Cross-mapping

Article · January 2005

DOI: 10.1145/1198555.1198583

J.P. Lewis
Google Inc.

Siggraph 2005 Digital Face Cloning

Course Notes

Frédéric Pighin
University of Southern California

J.P. Lewis
Graphics Primitives

Motivation and Overview

Synthesizing a digital face that is indistinguishable from a real one has long been an elusive goal for the graphics community. Driven by the need to replace actors in difficult stunts, recent efforts have focused on capturing digital replicas of real actors. Thanks to hardware and software innovations this approach has yielded digital faces of unprecedented realism.

Digital Cloning describes the process of capturing an actor's performance and optionally their likeness in a digital model. The captured performance can be used as a virtual stunt double, or mapped to a physically distinct character such as a child or animal. Although photo-real virtual stunt doubles are now commonly used in brief and distant movie shots, project demands are gradually increasing the duration and scale of virtual actor effects. Performance cloning to drive a distinct character enabled Polar Express and is frequently considered on other projects.

Despite these successes, Digital Cloning remains a challenging problem, in large part because human observers are particularly sensitive to the realism and subtleties of human faces. Recent virtual actors are able to fool most observers for only a few seconds, and some observers feel that cloned performances do not map naturally to distinct geometric models.

In this course we will present the diverse techniques that comprise the current state of the art in digital face cloning. Topics will include motion capture (marker based and dense), lighting capture, skin simulation, cross mapping algorithms, derivation of facial models from data, and audience perception of realism. Given the wealth of research on digital face cloning and its widespread applications, this area would benefit from a Siggraph course.

Instructors will include experts from both the entertainment industry and academic research. The course will inform the graphics community about the latest advances in this rapidly developing field and outline some of the remaining challenges in digitally cloning the human face. We hope that the course will help in defining new approaches for creating photorealistic digital faces.

Course attendees should have a background in computer graphics and mathematics in order to fully understand all the presentations.

The course is organized in two main sections. The first section spans the morning and focus on the technologies for capturing and modeling a digital face. The second section spans most of the afternoon and present case studies. These studies describe recent successful attempt from the industry at digitizing realistic faces. We will conclude the course with a discussion on the open issues in digital face cloning.

The course schedule, speaker bios, and table of contents follow.

Course schedule

8:30	Introduction and Overview **Fred Pighin**
9:00	Face Scanning and Dense Motion Capture Technologies **Li Zhang**
10:00	Break
10:15	Reflectance Modeling and Capture **Paul Debevec and J.P. Lewis**
11:15	Facial Parameterization and Cross-Mapping **J.P. Lewis and Fred Pighin**
12:15	Lunch
1:30	Case Study: Face Cloning at ILM **Christophe Hery and Steve Sullivan**
2:15	Case Study: The Gemini Man **Lance Williams**
3:00	Case study: Leaping the Uncanny Valley with Data (Face Cloning in the Matrix sequels) **George Borshukov**
3:45	Break
4:00	Case study: Polar Express **David Bennett**
4:45	Perception of Facial Realism **J.P. Lewis**
5:15	Panel on the Future of Digital Face Cloning **All speakers.**

Bios

Co-Organizers

J.P. Lewis
Graphics Primitives
J.P. Lewis has consulted on three film industry digital cloning efforts and was codeveloper of the image-domain skin subsurface scattering simulation used on the Matrix sequels. He has authored research papers computer graphic facial topics including eye movement and derivation of blendshape representations, and his early work on automatic lip-movement is described in Parke and Waters's book Computer Facial Animation.

Frédéric Pighin
USC Institute for Creative Technologies
Fred Pighin's pioneering work on the construction and animation of 3D facial models from images is widely recognized. More recently he has contributed research in areas including eye movement, automatic partitioning of blendshape models, and emotional speech animation. His published research on these topics appears in journals and conferences including SIGGRAPH, ACM Transactions on Graphics, ACM/Eurographics Symposium on Computer Animation, and ICCV. Fred was the primary organizer of the Frontiers of Facial Animation workshop held at the Institute for Creative Technologies at USC following SIGGRAPH 2004.

Lecturers

George Borshukov
Electronic Arts
George Borshukov holds a Masters Degree from the University of California at Berkeley where he was one of the creators of "The Campanile Movie" (SIGGRAPH 1997). He was Technical Designer for the "Bullet Time" sequences in The Matrix, and received an Academy Scientific/Technical award for image-based rendering technology used on this film. Borshkov lead the development of photoreal digital actors seen in the Matrix sequels, and received a Visual Effect Society Award for this work. Borshukov is now a Computer Graphics Supervisor at Electronic Arts, where he is focusing on setting the standard for next-generation interactive entertainment.

David Bennett
Sony Pictures Imageworks
David Bennet is a scientist in the cross-mapping group at Imageworks. Prior to joining Sony, he was a member of the operational staff at the National Geospatial Intelligence Agency and founder of the X7 offshore entity. He received his Raisin Juteux d'Édification from the Institut Suisse des Fulminations Cartographique, where his controversial efforts inspired the Theban Mapping Incident series on BBC-4. In his spare time David enjoys ambulation and other sports.

Paul Debevec
USC Institute for Creative Technologies
Paul Debevec received a Ph.D. in Computer Science at UC Berkeley in 1996 and conducts research in computer graphics and vision at USC's Institute for Creative Technologies. He has co-authored seven SIGGRAPH papers and five Electronic Theater animations involving research in image-based modeling, rendering, and lighting. Debevec has researched techniques for capturing natural illumination using high dynamic range photography to realistically illuminate virtual actors, and has led the development of several Light Stage systems for capturing realistic human reflectance properties and relightable performances for motion pictures. Debevec received ACM SIGGRAPH's Significant New Researcher Award in 2001.

Christophe Hery
Industrial Light + Magic
Christophe Hery received his degree from the Ecole Superieure des Travaux Publics, and has worked at computer graphics companies including BUF and TDI. At Industrial Light and Magic Hery is a R&D lead, working on skin research and related rendering issues. In 2003, Hery was recipient of a Technical Achievement Award for the development of methods for rendering skin using subsurface scattering techniques. He is currently working on a system for creating talking digital clones. Christophe has frequently participated in SIGGRAPH courses, panels, and sketches, most recently including the 2001 and 2003 Renderman Courses and 2004 Crowd simulation course.

Steve Sullivan
Industrial Light + Magic
Steve Sullivan received a Ph.D. in electrical engineering from the University of Illinois at Urbana-Champaign. At Rhythm and Hues he introduced computer vision techniques and developed 3D tracking with and without models. Currently he is Principal Engineer at Industrial Light & Magic, where he founded and leads the computer vision group. He received an academy Technical Achievement Award for the development of ILM’s structure and motion recovery system. Sullivan’s published research appears in IEEE PAMI and elsewhere.

Lance Williams
Applied Minds, Inc.
Lance Williams received a PhD in computer graphics from the University of Utah, and worked in the historic Graphics Lab at New York Institute of Technology. He was inventor of the Z-buffer shadow mapping and “mip” texture mapping techniques, as well as the pioneer of the facial performance cloning idea. Lance received SIGGRAPH’s Coons Award in 2001, and a motion picture Technical Academy Award in 2002. He was Chief Scientist at Disney Feature Animation, and has worked at Dream-works SKG, Apple Advanced Technology Group, and Jim Henson. He is currently pursuing proteomics research at Applied Minds.

Li Zhang
The University of Washington
Li Zhang obtained his B.E. in Automation, with a minor in Radio Technology and Information System, at Tsinghua University in Beijing, P.R. China, in 1998. He then began his PhD study in the Robotics Institute at Carnegie Mellon University. After two years, he transferred, with his advisor Steve Seitz, to the Department of Computer Science and Engineering at University of Washington, where he is now a PhD candidate under the supervision of Professors Steve Seitz and Brian Curless. Li Zhang is generally interested in Computer Vision and Graphics problems and is focused on 3D shape and motion recovery from videos with its applications in animation and medical robotics. Li Zhang is expecting to obtain his Ph.D. degree in the June of 2005.

Contents

Powerpoint Presentations

Included Papers

Siggraph 2005 course notes - Digital Face Cloning

Introduction

Frédéric Pighin J.P. Lewis

University of Southern California

Overview

By far the most challenging aspect of a photoreal actor is the creation of a digital face that can stand up to close scrutiny. The human face is an extremely complex biomechanical system that is very difficult to model. Human skin has unique reflectance properties that are challenging to simulate accurately. Moreover the face can convey subtle emotions through minute motions. We do not know the control mechanism of these motions. All these issues combine to make the human face one of the most challenging object to model using computer graphics.

In recent years, a novel approach to synthesizing realistic faces has appeared. Driven by the need to replace actors in difficult stunts, recent efforts have focused on capturing digital replicas of real actors. This approach, called *Digital Face Cloning*, attempts to create a digital face by replicating in parts the face of a real-performer. Digital face cloning describes the process of capturing an actor's performance and optionally their likeness in a digital model. With this technique, the human face is no longer considered as a biomechanical system but as a real object that can be digitized to produce a synthetic replica. This replication process has been made possible through a set of recently developed technologies that allow the recording of a performer's face. For instance, a face scanner can be used to to recover the geometry and some of the reflectance properties of the face. The motion of a face can be recorded using a motion capture system. Hundreds of points on the face can be tracked and mapped onto a virtual character whose facial motions will mimic those of the performer.

Advances in software technology have also been instrumental for cloning digital faces.

One of the most promising technology for replicating the motion of the human face is dense motion capture technology. Traditionally motion capture systems are limited to at most a few hundred markers on a human face. At this resolution, many of the small scale details (e.g. wrinkles) of the face might be missed. Dense motion capture allows the tracking of tens of thousands of facial locations. Generally the output of dense motion capture is a sequence of detailed face meshes. The meshes can then be registered so that facial features are in topological correspondence throughout the sequence. The end result is high-fidelity three-dimensional representation of the actor's performance.

Equally important is the ability to render faces realistically. Human skin is particularly difficult to render. This difficulty has two origins. First, human skin is somewhat transparent and exhibit multiple scattering effects. Simulating this phenomena requires considering skin as a volumetric object. Second, the skin's reflectance varies in space but also in time. Through the contraction and dilatation of blood vessels the reflectance of the skin varies as the face changes facial expression. To make things worse, the appearance of the face is also determined by wrinkles, pores, follicles, and other surface details whose size is often less than a millimeter. Given these challenges recent research has focussed on digitally capturing these properties with measuring devices. The purpose of these devices is to measure a particular performer's skin BRDF.

In the rest of this course, we will describe in more details the state of the art in facial geometry, motion, and BRDF acquisition. A critical issue in face cloning is the ability to remap motion captured on one performer into a digital character. This issue, called cross-mapping will be discussed in the course.

Finally and most importantly, we will present a few practical examples of successful digital face cloning systems that have been used in production.

In the rest of this introduction, we provide a brief historical overview of the field and we discuss some major issues related to digital face cloning.

History

The idea of creating artificial humans greatly predates computer graphics. It has haunted popular imagination well before computers were invented. For instance, in Jewish folklore the Golem is an artificially created human supernaturally endowed with life. More recently Mary Shelley's Frankenstein is a classic of gothic literature.

From a computer graphics point of view, we can look at the progression of digital face cloning from two perspectives. First, the film industry has long tried to include synthetic actors in movies. This has been motivated primarily by the need to replace actors in difficult stunts, but other purposes are becoming more common, including the reproduction of persons that are no longer living. Second, we will survey university research on this problem and consider cloning techniques that may be adopted in the future.

Film industry

The emergence of believable computer graphics (CG) actors remains a significant challenge in our field, and a successful demonstration of photoreal actors would be considered a milestone.

In fact considerable albeit gradual progress towards this goal has already occurred, and a singular "breakthrough" appearance of virtual actors is perhaps unlikely. CG humans have appeared in movies since *Futureworld* (1976), and CG stunt doubles have made distant and brief appearances in films in the 1990s including *Terminator 2* (1991), *Jurrasic Park* (1993, body only) and *Titanic* (1997). In recent years these stunt doubles have had somewhat larger and longer (though still brief) appearances, and have taken the form of recognizable actors in films such as *Space Cowboys* (2000) and *Enemy at the Gates* (2001). The current state of the art is represented by digital stuntpeople in *Spiderman II*, *The Matrix: Revolutions* (with a more-than-10-second full-screen shot of two virtual clones), and *Lemony Snicket* (with a number of CG shots of a baby intercut with the real baby). The corresponding technical developments, including implementations of lighting capture, subsurface scattering, and dense motion capture, are outlined in Siggraph sketches [19, 9, 2].

Although the progress in the last decade has been substantial, there is even further to go. Current technology can produce only brief cloned shots at considerable cost, and although the results are remarkable examples of computer graphics, they do not actually deceive a majority of observers for more than a few seconds. Moreover, several large scale film industry attempts to produce a CG "lead" character have been attempted and subsequently abandoned. These include a proposed remake of *The Incredible Mr. Limpet* in 1998, and Disney's Gemini Man attempt in 2000.

Academia

Facial animation has long been a very active research area. We do not pretend to cover all relevant research in this section but provide a broad historical overview of the field. More details studies can be found in other sections of the course notes.

We can tentatively date the beginnings of research on digital face cloning with the use of photographs for face modeling. One of the earliest efforts in this field was the pioneering system developed by Fred

Parke [14, 15] who made use of two orthogonal photographs and patterns painted on a performer's face to recover 3D facial geometry. Later Pighin et al. [16] extended this technique to an arbitrary number of images and texture extraction. With the advent of range scanning researchers have explored the use of facial scan to automatically model faces. The resulting range and color data can be fitted with a structured face mesh, augmented with a physically based model of skin and muscles [11, 10, 20, 22]. Haber et al. [7] propose a similar system using an improved facial model.

The idea of recovering facial geometry from images led naturally to motion estimation from video. An early system by Lance Williams [23] tracks the 2D face motion of a performer from a single video stream. Later, Brian Guenter et al. [6] extend this approach to 3D recovery from multiple video streams. Other researchers have explored marker-less tracking techniques, often relying on more sophisticated models such as blend shape (linear) models [17, 1], bi-linear models [21], or physically-based models [20]. In this perspective, a promising new research direction is the recovery of dense geometry from video. The UCAP system [3] used five hi-def cameras to recover the facial geometry and texture at every frame (requiring manual assistance, but no structured light), whereas the system developed by Li Zhang et al. [24] uses six video cameras and two structured light projectors to recover the facial geometry completely automatically.

To enable rendering from a different point of view and with a different lightning environment, researchers have explored how to capture the reflectance property of a face. For instance Marschner et al. [13] measured the human skin BRDF using a small number of images under different lighting conditions and re-used these measurements to realistically render a performance driven face [12]. In a more data-driven approach, Debevec et al. [4] used an image-based technique to render human faces in arbitrary lighting environments by recombining a large set of basis images from different lighting directions. Finally, Tim Hawkins et al. [8] extended this work by capturing the performer in a variety of expressions to build a blend shape model that includes reflectance information.

Main issues

The process of digital face cloning can be broken into two main steps. First, during the *recording step*, the face of the performer is recorded. This recording can take several aspects depending on how the recorded data will be used. To build an accurate replica, the geometry, the reflectance, and the motion of the performer all need to be recorded.

Second, during the *synthesis step*, the recorded data is reused to create a digital sequence. In the simplest case, only a specific performance needs to be captured and synthesized. In such a case there is usually no much need to modify the recorded data. In more complex cases, the data might need to be extensively modified. For instance, this may be because the data is remapped onto a different character, or because the character is placed in a different lighting environment, or maybe the motion has to be modified.

In an abstract sense, the data collected represent a set of samples from a space of facial properties. For instance, a face scan might be considered as a sample from a space of facial expressions, a recorded motion would belong to a space of motions. During the second step, these samples are then used to reconstruct portions of that space. In this perspective, the synthesis step can be seen as a resampling problem where the original samples are altered to meet the needs of a production. How this resampling is done is key to the process of digital face cloning. In general, this raises the issue of figuring out how the recorded properties of the face behave between the samples.

For example let us consider the space of facial expressions for a performer: we might have scanned his face in a neutral expression and in a angry expression. To generate an intermediate expression between these two sampled expressions we have to make some assumptions about how the face behave between the samples. These assumptions form what we will call a *representation* for the data. For instance, a BRDF representation of the skin might be used to resample reflectance data, or frequency decomposition might be used to modify facial motions. Another interesting example is a blend shape model. In this case resampling is done using correspondences between facial features (i.e., domain knowledge) and linear interpolation.

Many representations or resampling techniques can be used to modify the data. Some of them are very simple such as nearest neighbor, where the closest (according to some appropriate distance metrics) is selected, others can be quite complex and for instance involve physically-based modeling. In general, there seems to be a wide spectrum of representations starting with the *weak representations* (point sampling, linear interpolation) up to the *strong representations* (biomechanics, physics-based models). Choosing the best representation depends on many factors. This decision mainly reflects a tradeoff between how many samples need to be recorded versus how complex the representation is. If the reconstructed space is densely sampled then a weak representation is more appropriate since the behavior between the samples can usually be modeled using some simple approximation (e.g. linear approximation). If on the other hand there are few samples to work with then a stronger (or richer) representation might be needed in order to model complex behavior between the samples. Often weak representations are limited to interpolating the samples so that the reconstructed space lies within the convex hull of the samples – whereas with a strong representation, using the same samples, it might be possible to extrapolate from the samples and cover a much wider portion of the space.

As we have discussed the use of weak models often relies on a dense set of samples. Depending on the properties of the face that is modeled obtaining sufficient sample data might be more or less tractable. For instance, it is relatively easy to take photographs of a face using different camera positions (e.g. for view interpolation) but it is significantly more difficult to densely sample the set of potential face motions. It also depends on the extend of the portion of the space that needs to be reconstructed. Obviously the smaller that extent is the easier it is to sample that portion of the space. Finally, the properties of the face that vary little or in smooth ways usually require much fewer samples to be modeled compared to other properties that are less smooth.

The representation issue is not only relevant to the synthesis step but also to the data gathering step. Data gathering sometimes necessarily makes use of a representation. For instance in model-based tracking, a representation of the face is used to track facial motion from a video stream. The representation itself might be based on some initial data (or prior in a Bayesian framework). The representation used for data gathering might not be appropriate for synthesis, in this case a different representation is needed. This raises the question of whether the same representation can be used for analysis and synthesis. One of the interesting issues is the parameterization of the data. Often during synthesis we are interested in choosing parameters that have an intuitive meaning so that they can be manipulated easily by an animator. On the other hand, for analysis, a good set of parameters is sometime selected according to different criteria such as orthogonality. This issue arises for instance in the parameterization of a blend shape model.

The synthesis step dictates what kind of samples need to be collected during the capture step. However, given a particular sample space, it is often unclear which specific samples need to be recorded. If the sampling process is expensive (e.g. motion capture) the issue of how many samples need to be recorded and which to record becomes critical. Unfortunately, often there are no rules to answer these questions. More likely, it is a trial and error process that can be guided by some study of the human face. For instance, even though work in psychology [5] stresses the role of six basic expressions, most blend shape systems use a significant larger number of expressions (e.g. 946 for Gollum in the Lord of the Rings [18]).

References

[1] V. Blanz, C. Basso, T. Poggio, and T. Vetter. Reanimating faces in images and video. In *Proc. of Eurographics*, 2003.

[2] George Borshukov and J. P. Lewis. Realistic human face rendering for “the matrix reloaded”. In *Proceedings of SIGGRAPH conference Sketches & applications*. ACM Press, 2003.

[3] George Borshukov, Dan Piponi, Oystein Larsen, J. P. Lewis, and Christina Tempelaar-Lietz. Universal capture: image-based facial animation for ”the matrix reloaded”. In *Proceedings of SIGGRAPH conference on Sketches & applications*. ACM Press, 2003.

[4] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the reflectance field of a human face. In *SIGGRAPH 2000 Conference Proceedings*, pages 35–42. ACM SIGGRAPH, July 2000.

[5] P. Ekman and W.V. Friesen. *Unmasking the face. A guide to recognizing emotions fron facial clues*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[6] B. Guenter, C. Grimm, D. Wood, H. Malvar, and F. Pighin. Making faces. In *SIGGRAPH 98 Conference Proceedings*, pages 55–66. ACM SIGGRAPH, July 1998.

[7] J. Haber, K. Khler, I. Albrecht, H. Yamauchi, and H.-P. Seidel. Face to face: From real humans to realistic facial animation. In *Proceedings Israel-Korea Binational Conference on Geometrical Modeling and Computer Graphics*, pages 37–46, 2001.

[8] Tim Hawkins, Andreas Wenger, Chris Tchou, Andrew Gardner, Fredrik Göransson, and Paul Debevec. Animatable facial reflectance fields. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*, pages 309–320, June 2004.

[9] Christophe Hery. Implementing a skin bssrdf (or several). SIGGRAPH course notes: Renderman, Theory and Practice, 2003.

[10] Y. Lee, D. Terzopoulos, and K. Waters. Realistic modeling for facial animation. In *SIGGRAPH 95 Conference Proceedings*, pages 55–62. ACM SIGGRAPH, August 1995.

[11] Y.C. Lee, D. Terzopoulos, and K. Waters. Constructing physics-based facial models of individuals. In *Proceedings of Graphics Interface 93*, pages 1–8, May 1993.

[12] Stephen Marschner, Brian Guenter, and Sashi Raghupathy. Modeling and rendering for realistic facial animation. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 231–242, June 2000.

[13] Stephen R. Marschner, Stephen H. Westin, Eric P. F. Lafortune, Kenneth E. Torrance, and Donald P. Greenberg. Image-based brdf measurement including human skin. In *Eurographics Rendering Workshop 1999*, June 1999.

[14] F.I. Parke. Computer generated animation of faces. *Proceedings ACM annual conference.*, August 1972.

[15] F.I. Parke. *A parametric model for human faces*. PhD thesis, University of Utah, Salt Lake City, Utah, December 1974. UTEC-CSc-75-047.

[16] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D.H. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH 98 Conference Proceedings*, pages 75–84. ACM SIGGRAPH, July 1998.

[17] F. Pighin, R. Szeliski, and D.H. Salesin. Resynthesizing facial animation through 3d model-based tracking. In *Proceedings, International Conference on Computer Vision*, 1999.

[18] B. Raitt. The making of Gollum. Presentation at U. Southern California Institute for Creative Technologies's *Frontiers of Facial Animation* Workshop, August 2004.

[19] Mark Sagar. Reflectance field rendering of human faces for “spider-man 2”. In *Proceedings of SIGGRAPH conference Sketches & applications*. ACM Press, 2004.

[20] D. Terzopoulos and K. Waters. Techniques for realistic facial modeling and animation. In Nadia Magnenat Thalmann and Daniel Thalmann, editors, *Computer Animation 91*, pages 59–74. Springer-Verlag, Tokyo, 1991.

[21] D. Vlasic, M. Brand, H. Pfister, and J. Popovic. Face transfer with multilinear models. In *Proceedings of ACM SIGGRAPH 2005*. ACM Press/Addison-Wesley Publishing Co., 2005.

[22] K. Waters. A muscle model for animating three-dimensional facial expression. In *SIGGRAPH 87 Conference Proceedings)*, volume 21, pages 17–24. ACM SIGGRAPH, July 1987.

[23] L. Williams. Performance-driven facial animation. In *SIGGRAPH 90 Conference Proceedings*, volume 24, pages 235–242, August 1990.

[24] Li Zhang, Noah Snavely, Brian Curless, and Steven M. Seitz. Spacetime faces: high resolution capture for modeling and animation. *ACM Trans. Graph.*, 23(3):548–558, 2004.

Modeling and Digitizing Human Facial Reflectance

Paul Debevec

USC Institute for Creative Technologies

This section of the course describes techniques for modeling and digitizing the reflectance properties of faces and using these properties to create realistic renderings of faces under new lighting conditions, from new viewing angles, and with different expressions and motion.

Facial reflectance is a complex interaction of light and tissue, and like most aspects of face modeling, is important to reproduce accurately. The complexities present several challenges for modeling, capturing, and rendering. Although skin reflectance generally has a diffuse component and a specular component, the components do not conform precisely to commonly used reflectance models. These components vary spatially across the face: changes in skin thickness and pigmentation alter the diffuse component, and the specular component varies with the characteristics of surface roughness and oiliness. The breadth and shape of the specular reflection further depend on wrinkles, pores, and follicles with important details present at fractions of a millimeter. Facial reflectance also varies with time, as facial motion alters the blood concentrations under the skin, and the shape of specular reflections changes as the skin expands and contracts. Furthermore, skin is significantly translucent, and light falling on one area of the face may scatter beneath the surface and shine out from a nearby area, typically picking up the color of the skin's pigment and blood. The skin is also just one part of the face - other parts such as the eyes, mouth, and hair can pose even greater challenges and are equally important to represent faithfully.

Today, there is no complete system for capturing a digital model of a person that models all of these effects. However, a number of techniques have made important inroads into these problems. These reflectance modeling processes span a range from fitting mathematical models to sparse reflectance measurements to capturing comprehensive image datasets of how a face transforms incident light into radiant illumination. Different techniques address different parts of the facial modeling problem: some techniques aim to create a digital model that can be driven by animation, while others aim to capture a person actually acting in a way that their performance can be inserted into a virtual scene. Following is an annotated bibliography of some of the relevant published work done to address these problems so far:

- Image-based BRDF Measurement Including Human Skin. Stephen R. Marschner, Stephen H. Westin, Eric P. F. Lafortune, Kenneth E. Torrance, Donald P. Greenberg. Eurographics Rendering Workshop 1999. 1999.

  This paper measured an aggregate skin BRDF (bidirectional reflectance distribution function) by photographing a person's forehead under different lighting and viewing directions. The authors fitted the data to the Lafortune reflectance model.

- Acquiring the Reflectance Field of a Human Face. Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, Mark Sagar. Proceedings of ACM SIGGRAPH 2000. pp. 145-156, 2000.

- Animatable Facial Reflectance Fields. Tim Hawkins, Andreas Wenger, Chris Tchou, Andrew Gardner, Fredrik Goransson, Paul Debevec. 2004 Eurographics Symposium on Rendering. 2004.

  The first paper acquired reflectance field datasets of a person's face as illuminated by a dense sampling of incident illumination directions, and showed that realistic renderings of the face could be made under novel illumination environments directly from the data. The authors also showed renderings made from novel viewpoints by using a reflectance model to transform the reflectance data. The latter paper acquired face datasets under variable illumination, expression, pose, and viewpoint, and analyzed them to create a morphable face model that could be rendered three-dimensionally under novel illumination environments and animation.

- Reflection from Layered Surfaces Due to Subsurface Scattering Pat Hanrahan, Wolfgang Krueger. Proceedings of SIGGRAPH 93. pp. 165-174, 1993.
- A Practical Model for Subsurface Light Transport. Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, Pat Hanrahan. Proceedings of ACM SIGGRAPH 2001. pp. 511-518, 2001.

  These publications introduced techniques for simulating the tranlucency of the skin using stochastic rendering techniques, making it possible for computer renderings of 3D models to exhibit this important aspect of skin reflectance. The latter publication showed how subsurface scattering characteristics could be measured from a real subject and simulated efficiently using realistic approximations to the complete scatting functions.

- Light Scattering From Human Hair Fibers. Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, Pat Hanrahan. ACM Transactions on Graphics. 22(3), pp. 780-791, 2003.

  As an example of facial reflectance beyond skin, this paper measured and modeled an improved characterization of the reflectance of hair. In particular, the paper modeled how the cellular structure of human hair strands yields two different specular highlights: one the color of the light, and one the color of the hair.

- Andreas Wenger, Andrew Gardner, Chris Tchou, Jonas Unger, Tim Hawkins, and Paul Debevec. Performance Relighting and Reflectance Transformation with Time-Multiplexed Illumination. Proc. SIGGRAPH 2005.

  This work presents a technique for capturing a spatially-varying model of a person's reflectance in real time. Using rapidly changing lighting from different directions, the moving subject is illuminated by a series of rapidly changing lighting conditions. From this data, the person's live performance is realistically rendered under novel illumination conditions.

Siggraph 2005 course notes - Digital Face Cloning

Cross-mapping

J.P. Lewis Frédéric Pighin

University of Southern California

Introduction

When done correctly, a digitally recorded facial performance is an accurate measurement of the performer's motions. As such it reflects all the idiosyncrasies of the performer. However, often the digital character that needs to be animated is not a digital replica of the performer. In this case, the decision to use performance capture might be motivated by cost issues, the desire to use a favorite actor regardless of the intended character, or the desire to portray an older, younger, or otherwise altered version of the actor. The many incarnations of Tom Hanks in *Polar Express* illustrate several of these scenarios.

In this scenario, the recorded (source) performance has to be adapted to the target character. In this section of the course, we examine different techniques for transferring or cross-mapping a recorded facial performance onto a digital face. We have grouped these techniques in several categories mostly as a function of whether they use a blendshape system for the source and/or the target face.

Cross-mapping techniques

Blendshape animation is one of the most widespread facial animation techniques. Thus, it is not surprising that many techniques consider the cross-mapping problem in the context of a blendshape system. If we have a set of blendshapes for the performer and one for the target character that correspond to the same expressions, then once the performance is mapped onto the source blendshapes, it can be mapped onto the target character by simply reusing the same weights [10].

Blendshape mapping. It is also possible to accomplish blendshape mapping when the source and target models have blendshapes with differing functions. The slides accompanying this session point out that, provided a skilled user can produce *N* "corresponding" poses of the source and target models, a matrix that converts the weights from source to target representations can be found with a linear system solve. This assumes of course that linearity is adequate – which is also an assumption of the underlying blendshape representation. More importantly, however, it assumes that animation of the source model is obtainable. In the case of performance driven animation however, the source model will *not* exactly represent the performance unless the model itself is obtained directly from that performance (e.g. by principal component analysis of dense capture). Thus, transferring the performance onto the source model is at issue (and, to the extent that this mapping can be solved, why not omit the source and map the performance directly to the target?).

Choe and Ko [3] invented a very effective technique for transferring a recorded performance onto a digital character. In this framework, the target face is animated as a set of blendshapes. The goal of the algorithm is to find for each frame of the performance the corresponding blending weights. To do this, they first create a corresponding set of blend shapes (or actuation basis) for the source face. Once this is done, the blending weights can simply be transfered from the source blendshapes to the target blendshapes. The main advantage of their approach is that it refines the source blendshapes as a function of the recorded performance. In this sense, it is tolerant to approximate modeling.

This technique starts by manually assigning a location (corresponding point) on the source model for each recorded marker. From these correspondences, a transformation (rigid transformation and scaling) is computed that maps the performance coordinate system onto the model coordinate system. This transformation takes care of the difference in orientation and scale between the performance and source models. It is estimated on the first frame and applied to all the frames of the performance. The following procedure is then applied for each frame. If a frame in the animation is considered as a vector, it can be written as a linear combination of the corresponding points in the blendshapes where the weights are the blending weights. This provides a set of linear equations where the blending weights are the unknowns. Augmented with a convexity constraint (i.e. all weights have to be non-negative and sum up to one), this system can be solved using quadratic programming. Their approach assumes that the source blendshapes can exactly represent the performance, which is generally not true of manually sculpted blendshape models. To address this issue, a geometric correction is performed by solving the same system for the position of the corresponding points. These two steps (blend weight estimation and geometric correction) are iterated until convergence. Finally, the displacement of the corresponding points are propagated to the model vertices using radial basis functions [2].

This work is presented in a muscle actuation framework where each blendshape corresponds to the actuation of a muscle or muscle group. However, it should equally apply to sets of blendshapes constructed with different philosophies.

Direct mapping to target blendshapes. The previous technique requires a set of blendshapes for the source face. Other researchers have investigated direct mappings between the source motions and the target blendshapes. For instance Buck et al. [1] developed a system for mapping 2D facial motion onto cartoon drawings. The input motion is estimated from video by tracking a sparse set of features whose configuration provides a simplified facial expression.

Their system is build on top of a library of cartoon drawings that represent key poses for different facial areas of the target character (e.g. mouth, forehead). These key drawings are blended together, much like a set of blendshapes, to create an animation. Their mapping algorithm relies on associating each key drawing with a particular configuration of the tracked features. This association is then generalized using a scattered data interpolation algorithm. The interpolation is performed using a partition of the space of potential feature configuration (i.e. simplified facial expression). The partition they use is a 2D Delaunay triangulation. To map a frame of input motion, first the triangle that contains that frame is determined; second the barycentric coordinates within the triangles are computed; finally these coordinates are used as blending weights to compute the combination of key drawings. To provide a 2D parameterization of the input space, a Principal Component Analysis is performed on some test data. The two first principal components (maximum variance) determine the reduced dimensionality space.

Tim Hawkins et al. [8] use the same technique to animate facial reflectance fields with a higher dimensional space.

Chuang and Bregler [4] described another approach to mapping a performance directly to a differing target model. The source in their technique is video, but similar thinking could be applied in mapping from three-dimensional motion capture. Important feature points on the video frames are tracked using the Eigenpoints technique in which the weights of an eigenimage fit are applied in parallel to vectors of two dimensional points associated with each of the basis image [6]. The tracked feature points in a new image frame can (after a coarse affine registration) be approximated as a linear combination of basis feature vectors, and similar basis shapes can be sculpted for the target model.

With this background, they present two ideas that lead to a robust cross-mapping. First, they show how to choose a basis from among the source frames. After experimenting with several plausible approaches it was found that the best basis (least error in representing the source performance) resulted from taking the source frame point vectors that result in the smallest and largest projection on the leading eigenvectors of the source performance principal component model. Secondly, they point out that reusing the source weights on a target model does not work well when the basis is large and does not exactly represent the source performance. In this situation errors in representing a novel source frame can sometimes be reduced with large counterbalanced positive and negative weights, which results in a poor shape when the weights are reused on the target. Instead, they require the weights to be non-negative. This prevents the previously described situation because negative weights are not available, and results in demonstrably better cross-mapping even though the error in representing the source is somewhat higher.

Geometric mapping. Using a blend shape system is not the only way to drive a synthetic face through performance capture. For instance, good results can also be achieved using radial basis functions [9]. Noh and Neumann [12] propose a different approach, called “Expression Cloning”, that does not rely on blend shapes. Their technique assumes that the source performance is given as an animated mesh (i.e. the topology and motion curves for every vertex). Their goal is to transfer the deformations of the source mesh onto the target mesh.

The first step of the algorithm is to find geometric correspondences between the source and target models. This is done by computing a sparse set of correspondences that are propagated to the rest of the mesh using scattered data interpolation (radial basis functions). The sparse correspondences are determined either manually or using some face-specific heuristics.

Once the two models are brought into dense correspondence the motion vectors (offsets from the initial or rest expression) can be transferred. This transfer is performed by assigning a local coordinate system to each vertex in the source and target models. These coordinates systems are determined by the normal of the mesh at that vertex. Transferring a motion vector can then be done by changing local coordinate systems. The motion can also be locally scaled by using the ratio of locally defined bounding boxes in the two models. An additional procedure takes care of the special case of the lip contact line and prevents any spurious interactions between the two lips.

Expression/Style learning. Wang et. al. [11] describe an ambitious machine-learning based system for cross-mapping. A data reduction manifold learning technique (local linear embedding, LLE) is first used to derive a mapping from animated expression geometry over time to a one dimensional manifold (curve) embedded in a low-dimensional (e.g. 3D) space. They then establish correspondences between curves for a given expression over different individuals (this includes different monotonic reparameterizations of cumulative length along the curve). Once the correspondences are established, the registered curves are averaged to produce a mean manifold for the particular expression. Evolution of the expression over time now corresponds to movement along this curve.

Next a mapping from the curve back to the facial geometry is constructed. First a mapping from points on the expression curve back to the actor’s changing facial geometry is obtained using an approximating variant of radial basis scattered interpolation. This mapping conflates the different “styles” of facial expressions of different people. Lastly, using the bilinear decomposition approach introduced in graphics by Chuang et. al. [5], the changing expression geometry is factored into components of facial expression and individual identity (thus, for a particular facial expression there is a linear model of how various individuals effect that frozen expression, and for any individual the evolution of the expression over time is also a linear combination of models).

Although the system deals with each facial expression in isolation, it hints at future advances in deriving useful higher level models from data.

Unexplored issues

Expression vs. motion cross-mapping. The different techniques we have described treat the cross-mapping issue as a geometric problem where each frame from the recorded performance is deformed to match the target character. Unfortunately, this might not respect the dynamics of the target character. To go beyond a straight per frame cross-mapping requires an approach that takes timing into account. There are basically two ways this can be tackled: using a physical approach or a data-driven approach.

A physically-based animation system could provide physical constraints for the target face. The cross-mapping algorithm would have to satisfy two types of constraints: matching the source performance but also respecting the physics of the target face. By weighting these constraints an animator could control how much of the source performer versus how much of the target character appears in the final animation.

Any data-driven approach must be carefully designed to minimize the "curse of dimensionality" issues introduced by additional dimensions of timing and expression. One approach might involve building a (small) database of motions for the target character. The performer could then act these same motions to create a corresponding source database. Using machine learning or interpolation techniques these matching motions could provide a time-dependent mapping from the source motion space to the target motion space.

Facial puppeteering. The human face can express a wide gamut of emotions and expressions that can vary widely both in intensity and meaning. The issue of cross-mapping raises the more general issue of using the human face as an animation input device not only for animating digital faces but any expressive digital object (e.g. a pen character does not have a face). This immediately raises the issue of "mapping" the facial expressions of the performer onto meaningful poses of the target character. Dontcheva et al. [7] tackles this issue in the context of mapping body gesture onto articulated character animations.

References

[1] Ian Buck, Adam Finkelstein, Charles Jacobs, Allison Klein, David H. Salesin, Joshua Seims, Richard Szeliski, and Kentaro Toyama. Performance-driven hand-drawn animation. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 101–108, June 2000.

[2] Martin D. Buhmann. *Radial Basis Functions : Theory and Implementations*. Cambridge University Press, 2003.

[3] Byoungwon Choe, Hanook Lee, and Hyeong-Seok Ko. Performance-driven muscle-based facial animation. *The Journal of Visualization and Computer Animation*, 12(2):67–79, May 2001.

[4] E. Chuang and C. Bregler. Performance driven facial animation using blendshape interpolation. *CS-TR-2002-02, Department of Computer Science, Stanford University*, 2002.

[5] E. Chuang, H. Deshpande, and C. Bregler. Facial expression space learning. In *Proceedings of Pacific Graphics*, 2002.

