are generally used to encode actual surface color data. However, a texture map can also be used to encode any form of mathematical function. A non-linear light falloff function can be encoded in a 1D texture map [Figure 9]. This map can be used to generate a cartoon shaded look by modulating the value in this lookup table with the surface color [**32**]. This use of texture maps to encode arbitrary mathematic functions is a very powerful idea that can enable a great deal of effects through multi-pass rendering. In fact it has been shown that by encoding functions in texture maps, a shading language as complicated as RenderMan could potentially be developed for real-time applications [**23**]. NVIDIA, the developer of the GeForce 256 chipset for gaming

applications have also shown demonstrations and coding with great potential for usable applications in this area of toon-shading of our 3D characters, animations, props, scenes and environments [**46**, **47**, **48**, **49**, **50**].

The current generation of graphics hardware allows for a great deal of rendering power. We have examined how multi-texture techniques can be used to create more realistic characters. However, we generally used the multi-texture rendering in fairly standard ways. The graphics hardware allows the textures to be combined using fairly flexible methods. For example, in addition to simply multiplying the textures together to get a final output, the value of each pixel can be added together causing a saturation effect. This is very useful for things like surfaces that emit light.

5.1.7 Camera Control Implementations

The ideal solution to bad camera control methods would be to present the drama to the character as much as possible while allowing the player full control. It is clear that the camera system in a story-driven game needs to be a crucial character. It needs to be aware of what the player is doing, what is going on in the world. The camera needs to be "intelligent" enough to find the best viewpoint to show the player what is going on without ruining the dramatic element. This camera needs to understand the principals of cinematography and must use this knowledge to effectively display the action to the viewer. Hollywood production has provided us with a rich base of knowledge in effective storytelling through cinematography. We build on the techniques of He, et al. [**51**] to create an intelligent camera system for effective display of interactive situations. The technique, termed "Implicit Interaction" uses input from the player to regulate the use of the expert camera system [**52**]. This allows us to show the interaction in a compelling way without removing control from the player.

5.2 Authoring production pipeline to support these new character animation features

In developing new techniques which will run interactively on today's gaming platforms, developers have always struggled with developing efficient authoring pipelines to allow artists to easily test their work on the target platform and to allow constant changes throughout the games development lifecycle.

The 3 main components (Asset Construction Pipeline, Game Engine Specific Creator/Editor, Run-time Game Engine) are still the preferred approach for new implementations. One innovation we would like to see more use of is a digital concept prototyping system used at the start of the Asset construction pipeline to allow ideas and cinematography concepts to be "played" with when it is "cheap" to do so in the game development lifecycle. One such system, called SpotMaster, has been developed to address this problem for the Advertising content creation industry. [**45**]

The Game Engine Specific Creator/Editor may be more integrated as a "plug-in" environment to the Asset construction pipeline or collection of integrated off-the-shelf general-purpose modeling/painting/animation/rendering tool suites.

We need to optimize the movement of art content to the game engine with a combination of layered tools and scripting interface systems. These systems need to tie and glue the process together with flexibility and without too much re-development overhead.

5.2.1 Model/texture data implementations for skeletal head/face and body components

Formal asset management is quite advantageous by allowing the artists to derive all needed model assets from high resolution base models. Specifically, mathematical descriptions, such as NURBS, can be used to create and store these high resolution, base models to facilitate the creation of many model resolutions that may be required.

Traditional authoring can support the new implementations we have suggested for creating the morph targets, for vertex morphing, and for creating the new maps for texture, bump and reflection.

5.2.2 Authoring the Animation

Traditional asset creation systems of today are quite adequate to support the modified character setups required for all of the new implementations we suggest, including: matrix deformation, FFD systems, IK for animation modifiers, interactive user/environment driven animation and automatic environment and character interaction.

For dynamic vertex movement some modification to general asset creation systems can be made to tag vertices which will be influenced by this dynamic system. Again the traditional systems are quite adequate to implement these features.

5.2.3 Developing the library - Motion capture, Performance animation and motion blending

One technique, which has been recently developed [**35**], shows great promise in developing accurate facial animation libraries. This technique developed as part of a system called the "Digital Cloning System," actually tied the specific character setup into the motion capture system in a closed loop iterative batch type process. This method uses a suite of image processing techniques to not only initially analyze the human performer, as is done traditionally, but uses the image processing tools to analyze the rendered output for every frame after the system automatically manipulates the facial game character setup. It does not matter how different the human actor captured is from the game character to be animated. When the system determines that the game character face matches the human performer face expression for a specific frame, within tolerance, the system then advances to the next frame. With this automated system, the human animator artist, who must build up the library of reference facial expressions in limited time, now can spend their time on adjusting the database rather than manually matching MoCap facial data to specific characters.

