A Virtual Environment for Review and Annotation of Character Animation

Project Report

CMPS 615 – Virtual Reality, Spring 2008

Jan-Phillip Tiesel
University of Louisiana at Lafayette
jpt4246@cacs.louisiana.edu

ABSTRACT

This text is a description of a course project conducted between January and May 2005 summarizing related work and implementation details.

1. MOTIVATION

In the movie industry, the term *dailies* refers to the daily review of recently filmed footage by members of the production crew. In the production pipeline of a CG movie, dailies commonly refers to the process of reviewing and critiquing the work completed by different departments involved in the filmmaking process. They are used by the movie's director to refine and steer individual departments' efforts, ensure that the film remains *on-vision* and that the overall filmmaking process stays on schedule [7].

Typically, the daily work completed by the character animation department is given special attention and thorough review as the motion of a CG character is critical in the illusion of life that is potentially perceived by the viewer [8]. The review of an animation sequence generally consists of repetitively watching and analyzing a pre-rendered video sequence (usually created with a much simpler lighting and rendering model than the one used for final rendering) and providing feedback to the artist. Often, a lot of attention has to be given to details like facial expressions, proper application of animation principles [9], or synchronization of lip movements and respective voice track.

Today, fast graphics hardware and established algorithms allow for the rendering of high-quality animations in real time. Presenting virtual characters' animation sequences in a virtual environment is not only a viable option, but might provide more insight into the quality and believability of the artistic expression conveyed through the character's motion. Comparing the way that artistic character animation – created with a stationary viewer in mind – is perceived in a VR system might also result in more knowledge on how virtual reality can be used more appropriately as a medium used for storytelling. As character animation has become ubiquitous in the entertainment industry – ranging from video games to commercials and feature films – evaluating its potential for VR applications seems like a logical step to me in oder to take on the "exciting challenge [of][...] learning what to do with the medium" [12].

2. RELATED WORK

2.1 Animation in Virtual Environments

While character animation has been employed successfully in commercial VR systems for some time (e.g. Disney's Aladdin VR system 1996 [12]), little published research is available on animation topics specific to virtual reality. An extensive amount of publications exists on the underlying algorithms and techniques used for computer animation (see 2.2) but little insight has been gained on the differences and uniqueness of VR as a storytelling medium using expressive and compelling characters.

Approaches to populate virtual environments with emotional agents exist, but are still in its infancy [11]. Other systems employ virtual characters (avatars) for specific learning tasks or in order to induce a certain emotional state in the user (e.g. [16]). Generally speaking, most research does not take into account the importance of a character's physical motion even when avatars are supposed to convey complex behaviors and emotional states. While the sophisticated creation of animation that serves complex communicative purposes has been studied extensively for traditional media (e.g. [8], [17]), the interaction of VR and animation – both being highly appealing types of media – has not been studied in depth.

General questions that should be raised include:

- Does 'better' character animation performed in a VE lead to increased perception of presence / mental immersion?
- Can the review of character animation in VR be a valuable tool in the evaluation of believable character animation?

2.2 General character animation techniques

While many approaches exist for creating procedural animation or using dynamic simulations for producing interesting motion, most character animation for movies, video games, and other media is created by artists using keyframe animation systems. With this approach, the artist has full control over the character's configuration in every frame of an animation while tradeoffs and more time-efficient compromises can be made (e.g. automatic interpolation between frames, using keyframes to control an IK simulation).

Two main approaches have been used extensively in the past to give an animator control over the deformation of a mesh of vertices: *Vertex Blending* [4] and *Shape Interpolation* techniques. Shape interpolation algorithms take into account several base meshes of identical topology that were created by the artist. It exposes control to a number of blending parameters which let the artist control the influences of each base mesh on the target mesh at any given time. The target mesh is created repeatedly by interpolating vertex positions of the employed base meshes using the defined blending parameters. While the basic algorithm is quite simple and has several restrictions (e.g. identical mesh topology for all base meshes), it was shown to be a powerful technique for creating subtle animation effects with full control over details. This is the reason why it is used widely for facial animation, where *blend shapes* are defined for a variety of emotional expressions and vocal sounds.