[6] M. Covell and C. Bregler. Eigen-points. In *Proc. IEEE International Conference on Image Processing*, pages vol 3 p 471–474, 1996. Lausanne, Switzerland, Sept 16-19 1996.

[7] Mira Dontcheva, Gary Yngve, and Zoran Popović. Layered acting for character animation. *ACM Transactions on Graphics*, 22(3):409–416, July 2003.

[8] Tim Hawkins, Andreas Wenger, Chris Tchou, Andrew Gardner, Fredrik Göransson, and Paul Debevec. Animatable facial reflectance fields. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*, pages 309–320, June 2004.

[9] J. Noh, D. Fidaleo, and U. Neumann. Animated deformations with radial basis functions. In *ACM Symposium on Virtual Realisty Software and Technology*, pages 166–174, 2000.

[10] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D.H. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH 98 Conference Proceedings*, pages 75–84. ACM SIGGRAPH, July 1998.

[11] Y. Wang, X. Huang, C.-S. Lee, S. Zhang, D. Samaras, D. Metaxas, A. Elgammal, and P. Huang. High resolution acquisition, learning, and transfer of dynamic 3-d facial expressions. In *Eurographics*, 2004.

[12] Jun yong Noh and Ulrich Neumann. Expression cloning. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 277–288, August 2001.

Implementing a skin BSSRDF (or several...)

Christophe Hery
Industrial Light + Magic

Introduction

In the seminal paper from 2001: *A Practical Model for Subsurface Light Transport* ([Jensen '01]), Henrik Wann Jensen et al employed the concept of a BSSRDF (a bidirectional surface scattering distribution function) as a means to overcome the limitations of BRDFs.

The following year, in *A Rapid Hierarchical Rendering Technique for Translucent Materials* ([Jensen '02]), Jensen and Buhler presented a caching method to accelerate the computation of the BSSRDF.

We will try here to give pointers about how to implement such systems, as well as how to use them in production.

1 BSSRDF 101

Skin is a multi-layered medium, in which photons tend to penetrate and scatter around many many times before bouncing back out. Certain wavelengths are attenuated differently according to the thickness. The path the photons take is highly complex and depends on the nature of the substrates as well as their energy. But one thing is almost certain: they rarely leave through the location where they entered, which is why a BSSRDF - a model for transport of light through the surface - is needed.

The full BSSRDF system presented by Jensen et al. consists of two terms: a single scattering component, which through path tracing gives us an approximate solution for the cases where light bounced only once inside the skin, and a multiple scattering component, through a statistical dipole point source diffusion approximation. The beauty of this model is that it can be implemented in a simple ray-tracer, and with the use of some trickery, in a Z-buffer renderer.

We are now going to go over the papers and extract the relevant information for shader writers, so please have your Proceedings handy.

1.1 Single scattering

We are trying to consider the different paths photons could take such that they would exit towards the camera at the shaded point after one unique bounce inside the skin. Since we know the camera viewpoint $\underline{\mathsf{I}}$ at the shaded surface position $\underline{\mathsf{P}}$, we can compute the refracted outgoing direction T_o. So let's select some samples P_{samp} along T_o

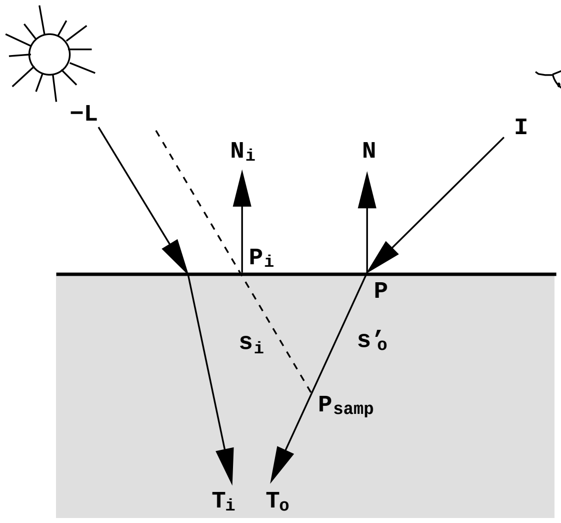


Figure 1: For single scattering, we march along the refracted outgoing ray T_o and project towards L.

(we'll figure out the density and the placement of those samples later), and let's try to find out how the light could have bounced back at those positions.

On the way in, the light should have refracted at the surface boundary. As mentioned in [Jensen '01], this is a hard problem for arbitrary geometry, and we make the approximation that the light did not refract. With this simplification, we can now find for each P_{samp} where the light first entered by simply tracing towards the light source.

In fact, we will not really trace towards the source position. Given the distances involved, we can consider the light to be uniform in direction and in color/intensity over the surface. So we simply trace along $\underline{\mathsf{L}}$ (the incident light direction at $\underline{\mathsf{P}}$) and multiply our result globally with $\underline{\mathsf{Cl}}$ (light energy received at $\underline{\mathsf{P}}$).

At this stage, we have: the distance the light traveled on the way in (s_i), and the distance the light traveled going out (s'_o). Prime signs here indicate, where appropriate, the refracted values. For instance, s'_o is the distance along the refracted outgoing ray, T_o, and it is a given, since we originally explicitly placed our sample P_{samp} along T_o from $\underline{\mathsf{P}}$ (in other words, $\mathsf{P}_{samp} = \underline{\mathsf{P}} + \mathsf{T}_o s'_o$).

The outscattered radiance for the single term, L_o, depends on the scattering coefficients, the phase function between the refracted directions, and some exponential falloff terms on s'_i and s'_o. But we do not know s'_i, the refracted distance before the bounce. The trick here is to use Snell's law to derive s'_i from s_i.

The other thing we are missing is the phase function. What is it? Simply stated, the phase function between two vectors $\vec{v}_1$ and $\vec{v}_2$ is a way to describe the amount of light scattered from the direction $\vec{v}_1$ into the direction $\vec{v}_2$. Knowing fully that we are not simulating, in CG, the light interactions one photon at a time, this phase becomes a probability density function (PDF), and for skin, it has been shown (for instance in [Pharr '01]) that the Henyey-Greenstein model is adequate.

Finally, we can try to make the solution converge faster. A naive implementation would have all P_{samp} uniformly selected along T_o. But this would require many many samples.

Let's look at L_0:

$$L_o(x_o,\vec{\omega}_o) = \frac{\sigma_s(x_o)Fp(\vec{\omega}_i'\cdot\vec{\omega}_o')}{\sigma_{tc}}e^{-s_i'\sigma_t(x_i)}e^{-s_o'\sigma_t(x_o)}L_i(x_i,\vec{\omega}_i).$$

This expression can clearly be rewritten as the product of the exponential falloff in s_o' with another function:

$$L_o(x_o,\vec{\omega}_o) = [L_i(x_i,\vec{\omega}_i)\frac{\sigma_s(x_o)Fp(\vec{\omega}_i'\cdot\vec{\omega}_o')}{\sigma_{tc}}e^{-s_i'\sigma_t(x_i)}]e^{-s_o'\sigma_t(x_o)}.$$

If we now pick the samples according to the PDF

$$x \sim \sigma_t e^{-\sigma_t x},$$

the integration in L_0 can be approximated by a simple summation. This is called importance sampling. By choosing

$$s_o' = \frac{-\log(random())}{\sigma_t},$$

we respect our PDF, and we can sum up all contributions without explicitly computing the falloff term in s_o'.

In pseudo shading language code, this becomes:

```
float singlescatter = 0;

vector To = normalize(refract(I,N,oneovereta));

for (i=0; i<nbrsamp; i+=1)
{
    float sp_o = -log(random())/sigma_t;
    point Psamp = P + To * sp_o;
    point (Pi,Ni) = trace(Psamp, L);
    float si = length(Psamp - Pi);
    float LdotNi = L.Ni;
    float sp_i = si * LdotNi
        / sqrt (1 - oneovereta*oneovereta * (1 - LdotNi*LdotNi));
    vector Ri, Ti; float Kri, Kti;
    fresnel (-L, Ni, oneovereta, Kri, Kti, Ri, Ti);
    Kti = 1-Kri;
    Ti  = normalize(Ti);
    float g2 = g*g;
    float phase = (1-g2) / pow(1+2*g*Ti.To+g2,1.5);
    singlescatter += exp(-sp_i*sigma_t) / sigma_tc * phase * Kti;
}

singlescatter *= Cl * PI * sigma_s / nbrsamp;
```

There is also a fresnel outgoing factor, but since it is to be applied for both the single and the diffusion terms, we are omitting it in this section.

It is that simple (in fact it is made simpler here, because we only deal with one wavelength)!

Lastly, note the trace call. It returns the position, the normal and potentially the textured diffuse color of the intersection of the ray going from P_{samp} in the $\underline{\mathsf{L}}$ direction. We will come back to it in section 3.

1.2 Diffusion scattering

We are trying to accumulate the observed profile of illumination over all positions on the surface. The dipole of point sources is a virtual construction to simulate the light attenuation on skin reported by medical scientists (the derivation of this model is beyond the scope of our discussion). The value we are summing up, at each sample, is the diffuse reflectance:

$$R_d(r) = \frac{\alpha'}{4\pi}[z_r(\sigma_{tr}d_r + 1)\frac{e^{-\sigma_{tr}d_r}}{d_r^3} + z_v(\sigma_{tr}d_v + 1)\frac{e^{-\sigma_{tr}d_v}}{d_v^3}]$$

Its expression mainly depends on the distance r from the shaded point to the sample, and also (directly and indirectly) on the scattering coefficients. The problem is once again the distribution of those samples. In fact, in [Jensen '02], a two-pass predistribution technique was shown (and we will come back to it in section 4). For now, let's assume that we want to place those samples at their appropriate locations during shading time. They need to lie on the surface and be located around $\underline{\mathsf{P}}$. The $\sigma_{tr}e^{-\sigma_{tr}d}$ terms are likely going to dominate the computation of R_d (both in times and in values). So we should try to put in a lot of effort in order to make them converge faster. Again we turn to the importance sampling theory, and we choose a PDF

$$x \sim \sigma_{tr}^2 e^{-\sigma_{tr}x}.$$

Why do we have a σ_{tr}^2 coefficient this time (where our PDF for single scattering had a simple σ_t multiplicator)? Well, we need to integrate in polar coordinates (whereas for single scattering we were marching along T_o in 1D), and a condition for a valid PDF is that its integral over the full domain comes to 1 (meaning that we have 100% chance to find a sample in the infinite range). This PDF corresponds to the cumulative density function (CDF)

$$P(R) = \int_0^R \sigma_{tr}^2 e^{-\sigma_{tr}r} r dr = 1 - e^{-\sigma_{tr}R}(1 + \sigma_{tr}R)$$

By the change of variable $u = \sigma_{tr}R$, we get

$$P(u) = 1 - e^{-u}(1 + u),$$

and through a numerical inversion of $P(u)$, we get a table of precomputed positions, that we can insert in the shader (we provide a C program in the Appendix that warps a uniform Poisson distribution into the one we want).

Again let's deal with the pseudo-code for one wavelength.

```
float diffusionscatter = 0;

uniform float scattdiffdist1[maxSamples] = { /* X table */ };
uniform float scattdiffdist2[maxSamples] = { /* Y table */ };

/* create a local frame of reference for the distribution */
vector  local = N;
vector  up    = vector "world"(0,1,0);
if (abs(local.up) > 0.9)
    up    = vector "world"(1,0,0);
vector  base1 = normalize(local^up);
vector  base2 = local^base1;

for (i=0; i<nbrsamp; i+=1)
{
    point Psamp = P + 1/sigma_tr *
        (base1 * scattdiffdist1[i] + base2 * scattdiffdist2[i]);
    point (Pi,Ni) = trace(Psamp, -local);
        /* make sure we are on the surface */

    float r = distance(P, Pi);
    float zr = sqrt(3.0*(1.0-alpha_prime)) / sigma_tr;
    float zv = A * zr;
    float dr = sqrt(r*r+zr*zr); /* distance to positive light */
    float dv = sqrt(r*r+zv*zv); /* distance to negative light */
    float sigma_tr_dr = sigma_tr*dr;
    float sigma_tr_dv = sigma_tr*dv;
    float Rd = (sigma_tr_dr+1.0) * exp(-sigma_tr_dr) * zr/(dr^3)
             + (sigma_tr_dv+1.0) * exp(-sigma_tr_dv) * zv/(dv^3);

    scattdiff += L.Ni * Rd / (sigma_tr*sigma_tr * exp(-sigma_tr*r));
        /* importance sampling weighting */
}

scattdiff *= Cl * (1-Fdr) * alpha_prime / nbrsamp;
```

Please be aware that, in this example, the distribution was done on the tangent plane at $\underline{P}$. Depending on the chosen tracing approach to reproject the samples on the actual geometry, this might not be the best solution.

2 Texture mapping and reparameterization

So far we have considered a shader that depends on some scattering coefficients. In reality, we need to texture them over our geometry. Painting them directly is non-trivial, and especially non-intuitive. In production, we tend to start from photographs of our live characters or maquettes. The question is: can we extract the variation of the relevant parameters from regular diffuse texture maps?

We are going to make some simplifications to make this conversion. First, we will assume that the influence of the single scattering is negligible compared to the diffusion term ([Jensen '02] provides a justification for skin materials). So let's assume that our diffuse texture map is the result of the diffusion scattering events under a uniform incident illumination, and try to solve for the scattering parameters which produce these colors.

In this case, we can approximate our BSSRDF as a BRDF:

$$R_d = \frac{\alpha'}{2}(1 + e^{-\frac{4}{3}A\sqrt{3(1-\alpha')}})e^{-\sqrt{3(1-\alpha')}},$$

which only depends on α' and A (indirectly η). This is also equal, by construction, to our diffuse map $\mathsf{C}_{\mathsf{map}}$. So we now have one equation with two unknowns: α' and η. If we fix η (at 1.3 for skin), we can get α' as a function of $\mathsf{C}_{\mathsf{map}}(= R_d)$. In truth, we cannot do this analytically, but we can build yet another table numerically.

We used the following code in Matlab to produce the table:

```
clear;
eta = 1.3;
Fdr = -1.440/(eta.^2) + 0.710/eta + 0.668 + 0.0636*eta;
A = (1 + Fdr)/(1 - Fdr);
alpha = 0:.01:1;
c = alpha.*(1+exp(-4/3*A*sqrt(3*(1-alpha))))
         .* exp(-sqrt(3*(1-alpha)));
C = 0:.001:2;
ALPHA = interp1(c,alpha,C);
```

And, in the shader, we can lookup those values in the following manner:

```
uniform float alpha_1_3[2001] = { /* put matlab table here */ };
float alpha_prime = alpha_1_3[floor(diffcolor*2000.0)];
```

Because α' is the reduced transport albedo, namely the ratio:

$$\alpha' = \frac{\sigma_s'}{\sigma_a + \sigma_s'},$$

this does not resolve directly into the scattering coefficients.

Even though σ'_s is given for some specific materials in [Jensen '01], it is still a very non-intuitive parameter, hence we replace our tuning factor with the diffuse mean free path $ld = 1/\sigma_{tr}$. It turns out that σ_{tr} and α' are all that is needed to control the diffusion scattering. For the single scattering, we can get:

$$\sigma'_t = \frac{1}{ld\sqrt{3\,(1-\alpha')}}$$

and

$$\sigma'_s = \alpha'\,\sigma'_t.$$

Lastly, it is worth mentioning that, ideally, the color inversion should be done at every sample for diffusion scattering, so that we get the correct R_d on those positions. For single scattering, the dependence on the inversion during the ray marching is embedded in the σ_{tc} denominator and σ_t term in the exponential.

3 BSSRDF through Z-buffers

If we can manage to *trace* only along the light direction, we can achieve a projection towards $\underline{\mathsf{L}}$ by some simple operations on the shadow buffers, of the kind:

```
uniform matrix shadCamSpace, shadNdcSpace;
textureinfo(shadMap, "viewingmatrix", shadCamSpace);
textureinfo(shadMap, "projectionmatrix", shadNdcSpace);
uniform matrix shadInvSpace = 1/shadCamSpace;
point Pi       = transform(shadCamSpace, Psamp);
point tmpPsamp = transform(shadNdcSpace, Psamp);
float shadNdcS = (1.0+xcomp(tmpPsamp)) * 0.5;
float shadNdcT = (1.0-ycomp(tmpPsamp)) * 0.5;
float zmap = texture(shadMap,
                     shadNdcS, shadNdcT, shadNdcS, shadNdcT,
                     shadNdcS, shadNdcT, shadNdcS, shadNdcT,
                     "samples", 1);
if (comp(shadNdcSpace, 3, 3) == 1) /* orthographic */
    setzcomp(Pi, zmap);
else
    Pi *= zmap/zcomp(Pi);
Pi = transform(shadInvSpace, Pi);
```

To access any other values at P_i (like N_i), if we produced an image of those values from the point of view of the light (for instance through a secondary display output), we simply lookup the texture, as in:

```
color CNi = texture(shadCNMap,
                    shadNdcS, shadNdcT, shadNdcS, shadNdcT,
                    shadNdcS, shadNdcT, shadNdcS, shadNdcT,
                    "samples", 1);
normal Ni;
setxcomp(Ni, comp(CNi,0));
setycomp(Ni, comp(CNi,1));
setzcomp(Ni, comp(CNi,2));
Ni = ntransform(shadInvSpace, Ni);
```

Of course, if we want to emulate the soft shadows cast from a non-point light source, we can super-sample our `shadNdcS` and `shadNdcT` lookup coordinates or any relevant tracing value in the shader.

This overall tracing method drops right in for single scattering (the trace is indeed along $\underline{\mathsf{L}}$), but it is more complex to make it work for diffusion scattering.

Let's think about it for a minute. We need to place some samples on the surface according to a specific distribution. Our assumption was that we would first position them on the tangent plane at $\underline{\mathsf{P}}$, then project them (along $-\underline{\mathsf{N}}$) to the surface. With our Z-buffers, all we have is light projections. So what if we distributed our samples in shadow space, then projected along $-\underline{\mathsf{L}}$? The surface distribution would definitely be skewed. Our importance sampling mechanism would not converge as fast as expected (it would still be faster than a simple uniform distribution though). In fact, by doing it this way, it is almost as if we would be building implicitly the $\underline{\mathsf{L}} \cdot \underline{\mathsf{N}}$ product. We should then just ”correct” the final result by omitting this factor.

In our pseudo-code from section 3, we just change:

- `vector local = N` into `vector local = L`,
- the trace with the method exposed above,
- the last line of the loop into `scattdiff += Rd / (sigma_tr*sigma_tr * exp(-sigma_tr*r));`

In practice, this twist on the theory works really well.

Ideally, we need 3 distributions (one per wavelength), ie 3 sets of fake trace Z-buffer projections. Not only is it a slow process (after all, `texture` is one of the most expensive operators in the shading language), but doing it that way creates a separation of the color channels, hence a very distracting color grain. The remedy is to do only one distribution with the minimum σ_{tr} (the wavelength that scatters the most).

Finally, since we use the shadow buffers as a way to represent the surface of our skin, we should not include in them any other geometry, like clothing or anything casting shadows from an environment (more on that later at paragraph 5.1). For the same reason, the resolution of the Z-buffers matters a great deal.

4 BSSRDF through a point cloud cache

As we said before (in section 1.2), the diffusion scattering term largely dominates the BSSRDF in the case of human skin rendering, to the point where we should think about dropping single scatter altogether from our model of skin. Single scatter can be made to produce good results though. One can balance it, tweak it, blur it, contain it with some area maps, so that its appearance is pleasing. However, from its very nature of being a directional term, the transitions of the illumination (especially the transitions from back lighting to front lighting) are very hard. For other materials with more isotropic scattering properties (scattering eccentricity closer to 0), such as marble, it becomes more critical to keep this term. In any case, when we speak about a skin BSSRDF from this section on, we will only consider the diffusion scattering.

Since the BSSRDF computation is fairly expensive, it is worth looking at acceleration methods. One observation we can make is that, no matter the pattern of distributions of the dipole samples, they are likely going to be extremely close from one shaded point to the next, yet we do indeed recompute their positions for each $\underline{\mathsf{P}}$. So one idea here would be to pre-distribute the samples on the surface, this time uniformly, then ”gather” them during the scattering loop. This is what [Jensen ’02] demonstrates.

There are several approaches one can take for this pre-distribution phase. One can build fancy Xsi scripts or mel commands that could achieve, through particle repulsion algorithms, a stabilized semi-Poisson set on the surface. One can implement a standalone Turk method ([Turk 1992]) as suggested in [Jensen ’02]. Or one can take advantage of a specific rendering engine. For example, in the case of Pixar’s Photorealistic Renderman, one can derive the positions directly from the micro-polygon grids through a dso (a la Matt Pharr’s `http://graphics.stanford.edu/∼mmp/code/dumpgrids.c`), or more directly with the native bake3d() shadeop. All in all, there are certainly many more ways to obtain a similar point cloud representation of the geometry.

So let’s assume that we managed to pre-generate this distribution, and that it is stored in a file for which we know the format. The relevant information for each sample should be its location, the normal, the coverage area it represents - in other words its local sampling density (this is necessary in the cases for which our distribution is not exactly uniform) - and, if possible, the diffuse texture at that position.

The first thing to do will be to compute the diffuse irradiance at each point, and store it back in the data structure. Again there are several ways to do this (in fact, if we opted to do the distribution through Matt Pharr’s method or with bake3d(), the value can already be present in the file, and we can step over this stage).

One hacky approach is to read the file one sample at a time through a shader DSO, then do an illuminance loop on the position and the normal just obtained, compute the $\underline{\mathsf{L}} \cdot \mathsf{N}_{cache}$ factor, and write the whole thing back. This is the code for such a scheme:

```
surface lightbake_srf
    (
    string inputScattCache  = "";
    string outputScattCache = "";
    float doFlipNormals = 0.0;
    )
{
    if (opencache(inputScattCache) == 1) {
        float index = indexcache();
        while(index >= 0.0) {
            point  cacheP = getPcache(index);
            normal cacheN = getNcache(index);
            cacheP = transform("world", "current", cacheP);
            if (doFlipNormals == 1)
                cacheN = ntransform("world", "current", -cacheN);
            else
                cacheN = ntransform("world", "current", cacheN);
            illuminance(cacheP, cacheN, PI/2) {
                float LdotN = L.cacheN;
                if (LdotN > 0.0) {
                    accumcache(index, Cl * LdotN / length(L));
                }
            }
            index = indexcache();
        }
        writecache(outputScattCache);
    }
}
```

This is a pretty weird surface shader indeed: apart from the illuminance statement, it uses only DSO calls to access our file format. The cache is opened and kept in as a static member of the DSO at the opencache stage, so that we can look up any specific position quickly. We are using an index as a kind of a pointer to the current sample, so all information about it can be read and written correctly.

A word of caution here: the light shaders need to be modified to do their own point sampling on the shadows; otherwise, the derivatives, as far as Renderman is concerned, will come from the successive cacheP values (which can jump all over the place, depending on how they are stored). Here is another trick. Instead of assigning the lightbake shader to a random visible piece of geometry, we use an RiPoint primitive, which has no derivatives by construction. The shadow calls are thus forced to point sample the buffers, without any editing of the light code. Another side benefit is that the overall computation is much faster this way.

To obtain the final beauty BSSRDF result, in yet another DSO, we read our cache file and store the data in memory as an octree for quick lookup. Given a search radius R, we can sum up the R_d contributions from each sample found in the proximity region around $\underline{\text{P}}$. This is the same R_d computation as presented in section 1-b:

```
float Rd = (sigma_tr_dr+1.0) * exp(-sigma_tr_dr) * zr/(dr^3)
         + (sigma_tr_dv+1.0) * exp(-sigma_tr_dv) * zv/(dv^3);
```

But we want to pre-multiply it by the correct precomputed radiance at each sample and its coverage area. Also, if we have the color information in the file, we can re-derive σ_{tr}, zr and zv through the alpha inversion as mentioned in paragraph 2. Note that the surface shader does not have to do any queries of the lights or any *tracing* anymore; it just needs to call the DSO.

This scatter gathering stage can still be fairly slow, as the cache might contain several thousands samples (to accurately represent the geometry and local details in the illumination and textures), so we use a level-of-detail approach. We generate some cached distribution at several densities. We search locally in the finer cache, and then, as the exponential falloffs in R_d tend towards 0, we look up in the more sparse files. This way, we can minimize the number of returned sample positions. [Jensen '02] describes a more formal data structure for this kind of optimization.

Alternatively, as recommended in Pixar's documentation, the dipoles can be summed on the point clouds themselves, through a utility like *ptfilter -ssdiffusion*. This is a highly efficient solution, as one does not incur the cost of integration at shading time, but its accuracy is as good as the density of the samples: as outlined in section 2, the alpha inversion can only be resolved on those points, thus potentially producing lower resolution texturing.

5 Extensions

5.1 Ambient occlusion, global illumination and cast shadows

Obviously the Z-buffer approach requires some sort of direct illumination, with some actual light sources. At first this makes it seem impossible to get an ambient occlusion (see last year's notes) contribution to participate in the scattering. Even though one can still do the ambient as a separate term in the shader, there is a sneaky way to merge the two together. The idea is to compute the occlusion, not through a shadeop or a ray-tracing pre-pass, but with a dome of lights. This method has recently been described on the web at: `http://www.andrew-whitehurst.net/amb_occlude.html`, so we are not going to cover it here.

Similarly, the skin position is assumed to be at the Z-buffer locations, so it does not seem easy to get correct cast shadows on our subject (we would not want to make the algorithm think that our surface is feet away from where it really is, right?). The solution to that is to generate a separate "outer" shadow buffer for each light, and scale the final result based on it.

On the other hand, there is no such limitation on the point cloud method of computing the scattering. Because we light the samples as a pre-pass, we can use any type of illumination algorithms on them, even any type of rendering engine, ahead of accumulating the dipoles in the final render.

For instance, we discovered that indirect diffuse computations have an impact on the look of the skin, and that they enhance the effects of the subsurface scattering. Ears tend to be a tiny bit too dark if not backlit, unless one first simulates at least one bounce of diffuse illumination.

5.2 Concept of blocking geometries

Up to this point the shader hasn't solved a small structural problem. It doesn't take into account the fact that inside the skin there are bones that are basically blocking some of the scattering effect. This extension is critical for small creatures and/or highly translucent materials, where one wants to give the impression of an obscuring entity inside the medium.

With the Z-buffer approaches, this is pretty trivial. We just need to generate yet another shadow buffer from the point of view of the scattering lights, a buffer of those inner blocking geometries. Then we can point sample it during the loops to get a z_{inner} value. By comparing our projected z_{map} and original sample point zcomp(P_i) with this z_{inner}, we can decide if and how much the sample participates in the illumination. This method is valid for both single and diffusion scattering.

If our BSSRDF is based on the point cloud approach, we can somehow simulate this effect by distributing samples on the inner geometries, summing the diffusion scattering they generate as seen from our skin shading point P, then subtracting this influence (or part of it) from our real surface diffusion scattering computation. This trick is certainly non-physical, but it seems to produce a pleasing effect.

6 Summary

We presented a set of techniques for implementing a production ready sub-surface scattering model in the shading language.

Acknowledgments

We would like to thank the following people who all contributed one way or another in turning the theory into a practical solution. Dan B Goldman, Andrea Maiolo and Hilmar Koch for their shading savviness, Tony Pelle for his ”pointscatter” tool and its viewer, Pixar and the Renderman team for integrating some of these methods natively, and the entire ILM R&D department for supporting this effort and letting us publish it.

makeScatAlphaTables.C:

```
//   utility to generate a table corresponding to the distribution:
//   1 - exp(-r) * (1+r)

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

static const   int numSampleTables = 1;
static const   int maxSamples = 128;
static float     sampleX[numSampleTables][maxSamples];
static float     sampleY[numSampleTables][maxSamples];

static void randomPoint (float &u, float &v)
{
   do {
         u = drand48() - 0.5;
         v = drand48() - 0.5;
   } while (u * u + v * v > 0.25);
}

// First use the best candidate algorithm from [Mitchell 1991]:
// Spectrally Optimal Sampling for Distribution Ray Tracing
// Don P. Mitchell, Siggraph Proceedings 1991

void initSampleTables()
{
   const int q = 10;

   for (int table = 0; table < numSampleTables; table++) {

         randomPoint (sampleX[table][0], sampleY[table][0]);

         for (int i = 1; i < maxSamples; i++) {

            float dmax = -1.0;

            for (int c = 0; c < i * q; c++) {

                  float u, v;
                  randomPoint (u, v);

                  float dc = 2.0;

                  for (int j = 0; j < i; j++) {

                     float dj =
                           (sampleX[table][j] - u) * (sampleX[table][j] - u) +
                           (sampleY[table][j] - v) * (sampleY[table][j] - v);
                     if (dc > dj)
                           dc = dj;
                  }

                  if (dc > dmax) {
                     sampleX[table][i] = u;
                     sampleY[table][i] = v;
```

```
                        dmax = dc;
                    }
                }
            }
        }
        return;
}

// Now take the uniform distribution and warp it into ours

inline float distFunc(float r)
{
    return(1.0-exp(-r)*(1.0+r));
}

float invDistFunc(float y, float tolerance)
{
    float test, diff;
    float x = y;
    while(1) {
        test = distFunc(x);
            diff = y - test;
            if (fabsf(diff) < tolerance) break;
            x = x + diff;
    }
    return(x);
}

void adaptSampleTables()
{
    float PI = fasin(1.0)*2.0;
    for (int i = 0; i < numSampleTables; i++) {
            for (int j = 0; j < maxSamples; j++) {
                float X = 2.0*sampleX[i][j]; // between -1 and 1
                float Y = 2.0*sampleY[i][j];
                float R = fsqrt(X*X+Y*Y);    // between 0 and 1
                float r = invDistFunc(R,.00001);
                float theta = fatan2(Y,X);   // between -PI and PI
                sampleX[i][j] = fcos(theta)*r;
                sampleY[i][j] = fsin(theta)*r;
            }
    }
}

// Finaly print the resulting tables

void printSampleTables()
{
    int i, j;

    printf ("uniform float scattdiffdist1[maxSamples] = {\n");
    for (i = 0; i < numSampleTables; i++) {
            for (j = 0; j < maxSamples; j++) {
                printf ("%f", sampleX[i][j]);
                if (j < maxSamples - 1)
                        printf (",");
            }
        if (i < numSampleTables - 1)
            printf (",\n");
    }
    printf ("\n};\n");

    printf ("uniform float scattdiffdist2[maxSamples] = {\n");
    for (i = 0; i < numSampleTables; i++) {
        for (j = 0; j < maxSamples; j++) {
            printf ("%f", sampleY[i][j]);
            if (j < maxSamples - 1)
                printf (",");
        }
        if (i < numSampleTables - 1)
            printf (",\n");
    }
    printf ("\n};\n");
}

void main()
{
    initSampleTables();
    adaptSampleTables();
    printSampleTables();
    exit (0);
}
```

Human Face Project

Walter Hyneman
Sony ImageWorks

Hiroki Itokazu
Walt Disney Feature Animation

Lance Williams
Applied Minds, Inc.

Xinmin Zhao
Walt Disney Feature Animation

Abstract

"The Human Face Project" is a short film documenting an effort at Walt Disney Feature Animation to track and animate human facial performance, which was shown in the SIGGRAPH 2001 Electronic Theater. This short paper outlines the techniques developed in this project, and demonstrated in that film.

The face tracking system we developed is exemplary of model-based computer vision, and exploits the detailed degrees of freedom of a geometric face model to confine the space of solutions. Optical flow and successive rerendering of the model are employed in an optimization loop to converge on model parameter estimates. The structure of the model permits very principled mapping of estimated expressions to different targets.

Of critical importance in media applications is the handling of details beyond the resolution or degrees of freedom of the tracking model. We describe behavioral modeling expedients for realizing these details in a plausible way in resynthesis.

Keywords: 3D facial animation, optical flow, optimization, visual servo, model-based coding.

1 Introduction

One of the oldest paradigms in the field of computer vision is construction of a three dimensional geometric model consistent with a two dimensional image, or registration of an existing model with an image or sequence of images. This was the approach undertaken by Larry Roberts in his pioneering work at Lincoln Labs [Roberts63]. Bruce Baumgart, whose wok in modeling has been very influential in the computer graphics community, extended Roberts' approach considerably, and aptly summarized it in an early paper [Baumgart75] "My approach to computer vision is best characterized as inverse computer graphics. In computer graphics, the world is represented in sufficient detail so that the image forming process can be numerically simulated to generate synthetic television images; in the inverse, perceived television pictures (from a real TV camera) are analyzed to compute detailed geometric models."

Subsequently, David Lowe [Lowe80] used "inverse computer graphics" to develop both academic and industrial robot vision systems. In the field of industrial robotics, the control of an effector using an electronic camera is temed, "visual servo," and an ample literature exists in this active area of model-based computer vision [Hutchinson96].

Fred Parke's pioneering work in 3D computer facial animation took place at the University of Utah in the early 1970's. Among the possible applications of his work, Parke mentioned the possibility of a kind of animated telecommunication, in which animation parameters could be transmitted to drive a CG model at the receiving end [Parke 82]. This has in fact been one of the more heavily-researched application areas for computer animation aside from flight simuylation, animated entertainment, and video games. Development has also continued in computer graphics and vision to enable model-based face tracking for general purposes [Essa94] [Pighin99].

Such use of computer graphic models (typically head-and-shoulders models of human beings) in image bandwidth compression, has led to increasingly refined methods for estimating model parameters that most closely match the video stream to be encoded. In such applications it is only natural to enlist renderings of the model in an iterative fitting process (e.g. [Girod97]).

The Human Face Project was based on this technique, and was informed by three principal areas of research and development: facial animation in computer graphics, "visual servo" in robotics, and model-based video codecs in picture bandwidth compression. Our particular application, motion picture visual effects, brought a particular emphasis to our implementation: performance capture with high detail, and high fidelity to a particular actor. Accuracy is much more important than speed in this application.

2 Model data

Our effort began with data collection of our principal performer, Price Pethel, a studio VP with a professional background in live action film, video, and visual effects, who was an active participant in the project for its duration. We taped video studies of his changing expressions, with features marked, and acquired 22 CyberWare 3D laser scans of his face in a gamut of extreme expressions (some examples: [Figure 1]).

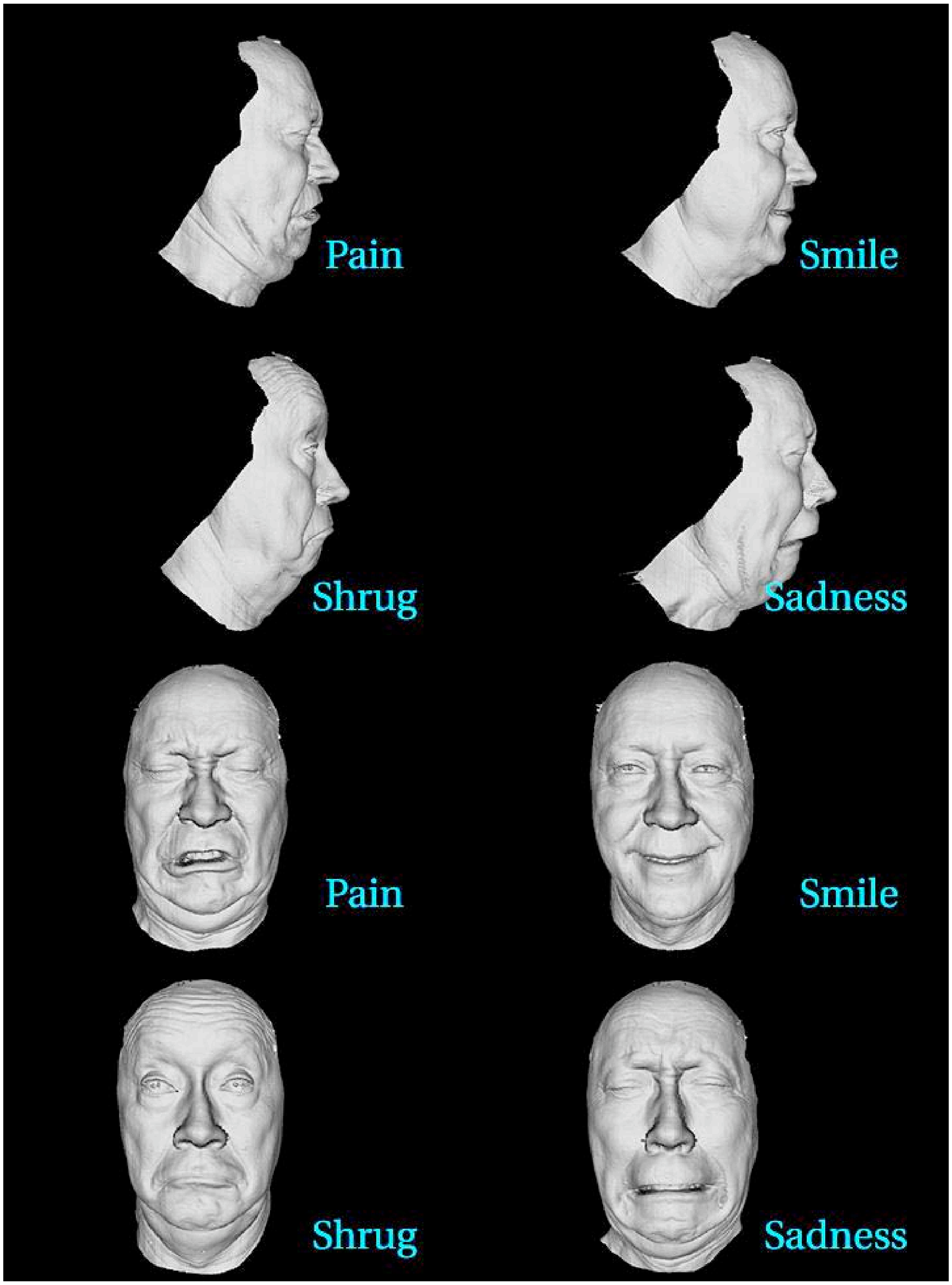


Figure 1. Four scanned expressions, front and side views.