6 Future Work with Emerging Technology

Consumer graphics hardware is currently being developed that will be able to perform sophisticated effects through programmable vertex processing circuits. In addition to providing more flexibility and speed to skeletal deformation and morphing techniques, this new functionality creates new opportunities beyond standard hardware implementations of software techniques.

An aspect of character rendering that we would like to explore and is difficult to deal with currently is self-shadowing. Most real-time lighting techniques assume the light is not occluded at each vertex. However, this is not very realistic. Self-shadows are a large part of facial features. Simply painting the shadows into the surface textures, as is often done in interactive applications, is not terribly realistic, as it ignores the changing direction of the light. Techniques like shadow volumes can solve this problem. However, shadow volumes require an additional rendering pass from the point-of-view of the light and this greatly slows rendering time. It is possible that in the future, this will not be a severe performance penalty. For now, though, self-shadowing remains an unsolved issue for real-time character animation.

Graphics hardware developers are preparing themselves for the next big change in hardware rendering. Programmable vertex and pixel shaders are set to appear this year on consumer graphics hardware. Microsoft has outlined, through their DX 8 specification, a method of exposing this functionality to developers. Developers will be able to create a custom program that controls the vertex transformation and pixel rendering pipeline. This will enable users to go beyond simple multi-pass texture mapping techniques. Some new work in these directions being introduced this year is coming from Stanford University, SGI and other institutions [**53**, **40**]

One of the areas where this greater flexibility will greatly improve the look of renders is in the rendering of surfaces that have anisotropic reflectance properties. Simple reflectance models such as Gouraud and Phong are not adequate for rendering objects with materials such as brushed metal. For objects like these, the use of bi-directional reflectance distribution functions (BRDF) will be used to create more realistic renderings of complex surfaces.

The use of vertex and pixel shaders will also allow the full hardware acceleration of non-photorealistic rendering, such as the cartoon shading we described above. The calculation of the light vector can be done with a vertex shader while the actual modulation and blending of textures is done with a pixel shader. Using this technique with DX8 capable hardware, a two-texture sketch shaded object could be rendered in a single rendering pass.

Additionally, the vertex shader system will allow the full matrix deformation system to be completely hardware accelerated without the severe limitation seen on current cards like the NVidia Geforce 256.

While the creation of programmable shaders for graphics hardware could complicate developers lives, the added flexibility and creativity possible will allow for much more compelling characters. This is the next logical step towards RenderMan-quality real-time animated content.

In the past, interactive developers sought to exploit every possible effect and mode on rendering hardware. Mode-X on PC VGA hardware and the hi-res rendering modes on the SNES system are examples of this. However, current consumer graphics hardware is changing so rapidly, many of these hidden and unexpected features are not always exploited. Developers seem content to wait till the features are improved on the next generation of card. For example, Microsoft's DirectX 7 included support for hardware-accelerated vertex blending. However, it was severely limited by allowing only two matrices to deform a triangle. This was completely inadequate for character animation and seemed to be completely unused by game developers who are content to stick with software

methods until the much improved DX 8 hardware appears. As long as graphics hardware continues to improve greatly every six months, we expect this trend to continue.

References

[1] Tice, S.E. and M.J. Fusco, “Digital Content Creation Systems” – Real-time Performance Animation based Production Pipelines, Included in the SIGGRAPH Course #1Notes - Character Motion Systems, SIGGRAPH 1993

[2] Williams, Lance, “Performance-driven facial Animation”. In Forest Baskett, editor, Computer Graphics (SIGGRAPH ’90 Proceedings), volume 24, pages 235-242, August 1990

[3] Ventriloquist System from Lips Inc. Cary, NC lipsinc.com

[4] Eberly, Dave, Numerical Design Limited, “Metrics for Level of Detail”, In Game Developers Conference 2000 Proceedings, March 8-12, 2000, pp. 173-190

[5] Goldfinger, Eliot, “Human Anatomy for Artists,” Oxford University Press, New York, New York, 1991

[6] Waters, Keith, “A Muscle Model for Animating 3D Facial Expression”, In Maureen C. Stone, editor, Computer Graphics (SIGGRAPH ’87 Proceedings) volume 21, pg. 17-24, July 1987.]