Vertex Blending (also referred to as *Vertex Skinning* [18] or *Skeleton Subspace Deformation*) is another interpolation technique which builds on the assumption of an underlying imaginary skeleton within the character that defines the deformation of certain mesh parts and as a whole constitutes the configuration of the character's body. In general, character models employing this approach are harder to control precisely than those using shape interpolation but allow animators to develop body posture quickly and block out the extreme poses of the character before his motions are refined further. This process resembles the stages of creating traditional cel animation and allows for much faster results and broad reusability of the underlying skeletal deformation models (commonly referred to as *character rigs* in the industry). [10] describes both algorithms in more detail and points out some common problems and limitations of the approaches which are beyond the scope of this text.

3. SYSTEM OVERVIEW

- Ability to load scenes including animated characters (+ synchronized voice audio track)
- Animation playback and real-time scene rendering
- UI allowing for intuitive and precise playback control (pause, rewind, reverse)
- UI allowing the user to leave spoken annotations

4. IMPLEMENTATION

4.1 Hardware

- Display: active stereo projection system
- Tracker: IS-900 hybrid tracking system, used to track position of head interaction device
- Sound: headset for playback and recording of voice annotations
- Interaction: IS-900 wand

4.2 Asset exchange format

One of the main challenges in order to import initial datasets into the growing application framework was to find and integrate a data exchange format that allows for export of complex scene information (containing geometry, materials, joint hierarchies and their respective bindings, animation curves, etc.) from an available software application capable of creating keyframed character animation. As I have worked extensively with Autodesk Maya in the past, I decided to look for a data format which would export the required data from Maya software into a format accessible through a high-level file abstraction layer.

The COLLADA digital asset exchange format [6] provides an open, well-documented scheme which allows exchanging a large variety of digital assets used commonly in CG applications and video games. In addition, the company *Feeling Software* provides an export tool for Autodesk Maya as well as an open-source library for C++ providing high-level abstractions of the underlying asset data (FCollada [14]). The actual scene file is stored in one XML file; access to the encapsulated objects was established using the FCollada library.

4.3 Toolkits & libraries

In order to shift the focus of the software development process towards intuitive interaction techniques and robust control over the complex hierarchical structure within the animated scene, I decided to use a variety of freely available software toolkits and libraries that provide functionality required for the system specifications.

- VR Juggler toolkit [15]

  VR Juggler 2.2 was used to simplify access to hardware (tracker, interaction devices) and to guarantee cross-platform compatibility of the employed graphics rendering routines using OpenGL. Configuration files were created for the specific hardware configuration in the VR lab

- Sonix (part of VR Juggler suite)

  Provides a high-level interface for sound playback that can be used with different underlying sound APIs like OpenAL or Audiere

- OpenAL Cross-Platform 3D Audio library [3]

  As Sonix does not provide functionality to capture audio samples from an input device, OpenAL was used to capture voice annotations and write them to a WAV file

- FCollada (by Feeling Software)

  see 4.2

- FreeType library [2]

FreeType 2.3.5 was used to create OpenGL textures containing anti-aliased text characters from TrueType font files. These were then rendered on simple quad geometry using OpenGL display list calls

4.4 Class hierarchy

In order to take advantage of the high-level abstraction of scene assets used in the FCollada library, I also tried to incorporate the idea of creating high-level abstractions of the underlying semantic objects and patterns into the overall software design. This resulted in a collection of classes that hide their underlying complexity and make the implementation of the rendering and interaction behavior inside the methods called repetitively by the VR Juggler main kernel loop clearly structured and comprehensible. As an overview of all classes created for the project (42 total) is beyond the scope of this text, two code snippets (Figure 1 and 2) are included to clarify the advantages of the described approach. For further details refer to the actual source code provided in digital form with this document.