For texture mapping, and modeling of fine details of the face, visual effects supervisor Hoyt Yeatman set up a system of synchronized 35mm film cameras with polarizing filters, and an enclosing set of synchronized strobe lights in diffusion boxes with cross-polarizing filters. Fast exposures and diffuse, uniform illumination allowed us to capture texture with no apparent lighting "baked in" (as in [Marschner99]). The cross-polarization allowed us to capture images without specular highlights, even in the eyes. We also captured the face with parallel polarizing filters on lights and cameras; differencing these images proved helpful in synthesizing fine facial relief.

In addition, we struck the actor's life mask and cast it in urethane, and made laser scans of dental molds of his teeth.

3 Surface modeling: "Hirokimation"

From the scanned head in a "neutral" expression, a NURBS mesh model was constructed (Figure 2). Note the loss of detail from the laser scan data. This simplification keeps the model from being too complex for rapid interaction. Later, detail is restored to the model in the form of a displacement map, which is invoked only in rendering.

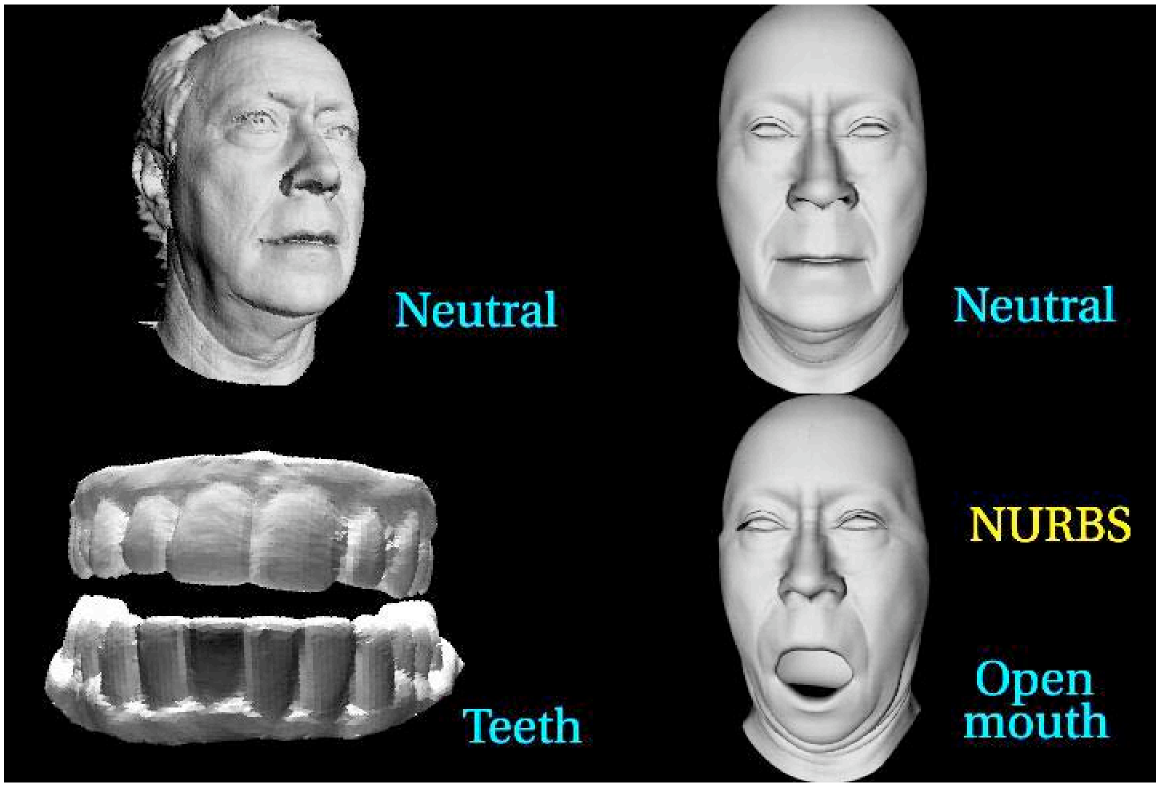


Figure 2. Neutral scan, teeth (left two images), NURBS model (right two).

Hiroki Itokazu, the chief modeler for our project, had previously developed at another studio a method for modeling individual muscle shapes of a specific actor. Beginning with a generic set of anatomically-based surface blendshapes (linear interpolation targets, in this case modeling the action of individual facial muscles) the modeler tries to duplicate each of the extreme facial expressions that were scanned. After getting as close as possible to an expression using the muscle blendshape weights, the modeler proceeds to adjust individual control points of the spline surface to exactly match the expression scanned. Once this matching surface has been created, the sculpted changes are mapped back to the contributing blendshapes by back propagation. That is, for all spline control points shared among several blendshapes, the sculpted changes are distributed according to the parameter weights of the respective shapes. After iterating through the gamut of expressions several times, a set of surface blendshapes corresponding to the contractions of individual facial muscles of the particular person we are modeling results. These shapes are used to track and animate the face.

4 Tracking

Given the head model with a complete set of muscle shapes as described, our process of tracking is simple. We record the actor we hafve modeled on film or video, using one or several calibrated cameras. The frames are digitized, and for each frame, the model head is positioned and oriented so as to correspond to the plate image, and the model muscle shapes are manipulated by hand to match the plate image expression. This is an exacting but direct approach to modeling the performance, and was the method originally employed to "Hirokimate" this linear surface model of facial musculature.

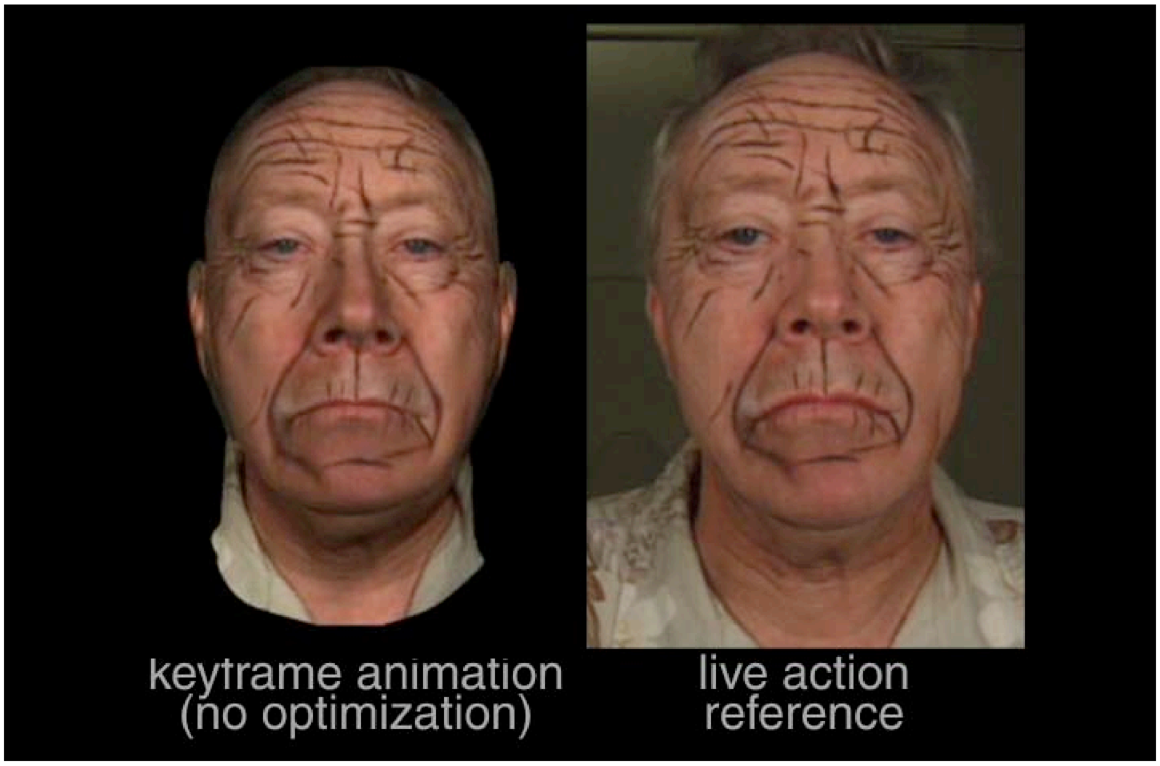


Figure 3. "Hirokimation"
CG model on left with muscles manually edited to match actor's expression on right.

We gained confidence in the ability of the model to match the actor's expressions in this way (Fig. 3).

To automate this process, we employ the following iteration, which we term the visual servo loop:

(a) Adjust head position and orientation, and adjust muscle shapes to approximate the plate image expression;
(b) render the head model;
(c) Compare the rendered image with the plate image:
    Is rendered image close enough to the plate image?
    If so, we are done.
    If not, but we are making progress, return to (a);
    Otherwise, EXIT, in failure.

In order for the visual servo process to converge, a basic requirement is that the actor's muscle model has to be expressive enough to be able to mimic all expressions the actor can assume. We'll assume in the following discussion that the performance is within the range of shapes we have modeled.

To render the head model, we need to know both external and internal camera parameters. We also need texture and lighting information.

To determine if the rendered image is close enough to the plate image, or to measure progress toward that goal, we need a metric to compare the rendered image and the plate image. Given such a metric, we can compute our visual servo loop using numerical optimization. The objective function to be minimized is a scalar function based on the metric, which encodes the distance between the rendered image and the plate image. The optimization process computes the head position, orientation, and muscle parameters that minimize the objective function.

Given: a face model H, controlled by a set of muscle shape variables x[0 . . . nmuscles-1] including head rigid-body variables, input plate images of camera p, I[p] with resolution [nx, ny], p in a set of cameras c[p=0 . . . ncams-1], and known lighting L.

Let the rendered model image at camera p be:

$$\mathrm{Im}[p] = \mathrm{Im}(x[0, \ldots \mathrm{nmuscles}-1], C[p], L)$$

We want to compute variables x[0, . . . nmuscles-1] to minimize:

$$f = \sum_{p=0}^{\mathrm{ncams}-1} \sum_{i=0}^{nx} \sum_{j=0}^{ny} (I[p][i,j] - \mathrm{Im}[p][i,j](x, C[p], L))^2 \qquad (1)$$

The optimizer we've employed in this work is L-BFGS [Zhu97] [Byrd95], a conjugate gradient solver which supports limits on each of the variables to be optimized. This method requires us to compute the first derivatives of the objective function against x, which includes all the muscle parameters, and the six degrees-of-freedom rigid body transform.

Finite differences are a simple way to compute the first derivative of function (1). If we are using central differences, then for each variable in x[0, n-1] we need to evaluate function f twice. That is, we need to evaluate f 2*n times to compute all the partial derivatives. If it takes k steps for the optimization to converge, each step requires 2*n function evaluations. To compute one frame, we need to evaluate the function O(2*n*k) times. For our model, this is relatively expensive, but most of the computation is rendering. This straightforward and robust approach gives very accurate results.

We can also estimate the derivatives using optical flow, by relating 2D pixel motion to the change in muscle parameters. Optical flow works very well if the movement between successive frames is small, and if pixels simpy change location without changing appearance. In many circumstances, including ours, these constraints are frequently violated.

We use optical flow based on [Black96]. We tracked several sequences successfully without iterating on the muscles; after first estimating the head transformation, we solve for it and the expression coefficients directly from flow. In general, however, the visual servo model governs the interpretation of the flowfield in a very useful way, accommodating large frame-to-frame excursions and countless local appearance changes resulting from occlusions, shading, highlights and shadows. Optical flow can be understood as a measure of image distance. Our experience indicates very simple metrics work with a 3D model; flow can be used to speed up the process but is unlikely to improve it.

5 Gaze tracking

For face regions with drastic appearance changes from frame to frame, like gaze direction and eye blinks, we applied a hybrid technique that uses optical flow equations but also a statistical appearance model. The details of this technique are described in [Bregler02]. A training set of example images is selected. Figure 4 shows a training image. In this case we used 15 different example images. Important morphing control points were labeled by hand (the green points in the example images). We also match the rotation of a 3D eye model to each frame. Using morphing, we extend this training set by interpolating between all pairs of hand-labeled images (in our example, from 15 to 330); the coordinates of key points and rotations of the eye model are similarly interpolated.

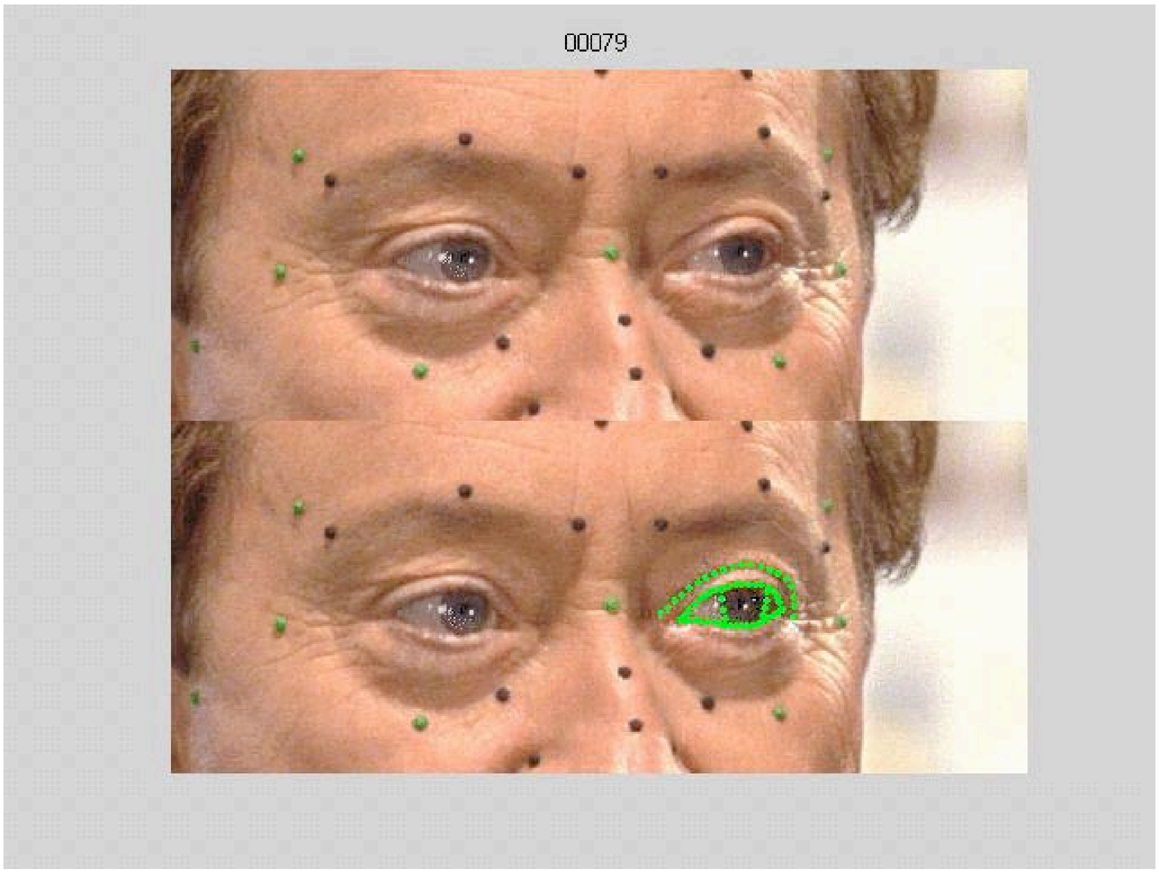


Figure 4. Eye tracker training image

Then the images of the expanded training set were aligned by a best affine fit, and principal component analysis was applied. The PCA stack of images, along with the associated feature points and eye rotations, constitute our statistical appearance model.

This model is used in the eye tracking process in a manner analogous to Lucas-Kanade affine tracking [Lucas81] [Baker04]. The tracker is initialized with a manually selected window on the first frame to be tracked (this first frame is ideally part of the training set, in which case this interactive match has already been performed). We formulate the image match as an optimization, solving a linear problem to find the best affine transformation of the window image to the next frame. After applying this transformation, we project the warped image onto the PCA stack to estimate a new window image. After ten iterations or so, the tracker converges on a warp and a set of PCA coefficients to match the next frame. The coefficients are used to interpolate the 3D model eye rotations.

Such an appearance model, and such a tracker, is appealing for studio visual effects. It is very general purpose, easily trainable, and can be "keyframed," in the sense that any problematic frames in a sequence to be tracked can be entered into the training set along with the desired outputs.

In subsequent work, we've successfully used visual servo directly on 3D eye models to track gaze. Unlike the image-based tracker, such an approach does not capture the correlation between gaze direction and the muscle shapes around the eye, which is visually very important. Model-based visual servo, on the other hand, has the conceptual advantage of being better suited to accommodate occlusion and varying illumination.

6 Modeling and rendering details

This work was originally targeted at live action visual effects for motion picture production. A particular scenario in which a well-known actor interacts with his younger self was envisioned. The task of the Human Face Project was to make such movies possible by creating totally convincing CG faces which could precisely duplicate the performance of human actors.

Since the time our test film was produced, significant advances in rendering human flesh have been achieved. In particular, the transillumination of flesh has been modeled by simulated subsurface scattering.

In the judgment of the authors, the project was extremely successful in tracking and animating human performance, but less successful in photorealistic rendering. Some of the techniques we developed, however, are complementary to methods of rendering CG humans today, and may be valuable in other contexts. To achieve our goals, we were obliged to animate faces with far more detail than we could track.

In section 1, we observed the loss of surface detail attendant on the NURBS modeling of our neutral-expression head model. The muscle blendshapes based on it are similarly simplified. In order to compensate for this loss of detail, we used the life mask we cast of our actor to create a high-resolution displacement map.

To do this, we had the cast digitized with a slow, very high-resolution structured-light projection optical scanner [ATOS01]. The result was a 3.3 million polygon surface model with impressive detail (Figure 5). We downsampled this model to a tractable resolution for interaction, and our modeler painstakingly fit our NURBS model to match it, first by matching position and orientation, then by adjusting the facial muscles, and finally, by moving control vertices to match it very closely. In doing so, he was compensating for the sagging of facial features weighed down by the molding compound. This sagging of facial features is the main drawback of casting faces of visual effects work, and is the main reason visual effects artists are enthusiastic about laser scans -- they capture the features as in life, undeformed by the casting process. We combined the best features of both approaches.

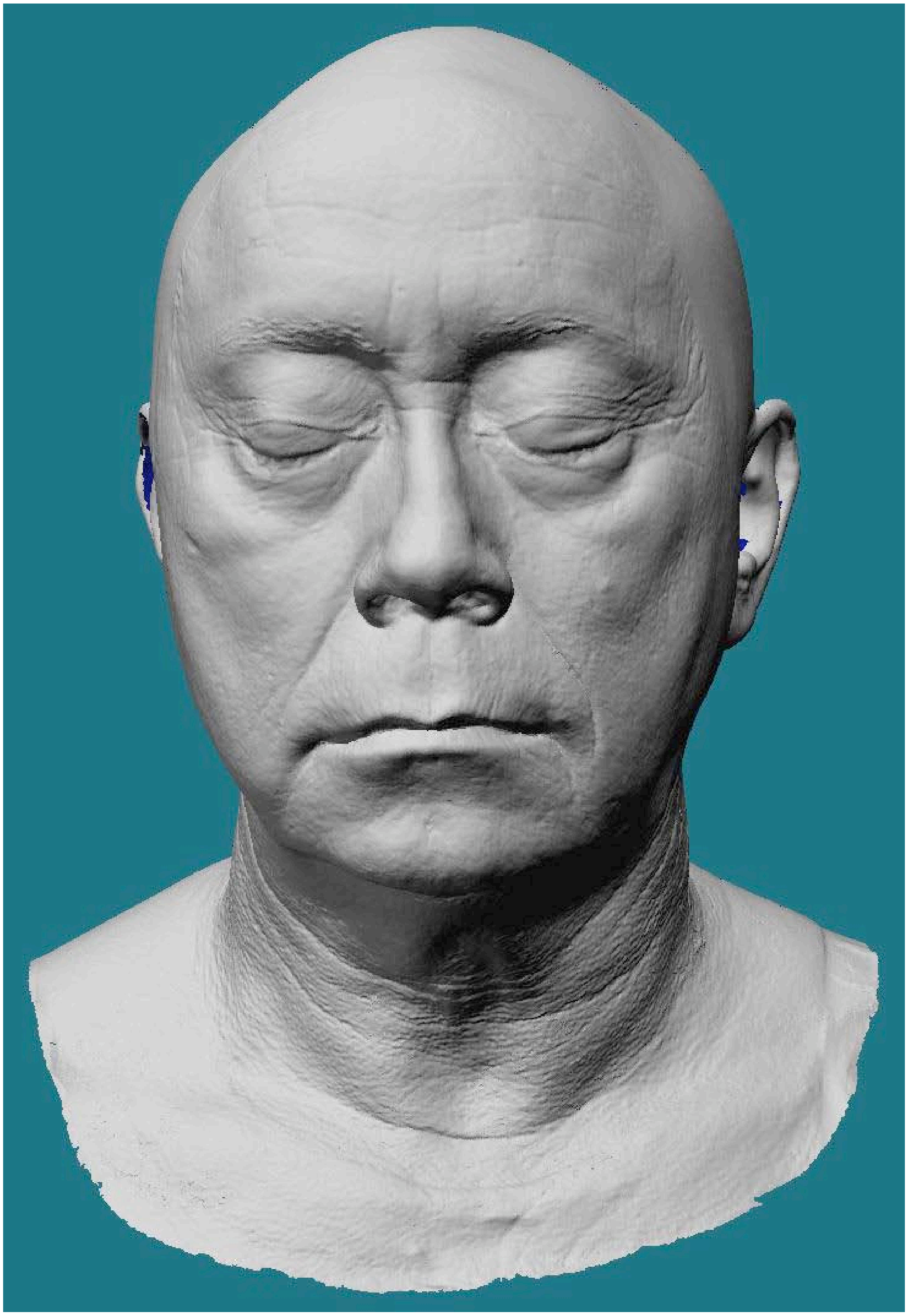


Figure 5. High resolution life mask scan data.

Having registered the NURBS model with a downsampled version of the detailed polygonal scan data, we read the registered NURBS head along with the scan data before downsampling into Cyslice®, a commercial modeling application, and used it to compute displacement maps for the patches in the NURBS model.

A still higher level of detail is required to reproduce very fine wrinkles, pores, and stubble. We extracted this level of detail from the high resolution images acquired for texture mapping, as described in section 1. When we shot cross-polarized texture images, we also shot parallel-polarized images, which emphasize highlights and surface detail. Because these images were not acquired in the same flash, but sequentially, they did not precisely register with the cross-polarized texture images. In general, because they are view dependent, highlights are a source of error for optical flow. In this case, however, because the misregistration was small, because the uniform diffuse illumination created corresponding dark features for wrinkles and light-trapping concavities in both images, and because specular reflections were present in only convex details in one of the images, we were able to use optical flow to register them. By differencing these registered images, we created a map which could easily be adapted to simulate high resolution surface relief.

The high resolution relief maps that resulted were composed with a lookup function that gave a nice shape to the profile of the wrinkles. Then, a sequence of linear filtering and morphological operations isolated pores from wrinkles and other surface features. We used these computed pore sites in the beard area to "grow" geometric stubble, and instanced fine downy hairs on the rest of the skin. These fine hairs are visible only in closeup, but have a softening effect on the shading in midrange shots.

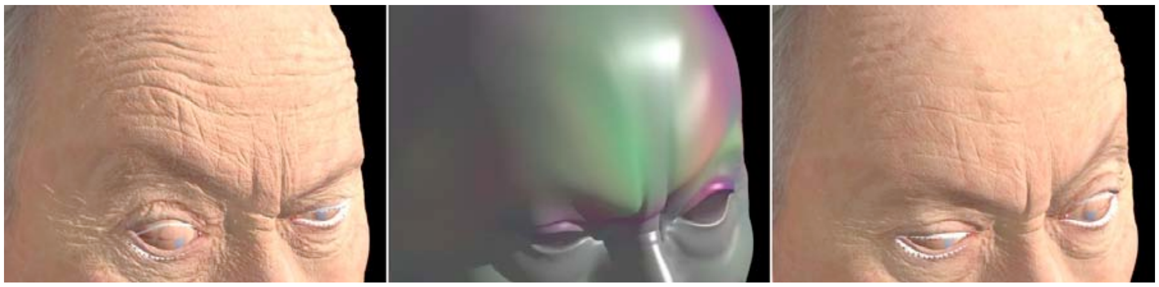


Figure 6. Left, all displacements. Center, tension map. Right, tension modulated wrinkles.

Having recognized and separated pores and wrinkles in the texture, we create a dynamic wrinkle displacement map. This map included hand-painted wrinkles and furrows based on our original laser-scanned expressions, and on video studies of changes in expression. Such surface displacements in CG animation typically do not animate; lumps, folds and wrinkles simply "go along for the ride," adding detail to the model. We envisioned "behavioral" displacements, which act in a manner consistent with the material behavior of the surface. Folds and wrinkles can arise naturally in a physical simulation, but our surface features would ideally match such details in the actor we model. Folds and wrinkles are governed not simply by current physical conditions, but accordingt to the history of the material; prior flexing creates variable elasticity, and the appearance of such features in skin is very individual. Also, at the level of detail we needed to work, a simulation such as a finite element model would be very computationally demanding.

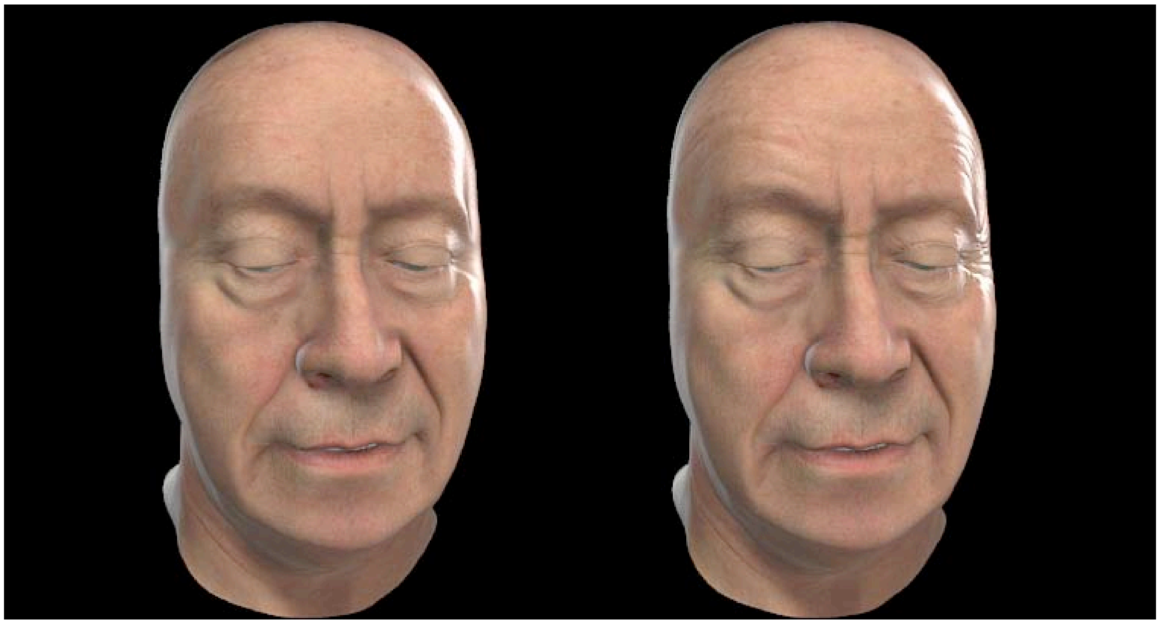


Figure 7. Model rendered without (left) and with (right) wrinkles.

Our solution is to filter the displacements into directional components, using pyramidal filter banks or Fourier-domain methods. We divide dynamic wrinkles into horizontal, vertical, diagonal and antidiagonal components with respect to the U, V parametrization. For behavioral animation, we compute the distance of each spline control vertex from each of eight neighbors in the neutral expression. At each animated frame, we compare the distances to their neutral values to compute stretch and compression on each axis. We term this a "tension map," and use the directional coefficients to modulate the directional components, which are summed to produce the wrinkles for that frame.

A simple way to think of this, is that wrinkles "flatten out" when they are aligned opposite an axis of sufficient stretch, while wrinkles aligned with the stretch will be unaffected. But the same process can be used to create compression wrinkles, like the furrows of the brow. In this case, sufficient compression interpolates the corrugations in. Figure 6 shows an early test of the concept. The unmodulated displacements are on the left, tension in U and V are displayed as intensity of red and green in the tension map at center, and the rightmost frame shows the displacements modulated by tension. Figure 7 shows a more refined rendering of our model with and without computed wrinkles.

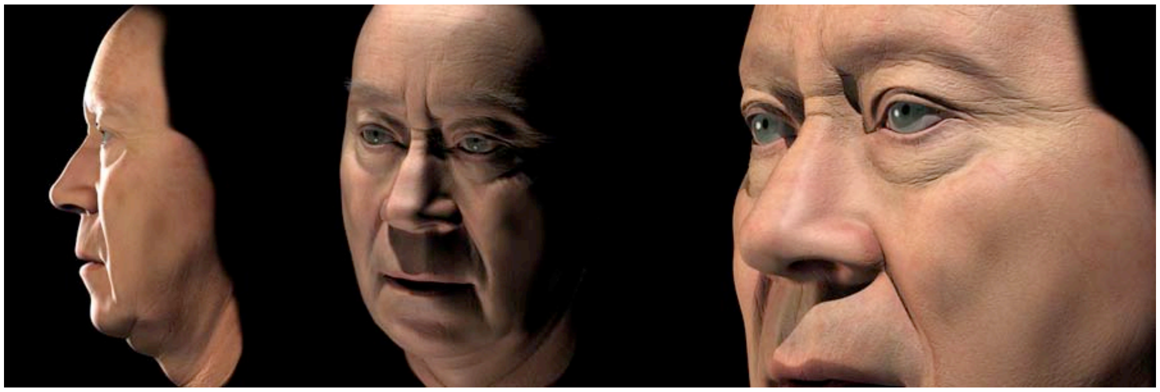


Figure 8. The model illuminated by CG lights

Our shading model is a BRDF model, based on Stephen Marschner's characterizations of human skin [Marschner99]. This shading method was successfully combined with on-set captures of high dynamic range environment maps. The result is significantly better than many commonly used shading models, capturing in particular the grazing-angle sheen that arises from lipid films in skin, and is so identifiably human. The environment map, captured by multiple photographic exposures of reflective spheres, or using a custom rig we built with digital cameras in a dual fisheye configuration, provided complex, detailed illumination that matched the plate photography.

Figure 8 illustrates the views of the model with CG lights. Figure 9 shows the same model illuminated by high dynamic range environment maps: the first two were cathedral and forest maps created by Paul Debevec and used with his permission [Debevec98], the third example is from the casino set of our demonstration film.

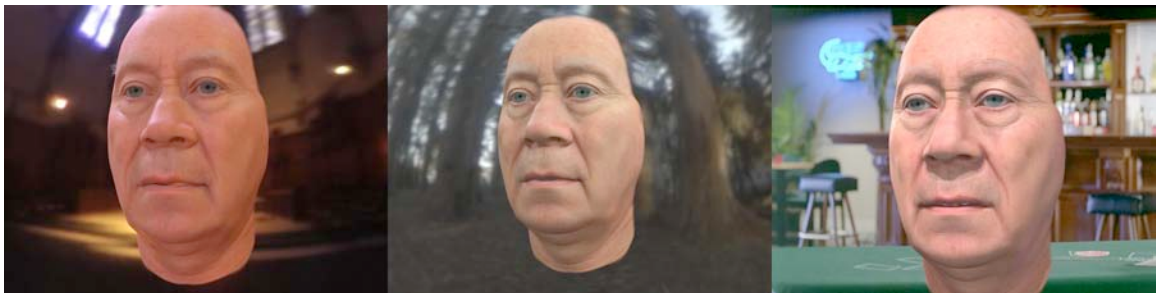


Figure 9. The model illuminated with environment maps.

7 Cross-mapping

There are many ways of modeling human facial expression. Some systems interpolate among whole faces in different expressions (like the expression scans in section 1). Others use the actions of muscle groups, as in FACS [Ekman98], or general purpose animation deformers implemented as clusters, wires, skeletons, lattice deformers, or force fields. MPEG-4 FAPS provide detailed expression changes, but are not strictly anatomically based.

Human facial expressions are innate, not learned. Blind children exhibit typical facial expressions for emotions such as fear, rage and joy, and human beings share many expressive features with other animals. Yet faces themselves are individual. The same muscle activations that shape one face into a characteristic expression will not produce the same shape on another face. But there is a strong sense in which they can be said to have the "same" expression.

Muscles do operate in groups, and people rarely train themselves to flex single facial muscles. But one could expect, as expressions change, to better model coarticulation of muscles by controlling them individually.

It is also true that separating our animation deformers into individual facial muscles permits us to better approximate such actions as rapidly moving the head so as to deform the flesh of the face by acceleration, or using the tongue to bulge the cheeks or lips. In can be argued that such actions are not well reproduced by models of facial muscles, of course, and actions such as puffing up the cheeks with air are a good argument for extra blendshapes that are not accounted forby the muscles of the face.

Our approach to cross-mapping a captured performance to another face is simply to directly apply the muscle parameters we track, to the muscle shapes of any other character we may model. Of course, it is possible to shape the mapping by biasing or scaling components, or applying motion signal processing. Figure 10 shows a tracked expression applied to multiple models.

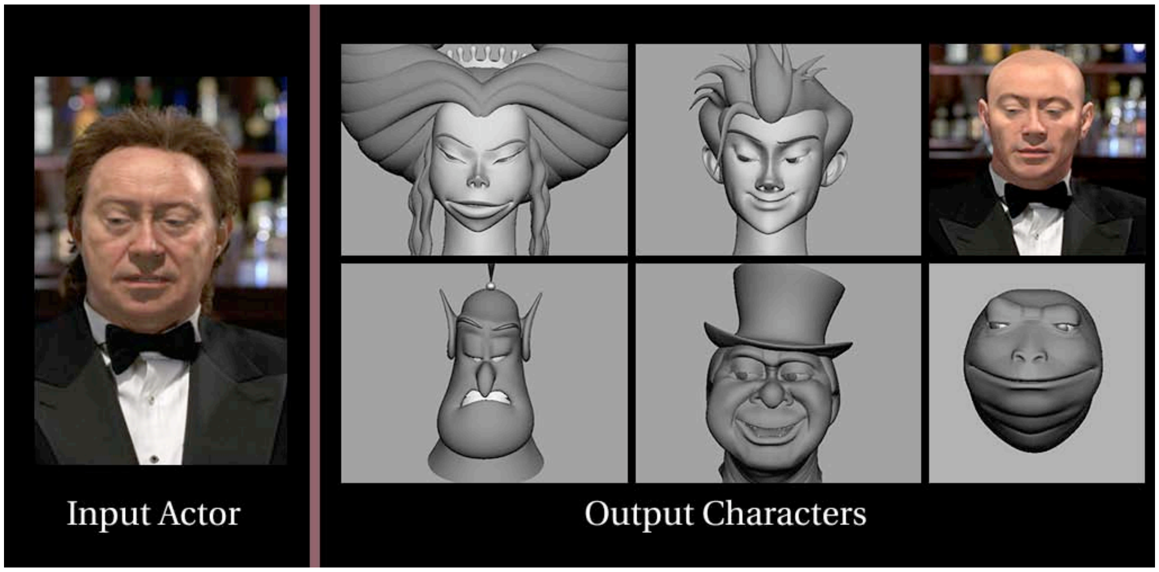


Figure 10. The actor's tracked expression applied to multiple models.

8 Summary and future work

The advantages of our image based performance capture include:

• Accuracy: We can achieve subpixel accuracy in registering our 3D model with source images. This allows us to capture subtle expressions and changes in expression.

• Robustness: Every pixel can contribute to the estimation of the muscle parameters. As a result, we have substantial immunity to noise and optical flow error. The model constrains the solution to expressions and actions the actor can perform.

• Flexibility: Our approach can work with one or multiple cameras, which need not be identical. It works with marked faces or unmarked faces. It can take conventional motion capture data (in the form of 3D tracked points) and, by fitting it with the muscle model, produce a more detailed result than scattered interpolation. It can also work with 2D tracking data, in the form of pixel coordinates of feature points.

• Animation: The model has on the order of 60-70 parameters, which control pose and expression in a direct and intuitive way. It is perfectly suitable for keyframe animation, rotomation, or puppeteering.

Compared with conventional motion capture approaches, the main disadvantages of our method are:

• Turnaround time: We are far from being able to drive a CG character in real time, while the actor is performing.

• Storage space: We need to store all the images before we can capture the performance. In the case of film, high definition video, and/;or multiple cameras, the storage requirement for all the images can be high.

9 Acknowledgments

This project was made possible by the efforts of a large team of talented people, to whom the authors extend their thanks.

Tracking and animation technology

Lewis Siegel
Dan Ruderman
John Lewis
Michael Karp
Jammie Friday
Chris Bregler

Modeling / rendering technology

Eric Enderton
Chyuan Huang
Sean Jenkins
Manuel Kraemer

Jason MacLeod
Jim Rothrock
Ken Wiatrak

Art, film and video

Jim McPherson
Price Pethel
Marta Recio
Pam Spertus
Hoyt Yeatman

10 References

[ATOS01] ATOS-II Advanced Topometric Scanner. 2001. http://www.gom-online.de/En/Products/atos2_vars_print.html, GOM mbH, Braunschweig, Germany.

[Baker04] Baker, S., and Matthews, I. 2004. Lucas-Kanade 20 Years On: A Unifying Framework. International Journal of Computer Vision, Vol. 56, No. 3, March, 2004, pp. 221 - 255.

[Baumgart75] Baumgart, B. 1975. A polyhedron representation for computer vision. In *AFIPS National Conference Proceedings*, volume 44, 589-596.

[Black96] Black, M. J., and Anandan, P. 1996. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. In *Computer Vision and Image Understanding* 63(1), 75-104.

[Bregler02] Bregler, C., Loeb, L., Chuang, E., and Deshpande, H. 2002. Turning to the masters: motion capturing cartoons. In Proceedings of SIGGRAPH 2002, ACM Press / ACM SIGGRAPH, J. Hughes, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 399-407.

[Byrd95] Byrd, R. H., Lu, P., and Nocedal,, J. 1995. A Limited Memory Algorithm for Bound Constrained Optimization. SIAM Journal on Scientific and Statistical Computing , 16, 5, pp. 1190-1208.