[7] Lander, J, “Flex Your Facial Animation Muscles”, Graphic Content Column in Game Developer Magazine, July 1999, pg. 12-18

[8] Volker Blanz and Thomas Vetter, Max-Planck-Institute for Computer Science. "A Morphable Model For The Synthesis of 3D Faces". In SIGGRAPH '99 Conference Proceedings, pg. 187-194. ACM SIGGRAPH, 1999.

[9] Matthew Brand. "Voice Puppetry", In SIGGRAPH '99 Conference Proceedings, pg. 21-28. ACM SIGGRAPH, 1999.

[10] Intel 3D Software Toolkits, www.intel.com

[11] Lander, J, “Over My Dead, Polygonal Body”, Skeletal Animation, Graphic Content Column of Game Developer Magazine, October 1999, pg. 17-22 www.darwin3d.com/gdm1999.htm#gdm1099

[12] Lander, J., “Read My Lips: Facial Animation Techniques,” Game Developer, June 1999

[13] Lander, J, “Skin Them Bones,” Graphic Content Column of Game Developer Magazine, October 1999, May 1998

[14] Geppetto Performance Facial and Body Animation System, QuantumWorks Corporation, www.quantumworks.com

[15] “Granny 3D Animation System”, RAD GameTools, www.radgametools.com

[16] Parke, F.I, Waters, K, “Computer Facial Animation”, A.K. Peters, Ltd. ISBN 1-56881-014-8, 1996

[17] Badler, N. I, Barsky, B. A., and Zeltzer, D, “Making Them Move”, Morgan Kaufman. ISBN 1-55860-106-6, 1991

[18] Jehee Lee and Sung Yong Shin, Computer Science Department Korea Advanced Institute of Science and Technology “A Hierarchical Approach for Human-like Figures”, In SIGGRAPH '99 Conference Proceedings, pg. 39-48 . ACM SIGGRAPH, 1999.

[19] Zoran Popovic and Andrew Witkin, Computer Science Department, Carnegie Mellon University, “Physically Based Motion Transformation”, In SIGGRAPH '99 Conference Proceedings, pg. 11-20. ACM SIGGRAPH, 1999.

[20] Badler, Norman, et. al, “Simulating Humans,” Oxford University Press, 1992 www.cis.upenn.edu/~badler/home.html

[21] Welman, Chris, “Inverse Kinematics and Geometric Contraints for Articulated Figure Manipulation”, Masters Thesis, Simon Fraser University, 1993. fas.sfu.ca/cs/research/groups/GMRL/pubs/

[22] Craig, John J., “Introduction to Robotics: Mechanics and Control”, Second Edition, Addison-Wesley, 1989.

[23] Lander, J., “Using Quaternions for Animation Part 2: Interpolation,” Graphic Content Column in Game Developer Magazine, April 1998

[24] Lander, J., "Oh My God, I Inverted Kinematics," Graphic Content Column in Game Developer Magazine, September 1998

[25] Lander, J., "Making Kinematics More Flexible," Graphic Content Column in Game Developer Magazine, November 1998

[26] Lander, J, "Procedural Animation - The Six Million Dollar Dolphin – Virtual Bionic Characters", Graphic Content Column of Game Developer Magazine, July 2000, pg. 13-16

[27] Lander, J, "Hardware-accelerated Procedural Animation -To Deceive is to Enchant: Programmable Animation, Graphic Content Column of Game Developer Magazine, May 2000, pg. 15-20

[28] Hecker, Chris, "How to Simulate a Ponytail," Game Developer, March 2000

[29] Jones, Doug and . Pai Doug L. Dinesh K, University of British Columbia, "Accurate Real Time Deformable Objects - ArtDefo In SIGGRAPH '99 Conference Proceedings, pg. 65-72. ACM SIGGRAPH, 1999

[30] Lander, J, "In This Corner … The Crusher" - Hardware-accelerated Deforming Characters, Graphic Content Column in Game Developer Magazine, June 2000, pg. 15-20

[31] Lander, J, "Devil in the Blue-Faceted Dress: Real-Time Cloth Animation" Real-time Cloth (and can be used for hair) on Characters -, Graphic Content Column in Game Developer Magazine, May 1999, pg. 17-21 www.darwin3d.com/gdm1999.htm#gdm0599

[32] Cohen-Or, Daniel, Fleishman, Shachar and Mann, Yair, Tel Aviv University, WebGlide Ltd, "Deep Compression for Streaming Texture Intensive Animations", ". In SIGGRAPH '99 Conference Proceedings, pg. 261-267. ACM SIGGRAPH, 1999.