While the FCollada library provided functionality for loading and accessing objects within a data file, it did not provide any implementation of scene graph operations, rendering or vertex blending functionality. Based on my knowledge of computer graphics and scene graph design, I implemented all functionality employing the data provided by the FCollada from scratch.

```
void CApplication::draw()
{
    glClear(GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
        informationConsole->render();
        virtualHand->render();

        glMultMatrixf( virtualUniverseTransform.mData );
        scene->render();
    glPopMatrix();
}
```

Figure 1: Main render loop showing high-level access to visual scene objects

Microsoft's Visual Studio 2005 C++ compiler was used to create the executable application which has been developed and tested under Microsoft Windows, but can be compiled cross-platform due to the operating system independent implementation of its components.

4.5 Vertex blending algorithm

As the basic vertex blending algorithm that was implemented for this project requires several per-vertex computations, it is highly suitable for optimization using custom vertex shaders that run on the GPU. The per-vertex computations are not interdependent, allowing for fast parallel computation exploiting the GPU architecture. The general approach is as follows:

1. Store complete per-vertex information in a vertex array that can be transferred permanently to GPU memory using a VBO (vertex buffer object). The per-vertex information contains
   - initial (bind pose) position
   - initial (bind pose) normal
   - number of joints influencing this vertex
   - array of joint IDs referring to influential joints
   - array of weights per influential joint
2. Store topology / connectivity information about the mesh as a VBO (typically set of indexed primitives)
3. Calculate the world transforms of scene graph nodes representing the skeleton joints (typically depending on animation curve(s))
4. Make transforms available to the vertex shader (typically using uniform matrix variables)
5. Execute OpenGL call which renders the mesh using our custom vertex shader
6. Continue with step 3.

```
// check whether any existing annotation is currently selected
CAnnotation* selectedAnnotation = scene->getSelectedAnnotation();
if (selectedAnnotation != 0)
{
    // play recorded sound annotation
    selectedAnnotation->startSoundPlayback();
    informationConsole->pushMessage("Start playing recorded
        annotation message...", 2.0f);
    // and set current time to the annotation timer
    scenePlaybackControl->setCurrentTime(
        selectedAnnotation->getTime() );
}
else
{
    // Add new annotation
    currentAnnotation =
        new CAnnotation(scenePlaybackControl->getCurrentTime());
    Matrix44f VU_wrt_Scene, World_wrt_VU;
    gmtl::invertFull(VU_wrt_Scene, scene->getTransform());
    gmtl::invertFull(World_wrt_VU, virtualUniverseTransform);

    currentAnnotation->setTransform(
        VU_wrt_Scene * World_wrt_VU * pointerTransform );
    scene->addAnnotation(currentAnnotation);
    interactionMode = ADD_ANNOTATION;
    informationConsole->pushMessage("Hold trigger and move
        wand to position annotation.");
}
```

Figure 2: Code snippet showing high-level access to scene structure

See Figure 3 for an abbreviated implementation of the vertex shader program (not shown: lighting calculation).

This method was successfully used to render simple real-time animated objects within the application, but it has an inherent restriction regarding the number of uniform variables available to store the updated joint transforms. This puts a constraint on the maximum number of joints in the scene that can be used without altering our previously described approach.