[Debevec98] Debevec, P. 1998. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of* SIGGRAPH 98.

[Ekman98] Ekman, P., and Rosenberg, E. (editors). What the Face Reveals: Basic and Applied Studies of Spontaneous Expression Using the Facial Action Coding System (Facs). (Series in Affective Science) Oxford University Press (February 01 1998).

[Eisert97] Eisert, P., and Girod, B. 1997. Model-based facial expression parameters from image sequences. In *Proceedings of the IEEE International Conference on Image Processing* (ICIP-97), 418-421.

[Hutchinson96] Hutchinson, S., Hager, G., Corke, P. 1996. A tutorial on visual servo control. In *IEEE Transactions on Robotics and Automation.*

[Lowe80] Lowe, D. 1980. Fitting parametrized three-dimensional models to images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13, 5. 441-450.

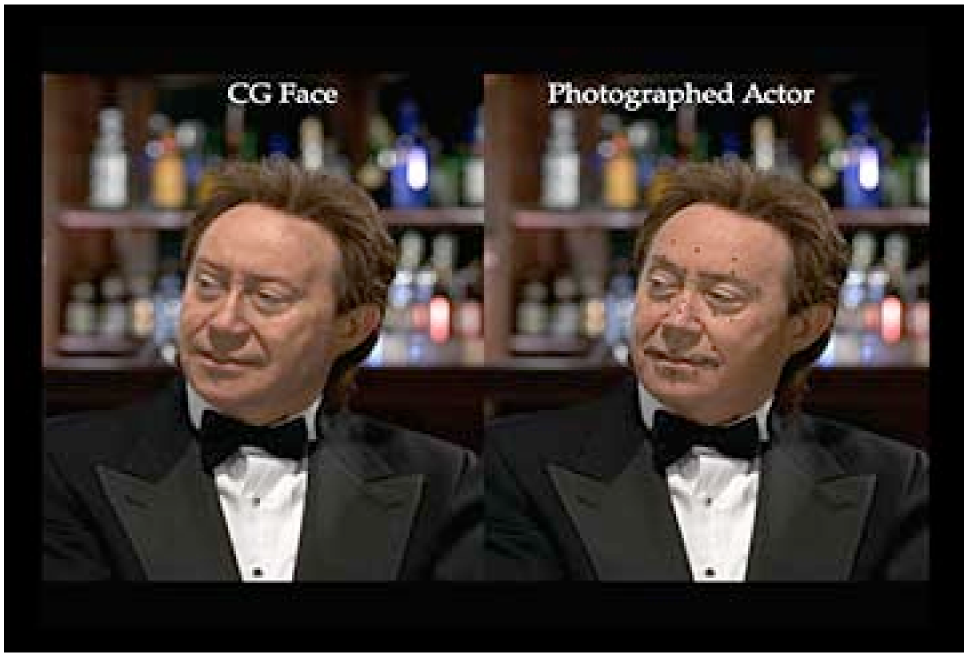
[Lucas81] Lucas, B.D., and Kanade, T. 1981. An Iterative Image Registration Technique with an Application to Stereo Vision. Proceedings of the 1981 DARPA Image Understanding Workshop, April, 1981, pp. 121-130.

[Marschner99] Marschner, S., Westin, S., Lafortune, E., Torrance, K., and Greenberg, D. 1999. Image-based BRDF measurement including human skin. In *10th Eurographics Rendering Workshop.*

[Pighin99] Pighin, F., Hecker, J., Lischinski, D., Szeliski, R., and Salesin, D. 1999. Resynthesizing facial animation through 3D model-based tracking. In Seventh IEEE International Conference on Computer Vision (ICCV '99), 143-150.

[Roberts63] Roberts, L. G. 1963. Machine perception of three-dimensional solids. *Lincoln Laboratory Technical Report* #315.

[Zhu97] Zhu, C., Byrd, R. H., and Nocedal, J. 1997. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. ACM Transactions on Mathematical Software, Vol 23, Num. 4, pp. 550 - 560.



After all this work, the actor meets his younger counterpart.

The Faces of "The Polar Express"

David Bennett

Sony Pictures Imageworks Inc.

Abstract

This paper covers methods used by Imageworks artist and technologists on *The Polar Express* ("PEX") for the entire facial pipeline. It includes a wide range of information from the shoot through to the final facial animation seen on the screen.

The main areas that will be covered:

- **Facial Performance Capture** – Defines performance capture and gives technical details about the shooting of the movie.
- **Facial Tracking** – Covers technical obstacles encountered with facial tracking and the solutions used to overcome these issues.
- **Facial Rigging** – Outlines how the rigs were designed to accommodate the motion capture and animation.
- **Facial Integration** – Covers the retargeting of the data to the character and other tools that were used to prepare the motion capture for the animation department.
- **Facial Animation** – Covers the tools and techniques that the animation department used to enhance the data.

1 Introduction

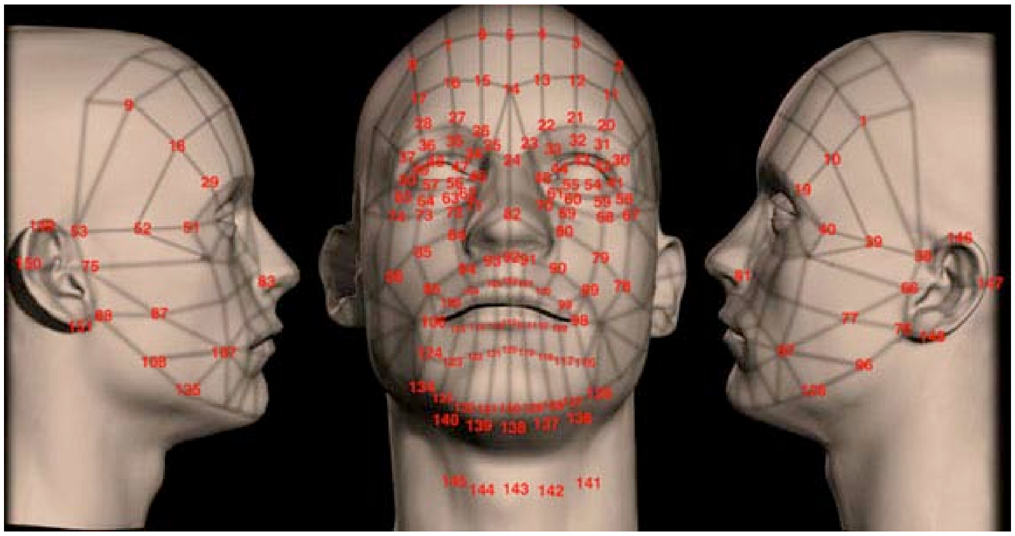
The Polar Express was the first film to successfully utilize facial motion capture for an entire CG movie. Although it has been attempted on other movies in the past such prior movies have not been able to employ this method properly or efficiently. At the beginning of PEX, it looked as if Imageworks would encounter the same fate and all of the facial motion capture data would have to be discarded. However, new technology was created by Imageworks' artists and engineers to make it possible to utilize facial motion capture. Once all the necessary tools were in place, and under a seemingly impossible deadline, a quick and precise process was designed that allowed realistic facial movement to be produced within a short timeframe.

2 Performance Capture

Performance capture is a technique that utilizes motion capture technology to portray the entire performance of the character. In traditional motion capture, the body and the face are recorded at different times and later blended together. Utilizing performance capture, the body and the face are captured at the same time to depict the entire performance of the actor.

The capture was challenging because there were only two months to prep for the shoot. Many changes and demands arose that required the technology to change often. Originally it was intended that the 10 feet x 10 feet capture volume was to capture the face and the body of two performers on the stage at the same time. It later became necessary to capture four people on the stage at the same time, which would allow for a more interactive shoot. Initially, 56 cameras captured the face and 8 cameras captured the body. When it was determined that it was necessary to capture 4 people in the volume at the same time, 8 more cameras were added to capture the body. During the shoot, 80 markers were used for the body and 152 markers were used for the face. When all four actors were on the floor at the same time the cameras were capturing 928 markers at 60fps. The actual shoot lasted 3 months and averaged 50 gigs of raw data every day.

3 Facial Tracking



The Imageworks Facial Tracking Department delivered 1400 facial takes within 1.5 years, which accounted for 16 hours of tracked data for the 96-minute movie. Described below are some of the key elements to facial tracking that became necessary to deliver high quality data for the facial system.

Stabilization is the process of removing the motion of the body from the facial markers. This is important because the retargeting calculations require a stationary input. Any slight shift in the alignment causes artificial facial changes on the character.

Stabilization is one of the most difficult aspects of the facial cleaning process. There is no place on the face that does not have some amount of slide relative to the underlying bone. As a result, this required the development of a tool that allowed the user control over how and when to adjust stabilization. The artists were then able to specify which markers controlled any of the six degrees of freedom over any frame range.

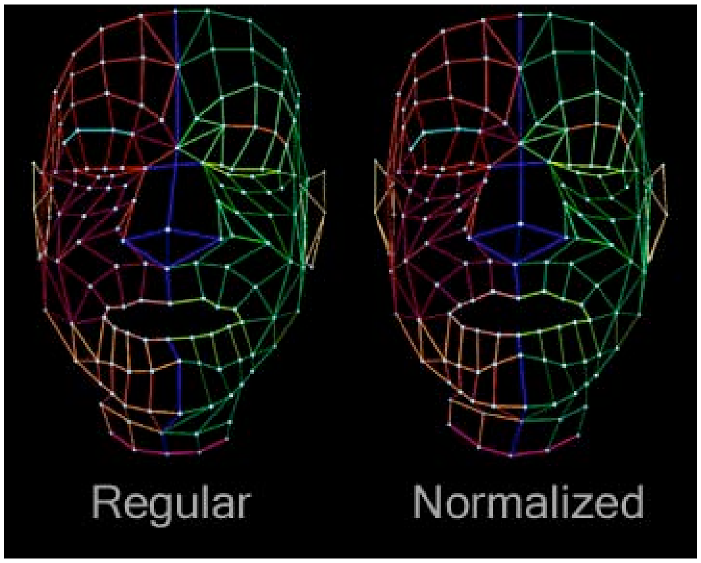
In addition, higher-level tools were then developed to automatically stabilize data. For example, tools that limited the forehead markers from moving side to side or away from the forehead. These, along with other tools, enabled more than 75% of the stabilization to be automatically completed.

T-pose: The t-pose (relaxed facial frame) frame was pasted to the start of the animation and checked for alignment. If the t-pose was not perfectly aligned with the rest of the take, the t-pose would have to be re-stabilized to the first frame of animation. This frame was later pasted to the start of every take to properly initialize the retargeting.

Normalization: Any two or more takes that are blended during a shot have to be normalized. Normalization is the process of adjusting the marker placement so that the positions of the markers in the t-pose are identical on every take. Normalization uses a master t-pose file, which is a single frame of a t-pose for an actor that is accepted as a perfectly neutral face with the best possible marker positions.

Normalization is important because the blending occurs on the marker data before the retargeting process. If the takes are not normalized, marker position differences create the effect of expression changes during blend regions.

Normalization consists of two steps. First, the take is oriented to allow the t-pose for that specific take to become perfectly aligned with the master t-pose. This relies on landmark marker positions, such as the corners of the eyes and the mouth, which were expected to change very little from one day to the next. Second, after the take is aligned, a constant offset is applied to each marker so that its position in the t-pose is identical to the position in the master t-pose. When done properly any number of takes can be blended together.



4 Facial Rigging

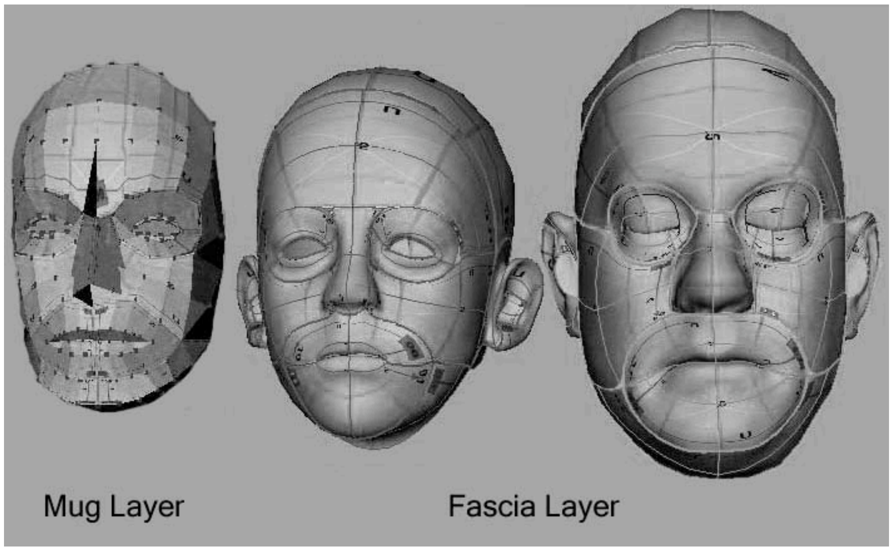
The facial set up on PEX was composed of several layers that ultimately controlled the hi-resolution geometry.

The base level of control is called the “mug” which is a series of degree 1 nurbs patches. The control vertices of the “mug” are controlled by the PFS muscle system plus the motion capture data, via a series of locators called “markers.” The motion capture data has freedom to go in any direction that it wants. The muscles must move along the offset vectors that have been pre-defined by the rigging department. The muscle and marker movement combined give one value that is passed on to the “fascia layer.”

The marker locations are determined by where the mocap data markers were placed on the actors’ faces. The layout of patches for the mug are determined by the location of the markers.

The mug influences the movement of the “fascia layer” or “fat layer” through deformers. The fascia patches are simplified nurbs patches based on the high-resolution geometry.

The high-resolution geometry has many deformer connections from the fascia patches that give the final shape to the skin.



5 Facial Integration

One of Imageworks' more complex challenges with the performance capture of PEX was the fact that adults performed most of the children's roles. This was an especially challenging problem, the performance from adult to child constantly had to be retargeted. As evidenced in the film, the conductor and hobo are perhaps the most realistic characters of the entire movie. This is mainly due to the fact that there was a one to one retargeting on these characters.

The difference in the retargeting became very obvious when observing the performance of Tom Hanks as the Hero Boy character. Early difficulties were faced when the data was retargeted to a smaller head, causing an expanding effect on the character. The fix for this was to adjust the parameters and retarget the character again.

Retargeting the data from the actor to the character was done by breaking the face down into muscle groups and scaling the motion of these muscle groups to match the proportions of the characters. The landmarks were measured on the face of the talent and destination character and scaled accordingly. The image below illustrates the parts of the face that where scaled together.

There were problems with retargeting with this method that did not always give the most desirable results because the actors' and characters' proportions are different. To fix this problem, David Bennett designed the Facial Control Rig (FCR). This tool allowed the user to edit the retargeted data directly. It had several offset layers, which made it a useful to offset or animate without destroying the original data. These offset layers also provide a useful way of scaling the data to get a more accurate retarget. This tool streamlined the animation process and made it possible to finish 3000-frame animations within a day or two.

6 Facial Animation

The Performance Facial System (PFS) was the tool used by animators to accent the motion capture and animate the face. Here are some key features of the PFS:

Muscles: are driven by curves on the face. These curves are meant to mimic the actual muscles of the face. The curves can be edited but not through the UI. When values are applied to muscles it creates an offset from the motion capture.

Poses: are created from assigning values to multiple muscles that create a very specific shape. The user can save this muscle value assignment as a pose with which they can use to animate.

Blending Motion Capture with Animation: in some instances it is necessary to blend out of facial motion capture data. In these scenarios the animator simply blends out to the retargeted data and then continues to animate. The main advantage was that it becomes possible to leave a certain percentage of the original retargeted data on the skin, which gives it a very life-like effect, yet still enables for animation over the top of it.

Tweak Clusters: The main use for tweak clusters is to add fine details. Tweak clusters allow the user to animate the individual vertices, or user-defined clusters of vertices, of the mesh directly. This gives the animator ultimate control over the facial animation.

7 Conclusion

As discussed, the realism created throughout *The Polar Express* was extremely difficult to accomplish. The closer one gets to realism within the world of animation, the more distortions from reality the audience is generally able to observe. While 100% realistic facial animation has not been reached, through continued development of technology and tools, and the creative input of talented individuals, such as those working on *The Polar Express*, this goal will one day be achieved.

8 Acknowledgments

I would like thank Alberto Menache for conceiving most of the facial motion capture pipeline.

I would like to thank the supervisors that span the three divisions that this work covers, Albert Hastings - Motion Capture Integration, Damien Gordon - Tracking Supervisor, David Shuab - Animation Supervisor.

Dennis Hauck for technical information regarding the tracking department.

Eugene Jeong for technical support with the characters rigs.

Frank Bonniewell and Mike Mumbuarer for many suggestions throughout the production of the Polar Express.

Last but not least, I’d like to express my thanks to Nikki Rodgers for her assistance on many technical papers, despite the fact that the technical jargon was somewhat of a mystery to her.

Siggraph 2005 course notes - Digital Face Cloning

Audience Perception of Clone Realism

J.P. Lewis

The digital clones that have appeared briefly in recent movies are remarkable examples of computer graphics. We cannot say “mission accomplished” yet however.

In this session we will partially address two questions:

1. are the current virtual actors really “photoreal”?
2. does everyone perceive an image to be equally realistic?

These questions will be investigated using an audience survey, in which both real and synthetic face images are viewed for various amounts of time, and people are asked to identify the synthetic faces.

The survey has been performed on previous audiences several times, and the results are interesting, and disappointing: *the majority of the viewers correctly identify most synthetic images in only 1/4 second.* This includes images resulting from major industry efforts involving many person-years of effort. On the other hand, a few CG face images actually do fool a relatively large proportion of viewers.

Further, we have asked both CG professionals and casual viewers to suggest what is wrong with the synthetic images. There is little consensus, and the varied responses include comments such as

- there is something wrong with the eyes
- the face is too perfect

The lack of specificity in these comments reveals a challenge to progress on digital cloning:

> We will have difficulty producing plausible virtual actors if we cannot clearly identify what is wrong with the existing ones.

Recently Lyu and Farid [1] demonstrated a program that automatically distinguishes between real synthetic images in many cases. The program operates on low-level image statistics without knowledge of the subject matter of the image. A wavelet representation is augmented with cross-scale correlation statistics, and these combined statistics are the input to trained classifiers (both linear discriminant and support vector machine). The high performance of the program clearly shows that current “photoreal” renderings are missing something. Unfortunately this program cannot indicate what is missing, at least in any usable form.

The viewing studies mentioned above suggests that further investigation in this area may be worthwhile. Remarkably, viewers appear to be more accurate at discriminating synthetic face images at the short (1/4 second) exposure than at longer exposures. This indicates that some problems with current renderings lie in fairly global characteristics that can be preattentively judged (eye movements to fixate on different details themselves require on the order of 200-250 msecs. each). Under longer observation, it is as if detail perception begins to dominate our decision making, and with all the details beautifully rendered, viewers are less sure that what they are seeing is synthetic.

In this section of the course we will summarize the results of previous viewing studies and then invite the course audience to directly experience these issues by participating in the survey (audience votes, several minutes). The issues raised here will be revisited in the subsequent panel discussion.

References

[1] S. Lyu and H. Farid. How realistic is photorealistic? *IEEE Trans. Signal Processing*, 53(2):845–850, Feb. 2005.

Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

Learning Controls for Blend Shape Based Realistic Facial Animation

Pushkar Joshi[1†], Wen C. Tien[1], Mathieu Desbrun[1] and Frédéric Pighin[2]

1 University of Southern Califronia, Computer Science Department
2 Institute for Creative Technologies, University of Southern California

Abstract
Blend shape animation is the method of choice for keyframe facial animation: a set of blend shapes (key facial expressions) are used to define a linear space of facial expressions. However, in order to capture a significant range of complexity of human expressions, blend shapes need to be segmented into smaller regions where key idiosyncracies of the face being animated are present. Performing this segmentation by hand requires skill and a lot of time. In this paper, we propose an automatic, physically-motivated segmentation that learns the controls and parameters directly from the set of blend shapes. We show the usefulness and efficiency of this technique for both, motion-capture animation and keyframing. We also provide a rendering algorithm to enhance the visual realism of a blend shape model.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

1. Introduction

The human face has always held a particular interest for the computer graphics community: its complexity is a constant challenge to our increasing ability to model, render, and animate lifelike synthetic objects. Facial animation requires a *deformable* model of the face to express the wide range of facial configurations related to speech or emotions. There are two traditional ways of creating deformable face models: using a physically-based model or a blend shape model. A physically-based model generally simulates various skin layers, muscles, fatty tissues, bones, and all the necessary components to approximate the real facial mechanics. A blend shape model, however, mostly disregards the mechanics; instead, it directly considers every facial expression as a linear combination of a few select facial expressions, the blend shapes. By varying the weights of the linear combination, a full range of facial expressions can be expressed with very little computation.

Nowadays, there are several options for creating blend shapes. A skilled digital artist can deform a base mesh into the different canonical shapes needed to cover the desired range of expressions. Alternatively, the blend shapes can be directly scanned by a range scanner from a real actor or a clay model. With this last technique, the scanned data needs to be registered in order to produce blend shapes that share a common topology [9] and can therefore be combined correctly.

To express a significant range of highly detailed expressions, digital animators often have to create large libraries of blend shapes. In this case a naive parameterization of the face model, one that would give a parameter for each blend shape, is not practical. In particular, the visual impact of changing the contribution of a blend shape might be diffcult to predict, leading to a tedious trial and error process for the user. Splitting the face geometry in several regions that can be specified individually somewhat alleviates this problem. By manipulating a smaller area the user is guaranteed that the modification will impact only a specific part of the face (e.g., the left eyebrow). However, segmenting the face manually is difficult and tedious. The segmentation should reflect the idiosyncracies of the face being modeled and provide editing and different level of details. In general, finding the right parameters and control knobs in a blend shape model is no simple task, and it often leverages an understanding of the mechanical structure of the face. In this paper we address

† e-mail: ppj@usc.edu

the problem of parameterization and control of blendshape models.

1.1. Related Work

Blend shape interpolation can be traced back to Parke's pioneering work in facial animation [12, 13]. This initial work has found many applications both, in computer graphics and in computer vision.

Parke's original idea was rapidly extended to a segmented face where the regions are blended individually [8], allowing a wider range of expressions. Traditionally these regions are defined *manually*. A prototypical example is the segmentation of a face into an upper region and a lower region: the upper region is used for expressing emotions, while the lower region expresses speech. Although this approximation is often used in practice, such an ad hoc separation does not reflect the subtle interdependencies appearing in reality.

Blend shape models have also found their way in the computer vision community where they help analyze face images and video. Blanz and Vetter [1] designed an algorithm that fits a blend shape model onto a single image. Their result is an estimate of the geometry and texture of the person's face. Pighin et al [15] extended this work by fitting their model to a whole sequence of images, allowing manipulation of the video by editing the fitted model throughout the video sequence.

There has been little research on interactive blend shape model manipulation with the exception of the work by Pighin et al [14]. They describe a keyframe animation system that uses a palette of facial expressions along with a painting interface to assign blending weights. The system gives the animator the freedom to assign the blending weights at the granularity of a vertex. This freedom is, in practice, a drawback: by not taking into account the physical limitations of the face, it is rather difficult to create realistic expressions. The system we propose is quite different: it does respect the mechanics of the face through an analysis of the physical properties of the data. In comparison, our system is more intuitive and helps generate plausible facial expressions. Choe et al [3] have done some interesting work on mapping motion onto a set of blend shapes, but where they build a segmentation of the face manually, we learn it from the data.

Segmentation is a very active topic in image processing and computer vision [7]. However, the problem we are addressing is very different from image or optical flow segmentation; our goal is to segment a 3D mesh that is a linear combination of sample meshes. Similarly, subdivision surfaces have become a popular representation for three-dimensional objects [16]. The goals of subdivision schemes as researched so far in the compter graphics community do not match that of this paper: the segmentation of a linear space of meshes.

1.2. Contribution and Overview

In this paper, we address the problems of *parameterization* and *control* of blend shape models. We design an *automatic technique* that extracts a set of parameters from a blend shape model. Instead of deriving our control mechanism from the biomechanics of the face, we learn it directly from the available data. This solution is thus specific to the processed blend shapes, and reflects the facial idiosyncracies present in data. We also demonstrate the usefulness of these parameters through two animation techniques: motion capture and keyframing. Finally, we propose a new rendering algorithm for blend shape models; one that addresses the problem of texture misregistration across blend shape textures.

We will describe our work by starting in section 2 with some definitions and notations. Section 3 and section 4 are then dedicated to the use of the model in motion capture animation and keyframing respectively. We also explain how some of the blur artifacts can be avoided while rendering the blend shape model in section 5. We finally conclude with a discussion of our results and ideas for future research.

2. Blend Shape Face Model

Setup We define a blend shape face model as being a convex linear combination of n basis vectors, each vector being one of the blend shapes. Each blend shape is a face model that includes geometry and texture. All the blend shape meshes for a given model share the same topology. The coordinates of a vertex $\boldsymbol{V}$ belonging to the blend shape model can then be written as follows:

$$\boldsymbol{V} = \sum_{i=1}^{n} \alpha_i \, \boldsymbol{V}_i$$

where the scalars α_i are the blending weights, $\boldsymbol{V}_i$ is the location of the vertex in the blend shape i, and n is the number of blend shapes. These weights must satisfy the convex constraint:

$$\alpha_i \geq 0, \text{ for all } i$$

and must sum to one for rotational and translational invariance:

$$\sum_{i=1}^{n} \alpha_i = 1$$

Similarly, the texture at a particular point of the blend shape model is a linear combination (i.e., alpha blending) of the blend shape textures with the same blending weights as those used for the geometry.

Learning Controls Spanning a complete range of facial expressions might require a large number of blend shapes. For instance, the facial animations of Gollum in the feature film

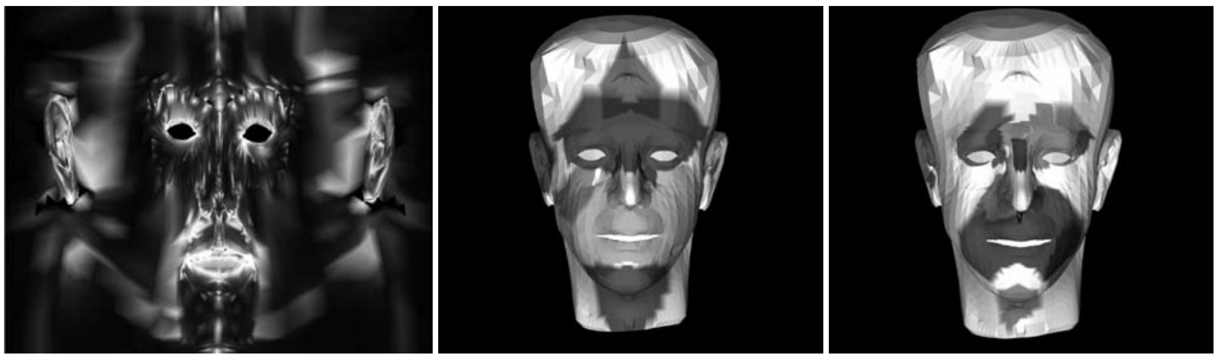


Figure 1: *Automatically generated regions: (a) Deformation map (the deformation in X, Y and Z directions is expressed as a respective RGB triplet) (b) Segmentation for a low threshold (c) and for a high threshold.*

The Two Towers required 675 blend shapes [6]. However, studies [5] have shown that it is possible to create complex and believable facial expressions using only a few blend shapes by combining smaller, local shapes. For instance, the face geometry can be split in three areas, one covering the mouth and the jaws, another covering the eyes, and the last one the eyebrows. If the regions can be manipulated independently, the number of possible combinations (and therefore the number of possible expressions) increases significantly. Although one can define the regions manually, it requires considerable skill and time, and needs to be performed each time a new character is animated. Instead, we propose a simple, automatic and fast (less than a minute for a typical blend shape model) segmentation process that leverages face deformation information directly from the input data to create meaningful blend regions.

Physical Model One of the simplest physical models for deformable objects is that of *linear elasticity*. The deformation of an object is measured by the displacement field $\boldsymbol{d}$ between each point's current position and its rest position. As explained, for instance, in Debunne et al [4], the governing equation of motion of a linear elastic model is the Lamé formulation:

$$\rho\boldsymbol{a} = \lambda\Delta\boldsymbol{d} + (\lambda+\mu)\nabla(\nabla\cdot\boldsymbol{d}) \tag{1}$$

In our current context, $\boldsymbol{d}$ is the displacement of the vertex from its position on the neutral face, ρ is the averaged face mass density, $\boldsymbol{a}$ is the vertex' acceleration, and λ and μ are the Lamé' coefficients that determine the material's behavior (related to Young's modulus and Poisson ratio). The interpretation of the previous equation is relatively simple: the laplacian vector $\Delta\boldsymbol{d}$ of the displacement field represents the propagation of deformation through the blend shape, while the second term represents the area-restoring force. These two second-order operators, null for any rigid deformation, are therefore *two complementary measures of deformation* of our face model. To further simplify our model, we will assume that the area distortion is negligible on a face (our tests confirm that this assumption does not change the results significantly); therefore, we only use the laplacian to segment the face into disjoint regions of similar amount of deformation, as explained next.

Segmentation Debunne et al [4] have introduced a simple discrete evaluation of the laplacian operator present in Eq. 1. We compute this discrete laplacian value at every vertex of every non-neutral (i.e., expressive) blend shape, and take the magnitude of the resulting vectors. This provides us with a deformation map for each expression. We gather these maps into a single deformation map M by computing for each vertex independently its maximum deformation value across all expressions. This resulting map (see Figure 3(a) - expressed as a vector map to show direction of deformation) measures the maximum amount of *local deformation* that our face model has for the blend shapes used. A fast segmentation can now be performed by simply splitting this map in the regions with low deformation, and those with high deformation. The threshold for this split can be chosen as:

$$threshold = \boldsymbol{D}[n\,t]$$

where $\boldsymbol{D}$ is the array of sorted deformation values, n is the size of this array and t is a scalar between 0 and 1.

That is, first sort all the deformation values, and then obtain the deformation at the position that is a function of the number of values. For instance, to generate the regions in Figure 3(b,c), we used $t = 0.25$ and $t = 0.75$ respectively. Depending on the threshold, disconnected regions are created all across the mesh. We automatically clean up the regions by absorbing isolated regions into larger regions and minimizing concavity of the regions. Finally, each region is extended by one vertex all around its boundary, in order to create an overlap with the neighboring regions. The result is a large, least-deformed region (i.e. the background), and a number of overlapping regions where there is generally more significant deformation in the range of expressions. These latter regions (see Figure 3(b,c)) correspond to vertices that generally undergo similar deformation: locally, each region deforms in a quasi-rigid way. Thus, linear blending in each of these regions will reconstruct much more detail of the target face expression as demonstrated in the next two sections.

3. Animation with Motion Capture

We express the motion in the motion capture data using the blend shape model. That is, we assume that the motion (or the per-frame position) of a motion marker can be expressed as a linear combination of corresponding points in the blend shapes. Namely:

$$\boldsymbol{M}_j = \sum_{i=1}^{n} \alpha_i \, \boldsymbol{V}_{ij}$$

where $\boldsymbol{M}_j$ is a location on the face whose motion was recorded and $\boldsymbol{V}_{ij}$ is the corresponding location in blendshape i. m is the number of motion markers and n the number of blend shapes (as in Choe et. al. [3])

Given several such equations, we find the blending weights α_i. We recast this as a minimization problem, where we need to minimize the sum of the differences:

$$\sum_{j=1}^{m} [\boldsymbol{M}_j - (\sum_{i=1}^{n} \alpha_i \, \boldsymbol{V}_{ij})]^2 \qquad (2)$$

The whole system is a linear system of equations where the unknowns α_i are the weights in the blend shape combination. By using an iterative quadratic programming solver [11], we obtain the optimal values of the blending weights α_i in the least squares sense. Solving this system is equivalent to orthogonally projecting the motion onto the set of blend shapes. In general equation 2 does not have an exact solution, since the motions can be more expressive than what the set of blend shapes allows. To produce an animated mesh that follows the motion more precisely we complement the projection on the blend shape basis by translating the vertices in the mesh by the residual $(\boldsymbol{M}_j - \sum_{i=1}^{n} \alpha_i \cdot \boldsymbol{V}_{ij})$. The residual, which is only known for a small set of points, is interpolated to the rest of the facial mesh using radial basis functions [10]. The final coordinates, $\boldsymbol{V}_j$, of a vertex on the face are then constructed using:

$$\boldsymbol{V}_j = \boldsymbol{P}_j + RBF(\boldsymbol{P}_j)$$

where $\boldsymbol{P}_j$ is the projection on the set of blend shape:

$$\boldsymbol{P}_j = \sum_{i=1}^{n} \alpha_i \, \boldsymbol{V}_{ij}$$

and $RBF(\boldsymbol{P}_j)$ is the interpolated residual at vertex $\boldsymbol{P}_j$:

$$RBF(\boldsymbol{P}_j) = \sum_{i=1}^{m} \exp(-\|\boldsymbol{M}_i - \boldsymbol{P}_j\|) \, \boldsymbol{C}_i \qquad (3)$$

In equation 3 the vectors $\boldsymbol{C}_i$ are computed using the known values of the residual at $\boldsymbol{M}_i$. Since the system of equations is linear in the unknowns, using linear least-squares provides an estimate of the unknowns [14]. Note that only applying the residual would have a different effect; by first projecting on the set of blend shapes we obtain a face geometry that reflects the blend shapes, then we apply the residual which brings the geometry closer to the motion. Choe et al [3]'s approach to mapping motion onto a set of blend shapes is very similar. The main difference is how the residual is taken into account. In their approach the blend shapes are modified to adapt them to the motions. We, on the other hand, use radial basis functions to modify the geometry on a per-frame basis. Their method would probably be more effective for processing a large quantity of motions, whereas ours would perform better on a small dataset.

Instead of solving the above system for the entire model, we solve for each region created using our automatic segmentation process. Doing so gives us localized control over the face mesh and results in better satisfaction of the spatial constraints. This also allows us to express a wide range of motion using only a limited number of blend shapes (ten, in our case).

For every frame and for every region, we construct the above minimization problem and obtain blending weights. The same weights are then used to obtain, for all vertices of the region, new positions that match the motion. Thus, for every frame of motion, we can solve a minimization problem to obtain the blending weights and consequently the face mesh that follows the motion capture data.

4. Keyframe Editing

Using our blend shape model, we can interactively construct face meshes that can be used as keyframes in a keyframing-based facial animation tool.

Creating Keyframes Creating a keyframe is similar to producing a frame in a motion capture sequence in that we need to specify control points (markers), their respective mappings, and spatial constraints (i.e. the positions of the markers). In our interface, the user can interactively specify all the above by clicking and dragging with the mouse on the face model. As in the process used in the motion capture application (see Eq. 2), we construct a minimization problem using the interactively specified constraints and obtain blending weights.

Regions and Region Hierarchy We can segment the blend shape model into regions using our automatic segmentation

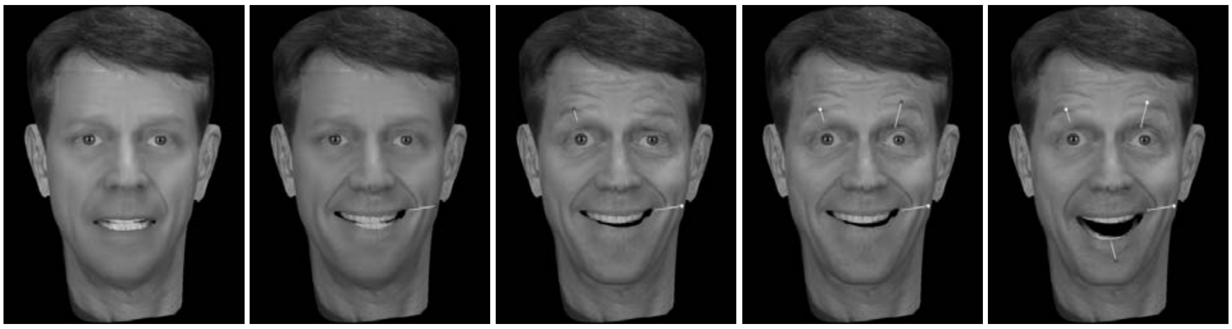


Figure 2: *Successive keyframe editing from coarse (left) to fine (right) level of details.*

technique. In order to allow keyframe editing at various levels of detail, we build a hierarchy of regions. This hierarchy is created by first running the segmentation algorithm described in section 2 with a high threshold value so as to generate small and localized regions. These regions constitute the lowermost region level. We can then merge regions iteratively so that contiguous regions are merged together as we generate higher region levels.

Motion Damping Some of the locations on the face do not move significantly throughout the set of blend shapes (e.g. tip of the nose). If we were to select such a location and try to deform it, using the interface describe so far, a small motion of the mouse would trigger a dramatic change in the facial expression. To reduce the sensitivity of the system we scale the displacement of the mouse according to a factor that is inversely proportional to the maximum displacement in the blend shape model at the selected point on the mesh.

Fig. 2 displays a sequence of manipulations performed on a keyframe. The successive keyframe editing is performed with increasing level of details to refine the facial expression in a localized manner.

5. Rendering Realistic Blend Shapes

Basic Process Rendering the blend shape model is pretty straightforward and can be done in two steps: first the consensus geometry is evaluated, and then it is rendered as many times as there are blend shapes in the model to blend the texture maps. This latter step is done by assigning to each vertex' alpha channel the corresponding weight for a given blend shape. To improve our renderings we decided not to blend the texture maps on the parts of the face whose texture should not vary as a function of the facial expression, in particular, in the hair, neck, and ears area. These areas are textured using the texture map of any blend shape (usually one corresponding to the neutral expression).

Realistic Textures Texture misregistration is a common problem with blend shape rendering for realistic facial animation. If the textures do not correspond at each point on the face geometry, combining them linearly will result in a blurred rendering. Thus, the frequency content of the rendered face images varies as a function of time. Figure 4 provides an illustration of this phenomenon. The leftmost image shows our model rendered with only one contributing blend shape. The middle image shows the rendered model with seven equally contributing blend shapes. In the middle rendering a lot of the details of the face texture have disappeared.