[33] Cabral, Brain, Olano, Marc, and Nemec, Philip, SGI, "Reflection Space Image Based Rendering" (implementable on graphics hardware), In SIGGRAPH '99 Conference Proceedings, pg. 165-170. ACM SIGGRAPH, 1999.

[34] Wolfgang Heidrich and Hans-Peter Seidel, Max-Planck-Institute for Computer Science. "Realistic, Hardware-accelerated Shading and Lighting". In SIGGRAPH '99 Conference Proceedings, pg. 171-178. ACM SIGGRAPH, 1999.

[35] Tackacs, B, Fromherz, T, Tice, S.E. and Metaxas D, "Digital Clones and Virtual Celebrities Facial Tracking, Gesture Recognition and Animation for the Movie Industry", ICCV'99 RATFG-RTS

[36] Guymon, Mel, "Talking Heads: Hierarchical Animation in Real-Time 3D Facial Animation – Artists Viewpoint", Artist's View column of Game Developer Magazine, July 1999, pg. 20-25

[37] Lander, J (2000), Using Technology to Create Believable 3D Characters, Course notes and slides, Game Developer's Conference 2000

[38] Hodgins, Jessica, et al., "Animating Human Athletics," Proceedings in Siggraph 1995, pp. 71-78.

[39] Uphill, Steve, "The RenderMan Companion," Addison Wesley, New York, 1990.

[40] Peercy, Mark, et al.,SGI, "Hardware Accelerated Programmable Shaders," Proceedings of Siggraph 2000, To Appear. http://www.cs.brown.edu/~tor/sig2000.html

[41] Lasseter, John, "Principles of Traditional Animation Applied to 3D Computer Animation," Computer Graphics, Vol 21. No. 4, 1987.

[42] Thomas & Johnston, "Disney Animation: The Illusion of Life," Abbeville Press, New York, 1984.

[43] Sederberg & Perry, "Free Form Deformations of Solid Geometric Models," Computer Graphics, Vol. 20, No. 4, 1986.

[44] Lander, J, "Hardware-accelerated Blending Textures - A Clean Start: washing Away the Millennium", Graphic Content Column of Game Developer Magazine, Dec. 1999, pg. 23-28 http://www.darwin3d.com/gdm1999.htm

[45] Tice, S.E., Lander, J, "SpotMaster – A Tool for quickly creating Advertising Commercial Prototypes suitable for focus group testing", QuantumWorks Corporation, September 1999 at Procter & Gamble presentation.

[**46**] Dietrich, Sim, Cartoon Rendering and Advanced Texture Features of the GeForce 256 Texture Matrix, Projective Textures, Cube Maps, Texture Coordinate Generation and DOTPRODUCT3 Texture Blending", NVIDIA Corporation, 2000

[**47**] Michael A. Kowalski, Ronen Barzel, Lee Markosian, J.D. Northrup, Lubomir Bourdev, Loring S. Holden, John F. Hughes, Brown University, Adobe Systems, Pixar, "Art-Based Rendering of Fur, Grass and Trees". In SIGGRAPH '99 Conference Proceedings, pg. 433-438. ACM SIGGRAPH, 1999.

[**48**] Rademacher, Paul, University of North Carolina at Chapel Hill, "View-Dependent Geometry". In SIGGRAPH '99 Conference Proceedings, pg. 439-446. ACM SIGGRAPH, 1999

[**49**] Lander, J, "Under the Shade of the Rendering Tree", Graphic Content Column of Game Developer Magazine, Feb 2000, pg. 17-21 www.darwin3d.com/gdm2000.htm

[**50**] Lander, J, "Shades of Disney: Opaquing a 3D World", Graphic Content Column of Game Developer Magazine, March 2000, pg. 15-20 www.darwin3d.com/gdm2000.htm

[**51**] He, L., Cohen, M., and Salesin, D., "The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing," Proceedings of SIGGRAPH, ACM SIGGRAPH, 1996, pp. 217-224.

[**52**] Lander, J , "Interactive Cinematography- Lights … Camera … Let's Have Some Action Already", Graphic Content Column of Game Developer Magazine, April 2000, pg. 15-20

[53] Pat Hanrahan, Bill Mark, Kekoa Proudfoot, Svetoslav Tzvetkov, Stanford University At other institutions: David Ebert, Wolfgang Heidrich, Philipp Slusallek, "Stanford Real-Time Programmable Shading Project" http:/]graphics.stanford.edu/projects/shading/