Limitations differ depending on graphics hardware, so that I also implemented a more general solution used in the current

```
//
// Additional vertex attributes
//
attribute float numJoints;
attribute vec4 jointID;
attribute vec4 jointWeight;

//
// Uniform application data
//
uniform mat4 jointTransform[32];

//
// Varying variables shared with fragment shader
//

void main() {
  int i, id, numJointsInteger;
  vec3 normal;

  numJointsInteger = int(numJoints);
  if (numJointsInteger > 0) {
    vec4 normalIn = vec4(gl_Normal, 0.0);
    vec4 normalOut = vec4(0.0, 0.0, 0.0, 0.0);
    vec4 position = vec4(0.0, 0.0, 0.0, 0.0);

    i = 0;
    id = 0;

    while (i < 4) {
      id = int(jointID[i]);

      if ((id >= 0) && (id < 32)) {
        position +=
          jointWeight[i] * (jointTransform[id] * gl_Vertex);

        normalOut +=
          jointWeight[i] * (jointTransform[id] * normalIn);
      }

      i++;
    }

    gl_Position = gl_ModelViewProjectionMatrix * position;
    normal = normalize(gl_NormalMatrix * normalOut.xyz);

  } else {
    // no skinning -> use regular vertex position and normal
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    normal = normalize(gl_NormalMatrix * gl_Normal);
  }

  ...
}
```

Figure 3: GLSL vertex shader code implementing vertex blending algorithm

system: All joint transforms are kept in main memory and vertex blending is done in the CPU before the transformed (or *blended*) vertices are sent to the graphics card. The previously described approach is still preferred, but due to time constraints, performance considerations had to be subordinated to a flexible, working solution capable of rendering complex scenes with a large number of joints.

In the future, the memory available to a vertex shader could be extended by 'misusing' texture memory to store a large number of joint transforms that is interpreted as an array of 4x4 matrices by the shader program. Thereby, the maximum number of joints will increase dramatically compared to the initial approach while only minimal changes to the implementation are necessary.

4.6 Asset acquisition

After receiving no reaction from online forum members on 11 second club [1] (see my post in 6.1), I tried contacting an animator who has won several online animation competitions and currently works for an animation studio in Singapore. He was willing to help and provide me with material but it turned out that he was not legally allowed to share the original Maya scene files with me due to copyright restrictions on the character rigs he used. They were provided by an animation school and can not be shared with anybody outside the school.

Unfortunately, I was not able to find an animated character sequence as elaborate as I had envisioned in order to evaluate its virtue when presented within the application. Instead, I incorporated a walking sequence that I had created as an undergraduate student in 2005 using the Generi Rig [13]. This sequence does not contain a voice track although the application framework allows for import and playback of a voice / sound track along with scene playback. Evaluation of the system using a motion sequence containing more expressive animation will be necessary to judge its capabilities.

5. FUTURE WORK

Keyframe animated sequences produced for use in feature films (or other media with the viewer maintaining a fixed point of view) are staged and created with previous knowledge about a spectator's single point of view [5]. Many animation principles – which were developed while working in traditional media types – are based on the assumption that this knowledge can be exploited by the animator in order to achieve desired effects on the viewer or to evoke certain emotional responses. Exaggerations in a certain motion that serve as aids in communicating information about the character's state of mind might not have the same magnitude or type of effect if the character is viewed from a different angle.

If we let the user control the viewpoint freely, we must take this major difference between VR and other media previously used for storytelling into account. Characters must look good from all vantage points and the readability of behaviors has to be ensured regardless of viewing angle. To me, this seems to be one of the major challenges to overcome in the future in order to make VR a compelling medium for storytelling.

6. APPENDIX

6.1 Online forum posting

=== Brief summary ===
I am looking for people willing to share Maya / COLLADA source files containing short clips of character animation. Those will be used within a Virtual Reality system to evaluate the questions raised in my research project.
==========

Hi everyone,

I am a graduate student in Computer Science and have been fascinated by the art of character animation for many years (and enjoy reading Richard Williams much more than any computer literature... :-)).