To alleviate this problem, we borrow an approach from the image processing community [2]: we base our blending on a band-pass decomposition of the textures. More specifically, we build a two level laplacian image pyramid out of each blend shape texture map. This results in the creation of two texture maps for each blend shape: the first is a low-pass version of the original texture, and the second is a signed detail texture. We then render the blend shape model as follow: we first render the lowpass texture maps and blend them together. Then we render the detail texture map of a single blend shape using the consensus geometry and add it to the previous rendering. The result is a rendering that both, better preserves the original spectral content of the blend shape textures and maintains the high frequency content constant throughout the animation. The rightmost rendering in figure 4 illustrates the improvement obtained by using this technique.

6. Results

We demonstrate the techniques described in this paper with a set of blend shapes modeled to capture the facial expression of an actor. We created ten blend shapes corresponding to extreme expressions. We used an image-based modeling technique similar to the one developed by Pighin et al [14]. Three photographs of the actor were processed to model each blend shape: front facing, 30 degree right, and 30 degree left. All the animations shown in the video were computed and rendered in real-time (30Hz) on a 1GhZ PC equipped with an NVidia GeForce 3 graphics card. We decided to animate the tongue, the lower teeth and the upper teeth in a simple procedural manner; they are moved rigidly and follow the motion of separate sets of manually selected points on the mesh. The eyeballs are moved rigidly according to the rigid motion of the head.

Motion Capture As described in section 3, we can project recorded motion onto the blend shape model. The accompanying video includes a few animated sequences that demonstrate this technique. The deformations of the face are very natural and reflect the actor's personality. Fig. 5 shows some of the frames obtained. The example shown uses only 10 blend shapes. To animate speech motion usually a much larger set of shapes needs to be used. We are able to animate the lips by using radial basis functions as described in section 3.

Keyframe Editing Also included in the video is a demonstration of the interactive tool described in section 4. The tool allows us to sculpt the face in a very intuitive way. We start manipulating the face with a set of coarse regions and refine the expression by using increasingly finer segmentations.

7. Future Work

We would like to improve our results in different ways. In particular, we feel our rendering algorithm would benefit from a more principled frequency analysis of the blend shapes texture maps. Using a feature preserving filter to separate the high frequency data might lead to better results. It would be interesting to try this technique on a non-human character; one for which segmenting the face might be more challenging and non-intuitive. We would also like to test our technique on a larger dataset of blend shapes. Finally, our segmentation technique only takes into account geometric information. We would like to extend it to also take advantage of the texture information.

Acknowledgements

The authors would like to thank J.P. Lewis for discussions about blend shape animation and Andrew Gardner for its initial development. This project was supported in part by National Science Foundation (CCR-0133983, DMS- 0221666, DMS-0221669, EEC-9529152) and the U.S. Army Research Institute for the Behavioral and Social Sciences under ARO contract number DAAD 19-99-D-0046. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the Department of the Army.

References

1. T. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH 99 Conference Proceedings*. ACM SIGGRAPH, August 1999.
2. P.J. Burt and E.H. Adelson. A multiresolution spline with application to image mosaics. *ACM Transaction on Graphics*, 2(4), October 1983.
3. B. Choe, H. Lee, and H. Ko. Performance-driven muscle-based facial animation. In *Proceedings of Computer Animation*, volume 12, pages 67–79, May 2001.
4. G. Debunne, M. Desbrun, M. Cani, and A. Barr. Adaptive simulation of soft bodies in real-time. In *Proceedings of Computer Animation 2000*, pages 15–20, May 2000.
5. P. Ekman and W.V. Friesen. *Unmasking the face. A guide to recognizing emotions fron facial clues*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.
6. J. Fordham. Middle earth strikes back. *Cinefex*, (92):71–142, 2003.
7. R.M. Haralick. Image segmentation survey. *Fundamentals in Computer Vision*, 1983.
8. J. Kleiser. A fast, efficient, accurate way to represent the human face. In *SIGGRAPH '89 Course Notes 22: State of the Art in Facial Animation*, 1989.
9. A. Lee, D. Dobkin, W. Sweldens, and P. Schröder. Multiresolution mesh morphing. In *Proceedings of SIGGRAPH 99*, pages 343–350, August 1999.
10. G.M. Nielson. Scattered data modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, January 1993.
11. J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 1999.
12. F.I. Parke. Computer generated animation of faces. *Proceedings ACM annual conference.*, August 1972.
13. F.I. Parke. *A parametric model for human faces*. PhD thesis, University of Utah, Salt Lake City, Utah, December 1974. UTEC-CSc-75-047.
14. F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D.H. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH 98 Conference Proceedings*, pages 75–84. ACM SIGGRAPH, July 1998.
15. F. Pighin, R. Szeliski, and D.H. Salesin. Resynthesizing facial animation through 3d model-based tracking. In *Proceedings, International Conference on Computer Vision*, 1999.
16. D. Zorin, P. Schröder, A. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for modeling and animation. In *SIGGRAPH 2000 Course Notes*. ACM SIGGRAPH, May 2000.

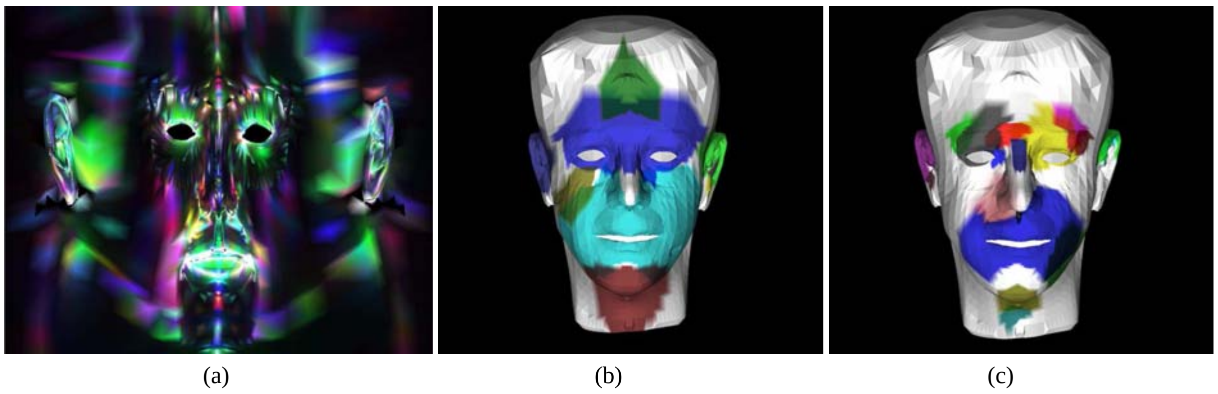


Figure 3: *Automatically generated regions: (a) Deformation map (the deformation in X, Y and Z directions is expressed as a respective RGB triplet) (b) Segmentation for a low threshold (c) and for a high threshold.*

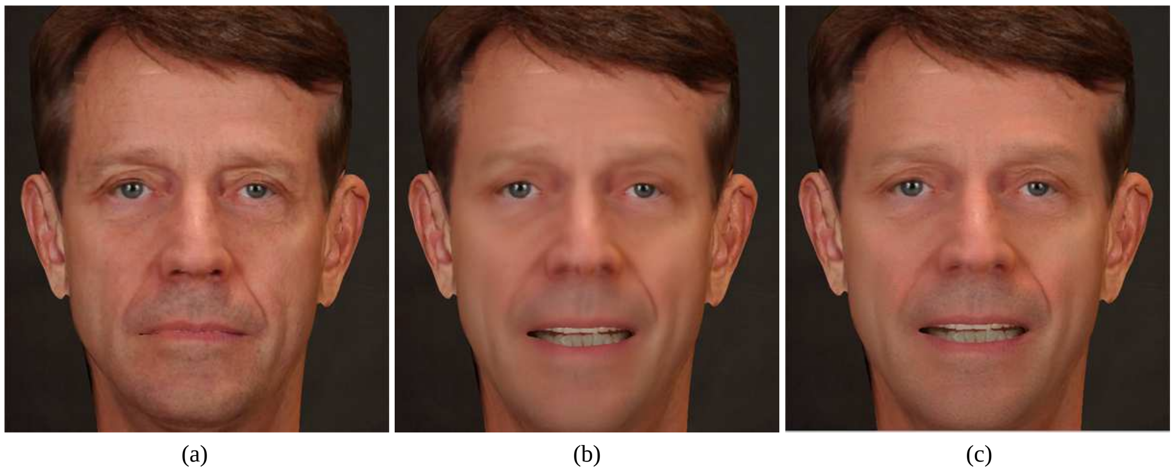


Figure 4: *Blend shape renderings (a) a single contributing blend shape (b) seven equally contributing blend shapes without detail texture (c) seven equally contributing blend shapes with detail texture*

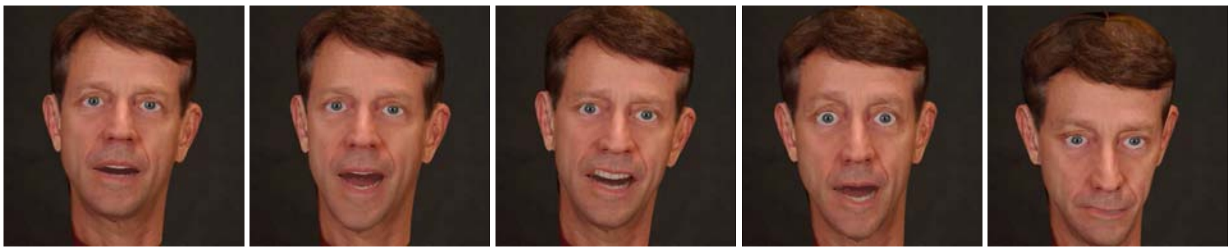


Figure 5: *Mapping motion capture data on a set of blend shapes*

Synthesizing Realistic Facial Expressions from Photographs

Frédéric Pighin *Jamie Hecker* *Dani Lischinski*[†] *Richard Szeliski*[‡] *David H. Salesin*

University of Washington [†]The Hebrew University [‡]Microsoft Research

Abstract

We present new techniques for creating photorealistic textured 3D facial models from photographs of a human subject, and for creating smooth transitions between different facial expressions by morphing between these different models. Starting from several uncalibrated views of a human subject, we employ a user-assisted technique to recover the camera poses corresponding to the views as well as the 3D coordinates of a sparse set of chosen locations on the subject's face. A scattered data interpolation technique is then used to deform a generic face mesh to fit the particular geometry of the subject's face. Having recovered the camera poses and the facial geometry, we extract from the input images one or more texture maps for the model. This process is repeated for several facial expressions of a particular subject. To generate transitions between these facial expressions we use 3D shape morphing between the corresponding face models, while at the same time blending the corresponding textures. Using our technique, we have been able to generate highly realistic face models and natural looking animations.

CR Categories: I.2.10 [Artificial Intelligence]: Vision and Scene Understanding — Modeling and recovery of physical attributes; I.3.7 [Computer Graphics]: Three-Dimensional Graphics — Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics — Color, shading, shadowing and texture.

Additional Keywords: facial modeling, facial expression generation, facial animation, photogrammetry, morphing, view-dependent texture-mapping

1 Introduction

There is no landscape that we know as well as the human face. The twenty-five-odd square inches containing the features is the most intimately scrutinized piece of territory in existence, examined constantly, and carefully, with far more than an intellectual interest. Every detail of the nose, eyes, and mouth, every regularity in proportion, every variation from one individual to the next, are matters about which we are all authorities.

— Gary Faigin [14],
from *The Artist's Complete Guide to Facial Expression*

Realistic facial synthesis is one of the most fundamental problems in computer graphics — and one of the most difficult. Indeed, attempts to model and animate realistic human faces date back to the early 70's [34], with many dozens of research papers published since. The applications of facial animation include such diverse fields as character animation for films and advertising, computer games [19], video teleconferencing [7], user-interface agents and avatars [44], and facial surgery planning [23, 45]. Yet no perfectly realistic facial animation has ever been generated by computer: no "facial animation Turing test" has ever been passed.

There are several factors that make realistic facial animation so elusive. First, the human face is an extremely complex geometric form. For example, the human face models used in Pixar's *Toy Story* had several thousand control points each [10]. Moreover, the face exhibits countless tiny creases and wrinkles, as well as subtle variations in color and texture — all of which are crucial for our comprehension and appreciation of facial expressions. As difficult as the face is to model, it is even more problematic to animate, since facial movement is a product of the underlying skeletal and muscular forms, as well as the mechanical properties of the skin and subcutaneous layers (which vary in thickness and composition in different parts of the face). All of these problems are enormously magnified by the fact that we as humans have an uncanny ability to read expressions — an ability that is not merely a learned skill, but part of our deep-rooted instincts. For facial expressions, the slightest deviation from truth is something any person will immediately detect.

A number of approaches have been developed to model and animate realistic facial expressions in three dimensions. (The reader is referred to the recent book by Parke and Waters [36] for an excellent survey of this entire field.) Parke's pioneering work introduced simple geometric interpolation between face models that were digitized by hand [34]. A radically different approach is performance-based animation, in which measurements from real actors are used to drive synthetic characters [4, 13, 47]. Today, face models can also be obtained using laser-based cylindrical scanners, such as those produced by Cyberware [8]. The resulting range and color data can be fitted with a structured face mesh, augmented with a physically-based model of skin and muscles [29, 30, 43, 46]. The animations produced using these face models represent the state-of-the-art in automatic physically-based facial animation.

For sheer photorealism, one of the most effective approaches to date has been the use of 2D morphing between photographic images [3]. Indeed, some remarkable results have been achieved in this way — most notably, perhaps, the Michael Jackson video produced by PDI, in which very different-looking actors are seemingly transformed into one another as they dance. The production of this video, however, required animators to painstakingly specify a few dozen carefully chosen correspondences between physical features of the actors in almost every frame. Another problem with 2D image morphing is that it does not correctly account for changes in viewpoint or object pose. Although this shortcoming has been recently addressed by a technique called "view morphing" [39], 2D morphing still lacks some of the advantages of a 3D model, such as the complete freedom of viewpoint and the ability to composite the image with other 3D graphics. Morphing has also been applied in 3D: Chen *et al.* [6] applied Beier and Neely's 2D morphing technique [3] to morph between cylindrical laser scans of human heads. Still, even in this case the animator must specify correspondences for every pair of expressions in order to produce a transition between them. More recently, Bregler *et al.* [5] used morphing of mouth regions to lip-synch existing video to a novel sound-track.

In this paper, we show how 2D morphing techniques can be combined with 3D transformations of a geometric model to automatically produce 3D facial expressions with a high degree of realism. Our process consists of several basic steps. First, we capture multiple views of a human subject (with a given facial expression) using cameras at arbitrary locations. Next, we digitize these photographs and manually mark a small set of initial corresponding points on the face in the different views (typically, corners of the eyes and mouth, tip of the nose, etc.). These points are then used to automatically recover the camera parameters (position, focal length, etc.) corresponding to each photograph, as well as the 3D positions of the marked points in space. The 3D positions are then used to deform a generic 3D face mesh to fit the face of the particular human subject. At this stage, additional corresponding points may be marked to refine the fit. Finally, we extract one or more texture maps for the 3D model from the photos. Either a single view-independent texture map can be extracted, or the original images can be used to perform view-dependent texture mapping. This whole process is repeated for the same human subject, with several different facial expressions. To produce facial animations, we interpolate between two or more different 3D models constructed in this way, while at the same time blending the textures. Since all the 3D models are constructed from the same generic mesh, there is a natural correspondence between all geometric points for performing the morph. Thus, transitions between expressions can be produced entirely automatically once the different face models have been constructed, without having to specify pairwise correspondences between any of the expressions.

Our modeling approach is based on photogrammetric techniques in which images are used to create precise geometry [31, 40]. The earliest such techniques applied to facial modeling and animation employed grids that were drawn directly on the human subject's face [34, 35]. One consequence of these grids, however, is that the images used to construct geometry can no longer be used as valid texture maps for the subject. More recently, several methods have been proposed for modeling the face photogrammetrically without the use of grids [20, 24]. These modeling methods are similar in concept to the modeling technique described in this paper. However, these previous techniques use a small predetermined set of features to deform the generic face mesh to the particular face being modeled, and offer no mechanism to further improve the fit. Such an approach may perform poorly on faces with unusual features or other significant deviations from the normal. Our system, by contrast, gives the user complete freedom in specifying the correspondences, and enables the user to refine the initial fit as needed. Another advantage of our technique is its ability to handle fairly arbitrary camera positions and lenses, rather than using a fixed pair that are precisely oriented. Our method is similar, in concept, to the work done in architectural modeling by Debevec *et al.* [9], where a set of annotated photographs are used to model buildings starting from a rough description of their shape. Compared to facial modeling methods that utilize a laser scanner, our technique uses simpler acquisition equipment (regular cameras), and it is capable of extracting texture maps of higher resolution. (Cyberware scans typically produce a cylindrical grid of 512 by 256 samples). The price we pay for these advantages is the need for user intervention in the modeling process.

We employ our system not only for creating realistic face models, but also for performing realistic transitions between different expressions. One advantage of our technique, compared to more traditional animatable models with a single texture map, is that we can capture the subtle changes in illumination and appearance (e.g., facial creases) that occur as the face is deformed. This degree of realism is difficult to achieve even with physically-based models, because of the complexity of skin folding and the difficulty of simulating interreflections and self-shadowing [18, 21, 32].

This paper also presents several new expression synthesis techniques based on extensions to the idea of morphing. We develop a morphing technique that allows for different regions of the face to have different "percentages" or "mixing proportions" of facial expressions. We also introduce a painting interface, which allows users to locally add in a little bit of an expression to an existing composite expression. We believe that these novel methods for expression generation and animation may be more natural for the average user than more traditional animation systems, which rely on the manual adjustments of dozens or hundreds of control parameters.

The rest of this paper is organized as follows. Section 2 describes our method for fitting a generic face mesh to a collection of simultaneous photographs of an individual's head. Section 3 describes our technique for extracting both view-dependent and view-independent texture maps for photorealistic rendering of the face. Section 4 presents the face morphing algorithm that is used to animate the face model. Section 5 describes the key aspects of our system's user interface. Section 6 presents the results of our experiments with the proposed techniques, and Section 7 offers directions for future research.

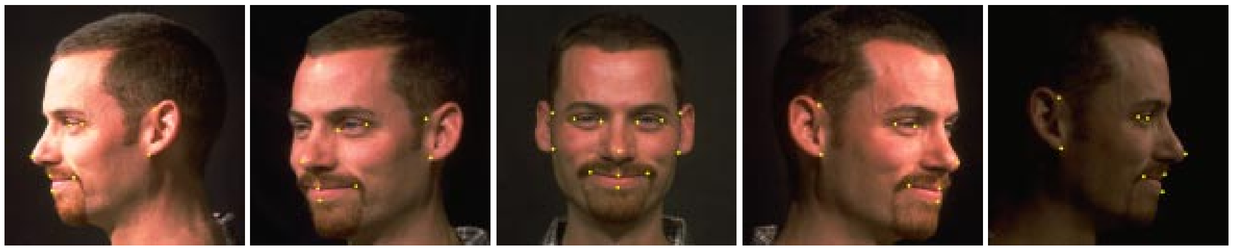
2 Model fitting

The task of the model-fitting process is to adapt a generic face model to fit an individual's face and facial expression. As input to this process, we take several images of the face from different viewpoints (Figure 1a) and a generic face model (we use the generic face model created with Alias|Wavefront [2] shown in Figure 1c). A few features points are chosen (13 in this case, shown in the frames of Figure 1a) to recover the camera pose. These same points are also used to refine the generic face model (Figure 1d). The model can be further refined by drawing corresponding curves in the different views (Figure 1b). The output of the process is a face model that has been adapted to fit the face in the input images (Figure 1e), along with a precise estimate of the camera pose corresponding to each input image.

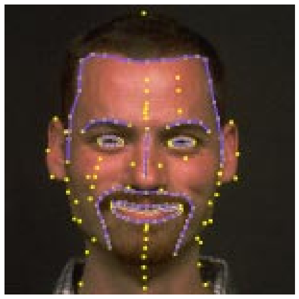
The model-fitting process consists of three stages. In the *pose recovery* stage, we apply computer vision techniques to estimate the viewing parameters (position, orientation, and focal length) for each of the input cameras. We simultaneously recover the 3D coordinates of a set of *feature points* on the face. These feature points are selected interactively from among the face mesh vertices, and their positions in each image (where visible) are specified by hand. The *scattered data interpolation* stage uses the estimated 3D coordinates of the feature points to compute the positions of the remaining face mesh vertices. In the *shape refinement* stage, we specify additional correspondences between facial vertices and image coordinates to improve the estimated shape of the face (while keeping the camera pose fixed).

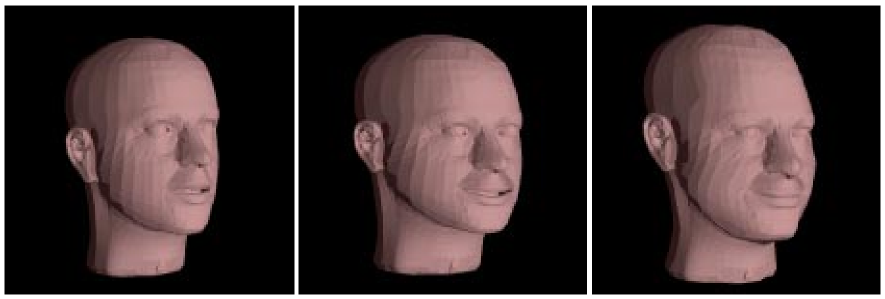
2.1 Pose recovery

Starting with a rough knowledge of the camera positions (e.g., frontal view, side view, etc.) and of the 3D shape (given by the generic head model), we iteratively improve the pose and the 3D shape estimates in order to minimize the difference between the predicted and observed feature point positions. Our formulation is based on the non-linear least squares structure-from-motion algorithm introduced by Szeliski and Kang [41]. However, unlike the method they describe, which uses the Levenberg-Marquardt algorithm to perform a complete iterative minimization over all of the unknowns simultaneously, we break the problem down into a series of linear least squares problems that can be solved using very simple



(a)





(b) (c) (d) (e)

Figure 1 Model-fitting process: (a) a set of input images with marked feature points, (b) facial features annotated using a set of curves, (c) generic face geometry (shaded surface rendering), (d) face adapted to initial 13 feature points (after pose estimation) (e) face after 99 additional correspondences have been given.

and numerically stable techniques [16, 37].

To formulate the pose recovery problem, we associate a rotation matrix $\boldsymbol{R}^k$ and a translation vector $\boldsymbol{t}^k$ with each camera pose k. (The three rows of $\boldsymbol{R}^k$ are $\boldsymbol{r}_x^k$, $\boldsymbol{r}_y^k$, and $\boldsymbol{r}_z^k$, and the three entries in $\boldsymbol{t}^k$ are t_x^k, t_y^k, t_z^k.) We write each 3D feature point as $\boldsymbol{p}_i$, and its 2D screen coordinates in the k-th image as (x_i^k, y_i^k).

Assuming that the origin of the (x, y) image coordinate system lies at the optical center of each image (i.e., where the optical axis intersects the image plane), the traditional 3D projection equation for a camera with a focal length f^k (expressed in pixels) can be written as

$$x_i^k = f^k \frac{\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k}{\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i + t_z^k} \qquad y_i^k = f^k \frac{\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k}{\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i + t_z^k} \tag{1}$$

(This is just an explicit rewriting of the traditional projection equation $\boldsymbol{x}_i^k \propto \boldsymbol{R}^k \boldsymbol{p}_i + \boldsymbol{t}^k$ where $\boldsymbol{x}_i^k = (x_i^k, y_i^k, f^k)$.)

Instead of using (1) directly, we reformulate the problem to estimate inverse distances to the object [41]. Let $\eta^k = 1/t_z^k$ be this inverse distance and $s^k = f^k \eta^k$ be a world-to-image scale factor. The advantage of this formulation is that the scale factor s^k can be reliably estimated even when the focal length is long, whereas the original formulation has a strong coupling between the f^k and t_z^k parameters.

Performing these substitution, we obtain

$$\begin{aligned} x_i^k &= s^k \frac{\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k}{1 + \eta^k \boldsymbol{r}_z^k \cdot \boldsymbol{p}_i} \\ y_i^k &= s^k \frac{\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k}{1 + \eta^k \boldsymbol{r}_z^k \cdot \boldsymbol{p}_i}. \end{aligned}$$

If we let $w_i^k = (1 + \eta^k(\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i))^{-1}$ be the inverse denominator, and collect terms on the left-hand side, we get

$$\begin{aligned} w_i^k \left(x_i^k + x_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k) \right) &= 0 \qquad (2) \\ w_i^k \left(y_i^k + y_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k) \right) &= 0 \end{aligned}$$

Note that these equations are linear in each of the unknowns that we wish to recover, i.e., $\boldsymbol{p}_i$, t_x^k, t_y^k, η^k, s^k, and $\boldsymbol{R}^k$, if we ignore the variation of w_i^k with respect to these parameters. (The reason we keep the w_i^k term, rather than just dropping it from these equations, is so that the linear equations being solved in the least squares step have the same magnitude as the original measurements (x_i^k, y_i^k). Hence, least-squares will produce a *maximum likelihood* estimate for the unknown parameters [26].)

Given estimates for initial values, we can solve for different subsets of the unknowns. In our current algorithm, we solve for the unknowns in five steps: first s^k, then $\boldsymbol{p}_i$, $\boldsymbol{R}^k$, t_x^k and t_y^k, and finally η^k. This order is chosen to provide maximum numerical stability given the crude initial pose and shape estimates. For each parameter or set of parameters chosen, we solve for the unknowns using linear least squares (Appendix A). The simplicity of this approach is a result of solving for the unknowns in five separate stages, so that the parameters for a given camera or 3D point can be recovered independently of the other parameters.

2.2 Scattered data interpolation

Once we have computed an initial set of coordinates for the feature points $\boldsymbol{p}_i$, we use these values to deform the remaining vertices on the face mesh. We construct a smooth interpolation function that gives the 3D displacements between the original point positions and the new adapted positions for every vertex in the original generic face mesh. Constructing such an interpolation function is a standard problem in scattered data interpolation. Given a set of known displacements $\boldsymbol{u}_i = \boldsymbol{p}_i - \boldsymbol{p}_i^{(0)}$ away from the original positions $\boldsymbol{p}_i^{(0)}$ at every constrained vertex i, construct a function that gives the displacement $\boldsymbol{u}_j$ for every unconstrained vertex j.

There are several considerations in choosing the particular data interpolant [33]. The first consideration is the embedding space, that is, the domain of the function being computed. In our case, we use the original 3D coordinates of the points as the domain. (An alternative would be to use some 2D parameterization of the surface mesh, for instance, the cylindrical coordinates described in Section 3.) We therefore attempt to find a smooth vector-valued function $\boldsymbol{f}(\boldsymbol{p})$ fitted to the known data $\boldsymbol{u}_i = \boldsymbol{f}(\boldsymbol{p}_i)$, from which we can compute $\boldsymbol{u}_j = \boldsymbol{f}(\boldsymbol{p}_j)$.

There are also several choices for how to construct the interpolating function [33]. We use a method based on *radial basis functions*, that is, functions of the form

$$\boldsymbol{f}(\boldsymbol{p}) = \sum_i \boldsymbol{c}_i \phi(\|\boldsymbol{p} - \boldsymbol{p}_i\|),$$

where $\phi(r)$ are radially symmetric basis functions. A more general form of this interpolant also adds some low-order polynomial terms to model global, e.g., affine, deformations [27, 28, 33]. In our system, we use an affine basis as part of our interpolation algorithm, so that our interpolant has the form:

$$\boldsymbol{f}(\boldsymbol{p}) = \sum_i \boldsymbol{c}_i \phi(\|\boldsymbol{p} - \boldsymbol{p}_i\|) + \boldsymbol{M}\boldsymbol{p} + \boldsymbol{t}, \qquad (3)$$

To determine the coefficients $\boldsymbol{c}_i$ and the affine components $\boldsymbol{M}$ and $\boldsymbol{t}$, we solve a set of linear equations that includes the interpolation constraints $\boldsymbol{u}_i = \boldsymbol{f}(\boldsymbol{p}_i)$, as well as the constraints $\sum_i \boldsymbol{c}_i = 0$ and $\sum_i \boldsymbol{c}_i \boldsymbol{p}_i^{\mathrm{T}} = 0$, which remove affine contributions from the radial basis functions.

Many different functions for $\phi(r)$ have been proposed [33]. After experimenting with a number of functions, we have chosen to use $\phi(r) = e^{-r/64}$, with units measured in inches.

Figure 1d shows the shape of the face model after having interpolated the set of computed 3D displacements at 13 feature points shown in Figure 1 and applied them to the entire face.

2.3 Correspondence-based shape refinement

After warping the generic face model into its new shape, we can further improve the shape by specifying additional correspondences. Since these correspondences may not be as easy to locate correctly, we do not use them to update the camera pose estimates. Instead, we simply solve for the values of the new feature points $\boldsymbol{p}_i$ using a simple least-squares fit, which corresponds to finding the point nearest the intersection of the viewing rays in 3D. We can then re-run the scattered data interpolation algorithm to update the vertices for which no correspondences are given. This process can be repeated until we are satisfied with the shape.

Figure 1e shows the shape of the face model after 99 additional correspondences have been specified. To facilitate the annotation process, we grouped vertices into polylines. Each polyline corresponds to an easily identifiable facial feature such as the eyebrow, eyelid, lips, chin, or hairline. The features can be annotated by outlining them with hand-drawn curves on each photograph where they are visible. The curves are automatically converted into a set of feature points by stepping along them using an arc-length parametrization. Figure 1b shows annotated facial features using a set of curves on the front view.

3 Texture extraction

In this section we describe the process of extracting the texture maps necessary for rendering photorealistic images of a reconstructed face model from various viewpoints.

The texture extraction problem can be defined as follows. Given a collection of photographs, the recovered viewing parameters, and the fitted face model, compute for each point $\boldsymbol{p}$ on the face model its texture color $T(\boldsymbol{p})$.

Each point $\boldsymbol{p}$ may be visible in one or more photographs; therefore, we must identify the corresponding point in each photograph and decide how these potentially different values should be combined

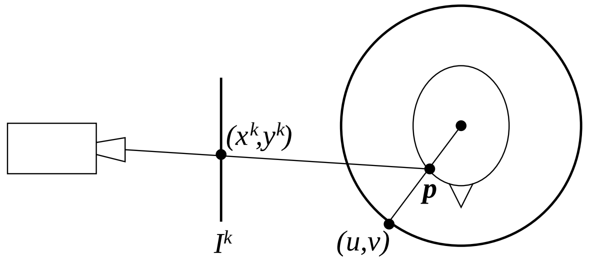


Figure 2 Geometry for texture extraction

(blended) together. There are two principal ways to blend values from different photographs: *view-independent blending*, resulting in a texture map that can be used to render the face from any viewpoint; and *view-dependent blending*, which adjusts the blending weights at each point based on the direction of the current viewpoint [9, 38]. Rendering takes longer with view-dependent blending, but the resulting image is of slightly higher quality (see Figure 3).

3.1 Weight maps

As outlined above, the texture value $T(\boldsymbol{p})$ at each point on the face model can be expressed as a convex combination of the corresponding colors in the photographs:

$$T(\boldsymbol{p}) = \frac{\sum_k m^k(\boldsymbol{p})\, I^k(x^k, y^k)}{\sum_k m^k(\boldsymbol{p})}.$$

Here, I^k is the image function (color at each pixel of the k-th photograph,) and (x^k, y^k) are the image coordinates of the projection of $\boldsymbol{p}$ onto the k-th image plane. The *weight map* $m^k(\boldsymbol{p})$ is a function that specifies the contribution of the k-th photograph to the texture at each facial surface point.

The construction of these weight maps is probably the trickiest and the most interesting component of our texture extraction technique. There are several important considerations that must be taken into account when defining a weight map:

1. *Self-occlusion:* $m^k(\boldsymbol{p})$ should be zero unless $\boldsymbol{p}$ is front-facing with respect to the k-th image and visible in it.
2. *Smoothness:* the weight map should vary smoothly, in order to ensure a seamless blend between different input images.
3. *Positional certainty:* $m^k(\boldsymbol{p})$ should depend on the “positional certainty” [24] of $\boldsymbol{p}$ with respect to the k-th image. The positional certainty is defined as the dot product between the surface normal at $\boldsymbol{p}$ and the k-th direction of projection.
4. *View similarity:* for view-dependent texture mapping, the weight $m^k(\boldsymbol{p})$ should also depend on the angle between the direction of projection of $\boldsymbol{p}$ onto the j-th image and its direction of projection in the new view.

Previous authors have taken only a subset of these considerations into account when designing their weighting functions. For example, Kurihara and Arai [24] use positional certainty as their weighting function, but they do not account for self-occlusion. Akimoto *et al.* [1] and Ip and Yin [20] blend the images smoothly, but address neither self-occlusion nor positional certainty. Debevec *et al.* [9], who describe a view-dependent texture mapping technique for modeling and rendering buildings from photographs, do address occlusion but do not account for positional certainty. (It should be noted, however, that positional certainty is less critical in photographs of buildings, since most buildings do not tend to curve away from the camera.)

To facilitate fast visibility testing of points on the surface of the face from a particular camera pose, we first render the face model using the recovered viewing parameters and save the resulting depth map from the Z-buffer. Then, with the aid of this depth map, we can quickly classify the visibility of each facial point by applying the viewing transformation and comparing the resulting depth to the corresponding value in the depth map.

3.2 View-independent texture mapping

In order to support rapid display of the textured face model from any viewpoint, it is desirable to blend the individual photographs together into a single texture map. This texture map is constructed on a virtual cylinder enclosing the face model. The mapping between the 3D coordinates on the face mesh and the 2D texture space is defined using a cylindrical projection, as in several previous papers [6, 24, 29].

For view-independent texture mapping, we will index the weight map m^k by the (u, v) coordinates of the texture being created. Each weight $m^k(u, v)$ is determined by the following steps:

1. Construct a feathered visibility map F^k for each image k. These maps are defined in the same cylindrical coordinates as the texture map. We initially set $F^k(u, v)$ to 1 if the corresponding facial point $\boldsymbol{p}$ is visible in the k-th image, and to 0 otherwise. The result is a binary visibility map, which is then smoothly ramped (feathered) from 1 to 0 in the vicinity of the boundaries [42]. A cubic polynomial is used as the ramping function.
2. Compute the 3D point $\boldsymbol{p}$ on the surface of the face mesh whose cylindrical projection is (u, v) (see Figure 2). This computation is performed by casting a ray from (u, v) on the cylinder towards the cylinder's axis. The first intersection between this ray and the face mesh is the point $\boldsymbol{p}$. (Note that there can be more than one intersection for certain regions of the face, most notably the ears. These special cases are discussed in Section 3.4.) Let $P^k(\boldsymbol{p})$ be the positional certainty of $\boldsymbol{p}$ with respect to the k-th image.
3. Set weight $m^k(u, v)$ to the product $F^k(u, v)\, P^k(\boldsymbol{p})$.

For view-independent texture mapping, we will compute each pixel of the resulting texture $T(u, v)$ as a weighted sum of the original image functions, indexed by (u, v).

3.3 View-dependent texture mapping

The main disadvantage of the view-independent cylindrical texture map described above is that its construction involves blending together resampled versions of the original images of the face. Because of this resampling, and also because of slight registration errors, the resulting texture is slightly blurry. This problem can be alleviated to a large degree by using a view-dependent texture map [9] in which the blending weights are adjusted dynamically, according to the current view.

For view-dependent texture mapping, we render the model several times, each time using a different input photograph as a texture map, and blend the results. More specifically, for each input photograph, we associate texture coordinates and a blending weight with each vertex in the face mesh. (The rendering hardware performs perspective-correct texture mapping along with linear interpolation of the blending weights.)

Given a viewing direction $\boldsymbol{d}$, we first select the subset of photographs used for the rendering and then assign blending weights to each of these photographs. Pulli *et al.* [38] select three photographs based on a Delaunay triangulation of a sphere surrounding the object. Since our cameras were positioned roughly in the same plane,

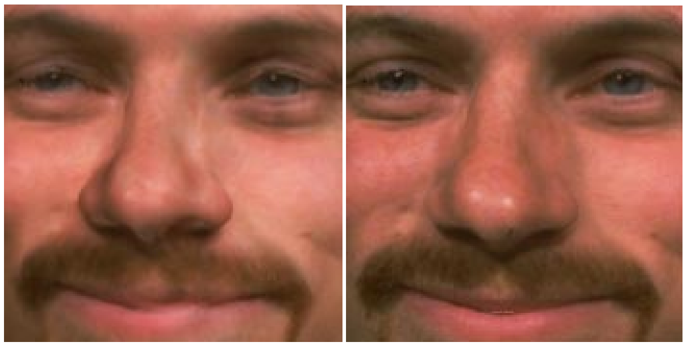


Figure 3 Comparison between view-independent (left) and view-dependent (right) texture mapping. Higher frequency details are visible in the view-dependent rendering.

we select just the two photographs whose view directions $\boldsymbol{d}^\ell$ and $\boldsymbol{d}^{\ell+1}$ are the closest to $\boldsymbol{d}$ and blend between the two.

In choosing the view-dependent term $V^k(\boldsymbol{d})$ of the blending weights, we wish to use just a single photo if that photo's view direction matches the current view direction precisely, and to blend smoothly between the nearest two photos otherwise. We used the simplest possible blending function having this effect:

$$V^k(\boldsymbol{d}) = \begin{cases} \boldsymbol{d}\cdot\boldsymbol{d}^k - \boldsymbol{d}^\ell\cdot\boldsymbol{d}^{\ell+1} & \text{if } \ell \le k \le \ell+1 \\ 0 & \text{otherwise} \end{cases}$$

For the final blending weights $m^k(\boldsymbol{p}, \boldsymbol{d})$, we then use the product of all three terms $F^k(x^k, y^k)\, P^k(\boldsymbol{p})\, V^k(\boldsymbol{d})$.

View-dependent texture maps have several advantages over cylindrical texture maps. First, they can make up for some lack of detail in the model. Second, whenever the model projects onto a cylinder with overlap, a cylindrical texture map will not contain data for some parts of the model. This problem does not arise with view-dependent texture maps if the geometry of the mesh matches the photograph properly. One disadvantage of the view-dependent approach is its higher memory requirements and slower speed due to the multi-pass rendering. Another drawback is that the resulting images are much more sensitive to any variations in exposure or lighting conditions in the original photographs.

3.4 Eyes, teeth, ears, and hair

The parts of the mesh that correspond to the eyes, teeth, ears, and hair are textured in a separate process. The eyes and teeth are usually partially occluded by the face; hence it is difficult to extract a texture map for these parts in every facial expression. The ears have an intricate geometry with many folds and usually fail to project without overlap on a cylinder. The hair has fine-detailed texture that is difficult to register properly across facial expressions. For these reasons, each of these facial elements is assigned an individual texture map. The texture maps for the eyes, teeth, and ears are computed by projecting the corresponding mesh part onto a selected input image where that part is clearly visible (the front view for eyes and teeth, side views for ears).

The eyes and the teeth are usually partially shadowed by the eyelids and the mouth respectively. We approximate this shadowing by modulating the brightness of the eye and teeth texture maps according to the size of the eyelid and mouth openings.

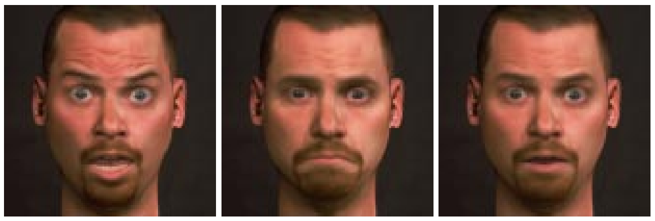


Figure 4 A global blend between "surprised" (left) and "sad" (center) produces a "worried" expression (right).

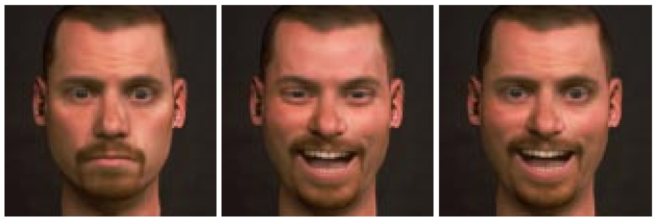


Figure 5 Combining the upper part of a "neutral" expression (left) with the lower part of a "happy" expression (center) produces a "fake smile" (right).

4 Expression morphing

A major goal of this work is the generation of continuous and realistic transitions between different facial expressions. We achieve these effects by morphing between corresponding face models.

In general the problem of morphing between arbitrary polygonal meshes is a difficult one [22], since it requires a set of correspondences between meshes with potentially different topology that can produce a reasonable set of intermediate shapes. In our case, however, the topology of all the face meshes is identical. Thus, there is already a natural correspondence between vertices. Furthermore, in creating the models we attempt to mark facial features consistently across different facial expressions, so that the major facial features correspond to the same vertices in all expressions. In this case, a satisfactory 3D morphing sequence can be obtained using simple linear interpolation between the geometric coordinates of corresponding vertices in each of the two face meshes.

Together with the geometric interpolation, we need to blend the associated textures. Again, in general, morphing between two images requires pairwise correspondences between images features [3]. In our case, however, correspondences between the two textures are implicit in the texture coordinates of the two associated face meshes. Rather than warping the two textures to form an intermediate one, the intermediate face model (obtained by geometric interpolation) is rendered once with the first texture, and again with the second. The two resulting images are then blended together. This approach is faster than warping the textures (which typically have high resolution), and it avoids the resampling that is typically performed during warping.

4.1 Multiway blend and localized blend

Given a set of facial expression meshes, we have explored ways to enlarge this set by combining expressions. The simplest approach is to use the morphing technique described above to create new facial expressions, which can be added to the set. This idea can be generalized to an arbitrary number of starting expressions by taking convex combinations of them all, using weights that apply both to the coordinates of the mesh vertices and to the values in the texture map. (Extrapolation of expressions should also be possible by allowing weights to have values outside of the interval $[0, 1]$; note, however, that such weights might result in colors outside of the allowable gamut.)

We can generate an even wider range of expressions using a localized blend of the facial expressions. Such a blend is specified by a set of blend functions, one for each expression, defined over the vertices of the mesh. These blend functions describe the contribution of a given expression at a particular vertex.

Although it would be possible to compute a texture map for each new expression, doing so would result in a loss of texture quality. Instead, the weights for each new blended expression are always factored into weights over the vertices of the original set of expressions. Thus, each blended expression is rendered using the texture map of an original expression, along with weights at each vertex, which control the opacity of that texture. The opacities are linearly interpolated over the face mesh using Gouraud shading.

4.2 Blend specification

In order to design new facial expressions easily, the user must be provided with useful tools for specifying the blending functions. These tools should satisfy several requirements. First, it should be possible to edit the blend at different resolutions. Moreover, we would like the specification process to be continuous so that small changes in the blend parameters do not trigger radical changes in the resulting expression. Finally, the tools should be intuitive to the user; it should be easy to produce a particular target facial expression from an existing set.

We explored several different ways of specifying the blending weights:

- *Global blend.* The blending weights are constant over all vertices. A set of sliders controls the mixing proportions of the contributing expressions. Figure 4 shows two facial expressions blended in equal proportions to produce a halfway blend.
- *Regional blend.* According to studies in psychology, the face can be split into several regions that behave as coherent units [11]. Usually, three regions are considered: one for the forehead (including the eyebrows), another for the eyes, and another for the lower part of the face. Further splitting the face vertically down the center results in six regions and allows for asymmetric expressions. We similarly partition the face mesh into several (softly feathered) regions and assign weights so that vertices belonging to the same region have the same weights. The mixing proportions describing a selected region can be adjusted by manipulating a set of sliders. Figure 5 illustrates the blend of two facial expressions with two regions: the upper part of the face (including eyes and forehead) and the lower part (including nose, mouth, and chin.)
- *Painterly interface.* The blending weights can be assigned to the vertices using a 3D painting tool. This tool uses a palette in which the "colors" are facial expressions (both geometry and color), and the "opacity" of the brush controls how much the expression contributes to the result. Once an expression is selected, a 3D brush can be used to modify the blending weights in selected areas of the mesh. The fraction painted has a gradual drop-off and is controlled by the opacity of the brush. The strokes are applied directly on the rendering of the current facial blend, which is updated in real-time. To improve the rendering speed, only the portion of the mesh that is being painted is re-rendered. Figure 7 illustrates the design of a debauched smile: starting with a neutral expression, the face is locally modified using three other expressions. Note that in the last step, the use of a partially transparent brush with the "sleepy" expression results in the actual geometry of the eyelids becoming partially lowered.

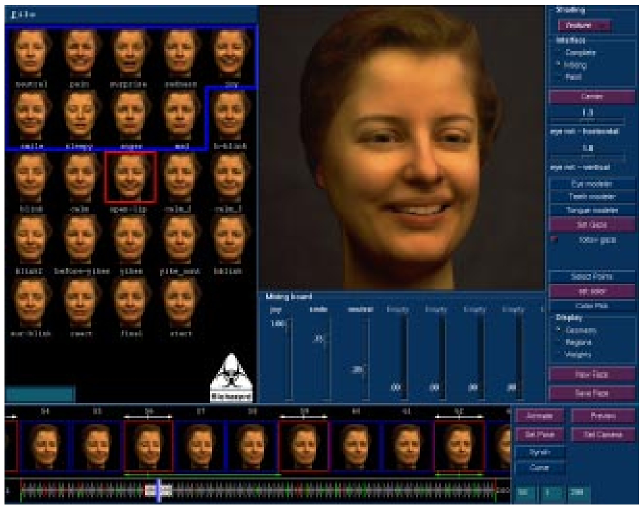


Figure 6 Animation interface. On the left is the "expression gallery"; on the right an expression is being designed. At the bottom expressions and poses are scheduled on the timeline.

Combining different original expressions enlarges the repertoire of expressions obtained from a set of photographs. The expressions in this repertoire can themselves be blended to create even more expressions, with the resulting expression still being representable as a (locally varying) linear combination of the original expressions.

5 User interface

We designed an interactive tool to fit a 3D face mesh to a set of images. This tool allows a user to select vertices on the mesh and mark where these curves or vertices should project on the images. After a first expression has been modeled, the set of annotations can be used as an initial guess for subsequent expressions. These guesses are automatically refined using standard correlation-based search. Any resulting errors can be fixed up by hand. The extraction of the texture map does not require user intervention, but is included in the interface to provide feedback during the modeling phase.

We also designed a keyframe animation system to generate facial animations. Our animation system permits a user to blend facial expressions and to control the transitions between these different expressions (Figure 6). The expression gallery is a key component of our system; it is used to select and display (as thumbnails) the set of facial expressions currently available. The thumbnails can be dragged and dropped onto the timeline (to set keyframes) or onto the facial design interface (to select or add facial expressions). The timeline is used to schedule the different expression blends and the changes in viewing parameters (pose) during the animation. The blends and poses have two distinct types of keyframes. Both types of keyframes are linearly interpolated with user-controlled cubic Bézier curves. The timeline can also be used to display intermediate frames at low resolution to provide a quick feedback to the animator. A second timeline can be displayed next to the composition timeline. This feature is helpful for correctly synchronizing an animation with live video or a soundtrack. The eyes are animated separately from the rest of the face, with the gaze direction parameterized by two Euler angles.

6 Results

In order to test our technique, we photographed both a man (J. R.) and a woman (Karla) in a variety of facial expressions. The photog-

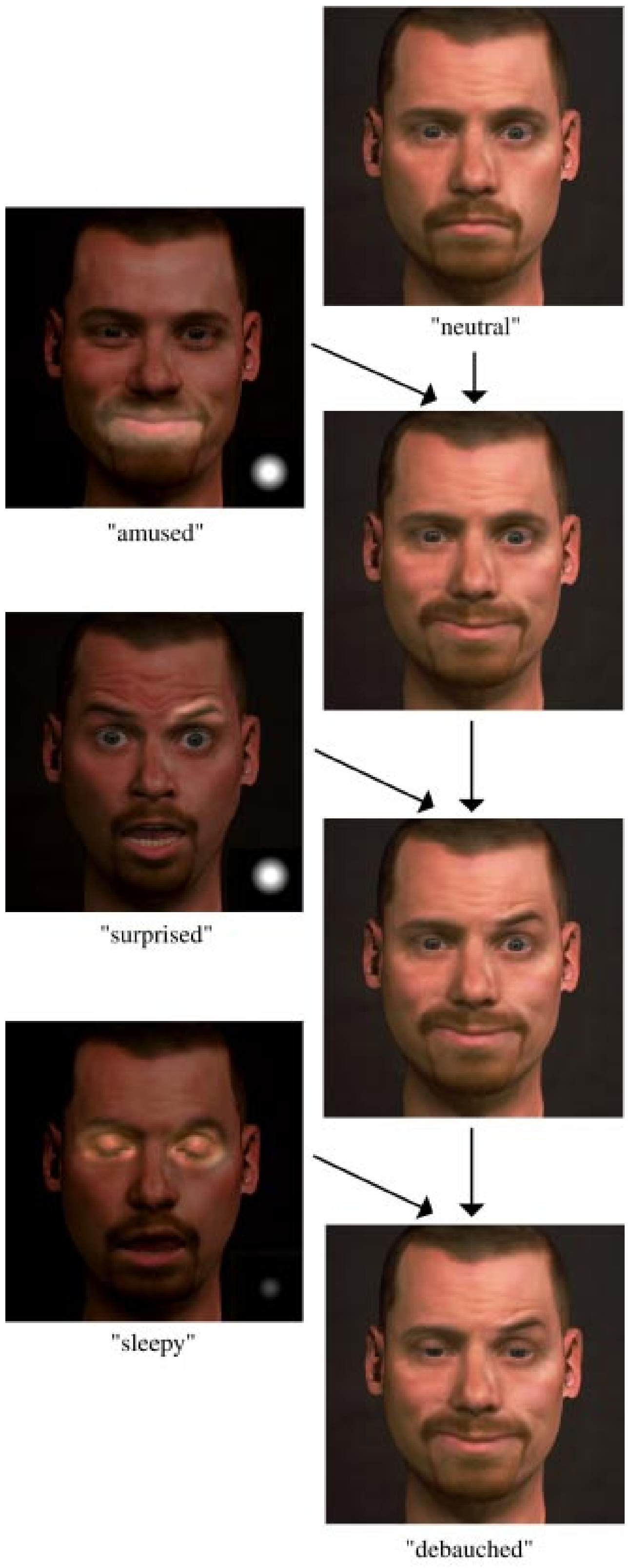


Figure 7 Painterly interface: design of a debauched smile. The right column shows the different stages of the design; the left column shows the portions of the original expressions used in creating the final expression. The "soft brush" used is shown at the bottom-right corner of each contributing expression.

raphy was performed using five cameras simultaneously. The cameras were not calibrated in any particular way, and the lenses had different focal lengths. Since no special attempt was made to illuminate the subject uniformly, the resulting photographs exhibited considerable variation in both hue and brightness. The photographs were digitized using the Kodak PhotoCD process. Five typical images (cropped to the size of the subject's head) are shown in Figure 1a.

We used the interactive modeling system described in Sections 2 and 3 to create the same set of eight face models for each subject: "happy," "amused," "angry," "surprised," "sad," "sleepy," "pained," and "neutral."

Following the modeling stage, we generated a facial animation for each of the individuals starting from the eight original expressions. We first created an animation for J. R. We then applied the very same morphs specified by this animation to the models created for Karla. For most frames of the animation, the resulting expressions were quite realistic. Figure 8 shows five frames from the animation sequence for J. R. and the purely automatically generated frames in the corresponding animation for Karla. With just a small amount of additional retouching (using the blending tools described in Section 4.2), this derivative animation can be made to look as good as the original animation for J. R.

7 Future work

The work described in this paper is just the first step towards building a complete image-based facial modeling and animation system. There are many ways to further enhance and extend the techniques that we have described:

Color correction. For better color consistency in facial textures extracted from photographs, color correction should be applied to simultaneous photographs of each expression.

Improved registration. Some residual ghosting or blurring artifacts may occasionally be visible in the cylindrical texture map due to small misregistrations between the images, which can occur if geometry is imperfectly modeled or not detailed enough. To improve the quality of the composite textures, we could locally warp each component texture (and weight) map before blending [42].

Texture relighting. Currently, extracted textures reflect the lighting conditions under which the photographs were taken. Relighting techniques should be developed for seamless integration of our face models with other elements.

Automatic modeling. Our ultimate goal, as far as the facial modeling part is concerned, is to construct a fully automated modeling system, which would automatically find features and correspondences with minimal user intervention. This is a challenging problem indeed, but recent results on 2D face modeling in computer vision [25] give us cause for hope.

Modeling from video. We would like to be able to create face models from video or old movie footage. For this purpose, we would have to improve the robustness of our techniques in order to synthesize face meshes and texture maps from images that do not correspond to different views of the same expression. Adding anthropomorphic constraints to our face model might make up for the lack of coherence in the data [48].

Complex animations. In order to create complex animations, we must extend our vocabulary for describing facial movements beyond blending between different expressions. There are several potential ways to attack this problem. One would be to adopt an action-unit-based system such as the Facial Action Coding System

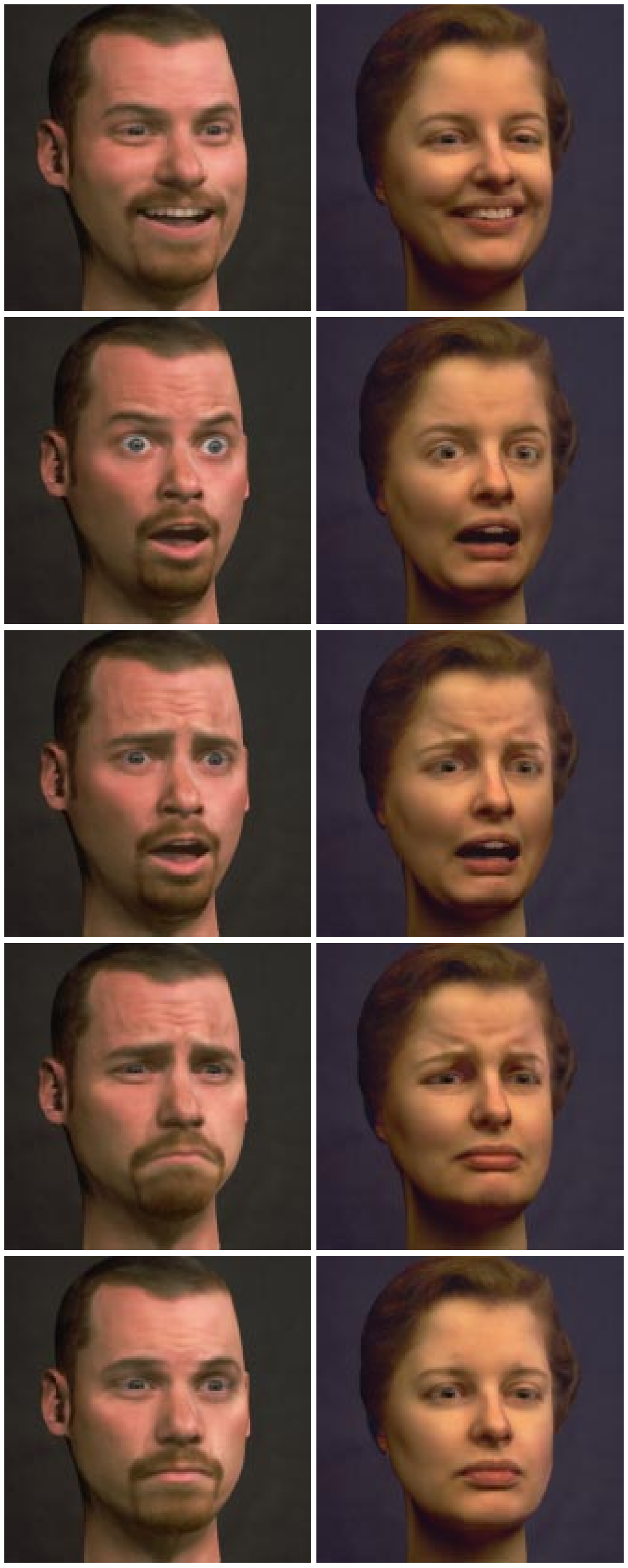

(a) (b)

Figure 8 On the left are frames from an original animation, which we created for J. R. The morphs specified in these frames were then directly used to create a derivative animation for Karla, shown on the right.

(FACS) [12]. Another possibility would be to apply modal analysis (principal component analysis) techniques to describe facial expression changes using a small number of motions [25]. Finding natural control parameters to facilitate animation and developing realistic-looking temporal profiles for such movements are also challenging research problems.

Lip-synching. Generating speech animation with our keyframe animation system would require a large number of keyframes. However, we could use a technique similar to that of Bregler *et al.* [5] to automatically lip-synch an animation to a sound-track. This would require the synthesis of face models for a wide range of visemes. For example, such database of models could be constructed using video footage to reconstruct face models automatically [17].

Performance-driven animation. Ultimately, we would also like to support performance-driven animation, i.e., the ability to automatically track facial movements in a video sequence, and to automatically translate these into animation control parameters. Our current techniques for registering images and converting them into 3D movements should provide a good start, although they will probably need to be enhanced with feature-tracking techniques and some rudimentary expression-recognition capabilities. Such a system would enable not only very realistic facial animation, but also a new level of video coding and compression techniques (since only the expression parameters would need to be encoded), as well as real-time control of avatars in 3D chat systems.

8 Acknowledgments

We would like to thank Katrin Petersen and Andrew Petty for modeling the generic face model, Cassidy Curtis for his invaluable advice on animating faces, and Joel Auslander and Jason Griffith for early contributions to this project. This work was supported by an NSF Presidential Faculty Fellow award (CCR-9553199), an ONR Young Investigator award (N00014-95-1-0728), and industrial gifts from Microsoft and Pixar.

References

[1] Takaaki Akimoto, Yasuhito Suenaga, and Richard S. Wallace. Automatic Creation of 3D Facial Models. *IEEE Computer Graphics and Applications*, 13(5):16–22, September 1993.

[2] Alias | Wavefront, Toronto, Ontario. *Alias V7.0*, 1995.

[3] Thaddeus Beier and Shawn Neely. Feature-based Image Metamorphosis. In *SIGGRAPH 92 Conference Proceedings*, pages 35–42. ACM SIGGRAPH, July 1992.

[4] Philippe Bergeron and Pierre Lachapelle. Controlling Facial Expressions and Body Movements in the Computer-Generated Animated Short "Tony De Peltrie". In *SIGGRAPH 85 Advanced Computer Animation seminar notes*. July 1985.

[5] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video Rewrite: Driving Visual Speech with Audio. In *SIGGRAPH 97 Conference Proceedings*, pages 353–360. ACM SIGGRAPH, August 1997.

[6] David T. Chen, Andrei State, and David Banks. Interactive Shape Metamorphosis. In *1995 Symposium on Interactive 3D Graphics*, pages 43–44. ACM SIGGRAPH, April 1995.

[7] Chang S. Choi, Kiyoharu, Hiroshi Harashima, and Tsuyoshi Takebe. Analysis and Synthesis of Facial Image Sequences in Model-Based Image Coding. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 4, pages 257 – 275. June 1994.

[8] Cyberware Laboratory, Inc, Monterey, California. *4020/RGB 3D Scanner with Color Digitizer*, 1990.

[9] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20. ACM SIGGRAPH, August 1996.

[10] Eben Ostby, Pixar Animation Studios. Personal communication, January 1997.

[11] Paul Ekman and Wallace V. Friesen. *Unmasking the Face. A guide to recognizing emotions fron facial clues*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[12] Paul Ekman and Wallace V. Friesen. *Manual for the Facial Action Coding System*. Consulting Psychologists Press, Inc., Palo Alto, California, 1978.

[13] Irfan Essa, Sumit Basu, Trevor Darrell, and Alex Pentland. Modeling, Tracking and Interactive Animation of Faces and Heads Using Input from Video. In *Computer Animation Conference*, pages 68–79. June 1996.

[14] Gary Faigin. *The Artist's Complete Guide to Facial Expression*. Watson-Guptill Publications, New York, 1990.

[15] Olivier Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.

[16] G. Golub and C. F. Van Loan. *Matrix Computation, third edition*. The John Hopkins University Press, Baltimore and London, 1996.

[17] Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Frédéric Pighin. Making Faces. In *SIGGRAPH 98 Conference Proceedings*. ACM SIGGRAPH, July 1998.

[18] Pat Hanrahan and Wolfgang Krueger. Reflection from Layered Surfaces Due to Subsurface Scattering. In *SIGGRAPH 93 Conference Proceedings*, volume 27, pages 165–174. ACM SIGGRAPH, August 1993.

[19] Bright Star Technologies Inc. *Beginning Reading Software.* Sierra On-Line, Inc., 1993.

[20] Horace H. S. Ip and Lijun Yin. Constructing a 3D Individualized Head Model from Two Orthogonal Views. *The Visual Computer*, 12:254–266, 1996.

[21] Gregory Ward J., Francis M. Rubinstein, and Richard D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *SIGGRAPH 88 Conference Proceedings*, volume 22, pages 85–92. August 1988.

[22] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape Transformation for Polyhedral Objects. In *SIGGRAPH 92 Proceedings Conference*, volume 26, pages 47–54. ACM SIGGRAPH, July 1992.

[23] Rolf M. Koch, Markus H. Gross, Friedrich R. Carls, Daniel F. von Büren, George Fankhauser, and Yoav I. H. Parish. Simulating Facial Surgery Using Finite Element Methods. In *SIGGRAPH 96 Conference Proceedings*, pages 421–428. ACM SIGGRAPH, August 1996.

[24] Tsuneya Kurihara and Kiyoshi Arai. A Transformation Method for Modeling and Animation of the Human Face from Photographs. In Nadia Magnenat Thalmann and Daniel Thalmann, editors, *Computer Animation 91*, pages 45–58. Springer-Verlag, Tokyo, 1991.

[25] A. Lanitis, C. J. Taylor, and T. F. Cootes. A Unified Approach for Coding and Interpreting Face Images. In *Fifth International Conference on Computer Vision (ICCV 95)*, pages 368–373. Cambridge, Massachusetts, June 1995.

[26] C. L. Lawson and R. J. Hansen. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, 1974.

[27] Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, and George Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. In *SIGGRAPH 95 Conference Proceedings*, pages 439–448. ACM SIGGRAPH, August 1995.

[28] Seung-Yong Lee, George Wolberg, Kyung-Yong Chwa, and Sung Yong Shin. Image Metamorphosis with Scattered Feature Constraints. *IEEE Transactions on Visualization and Computer Graphics*, 2(4), December 1996.

[29] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic Modeling for Facial Animation. In *SIGGRAPH 95 Conference Proceedings*, pages 55–62. ACM SIGGRAPH, August 1995.

[30] Yuencheng C. Lee, Demetri Terzopoulos, and Keith Waters. Constructing Physics-Based Facial Models of Individuals. In *Proceedings of Graphics Interface 93*, pages 1–8. May 1993.

[31] Francis H. Moffitt and Edward M. Mikhail. *Photogrammetry*. Harper & Row, New York, 3 edition, 1980.

[32] Shree K. Nayar, Katsushi Ikeuchi, and Takeo Kanade. Shape from Interreflections. *International Journal of Computer Vision*, 6:173–195, 1991.

[33] Gregory M. Nielson. Scattered Data Modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, January 1993.

[34] Frederic I. Parke. Computer Generated Animation of Faces. *Proceedings ACM annual conference.*, August 1972.

[35] Frederic I. Parke. A Parametric Model for Human Faces. PhD thesis, University of Utah, Salt Lake City, Utah, December 1974. UTEC-CSc-75-047.

[36] Frederic I. Parke and Keith Waters. *Computer Facial Animation*. A K Peters, Wellesley, Massachusetts, 1996.

[37] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, second edition, 1992.

[38] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Proc. 8th Eurographics Workshop on Rendering*. June 1997.

[39] Steven M. Seitz and Charles R. Dyer. View Morphing. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 21–30. ACM SIGGRAPH, August 1996.

[40] Chester C. Slama, editor. *Manual of Photogrammetry*. American Society of Photogrammetry, Falls Church, Virginia, fourth edition, 1980.

[41] Richard Szeliski and Sing Bing Kang. Recovering 3D Shape and Motion from Image Streams using Nonlinear Least Squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, March 1994.

[42] Richard Szeliski and Heung-Yeung Shum. Creating Full View Panoramic Image Mosaics and Texture-Mapped Models. In *SIGGRAPH 97 Conference Proceedings*, pages 251–258. ACM SIGGRAPH, August 1997.

[43] Demetri Terzopoulos and Keith Waters. Physically-based Facial Modeling, Analysis, and Animation. *Journal of Visualization and Computer Animation*, 1(4):73–80, March 1990.

[44] Kristinn R. Thórisson. Gandalf: An Embodied Humanoid Capable of Real-Time Multimodal Dialogue with People. In *First ACM International Conference on Autonomous Agents*. 1997.

[45] Michael W. Vannier, Jeffrey F. Marsh, and James O. Warren. Three-dimentional Computer Graphics for Craniofacial Surgical Planning and Evaluation. In *SIGGRAPH 83 Conference Proceedings*, volume 17, pages 263–273. ACM SIGGRAPH, August 1983.

[46] Keith Waters. A Muscle Model for Animating Three-Dimensional Facial Expression. In *SIGGRAPH 87 Conference Proceedings)*, volume 21, pages 17–24. ACM SIGGRAPH, July 1987.

[47] Lance Williams. Performance-Driven Facial Animation. In *SIGGRAPH 90 Conference Proceedings*, volume 24, pages 235–242. August 1990.

[48] Z. Zhang, K. Isono, and S. Akamatsu. Euclidean Structure from Uncalibrated Images Using Fuzzy Domain Knowledge: Application to Facial Images Synthesis. In *Proc. International Conference on Computer Vision (ICCV'98)*. January 1998.

A Least squares for pose recovery

To solve for a subset of the parameters given in Equation (2), we use linear least squares. In general, given a set of linear equations of the form

$$\boldsymbol{a}_j \cdot \boldsymbol{x} - b_j = 0, \tag{4}$$

we solve for the vector $\boldsymbol{x}$ by minimizing

$$\sum_j (\boldsymbol{a}_j \cdot \boldsymbol{x} - b_j)^2. \tag{5}$$

Setting the partial derivative of this sum with respect to $\boldsymbol{x}$ to zero, we obtain

$$\sum_j (\boldsymbol{a}_j \boldsymbol{a}_j^T)\boldsymbol{x} - b_j \boldsymbol{a}_j = 0, \tag{6}$$

i.e., we solve the set of *normal equations* [16]

$$\left(\sum_j \boldsymbol{a}_j \boldsymbol{a}_j^T \right) \boldsymbol{x} = \sum_j b_j \boldsymbol{a}_j. \tag{7}$$

More numerically stable methods such as *QR* decomposition or Singular Value Decomposition [16] can also be used to solve the least squares problem, but we have not found them to be necessary for our application.

To update one of the parameters, we simply pull out the relevant linear coefficient $\boldsymbol{a}_j$ and scalar value b_j from Equation (2). For example, to solve for $\boldsymbol{p}_i$, we set

$$\begin{aligned} \boldsymbol{a}_{2k+0} &= w_i^k(x_i^k \eta^k \boldsymbol{r}_z^k - s^k \boldsymbol{r}_x^k), \qquad b_{2k+0} = w_i^k(s^k t_x^k - x_i^k) \\ \boldsymbol{a}_{2k+1} &= w_i^k(y_i^k \eta^k \boldsymbol{r}_z^k - s^k \boldsymbol{r}_y^k), \qquad b_{2k+1} = w_i^k(s^k t_y^k - y_i^k). \end{aligned}$$

For a scalar variable like s^k, we obtain scalar equations

$$\begin{aligned} \boldsymbol{a}_{2k+0} &= w_i^k(\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k), \qquad b_{2k+0} = w_i^k \left(x_i^k + x_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) \right) \\ \boldsymbol{a}_{2k+1} &= w_i^k(\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k), \qquad b_{2k+1} = w_i^k \left(y_i^k + y_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) \right). \end{aligned}$$

Similar equations for $\boldsymbol{a}_j$ and b_j can be derived for the other parameters t_x^k, t_y^k, and η^k. Note that the parameters for a given camera k or 3D point i can be recovered independently of the other parameters.

Solving for rotation is a little trickier than for the other parameters, since $\boldsymbol{R}$ must be a valid rotation matrix. Instead of updating the elements in $\boldsymbol{R}_k$ directly, we replace the rotation matrix $\boldsymbol{R}^k$ with $\tilde{\boldsymbol{R}}\boldsymbol{R}^k$ [42], where $\tilde{\boldsymbol{R}}$ is given by Rodriguez's formula [15]:

$$\tilde{\boldsymbol{R}}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta \boldsymbol{X}(\hat{\boldsymbol{n}}) + (1 - \cos\theta)\boldsymbol{X}^2(\hat{\boldsymbol{n}}), \tag{8}$$

where θ is an incremental rotation angle, $\hat{\boldsymbol{n}}$ is a rotation axis, and $\boldsymbol{X}(\boldsymbol{v})$ is the cross product operator

$$\boldsymbol{X}(\boldsymbol{v}) = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}. \tag{9}$$

A first order expansion of $\tilde{\boldsymbol{R}}$ in terms of the entries in $\boldsymbol{v} = \theta\hat{\boldsymbol{n}} = (v_x, v_y, v_z)$ is given by $\boldsymbol{I} + \boldsymbol{X}(\boldsymbol{v})$.

Substituting into Equation (2) and letting $\boldsymbol{q}_i = \boldsymbol{R}^k \boldsymbol{p}_i$, we obtain

$$\begin{aligned} w_i^k \left(x_i^k + x_i^k \eta^k (\tilde{\boldsymbol{r}}_z^k \cdot \boldsymbol{q}_i) - s^k (\tilde{\boldsymbol{r}}_x^k \cdot \boldsymbol{q}_i + t_x^k) \right) &= 0 \\ w_i^k \left(y_i^k + y_i^k \eta^k (\tilde{\boldsymbol{r}}_z^k \cdot \boldsymbol{q}_i) - s^k (\tilde{\boldsymbol{r}}_y^k \cdot \boldsymbol{q}_i + t_y^k) \right) &= 0, \end{aligned} \tag{10}$$

where $\tilde{\boldsymbol{r}}_x^k = (1, -v_z, v_y)$, $\tilde{\boldsymbol{r}}_y^k = (v_z, 1, -v_x)$, $\tilde{\boldsymbol{r}}_z^k = (-v_y, v_x, 1)$, are the rows of $[\boldsymbol{I} + \boldsymbol{X}(\boldsymbol{v})]$. This expression is linear in (v_x, v_y, v_z), and hence leads to a 3×3 set of normal equations in (v_x, v_y, v_z). Once the elements of $\boldsymbol{v}$ have been estimated, we can compute θ and $\hat{\boldsymbol{n}}$, and update the rotation matrix using

$$\boldsymbol{R}^k \leftarrow \tilde{\boldsymbol{R}}(\hat{\boldsymbol{n}}^k, \theta^k)\boldsymbol{R}^k.$$

Making Faces

Brian Guenter[†] Cindy Grimm[†] Daniel Wood[‡]
Henrique Malvar[†] Fredrick Pighin[‡]
[†]Microsoft Corporation [‡]University of Washington

ABSTRACT

We have created a system for capturing both the three-dimensional geometry and color and shading information for human facial expressions. We use this data to reconstruct photorealistic, 3D animations of the captured expressions. The system uses a large set of sampling points on the face to accurately track the three dimensional deformations of the face. Simultaneously with the tracking of the geometric data, we capture multiple high resolution, registered video images of the face. These images are used to create a texture map sequence for a three dimensional polygonal face model which can then be rendered on standard 3D graphics hardware. The resulting facial animation is surprisingly life-like and looks very much like the original live performance. Separating the capture of the geometry from the texture images eliminates much of the variance in the image data due to motion, which increases compression ratios. Although the primary emphasis of our work is not compression we have investigated the use of a novel method to compress the geometric data based on principal components analysis. The texture sequence is compressed using an MPEG4 video codec. Animations reconstructed from 512x512 pixel textures look good at data rates as low as 240 Kbits per second.

CR Categories: I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism: Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1 Introduction

One of the most elusive goals in computer animation has been the realistic animation of the human face. Possessed of many degrees of freedom and capable of deforming in many ways the face has been difficult to simulate accurately enough to convince the average person that a piece of computer animation is actually an image of a real person.

We have created a system for capturing human facial expression and replaying it as a highly realistic 3D "talking head" consisting of a deformable 3D polygonal face model with a changing texture map. The process begins with video of a live actor's face, recorded from multiple camera positions simultaneously. Fluorescent colored 1/8" circular paper fiducials are glued on the actor's face and their 3D position reconstructed over time as the actor talks and emotes. The 3D fiducial positions are used to distort a 3D polygonal face model in mimicry of the distortions of the real face. The fiducials are removed using image processing techniques and the video streams from the multiple cameras are merged into a single texture map. When the resulting fiducial-free texture map is applied to the 3D reconstructed face mesh the result is a remarkably life-like 3D animation of facial expression. Both the time varying texture created from the video streams and the accurate reproduction of the 3D face structure contribute to the believability of the resulting animation.

Our system differs from much previous work in facial animation, such as that of Lee [10], Waters [14], and Cassel [3], in that we are not synthesizing animations using a physical or procedural model of the face. Instead, we capture facial movements in three dimensions and then replay them. The systems of [10], [14] are designed to make it relatively easy to animate facial expression manually. The system of [3] is designed to automatically create a dialog rather than faithfully reconstruct a particular person's facial expression. The work of Williams [15] is most similar to ours except that he used a single static texture image of a real person's face and tracked points only in 2D. The work of Bregler et al [2] is somewhat less related. They use speech recognition to locate visemes[1] in a video of a person talking and then synthesize new video, based on the original video sequence, for the mouth and jaw region of the face to correspond with synthetic utterances. They do not create a three dimensional face model nor do they vary the expression on the remainder of the face. Since we are only concerned with capturing and reconstructing facial performances out work is unlike that of [5] which attempts to recognize expressions or that of [4] which can track only a limited set of facial expressions.

An obvious application of this new method is the creation of believable virtual characters for movies and television. Another application is the construction of a flexible type of video compression. Facial expression can be captured in a studio, delivered via CDROM or the internet to a user, and then reconstructed in real time on a user's computer in a virtual 3D environment. The user can select any arbitrary position for the face, any virtual camera viewpoint, and render the result at any size.

One might think the second application would be difficult to achieve because of the huge amount of video data required for the time varying texture map. However, since our system generates accurate 3D deformation information, the texture image data is precisely registered from frame to frame. This reduces most of the variation in image intensity due to geometric motion, leaving primarily shading and self shadowing effects. These effects tend to be of low spatial frequency and can be compressed very efficiently. The compressed animation looks good at data rates of 240 kbits per second for texture image sizes of 512x512 pixels, updating at 30 frames per second.

The main contributions of the paper are a method for robustly capturing both a 3D deformation model and a registered texture image sequence from video data. The resulting geometric and texture data can be compressed, with little loss of fidelity, so that storage

[1] Visemes are the visual analog of phonemes.

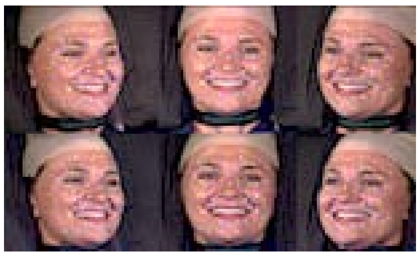


Figure 1: The six camera views of our actress' face.

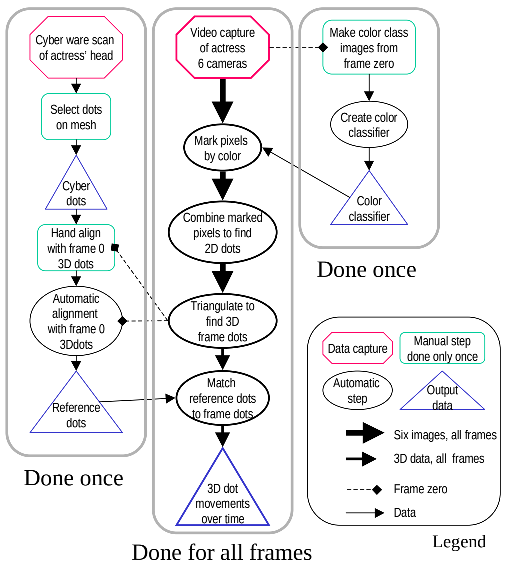

Cyber ware scan of actress' head
Select dots on mesh
Cyber dots
Hand align with frame 0 3D dots
Automatic alignment with frame 0 3Ddots
Reference dots
Done once
Video capture of actress 6 cameras
Mark pixels by color
Combine marked pixels to find 2D dots
Triangulate to find 3D frame dots
Match reference dots to frame dots
3D dot movements over time
Done for all frames
Make color class images from frame zero
Create color classifier
Color classifier
Done once
Data capture
Manual step done only once
Automatic step
Output data
Six images, all frames
3D data, all frames
Frame zero
Data
Legend

Figure 2: The sequence of operations needed to produce the labeled 3D dot movements over time.

requirements are reasonable for many applications.