Currently, I am working as a research assistant at a Virtual Reality lab. As part of my thesis research, I would like to evaluate the implications and benefits of bringing believable animation (not "programmer's art") into an immersive system (e.g. a CAVE `http://graphics.ethz.ch/Downloads/Publications/Papers/2004/spie04/cave.jpg`). To start out with, I would like to find out whether presenting simple playblasts in a "true" 3D environment makes a difference in judging the quality of the animation and / or identifying flaws. As we will use a head tracking system, the user will be able to move around in the scene in order to change his / her point of view. You can find more details about the project here: `http://www.jptiesel.de/Project_Outline.pdf`

As I know there are a lot of gifted and skilled people out here on this site, I hope to get some help from a few of you: In order to present expressive and believable character animation, my own attempts at this will not be sufficient. That is why I hope to find somebody to provide me with Maya / COLLADA sequences (preferably WIP files at different stages [blocking, breakdowns, ...]) that I can load and play in the VR system. Any work in progress from a 11 Second Club contest would be particularly helpful, as it is a good length for a quick review and contains a voice track.

Of course, your work will be properly credited (at least in my report, maybe I can even get a research paper out of this) and not shared with anybody else without your consent. I also hope to be able to put a video together in a few months showing the system 'in action' and share some of the results.

Please shoot me an email if you also think that this is an interesting project and are willing to help me to get it going. I can provide for FTP space to upload files.

Thanks for staying with me all the way to the end of this post, keep up the great work!!!

Jan-Phillip Tiesel, jpt4246@gmail.com

7. REFERENCES

[1] 11 second club: Forum. `http://www.11secondclub.com/forum`. Accessed 5/13/08.

[2] The freetype project: a free, high-quality and portable font engine. `http://www.freetype.org`. Accessed 5/13/08.

[3] Openal: A free (lgpl-ed) and open source, cross platform audio library used for 3d and 2d sound. `http://www.openal.org`. Accessed 5/13/08.

[4] T. Akenine-Möller and E. Haines. *Real-time Rendering*. A.K. Peters Ltd., 2nd edition, 2002.

[5] Marty Altman. Personal discussion. 4/7/08.

[6] M. Barnes. Collada: Digital asset schema release 1.4.1 specification. `http://www.khronos.org/files/collada_spec_1_4.pdf`, 2006. Accessed 3/29/08.

[7] Dane Edward Bettis. Digital production pipelines: examining structures and methods in the computer effects industry. Master's thesis, Texas A&M University, May 2005. Available online at `http://txspace.tamu.edu/bitstream/1969.1/2406/1/etd-tamu-2005A-VIZA-Bettis.pdf`.

[8] Ollie Johnston and Frank Thomas. *The Illusion of Life. Disney Animation.* Abbeville Press, New York, 1st ed. edition, 1981.

[9] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 35–44, New York, NY, USA, 1987. ACM.

[10] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[11] Maic Masuch, Knut Hartman, and Grit Schuster. Emotional agents for interactive environments. In *C5 '06: Proceedings of the Fourth International Conference on Creating, Connecting and Collaborating through Computing*, pages 96–102, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Randy Pausch, Jon Snoddy, Robert Taylor, Scott Watson, and Eric Haseltine. Disney's aladdin: first steps toward storytelling in virtual reality. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 193–203, New York, NY, USA, 1996. ACM.

[13] Andrew Silke. Generi rig. `http://www.andrewsilke.com/generi_rig/generi_rig.html`. Accessed 3/29/08.

[14] Feeling Software. Feeling software: Fcollada. `http://www.feelingsoftware.com/content/view/62/76`, 2008. Accessed 3/29/08.

[15] VR Juggler Development Team. The vr juggler suite. `http://www.vrjuggler.org`. Accessed 5/13/08.

[16] Marco Vala, Pedro Sequeira, Ana Paiva, and Ruth Aylett. Fearnot! demo: a virtual environment with synthetic characters to help bullying. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–2, New York, NY, USA, 2007. ACM.

[17] Richard Williams. *The Animator's Survival Kit: A Manual of Methods, Principles, and Formulas for Classical, Computer, Games, Stop Motion, and Internet Animators.* Faber & Faber, 2002.

[18] R. Woodland. Filling the gaps: Advanced animation using stitching and skinning. *Game Programming Gems*, pages 476–483, 2000.