Section 2 of the paper explains the data capture stage of the process. Section 3 describes the fiducial correspondence algorithm. In Section 4 we discuss capturing and moving the mesh. Sections 5 and 6 describe the process for making the texture maps. Section 7 of the paper describes the algorithm for compressing the geometric data.

2 Data Capture

We used six studio quality video cameras arranged in the pattern shown in Plate 1 to capture the video data. The cameras were synchronized and the data saved digitally. Each of the six cameras was individually calibrated to determine its intrinsic and extrinsic parameters and to correct for lens distortion. The details of the calibration process are not germane to this paper but the interested reader can find a good overview of the topic in [6] as well as an extensive bibliography.

We glued 182 dots of six different colors onto the actress' face. The dots were arranged so that dots of the same color were as far apart as possible from each other and followed the contours of the face. This made the task of determining frame to frame dot correspondence (described in Section 3.3) much easier. The dot pattern was chosen to follow the contours of the face (i.e., outlining the eyes, lips, and nasio-labial furrows), although the manual application of the dots made it difficult to follow the pattern exactly.

The actress' head was kept relatively immobile using a padded foam box; this reduced rigid body motions and ensured that the actress' face stayed centered in the video images. Note that rigid body motions can be captured later using a 3D motion tracker, if desired.

The actress was illuminated with a combination of visible and near UV light. Because the dots were painted with fluorescent pigments the UV illumination increased the brightness of the dots significantly and moved them further away in color space from the colors of the face than they would ordinarily be. This made them easier to track reliably. Before the video shoot the actress' face was digitized using a cyberware scanner. This scan was used to create the base 3D face mesh which was then distorted using the positions of the tracked dots.

3 Dot Labeling

The fiducials are used to generate a set of 3D points which act as control points to warp the cyberware scan mesh of the actress' head. They are also used to establish a stable mapping for the textures generated from each of the six camera views. This requires that each dot have a unique and consistent label over time so that it is associated with a consistent set of mesh vertices.

The dot labeling begins by first locating (for each camera view) connected components of pixels which correspond to the fiducials. The 2D location for each dot is computed by finding the two dimensional centroid of each connected component. Correspondence between 2D dots in different camera views is established and potential 3D locations of dots reconstructed by triangulation. We construct a reference set of dots and pair up this reference set with the 3D locations in each frame. This gives a unique labeling for the dots that is maintained throughout the video sequence.

A flowchart of the dot labeling process is shown in Figure 2. The left side of the flowchart is described in Section 3.3.1, the middle in Sections 3.1, 3.2, and 3.3.2, and the right side in Section 3.1.1.

3.1 Two-dimensional dot location

For each camera view the 2D coordinates of the centroid of each colored fiducial must be computed. There are three steps to this process: color classification, connected color component generation, and centroid computation.

First, each pixel is classified as belonging to one of the six dot colors or to the background. Then depth first search is used to locate connected blobs of similarly colored pixels. Each connected colored blob is grown by one pixel to create a mask used to mark those pixels to be included in the centroid computation. This process is illustrated in Figure 4.

The classifier requires the manual marking of the fiducials for one frame for each of the six cameras. From this data a robust color classifier is created (exact details are discussed in Section 3.1.1). Although the training set was created using a single frame of a 3330 frame sequence, the fiducial colors are reliably labeled throughout the sequence. False positives are quite rare, with one major exception, and are almost always isolated pixels or two pixel clusters. The majority of exceptions arise because the highlights on the teeth and mouth match the color of the white fiducial training set. Fortunately, the incorrect white fiducial labelings occur at consistent 3D locations and are easily eliminated in the 3D dot processing stage.

The classifier generalizes well so that even fairly dramatic changes in fiducial color over time do not result in incorrect classification. For example, Figure 5(b) shows the same green fiducial in two different frames. This fiducial is correctly classified as green in both frames.

The next step, finding connected color components, is complicated by the fact that the video is interlaced. There is significant field to field movement, especially around the lips and jaw, sometimes great enough so that there is no spatial overlap at all between the pixels of a fiducial in one field and the pixels of the same fiducial in the next field. If the two fields are treated as a single frame then a single fiducial can be fragmented, sometimes into many pieces.

One could just find connected color components in each field and use these to compute the 2D dot locations. Unfortunately, this does not work well because the fiducials often deform and are sometimes partially occluded. Therefore, the threshold for the number of pixels needed to classify a group of pixels as a fiducial has to be set very low. In our implementation any connected component which has more than three pixels is classified as a fiducial rather than noise. If just the connected pixels in a single field are counted then the threshold would have to be reduced to one pixel. This would cause many false fiducial classifications because there are typically a few 1 pixel false color classifications per frame and 2 or 3 pixel false clusters occur occasionally. Instead, we find connected components and generate lists of potential 2D dots in each field. Each potential 2D dot in field one is then paired with the closest 2D potential dot in field two. Because fiducials of the same color are spaced far apart, and because the field to field movement is not very large, the closest potential 2D dot is virtually guaranteed to be the correct match. If the sum of the pixels in the two potential 2D dots is greater than three pixels then the connected components of the two 2D potential dots are merged, and the resulting connected component is marked as a 2D dot.

The next step is to find the centroid of the connected components marked as 2D dots in the previous step. A two dimensional gradient magnitude image is computed by passing a one dimensional first derivative of Gaussian along the x and y directions and then taking the magnitude of these two values at each pixel. The centroid of the colored blob is computed by taking a weighted sum of positions of the pixel (x, y) coordinates which lie inside the gradient mask, where the weights are equal to the gradient magnitude.

3.1.1 Training the color classifier

We create one color classifier for each of the camera views, since the lighting can vary greatly between cameras. In the following discussion we build the classifier for a single camera.

The data for the color classifier is created by manually marking the pixels of frame zero that belong to a particular fiducial color. This is repeated for each of the six colors. The marked data is stored as 6 *color class images*, each of which is created from the original camera image by setting all of the pixels *not* marked as the given color to black (we use black as an out-of-class label because pure black never occurred in any of our images). A typical color class image for the yellow dots is shown in Figure 3. We generated the color class images using the “magic wand” tool available in many image editing programs.

A seventh color class image is automatically created for the background color (e.g., skin and hair) by labeling as out-of-class any pixel in the image which was previously marked as a fiducial in any of the fiducial color class images. This produces an image of the face with black holes where the fiducials were.

The color classifier is a discrete approximation to a nearest neighbor classifier [12]. In a nearest neighbor classifier the item to be classified is given the label of the closest item in the training set, which in our case is the color data contained in the color class images. Because we have 3 dimensional data we can approximate the nearest neighbor classifier by subdividing the RGB cube uniformly into voxels, and assigning class labels to each RGB voxel. To classify a new color you quantize its RGB values and then index into the cube to extract the label.

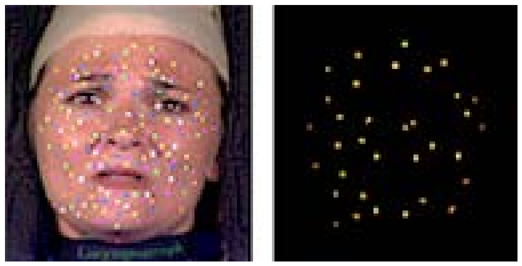


Figure 3: An image of the actress’s face. A typical training set for the yellow dots, selected from the image on the left.

To create the color classifier we use the color class images to assign color classes to each voxel. Assume that the color class image for color class C_i has n distinct colors, $c_1 ... c_n$. Each of the voxels corresponding to the color c_j is labeled with the color class C_i. Once the voxels for all of the known colors are labeled, the remaining unlabeled voxels are assigned labels by searching through all of the colors in each color class C_i and finding the color closest to p in RGB space. The color p is given the label of the color class containing the nearest color. Nearness in our case is the Euclidean distance between the two points in RGB space.

If colors from different color classes map to the same sub-cube, we label that sub-cube with the background label since it is more important to avoid incorrect dot labeling than it is to try to label every dot pixel. For the results shown in this paper we quantized the RGB color cube into a 32x32x32 lattice.

3.2 Camera to camera dot correspondence and 3D reconstruction

In order to capture good images of both the front and the sides of the face the cameras were spaced far apart. Because there are such extreme changes in perspective between the different camera views, the projected images of the colored fiducials are very different. Figure 5 shows some examples of the changes in fiducial shape and color between camera views. Establishing fiducial correspondence between camera views by using image matching techniques such as optical flow or template matching would be difficult and likely to generate incorrect matches. In addition, most of the camera views will only see a fraction of the fiducials so the correspondence has to be robust enough to cope with occlusion of fiducials in some of the camera views. With the large number of fiducials we have placed on the face false matches are also quite likely and these must be detected and removed. We used ray tracing in combination with a RANSAC [7] like algorithm to establish fiducial correspondence and to compute accurate 3D dot positions. This algorithm is robust to occlusion and to false matches as well.

First, all potential point correspondences between cameras are generated. If there are k cameras, and n 2D dots in each camera view then $\begin{pmatrix} k \\ 2 \end{pmatrix} n^2$ point correspondences will be tested. Each correspondence gives rise to a 3D candidate point defined as the closest point of intersection of rays cast from the 2D dots in the

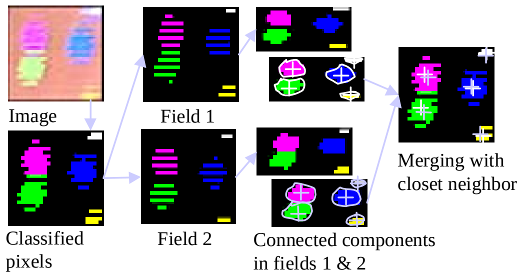


Image
Field 1
Classified pixels
Field 2
Connected components in fields 1 & 2
Merging with closet neighbor

Figure 4: Finding the 2D dots in the images.

two camera views. The 3D candidate point is projected into each of the two camera views used to generate it. If the projection is further than a user-defined epsilon, in our case two pixels, from the centroid of either 2D point then the point is discarded as a potential 3D point candidate. All the 3D candidate points which remain are added to the 3D point list.

Each of the points in the 3D point list is projected into a reference camera view which is the camera with the best view of all the fiducials on the face. If the projected point lies within two pixels of the centroid of a 2D dot visible in the reference camera view then it is added to the list of potential 3D candidate positions for that 2D dot. This is the list of potential 3D matches for a given 2D dot.

For each 3D point in the potential 3D match list, $\binom{n}{3}$ possible combinations of three points in the 3D point list are computed and the combination with the smallest variance is chosen as the true 3D position. Then all 3D points which lie within a user defined distance, in our case the sphere subtended by a cone two pixels in radius at the distance of the 3D point, are averaged to generate the final 3D dot position. This 3D dot position is assigned to the corresponding 2D dot in the reference camera view.

This algorithm could clearly be made more efficient because many more 3D candidate points are generated then necessary. One could search for potential camera to camera correspondences only along the epipolar lines and use a variety of space subdivision techniques to find 3D candidate points to test for a given 2D point. However, because the number of fiducials in each color set is small (never more than 40) both steps of this simple and robust algorithm are reasonably fast, taking less than a second to generate the 2D dot correspondences and 3D dot positions for six camera views. The 2D dot correspondence calculation is dominated by the time taken to read in the images of the six camera views and to locate the 2D dots in each view. Consequently, the extra complexity of more efficient stereo matching algorithms does not appear to be justified.

3.3 Frame to frame dot correspondence and labeling

We now have a set of unlabeled 3D dot locations for each frame. We need to assign, across the entire sequence, consistent labels to the 3D dot locations. We do this by defining a reference set of dots D and matching this set to the 3D dot locations given for each frame. We can then describe how the reference dots move over time as follows: Let $d_j \in D$ be the neutral location for the reference dot j. We define the position of d_j at frame i by an offset, i.e.,

$$d_j^i = d_j + \vec{v}_j^i \tag{1}$$

Because there are thousands of frames and 182 dots in our data

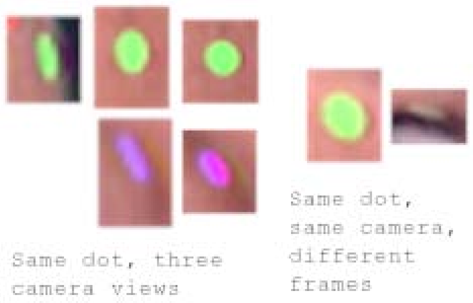

Same dot, three camera views
Same dot, same camera, different frames

Figure 5: Dot variation. Left: Two dots seen from three different cameras (the purple dot is occluded in one camera's view). Right: A single dot seen from a single camera but in two different frames.

set we would like the correspondence computation to be automatic and quite efficient. To simplify the matching we used a fiducial pattern that separates fiducials of a given color as much as possible so that only a small subset of the unlabeled 3D dots need be checked for a best match. Unfortunately, simple nearest neighbor matching fails for several reasons: some fiducials occasionally disappear, some 3D dots may move more than the average distance between 3D dots of the same color, and occasionally extraneous 3D dots appear, caused by highlights in the eyes or teeth. Fortunately, neighboring fiducials move similarly and we can exploit this fact, modifying the nearest neighbor matching algorithm so that it is still efficient but also robust.

For each frame i we first move the reference dots to the locations found in the previous frame. Next, we find a (possibly incomplete) match between the reference dots and the 3D dot locations for frame i. We then move each matched reference dot to the location of its corresponding 3D dot. If a reference dot does not have a match we "guess" a new location for it by moving it in the same direction as its neighbors. We then perform a final matching step.

3.3.1 Acquiring the reference set of dots

The cyberware scan was taken with the dots glued onto the face. Since the dots are visible in both the geometric and color information of the scan, we can place the reference dots on the cyberware model by manually clicking on the model. We next need to align the reference dots and the model with the 3D dot locations found in frame zero. The coordinate system for the cyberware scan differs from the one used for the 3D dot locations, but only by a rigid body motion plus a uniform scale. We find this transform as follows: we first hand-align the 3D dots from frame zero with the reference dots acquired from the scan, then call the matching routine described in Section 3.3.2 below to find the correspondence between the 3D dot locations, f_i, and the reference dots, d_i. We use the method described in [9] to find the exact transform, T, between the two sets of dots. Finally, we replace the temporary locations of the reference dots with $d_i = f_i$.
and use T^{-1} to transform the cyberware model into the coordinate system of the video 3D dot locations.

3.3.2 The matching routine

The matching routine is run twice per frame. We first perform a conservative match, move the reference dots (as described below in Section 3.3.3), then perform a second, less conservative, match. By moving the reference dots between matches we reduce the problem of large 3D dot position displacements.

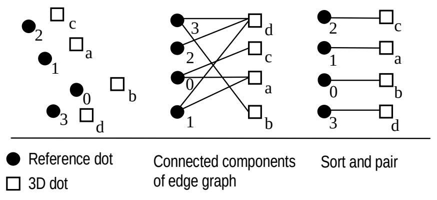


Figure 6: Matching dots.

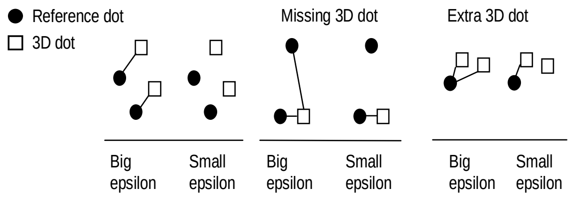


Figure 7: Examples of extra and missing dots and the effect of different values for ϵ.

The matching routine can be thought of as a graph problem where an edge between a reference dot and a frame dot indicates that the dots are potentially paired (see Figure 6). The matching routine proceeds in several steps; first, for each reference dot we add an edge for every 3D dot of the same color that is within a given distance ϵ. We then search for connected components in the graph that have an equal number of 3D and reference dots (most connected components will have exactly two dots, one of each type). We sort the dots in the vertical dimension of the plane of the face and use the resulting ordering to pair up the reference dots with the 3D dot locations (see Figure 6).

In the video sequences we captured, the difference in the 3D dot positions from frame to frame varied from zero to about 1.5 times the average distance separating closest dots. To adjust for this, we run the matching routine with several values of ϵ and pick the run that generates the most matches. Different choices of ϵ produce different results (see Figure 7): if ϵ is too small we may not find matches for 3D dots that have moved a lot. If ϵ is too large then the connected components in the graph will expand to include too many 3D dots. We try approximately five distances ranging from 0.5 to 1.5 of the average distance between closest reference dots.

If we are doing the second match for the frame we add an additional step to locate matches where a dot may be missing (or extra). We take those dots which have not been matched and run the matching routine on them with smaller and smaller ϵ values. This resolves situations such as the one shown on the right of Figure 7.

3.3.3 Moving the dots

We move all of the matched reference dots to their new locations then interpolate the locations for the remaining, unmatched reference dots by using their nearest, matched neighbors. For each reference dot we define a valid set of neighbors using the routine in Section 4.2.1, ignoring the blending values returned by the routine.

To move an unmatched dot d_k we use a combination of the offsets of all of its valid neighbors (refer to Equation 1). Let $n_k \subset D$ be the set of neighbor dots for dot d_k. Let $\hat{n}_k$ be the set of neighbors that have a match for the current frame i. Provided $\hat{n}_k \neq \emptyset$, the offset vector for dot d_k^i is calculated as follows: let $\vec{v}_j^i = d_j^i - d_j$ be the offset of dot j (recall that d_j is the initial position for the reference dot j).

$$\vec{v}_k^i = \frac{1}{||\hat{n}_k||} \sum_{d_j^i \in \hat{n}_k} \vec{v}_j^i$$

If the dot has no matched neighbors we repeat as necessary, treating the moved, unmatched reference dots as matched dots. Eventually, the movements will propagate through all of the reference dots.

4 Mesh construction and deformation

4.1 Constructing the mesh

To construct a mesh we begin with a cyberware scan of the head. Because we later need to align the scan with the 3D video dot data, we scanned the head with the fiducials glued on. The resulting scan suffers from four problems:

- The fluorescent fiducials caused “bumps” on the mesh.
- Several parts of the mesh were not adequately scanned, namely, the ears, one side of the nose, the eyes, and under the chin. These were manually corrected.
- The mesh does not have an opening for the mouth.
- The scan has too many polygons.

The bumps caused by the fluorescent fiducials were removed by selecting the vertices which were out of place (approximately 10-30 surrounding each dot) and automatically finding new locations for them by blending between four correct neighbors. Since the scan produces a rectangular grid of vertices we can pick the neighbors to blend between in (u, v) space, i.e., the nearest valid neighbors in the positive and negative u and v direction.

The polygons at the mouth were split and then filled with six rows of polygons located slightly behind the lips. We map the teeth and tongue onto these polygons when the mouth is open.

We reduced the number of polygons in the mesh from approximately $460,000$ to 4800 using Hoppe’s simplification method [8].

4.2 Moving the mesh

The vertices are moved by a linear combination of the offsets of the nearest dots (refer to Equation 1). The linear combination for each vertex v_j is expressed as a set of blend coefficients, α_k^j, one for each dot, such that $\sum_{d_k \in D} \alpha_k^j = 1$ (most of the α_k^js will be zero). The new location p_j^i of the vertex v_j at frame i is then

$$p_j^i = p_j + \sum_k \alpha_k^j ||d_k^i - d_k||$$

where p_j is the initial location of the vertex v_j.

For most of the vertices the α_k^js are a weighted average of the closest dots. The vertices in the eyes, mouth, behind the mouth, and outside of the facial area are treated slightly differently since, for example, we do not want the dots on the lower lip influencing vertices on the upper part of the lip. Also, although we tried to keep the head as still as possible, there is still some residual rigid body motion. We need to compensate for this for those vertices that are not directly influenced by a dot (e.g., the back of the head).

We use a two-step process to assign the blend coefficients to the vertices. We first find blend coefficients for a grid of points evenly distributed across the face, then use this grid of points to

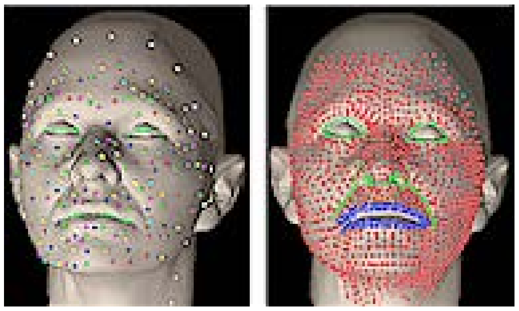


Figure 8: Left: The original dots plus the extra dots (in white). The labeling curves are shown in light green. Right: The grid of dots. Outline dots are green or blue.

assign blend coefficients to the vertices. This two-step process is helpful because both the fluorescent fiducials and the mesh vertices are unevenly distributed across the face, making it difficult to get smoothly changing blend coefficients.

The grid consists of roughly 1400 points, evenly distributed and placed by hand to follow the contours of the face (see Figure 8). The points along the nasolabial furrows, nostrils, eyes, and lips are treated slightly differently than the other points to avoid blending across features such as the lips.

Because we want the mesh movement to go to zero outside of the face, we add another set of unmoving dots to the reference set. These new dots form a ring around the face (see Figure 8) enclosing all of the reference dots. For each frame we determine the rigid body motion of the head (if any) using a subset of those reference dots which are relatively stable. This rigid body transformation is then applied to the new dots.

We label the dots, grid points, and vertices as being *above*, *below*, or *neither* with respect to each of the eyes and the mouth. Dots which are *above* a given feature can not be combined with dots which are *below* that same feature (or vice-versa). Labeling is accomplished using three curves, one for each of the eyes and one for the mouth. Dots directly above (or below) a curve are labeled as *above* (or *below*) that curve. Otherwise, they are labeled *neither*.

4.2.1 Assigning blends to the grid points

The algorithm for assigning blends to the grid points first finds the closest dots, assigns blends, then filters to more evenly distribute the blends.

Finding the ideal set of reference dots to influence a grid point is complicated because the reference dots are not evenly distributed across the face. The algorithm attempts to find two or more dots distributed in a rough circle around the given grid point. To do this we both compensate for the dot density, by setting the search distance using the two closest dots, and by checking for dots which will both “pull” in the same direction.

To find the closest dots to the grid point p we first find δ_1 and δ_2, the distance to the closest and second closest dot, respectively. Let $D_n \subset D$ be the set of dots within $1.8\frac{\delta_1+\delta_2}{2}$ distance of p whose labels do not conflict with p’s label. Next, we check for pairs of dots that are more or less in the same direction from p and remove the furthest one. More precisely, let $\hat{v}_i$ be the normalized vector from p to the dot $d_i \in D_n$ and let $\hat{v}_j$ be the normalized vector from p to the dot $d_j \in D_n$. If $\hat{v}_1 \cdot \hat{v}_2 > 0.8$ then remove the furthest of d_i and d_j from the set D_n.

We assign blend values based on the distance of the dots from p. If the dot is not in D_n then its corresponding α value is 0. For the dots in D_n let $l_i = \frac{1.0}{||d_i - p||}$. Then the corresponding α’s are

$$\alpha_i = \frac{l_i}{(\sum_{d_i \in D_n} l_i)}$$

We next filter the blend coefficients for the grid points. For each grid point we find the closest grid points – since the grid points are distributed in a rough grid there will usually be 4 neighboring points – using the above routine (replacing the dots with the grid points). We special case the outlining grid points; they are only blended with other outlining grid points. The new blend coefficients are found by taking 0.75 of the grid point’s blend coefficients and 0.25 of the average of the neighboring grid point’s coefficients. More formally, let $g_i = [\alpha_0, \ldots, \alpha_n]$ be the vector of blend coefficients for the grid point i. Then the new vector g_i' is found as follows, where N_i is the set of neighboring grid points for the grid point i:

$$g_i' = 0.75 g_i + \frac{0.25}{||N_i||} \sum_{j \in N_i} g_j$$

We apply this filter twice to simulate a wide low pass filter.

To find the blend coefficients for the vertices of the mesh we find the closest grid point with the same label as the vertex and copy the blend coefficients. The only exception to this is the vertices for the polygons inside of the mouth. For these vertices we take β of the closest grid point on the top lip and $1.0 - \beta$ of the closest grid point on the bottom lip. The β values are 0.8, 0.6, 0.4, 0.25, and 0.1 from top to bottom of the mouth polygons.

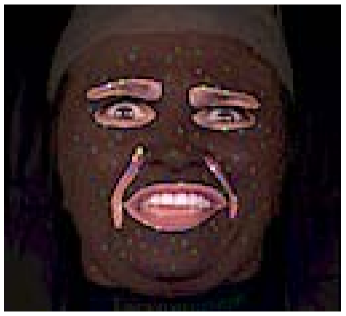


Figure 9: Masks surrounding important facial features. The gradient of a blurred version of this mask is used to orient the low-pass filters used in the dot removal process.

5 Dot removal

Before we create the textures, the dots and their associated illumination effects have to be removed from the camera images. Interreflection effects are surprisingly noticeable because some parts of the face fold dramatically, bringing the reflective surface of some dots into close proximity with the skin. This is a big problem along the naso-labial furrow where diffuse interreflection from the colored dots onto the face significantly alters the skin color.

First, the dot colors are removed from each of the six camera image sequences by substituting skin texture for pixels which are covered by colored dots. Next, diffuse interreflection effects and any remaining color casts from stray pixels that have not been properly substituted are removed.

The skin texture substitution begins by finding the pixels which correspond to colored dots. The nearest neighbor color classifier

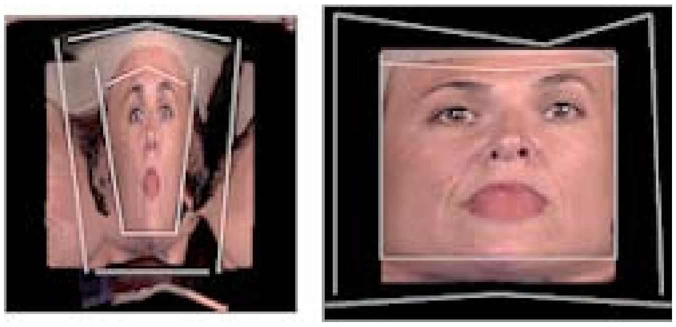


Figure 10: Standard cylindrical texture map. Warped texture map that focuses on the face, and particularly on the eyes and mouth. The warp is defined by the line pairs shown in white.

described in Section 3.1.1 is used to mark all pixels which have any of the dot colors. A special training set is used since in this case false positives are much less detrimental than they are for the dot tracking case. Also, there is no need to distinguish between dot colors, only between dot colors and the background colors. The training set is created to capture as much of the dot color and the boundary region between dots and the background colors as possible.

A dot mask is generated by applying the classifier to each pixel in the image. The mask is grown by a few pixels to account for any remaining pixels which might be contaminated by the dot color. The dot mask marks all pixels which must have skin texture substituted.

The skin texture is broken into low spatial frequency and high frequency components. The low frequency components of the skin texture are interpolated by using a directional low pass filter oriented parallel to features that might introduce intensity discontinuities. This prevents bleeding of colors across sharp intensity boundaries such as the boundary between the lips and the lighter colored regions around the mouth. The directionality of the filter is controlled by a two dimensional mask which is the projection into the image plane of a three dimensional polygon mask lying on the 3D face model. Because the polygon mask is fixed on the 3D mesh, the 2D projection of the polygon mask stays in registration with the texture map as the face deforms.

All of the important intensity gradients have their own polygon mask: the eyes, the eyebrows, the lips, and the naso-labial furrows (see 9). The 2D polygon masks are filled with white and the region of the image outside the masks is filled with black to create an image. This image is low-pass filtered. The intensity of the resulting image is used to control how directional the filter is. The filter is circularly symmetric where the image is black, i.e., far from intensity discontinuities, and it is very directional where the image is white. The directional filter is oriented so that its long axis is orthogonal to the gradient of this image.

The high frequency skin texture is created from a rectangular sample of skin texture taken from a part of the face that is free of dots. The skin sample is highpass filtered to eliminate low frequency components. At each dot mask pixel location the highpass filtered skin texture is first registered to the center of the 2D bounding box of the connected dot region and then added to the low frequency interpolated skin texture.

The remaining diffuse interreflection effects are removed by clamping the hue of the skin color to a narrow range determined from the actual skin colors. First the pixel values are converted from RGB to HSV space and then any hue outside the legal range is clamped to the extremes of the range. Pixels in the eyes and mouth, found using the eye and lip masks shown in Figure 9, are left unchanged.

Some temporal variation remains in the substituted skin texture due to imperfect registration of the high frequency texture from frame to frame. A low pass temporal filter is applied to the dot mask regions in the texture images, because in the texture map space the dots are relatively motionless. This temporal filter effectively eliminates the temporal texture substitution artifacts.

6 Creating the texture maps

Figure 11 is a flowchart of the texture creation process. We create texture maps for every frame of our animation in a four-step process. The first two steps are performed only once per mesh. First we define a parameterization of the mesh. Second, using this parameterization, we create a *geometry map* containing a location on the mesh for each texel. Third, for every frame, we create six preliminary texture maps, one from each camera image, along with weight maps. The weight maps indicate the relative quality of the data from the different cameras. Fourth, we take a weighted average of these texture maps to make our final texture map.

We create an initial set of texture coordinates for the head by tilting the mesh back 10 degrees to expose the nostrils and projecting the mesh vertices onto a cylinder. A texture map generated using this parametrization is shown on the left of Figure 10. We specify a set of line pairs and warp the texture coordinates using the technique described by Beier and Neely[1]. This parametrization results in the texture map shown on the right of Figure 10. Only the front of the head is textured with data from the six video streams.

Next we create the geometry map containing a mesh location for each texel. A mesh location is a triple (k, β_1, β_2) specifying a triangle k and barycentric coordinates in the triangle (β_1, β_2, $1 - \beta_1 - \beta_2$). To find the triangle identifier k for texel (u, v) we exhaustively search through the mesh's triangles to find the one that contains the texture coordinates (u, v). We then set the β_is to be the barycentric coordinates of the point (u, v) in the texture coordinates of the triangle k. When finding the mesh location for a pixel we already know in which triangles its neighbors above and to the left lie. Therefore, we speed our search by first searching through these triangles and their neighbors. However, the time required for this task is not critical as the geometry map need only be created once.

Next we create preliminary texture maps for frame f one for each camera. This is a modified version of the technique described in [11]. To create the texture map for camera c, we begin by deforming the mesh into its frame f position. Then, for each texel, we get its mesh location, (k, β_1, β_2), from the geometry map. With the 3D coordinates of triangle k's vertices and the barycentric coordinates β_i, we compute the texel's 3D location t. We transform t by camera c's projection matrix to obtain a location, (x, y), on camera c's image plane. We then color the texel with the color from camera c's image at (x, y). We set the texel's weight to the dot product of the mesh normal at t, $\hat{n}$, with the direction back to the camera, $\hat{d}$ (see Figure 12). Negative values are clamped to zero. Hence, weights are low where the camera's view is glancing. However, this weight map is not smooth at triangle boundaries, so we smooth it by convolving it with a Gaussian kernel.

Last, we merge the six preliminary texture maps. As they do not align perfectly, averaging them blurs the texture and loses detail. Therefore, we use only the texture map of our bottom, center camera for the center 46 % of the final texture map. We smoothly transition (over 23 pixels) to using a weighted average of each preliminary texture map at the sides.

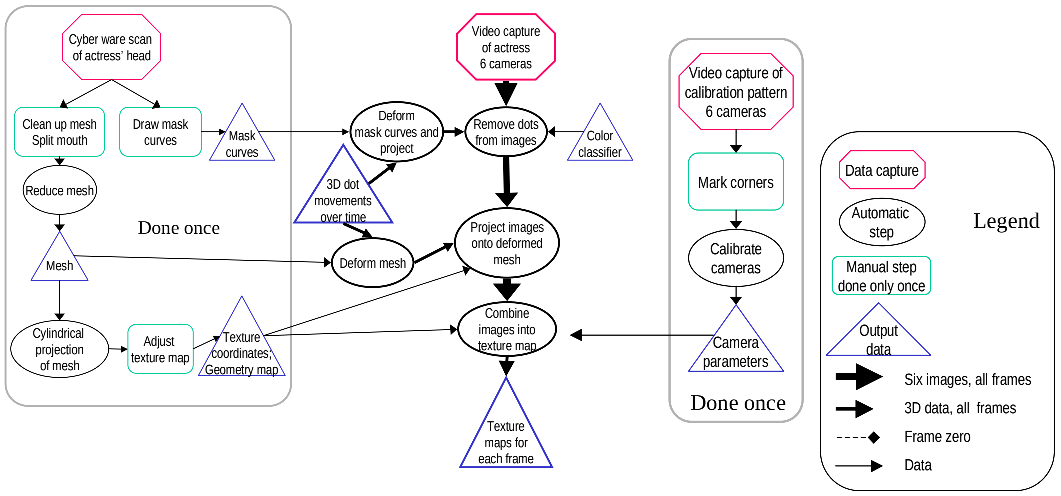

Cyber ware scan of actress' head
Clean up mesh Split mouth
Draw mask curves
Mask curves
Reduce mesh
Done once
Mesh
Cylindrical projection of mesh
Adjust texture map
Texture coordinates, Geometry map
Video capture of actress 6 cameras
Deform mask curves and project
Remove dots from images
Color classifier
3D dot movements over time
Deform mesh
Project images onto deformed mesh
Combine images into texture map
Texture maps for each frame
Video capture of calibration pattern 6 cameras
Mark corners
Calibrate cameras
Camera parameters
Done once
Data capture
Automatic step
Legend
Manual step done only once
Output data
Six images, all frames
3D data, all frames
Frame zero
Data

Figure 11: Creating the texture maps.

We texture the parts of the head not covered by the aforementioned texture maps with the captured reflectance data from our Cyberware scan, modified in two ways. First, because we replaced the mesh's ears with ears from a stock mesh (Section 4.1), we moved the ears in the texture to achieve better registration. Second, we set the alpha channel to zero (with a soft edge) in the region of the texture for the front of the head. Then we render in two passes to create an image of the head with both texture maps applied.

7 Compression

7.1 Principal Components Analysis

The geometric and texture map data have different statistical characteristics and are best compressed in different ways. There is significant long-term temporal correlation in the geometric data since similar facial expressions occur throughout the sequence. The short term correlation of the texture data is significantly increased over that of the raw video footage because in the texture image space the fiducials are essentially motionless. This eliminates most of the intensity changes associated with movement and leaves primarily shading changes. Shading changes tend to have low spatial frequencies and are highly compressible. Compression schemes such as MPEG, which can take advantage of short term temporal correlation, can exploit this increase in short term correlation.

For the geometric data, one way to exploit the long term correlation is to use principal component analysis. If we represent our data set as a matrix A, where frame i of the data maps column i of A, then the first principal component of A is

$$\max_u (A^T u)^T (A^T u) \qquad (2)$$

The u which maximizes Equation 2 is the eigenvector associated with the largest eigenvalue of AA^T, which is also the value of the maximum. Succeeding principal components are defined similarly, except that they are required to be orthogonal to all preceding principal components, i.e., $u_i^T u_j = 0$ for $j \neq i$. The principal components form an orthonormal basis set represented by the matrix U where the columns of U are the principal components of A ordered by eigenvalue size with the most significant principal component in the first column of U.

The data in the A matrix can be projected onto the principal component basis as follows:

$$W = U^T A$$

Row i of W is the projection of column A_i onto the basis vector u_i. More precisely, the jth element in row i of W corresponds to the projection of frame j of the original data onto the ith basis vector. We will call the elements of the W matrix projection *coefficients*.

Similarly, A can be reconstructed exactly from W by multiplication by the basis set, i.e., $A = UW$.
The most important property of the principal components for our purposes is that they are the best linear basis set for reconstruction in the l_2 norm sense. For any given matrix U_k, where k is the number of columns of the matrix and $k < rank(A)$, the reconstruction error

$$e = ||A - U_k U_k^T A||_F^2 \qquad (3)$$

where $||B||_F^2$ is the Frobenius norm defined to be

$$||B||_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij}^2 \qquad (4)$$

will be minimized if U_k is the matrix containing the k most significant principal components of A.

We can compress a data set A by quantizing the elements of its corresponding W and U matrices and entropy coding them. Since the compressed data cannot be reconstructed without the principal component basis vectors both the W and U matrices have to be compressed. The basis vectors add overhead that is not present with basis sets that can be computed independent of the original data set, such as the DCT basis.

For data sequences that have no particular structure the extra overhead of the basis vectors would probably out-weigh any gain in compression efficiency. However, for data sets with regular frame to frame structure the residual error for reconstruction with the principal component basis vectors can be much smaller than for other bases. This reduction in residual error can be great enough to compensate for the overhead bits of the basis vectors.

The principal components can be computed using the singular value decomposition (SVD) [13]. Efficient implementations of this algorithm are widely available. The SVD of a matrix A is

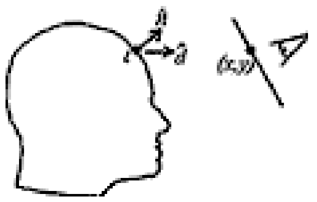


Figure 12: Creating the preliminary texture map.

$$A = U\Sigma V^T \tag{5}$$

where the columns of U are the eigenvectors of AA^T, the singular values, σ_i, along the diagonal matrix Σ are the square roots of the eigenvalues of AA^T, and the columns of V are the eigenvectors of A^TA. The ith column of U is the ith principal component of A. Computing the first k left singular vectors of A is equivalent to computing the first k principal components.

7.2 Geometric Data

The geometric data has the long term temporal coherence properties mentioned above since the motion of the face is highly structured. The overhead of the basis vectors for the geometric data is fixed because there are only 182 fiducials on the face. The maximum number of basis vectors is $182 * 3$ since there are three numbers, x, y, and z, associated with each fiducial. Consequently, the basis vector overhead steadily diminishes as the length of the animation sequence increases.

The geometric data is mapped to matrix form by taking the 3D offset data for the ith frame and mapping it the ith column of the data matrix A_g. The first k principal components, U_g, of A_g are computed and A_g is projected into the U_g basis to give the projection coefficients W_g.

There is significant correlation between the columns of projection coefficients because the motion of the dots is relatively smooth over time. We can reduce the entropy of the quantized projection coefficients by temporally predicting the projection coefficients in column i from column $i-1$, i.e., $c_i = c_{i-1} + \Delta_i$ where we encode Δ_i.

For our data set, only the projection coefficients associated with the first 45 principal components, corresponding to the first 45 rows of W_g, have significant temporal correlation so only the first 45 rows are temporally predicted. The remaining rows are entropy coded directly. After the temporal prediction the entropy is reduced by about 20 percent (Figure 13).

The basis vectors are compressed by choosing a peak error rate and then varying the number of quantization levels allocated to each vector based on the standard deviation of the projection coefficients for each vector.

We visually examined animation sequences with W_g and U_g compressed at a variety of peak error rates and chose a level which resulted in undetectable geometric jitter in reconstructed animation. The entropy of W_g for this error level is 26 Kbits per second and the entropy of U_g is 13 kbits per second for a total of 40 kbits per second for all the geometric data. These values were computed for our 3330 frame animation sequence.

8 Results

Figure 16 shows some typical frames from a reconstructed sequence of 3D facial expressions. These frames are taken from a 3330 frame animation in which the actress makes random expressions while reading from a script[2].

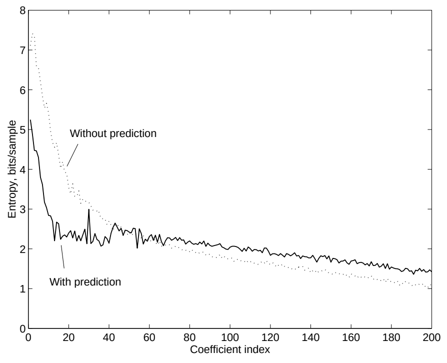


Figure 13: Reduction in entropy after temporal prediction.

The facial expressions look remarkably life-like. The animation sequence is similarly striking. Virtually all evidence of the colored fiducials and diffuse interreflection artifacts is gone, which is surprising considering that in some regions of the face, especially around the lips, there is very little of the actress' skin visible – most of the area is covered by colored fiducials.

Both the accurate 3D geometry and the accurate face texture contribute to the believability of the reconstructed expressions. Occlusion contours look correct and the subtle details of face geometry that are very difficult to capture as geometric data show up well in the texture images. Important examples of this occur at the nasolabial furrow which runs from just above the nares down to slightly below the lips, eyebrows, and eyes. Forehead furrows and wrinkles also are captured. To recreate these features using geometric data rather than texture data would require an extremely detailed 3D capture of the face geometry and a resulting high polygon count in the 3D model. In addition, shading these details properly if they were represented as geometry would be difficult since it would require computing shadows and possibly even diffuse interreflection effects in order to look correct. Subtle shading changes on the smooth parts of the skin, most prominent at the cheekbones, are also captured well in the texture images.

There are still visible artifacts in the animation, some of which are polygonization or shading artifacts, others of which arise because of limitations in our current implementation.

Some polygonization of the face surface is visible, especially along the chin contour, because the front surface of the head contains only 4500 polygons. This is not a limitation of the algorithm – we chose this number of polygons because we wanted to verify that believable facial animation could be done at polygon resolutions low enough to potentially be displayed in real time on inexpensive ($200) 3D graphics cards[3]. For film or television work, where real time rendering is not an issue, the polygon count can be made much higher and the polygonization artifacts will disappear. As graphics hardware becomes faster the differential in quality between offline and online rendered face images will diminish.

Several artifacts are simply the result of our current implementation. For example, occasionally the edge of the face, the tips of the nares, and the eyebrows appear to jitter. This usually occurs when dots are lost, either by falling below the minimum size threshold or by not being visible to three or more cameras. When a dot is lost the algorithm synthesizes dot position data which is

[2]The rubber cap on the actress' head was used to keep her hair out of her face.

[3]In this paper we have not addressed the issue of real time texture decompression and rendering of the face model, but we plan to do so in future work

usually incorrect enough that it is visible as jitter. More cameras, or better placement of the cameras, would eliminate this problem. However, overall the image is extremely stable.

In retrospect, a mesh constructed by hand with the correct geometry and then fit to the cyberware data [10] would be much simpler and possibly reduce some of the polygonization artifacts.

Another implementation artifact that becomes most visible when the head is viewed near profile is that the teeth and tongue appear slightly distorted. This is because we do not use correct 3D models to represent them. Instead, the texture map of the teeth and tongue is projected onto a sheet of polygons stretching between the lips. It is possible that the teeth and tongue could be tracked using more sophisticated computer vision techniques and then more correct geometric models could be used.

Shading artifacts represent an intrinsic limitation of the algorithm. The highlights on the eyes and skin remain in fixed positions regardless of point of view, and shadowing is fixed at the time the video is captured. However, for many applications this should not be a limitation because these artifacts are surprisingly subtle. Most people do not notice that the shading is incorrect until it is pointed out to them, and even then frequently do not find it particularly objectionable. The highlights on the eyes can probably be corrected by building a 3D eye model and creating synthetic highlights appropriate for the viewing situation. Correcting the skin shading and self shadowing artifacts is more difficult. The former will require very realistic and efficient skin reflectance models while the latter will require significant improvements in rendering performance, especially if the shadowing effect of area light sources is to be adequately modeled. When both these problems are solved then it will no longer be necessary to capture the live video sequence – only the 3D geometric data and skin reflectance properties will be needed.

The compression numbers are quite good. Figure 14 shows a single frame from the original sequence, the same frame compressed by the MPEG4 codec at 460 Kbps and at 260 KBps. All of the images look quite good. The animated sequences also look good, with the 260 KBps sequence just beginning to show noticeable compression artifacts. The 260 KBps video is well within the bandwidth of single speed CDROM drives. This data rate is probably low enough that decompression could be performed in real time in software on the fastest personal computers so there is the potential for real time display of the resulting animations. We intend to investigate this possibility in future work.

There is still room for significant improvement in our compression. A better mesh parameterization would significantly reduce the number of bits needed to encode the eyes, which distort significantly over time in the texture map space. Also the teeth, inner edges of the lips, and the tongue could potentially be tracked over time and at least partially stabilized, resulting in a significant reduction in bit rate for the mouth region. Since these two regions account for the majority of the bit budget, the potential for further reduction in bit rate is large.

9 Conclusion

The system produces remarkably lifelike reconstructions of facial expressions recorded from live actors' performances. The accurate 3D tracking of a large number of points on the face results in an accurate 3D model of facial expression. The texture map sequence captured simultaneously with the 3D deformation data captures details of expression that would be difficult to capture any other way. By using the 3D deformation information to register the texture maps from frame to frame the variance of the texture map sequence is significantly reduced which increases its compressibility. Image quality of 30 frame per second animations, reconstructed at approximately 300 by 400 pixels, is still good at data rates as low as 240 Kbits per second, and there is significant potential for lowering this bit rate even further. Because the bit overhead for the geometric data is low in comparison to the texture data one can get a 3D talking head, with all the attendant flexibility, for little more than the cost of a conventional video sequence. With the true 3D model of facial expression, the animation can be viewed from any angle and placed in a 3D virtual environment, making it much more flexible than conventional video.

References

[1] BEIER, T., AND NEELY, S. Feature-based image metamorphosis. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 35–42.

[2] BREGLER, C., COVELL, M., AND SLANEY, M. Video rewrite: Driving visual speech with audio. *Computer Graphics 31*, 2 (Aug. 1997), 353–361.

[3] CASSELL, J., PELACHAUD, C., BADLER, N., STEEDMAN, M., ACHORN, B., BECKET, T., DOUVILLE, B., PREVOST, S., AND STONE, M. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. *Computer Graphics 28*, 2 (Aug. 1994), 413–420.

[4] DECARLO, D., AND METAXAS, D. The integration of optical flow and deformable models with applications to human face shape and motion estimation. *Proceedings CVPR* (1996), 231–238.

[5] ESSA, I., AND PENTLAND, A. Coding, analysis, interpretation and recognition of facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 7 (1997), 757–763.

[6] FAUGERAS, O. *Three-dimensional computer vision*. MIT Press, Cambridge, MA, 1993.

[7] FISCHLER, M. A., AND BOOLES, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (Aug. 1981), 381–395.

[8] HOPPE, H. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 99–108. held in New Orleans, Louisiana, 04-09 August 1996.

[9] HORN, B. K. P. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America 4*, 4 (Apr. 1987).

[10] LEE, Y., TERZOPOULOS, D., AND WATERS, K. Realistic modeling for facial animation. *Computer Graphics 29*, 2 (July 1995), 55–62.

[11] PIGHIN, F., AUSLANDER, J., LISHINSKI, D., SZELISKI, R., AND SALESIN, D. Realistic facial animation using image based 3d morphing. Tech. Report TR-97-01-03, Department of Computer Science and Engineering, University of Washington, Seattle, Wa, 1997.

[12] SCHÜRMANN, J. *Pattern Classification: A Unified View of Statistical and Neural Approaches*. John Wiley and Sons, Inc., New York, 1996.

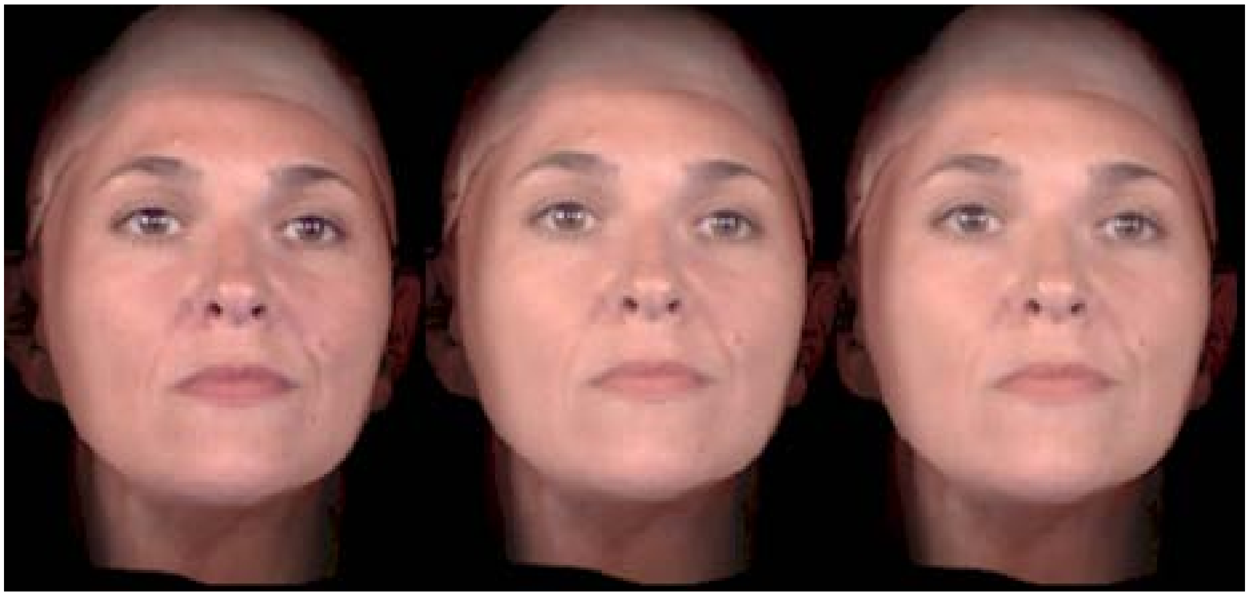


Figure 14: Left to Right: Mesh with uncompressed textures, compressed to 400 kbits/sec, and compressed to 200 kbits/sec

[13] STRANG. *Linear Algebra and its Application*. HBJ, 1988.

[14] WATERS, K. A muscle model for animating three-dimensional facial expression. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), M. C. Stone, Ed., vol. 21, pp. 17–24.

[15] WILLIAMS, L. Performance-driven facial animation. *Computer Graphics 24*, 2 (Aug. 1990), 235–242.

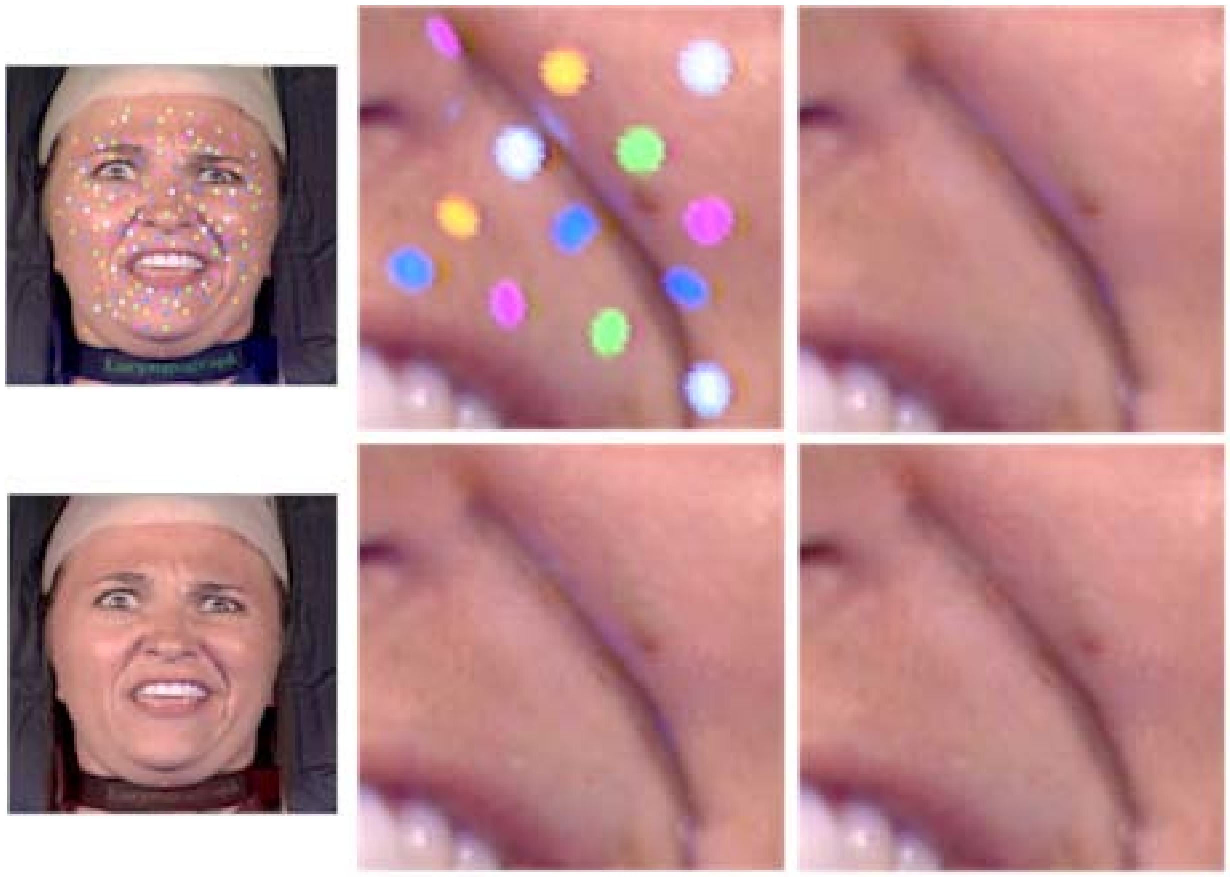


Figure 15: Face before and after dot removal, with details showing the steps in the dot removal process. From left to right, top to bottom: Face with dots, dots replaced with low frequency skin texture, high frequency skin texture added, hue clamped.

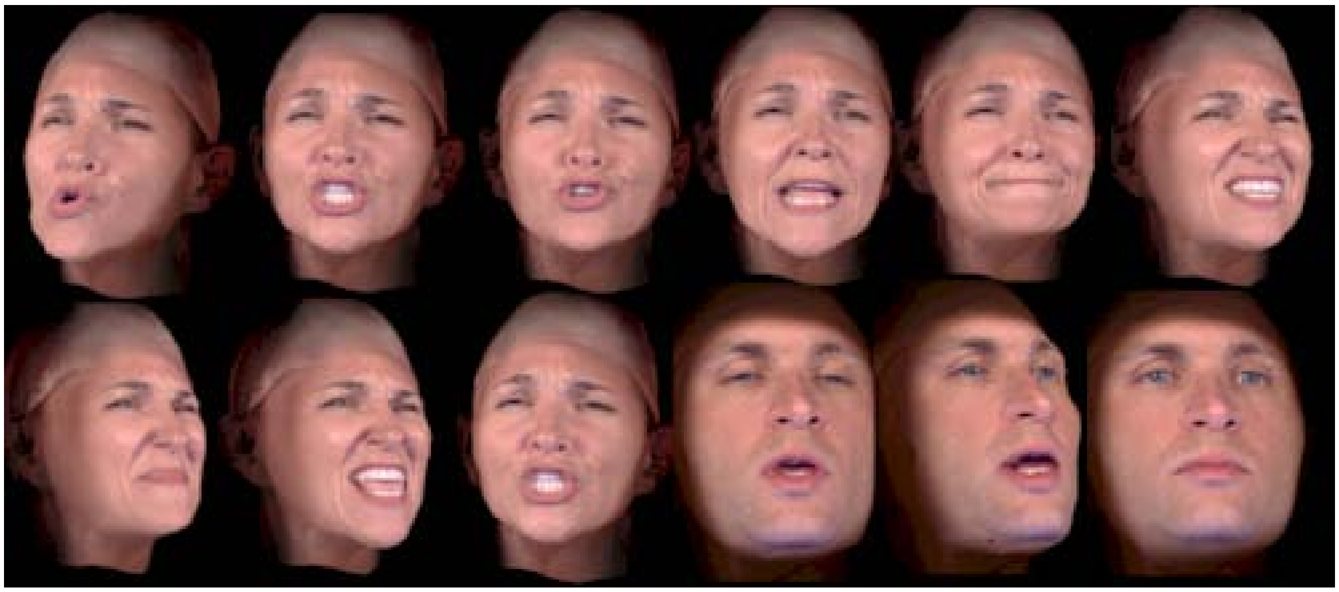


Figure 16: Sequence of rendered images of textured mesh.

Digital Face Cloning

Henrik Wann Jensen[*]
University of California, San Diego

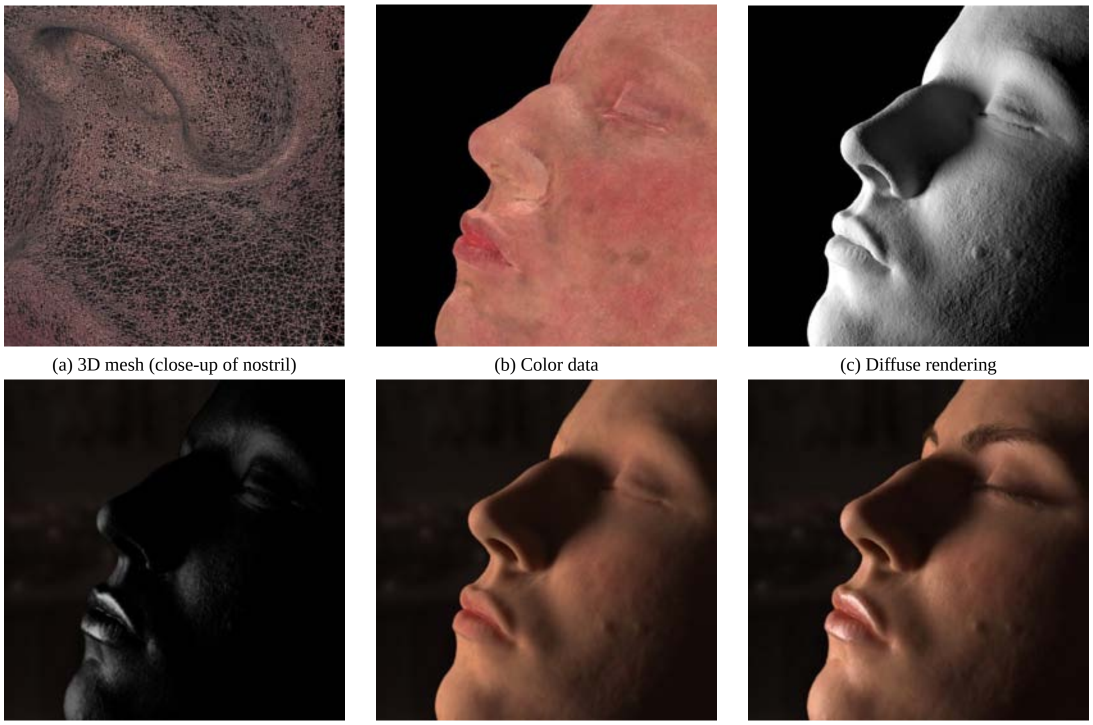

(a) 3D mesh (close-up of nostril) (b) Color data (c) Diffuse rendering
(d) Oily layer (e) Subsurface scattering (f) Final result

Figure 1: These images illustrate the steps in recreating a human face using 3D scanning (top row) and a skin BSSRDF model (bottom row).

Abstract

This sketch describes the process and the technology used in the creation of a digital clone of a human face for a story in the November 2002 issue of National Geographic on skin[Swerdlow 2002].

Digital Face Cloning

In the Spring of 2002 National Geographic contacted me to hear if I would be interested in rendering a human face for a story on skin. The challenge would be to reproduce the appearance of a face given a 3D model.

Scanning: The first challenge was the acquisition of an accurate 3D model of a face. Since the face would be rendered as a close-up spread over two pages the face model had to be very accurate, with enough resolution to capture fine details such as pores. Unfortunately, it is not possible with existing techniques to scan skin with this accuracy due to subsurface scattering which blurs the light as it interacts with the skin. To achieve sufficient accuracy a company (Arius3D) has developed a technique where they create a cast of a human face. The cast is flexible enough to represent pores and sufficiently opaque that it can be scanned with very high resolution (100μm). With this technology, they were able to produce a face model with 13 million triangles. A slightly simplified version is shown in Figure 1a - the smallest triangles in this image are representative for the detail in the full model. In addition, photographs of the model were aligned with the 3D model to obtain a rough estimate of the skin color (Figure 1b). A white diffuse rendering of the geometry is shown in Figure1c.

Rendering: To accurately render a human face it is necessary to take into account the two key elements of light interaction with human skin: subsurface scattering and light reflection of the oily layer at the surface the skin. Since the scanned model did not have any information about oil, the first task was to create a program that made it possible to interactively adjust the oil layer density over the face. A rendering of the final oily layer (using a Torrance Sparrow model) is shown in Figure 1d.

A full simulation of subsurface scattering requires a BSSRDF model [Jensen et al. 2001] as well as scattering and absorption parameters for the skin. These parameters were not available and instead the skin parameters measured in [Jensen et al. 2001] were used as a starting point. Furthermore, a simple heuristic was developed to locally adjust the scattering parameters slightly based on the color information from the scanning process. Using the BSSRDF model we obtained the image in Figure 1e, which shows the subsurface scattering component of the skin.

To obtain the final face image it was necessary to add facial hair to the model. Most important was the lack of eyebrows. The eyebrows and eyelashes were modeled by Steve Worley using his commercial Sasquatch program (www.worley.com), and rendered using the Kajiya model for hair [Kajiya and Kay 1989].

Note, that the rendering of the final face model had to be done in roughly one week. The final combined result is shown in Figure 1f, and this image appeared in the 2002 November issue of National Geographic [Swerdlow 2002], and was used on the cover of the Dutch edition. For the same article, National Geographic had the human subject photographed under roughly the same lighting conditions as in the simulation in order to compare with the results of the computer generated skin. Unfortunately, the subject used a significant amount of makeup for these closeup photographs, which almost eliminated the presence of subsurface scattering in her skin.

Future challenges includes a more detailed skin model as well as the ability to animate the face.

References

Jensen, H. W., Marschner, S., Levoy, M., and Hanrahan, P. 2001. A practical model for subsurface light transport. *Proceedings of SIGGRAPH '2001*, 511–518.

Kajiya, J., and Kay, T. 1989. Rendering fur with three dimensional textures. *Proceedings of SIGGRAPH 1989*.

Swerdlow, J. 2002. Unmasking skin. *National Geographic*, 36–63.

[*]e-mail: henrik@cs.ucsd.edu

A Rapid Hierarchical Rendering Technique for Translucent Materials

Henrik Wann Jensen
Stanford University

Juan Buhler
PDI/DreamWorks

Abstract

This paper introduces an efficient two-pass rendering technique for translucent materials. We decouple the computation of irradiance at the surface from the evaluation of scattering inside the material. This is done by splitting the evaluation into two passes, where the first pass consists of computing the irradiance at selected points on the surface. The second pass uses a rapid hierarchical integration technique to evaluate a diffusion approximation based on the irradiance samples. This approach is substantially faster than previous methods for rendering translucent materials, and it has the advantage that it integrates seamlessly with both scanline rendering and global illumination methods. We show several images and animations from our implementation that demonstrate that the approach is both fast and robust, making it suitable for rendering translucent materials in production.

Keywords: Subsurface scattering, BSSRDF, reflection models, light transport, diffusion theory, global illumination, realistic image synthesis

1 Introduction

Translucent materials are frequently encountered in the natural world. Examples include snow, plants, milk, cheese, meat, human skin, cloth, marble, and jade. The degree of translucency may vary, but the characteristic appearance is distinctly smooth and soft as a result of light scattering inside the objects, a process known as subsurface scattering. Subsurface scattering diffuses the scattered light and blurs the effect of small geometric details on the surface, softening the overall look. In addition, scattered light can pass through translucent objects; this is particularly noticeable when the objects are lit from behind. To render these phenomena and capture the true appearance of translucent materials it is therefore necessary to simulate subsurface scattering.

Traditionally subsurface scattering has been approximated as Lambertian diffuse reflection. This was later improved by Hanrahan and Krueger [1993] with an analytic term for single scattering in order to account for important directional effects. They also proposed a method for simulating subsurface scattering by tracing photons through the material, but in the end they used a BRDF (Bidirectional Reflectance Distribution Function [Nicodemus et al. 1977]) to represent the final model. A BRDF only accounts for scattering at a single point, and it cannot be used to simulate light transport within the material between different points on the surface. This requires treating the material as a participating medium with a surface. This was done by Dorsey et al. [1999] who used photon mapping to simulate subsurface scattering in weathered stone. Pharr and Hanrahan [2000] introduced the concept of scattering equations and demonstrated how this concept could be used to simulate subsurface scattering more efficiently than traditional Monte Carlo ray tracing.

More recently, Koenderink and van Doorn [2001] and Jensen et al. [2001] proposed modeling the scattering of light in translucent materials as a diffusion process. The diffusion approximation works particularly well in highly scattering media where traditional Monte Carlo ray tracing becomes very expensive [Stam 1995]. Jensen et al. [2001] suggested a simple analytical dipole diffusion approximation and found this model to be in good agreement with measurements of light scattered from translucent materials. They used this approximation to formulate a complete BSSRDF (Bidirectional Scattering Surface Reflectance Distribution Function [Nicodemus et al. 1977]), which relates outgoing radiance at a point to incident flux at all points on the surface. Finally, they evaluate the BSSRDF by sampling the incident flux on the surface.

The BSSRDF approximation [Jensen et al. 2001] is much faster than Monte Carlo photon tracing. However, since it requires sampling the incident flux distribution at the surface, it is still more expensive to evaluate than a traditional BRDF. It is particularly expensive for highly translucent materials where light can scatter a long distance within the material. Another difficulty with the approach is that it only includes internal scattering in the material due to direct illumination from the light sources. It is not obvious how to extend the sampling technique to include global illumination as well.

In this paper we introduce a fast and general two-pass rendering technique for translucent materials. Our approach is based on two key ideas. The first idea is to decouple of the computation of the incident illumination from the evaluation of the BSSRDF by using a two-pass approach. In the first pass, we compute the irradiance at selected points on the surface, and in the second pass we evaluate a diffusion approximation using the pre-computed irradiance samples. The second idea is to use a rapid hierarchical evaluation of the diffusion approximation using the pre-computed irradiance samples. This approach is substantially faster than directly sampling the BSSRDF since it only evaluates the incident illumination once at a given surface location, and it is particularly efficient for highly translucent materials where sampling the BSSRDF is costly. To evaluate the irradiance, we can use standard rendering techniques including scanline rendering and global illumination methods. This means that we can compute the effects of indirect illumination on translucent materials. Furthermore, our results do not suffer from any high-frequency Monte Carlo sampling noise since the hierarchical evaluation is deterministic. This is a great advantage for animations where this type of noise is particularly noticeable.

Another contribution of this paper is a reformulation of the scattering parameters for translucent materials. We show how the intrinsic scattering properties of translucent materials can be computed from two intuitive parameters: a diffuse reflectance and an average scattering distance. Finally, we show several results from our implementation of the method in a scanline renderer as well as a Monte Carlo ray tracer. Our results indicate that the hierarchical evaluation technique is fast and robust, and capable of rendering images and animations of translucent objects in complex lighting environments.

2 Light Diffusion in Translucent Materials

The scattering of light within a medium is described by the radiative transport equation [Chandrasekhar 1960]:

$$(\vec{\omega} \cdot \nabla) L(x, \vec{\omega}) = -\sigma_t L(x, \vec{\omega}) + \sigma_s L_i(x, \vec{\omega}) + s(x, \vec{\omega}). \quad (1)$$

Here, L is the radiance, s is a source term, σ_s is the scattering coefficient, σ_a is the absorption coefficient, σ_t is defined as $\sigma_a + \sigma_s$, and L_i is the in-scattered radiance:

$$L_i(x, \vec{\omega}) = \int_{4\pi} p(\vec{\omega}, \vec{\omega}') L(x, \vec{\omega}') d\vec{\omega}'. \quad (2)$$

The phase function, p, specifies the spherical distribution of the scattered light. It is normalized, $\int_{4\pi} p(\vec{\omega}, \vec{\omega}') d\vec{\omega}' = 1$, and we assume it only depends on the cosine of the scattering angle, $p(\vec{\omega}, \vec{\omega}') = p(\vec{\omega} \cdot \vec{\omega}')$. The mean cosine, g, of the scattering angle is:

$$g = \int_{4\pi} p(\vec{\omega}, \vec{\omega}')(\vec{\omega} \cdot \vec{\omega}') d\vec{\omega}'. \quad (3)$$

The value of $g \in [-1, 1]$ indicates the type of scattering in the medium. $g = 0$ is isotropic scattering, $g < 0$ is backwards scattering and $g > 0$ is forward scattering. Most translucent materials are strongly forward scattering with $g > 0.7$ (skin for example has $0.7 < g < 0.9$ [Gemert et al. 1989]). Such strongly peaked phase functions are costly to simulate in media with multiple scattering since the probability of sampling in the direction of the light sources will be low in most situations. The difficulty of sampling further increases with the distance to the light sources. In this case we can benefit from a powerful technique known as the *similarity of moments* [Wyman et al. 1980], which allows us to change the scattering properties of the medium without significantly influencing the actual distribution of light. Specifically, we can modify the medium to have isotropic scattering ($g = 0$) by changing the scattering coefficient to

$$\sigma_s' = (1 - g)\sigma_s, \quad (4)$$

where σ_s' is the *reduced* scattering coefficient. The absorption coefficient remains unchanged, and we get the reduced extinction coefficient $\sigma_t' = \sigma_s' + \sigma_a$.

Equation 1 is a five-dimensional integro-differential equation, and even in media with isotropic scattering it is in most cases difficult to solve. One approach is to expand radiance into a truncated series of spherical harmonics. For this purpose we divide the radiance into two components: the unscattered radiance, L_u, and the scattered (diffuse) radiance, L_d. The unscattered radiance is reduced as a function of the distance traveled through the medium [Ishimaru 1978]:

$$L_u(x + \Delta x, \vec{\omega}) = e^{-\sigma'_t \Delta x} L_u(x, \vec{\omega}). \quad (5)$$

The average distance at which the light is scattered, the mean-free path, is $\ell_u = 1/\sigma_t'$.

The diffusion approximation uses the first four terms of the spherical harmonic expansion to represent L_d:

$$L_d(x, \vec{\omega}) \approx F_t(x) + \frac{3}{4}\pi \vec{E}(x) \cdot \vec{\omega}. \quad (6)$$

The 0th-order spherical harmonic, the radiant fluence, F_t, is $F_t(x) = \int_{4\pi} L_d(x, \vec{\omega}') d\vec{\omega}'$, and the 3 terms of the 1st-order spherical harmonic, the vector irradiance, $\vec{E}$, is $\vec{E} = \int_{4\pi} L_d(x, \vec{\omega}')\vec{\omega}' d\vec{\omega}'$. Note that L_d cannot be purely diffuse as this would result in zero flux within the medium. Instead L_d is approximated as being mostly diffuse, but with a preferential direction (as indicated by $\vec{E}$) to the overall flow of the flux.

The diffusion approximation is particularly effective in highly scattering media at some distance from the light sources as well as in regions with rapidly changing scattering properties. This is due to the natural smoothing resulting from multiple scattering [Stam 1995]. More precisely, the diffusion approximation has been shown [Furutso 1980] to be accurate when $\sigma_a/\sigma_t \ll 1 - g^2$.

Applying the diffusion approximation (Equation 6) to the radiative transport equation (Equation 1) yields the diffusion equation (the details of the derivation can be found in [Ishimaru 1978]):

$$\frac{1}{3\sigma_t'} \nabla^2 F_t(x) = \sigma_a F_t(x) - S_0(x) + \frac{1}{\sigma_t'} \nabla \cdot \vec{S}_1(x). \quad (7)$$

Here, S_0 and S_1 represents the 0th- and the 1st-order spherical harmonics expansion of the source term, similar to the expansion for diffuse radiance.

The diffusion equation can be solved analytically for special cases [Ishimaru 1978], or by using a multigrid approach [Stam 1995]. In the case of translucent materials, we are interested in the outgoing radiance at the material surface as a function of the incoming radiance. Jensen et al. [2001] use a dipole diffusion approximation for a point source in a semi-infinite medium. The point source is an approximation of an incoming beam of light for which it is assumed that all light scatters at a depth of one mean-free path below the surface. The dipole diffusion approximation results in the following expression for the radiant exitance, M_o, at surface location x_o due to incident flux, $\Phi_i(x_i)$, at x_i:

$$dM_o(x_o) = d\Phi_i(x_i) \frac{\alpha'}{4\pi} \left\{ C_1 \frac{e^{-\sigma_{tr} d_r}}{d_r^2} + C_2 \frac{e^{-\sigma_{tr} d_v}}{d_v^2} \right\}, \quad (8)$$

where

$$C_1 = z_r \left(\sigma_{tr} + \frac{1}{d_r} \right) \text{ and } C_2 = z_v \left(\sigma_{tr} + \frac{1}{d_v} \right). \quad (9)$$

Here, $\alpha' = \sigma_s'/\sigma_t'$ is the reduced albedo, $\sigma_{tr} = \sqrt{3\sigma_a\sigma_t'}$ is the effective transport extinction coefficient, $d_r = \sqrt{r^2 + z_r^2}$ is the distance to the real light source, $d_v = \sqrt{r^2 + z_v^2}$ is the distance to the virtual source, $r = ||x_o - x_i||$ is the distance from x_o to the point of illumination, and $z_r = \ell_u$ and $z_v = \ell_u(1 + 4/3A)$ are the distance from the the dipole lights to the surface (shown in Figure 2). Finally, the boundary condition for mismatched interfaces is taken into account by the A term which is computed as $A = (1 + F_{dr})/(1 - F_{dr})$, where the diffuse Fresnel term, F_{dr} is approximated from the relative index of refraction η by [Groenhuis et al. 1983]:

$$F_{dr} = -\frac{1.440}{\eta^2} + \frac{0.710}{\eta} + 0.668 + 0.0636\eta. \quad (10)$$

In addition to Equation 8 the BSSRDF includes a single scattering term (see [Jensen et al. 2001] for the details).

2.1 The Importance of Multiple Scattering

The diffuse term is the most costly to sample for translucent materials since it depends on lighting from a large fraction of the material surface. We can approximate the average distance, $\ell_d = 1/\sigma_{tr}$,

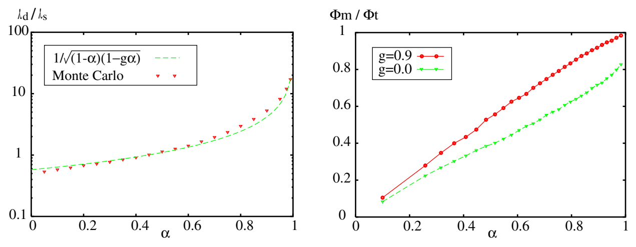


Figure 1: These graphs show the effect of increasing the scattering albedo of the material. The left graph shows the average scattering distance for diffuse radiance divided by the mean-free path for single scattering (for $g = 0.9$) as predicted by Equation 11 and estimated using a Monte Carlo photon simulation. The graph on the right shows the fraction of the radiant exitance that is due to multiple scattering (estimated with a Monte Carlo photon simulation). The important thing to notice in the two graphs is that the diffuse radiance scatters much further, and that it becomes increasingly important as the albedo gets closer to one.

along the surface that the diffused radiance scatters by assuming that the exponential term dominates in Equation 8. By dividing this distance with the mean-free path, $\ell_s = 1/\sigma_t$, of single-scattered light, we can estimate the relative scattered distance of the two within the medium:

$$\frac{\ell_d}{\ell_s} = \frac{\sigma_t}{\sigma_{tr}} = \frac{1}{\sqrt{3(1-\alpha)(1-g\alpha)}}. \qquad (11)$$

Note how the ratio depends only on the albedo, α, and the scattering anisotropy, g. Figure 1 shows a plot of this equation and a comparison with a Monte Carlo photon simulation. For the photon simulation, we traced photons from random directions towards a translucent material and recorded the average distance at which the photons left the surface again after scattering inside the material. This distance divided by ℓ_s is shown in the graph. For the simulation we used the Henyey-Greenstein phase function [Henyey and Greenstein 1941] and the photons are scattered using the approach described by Hanrahan and Krueger [1993]. Despite several assumptions about the average scattering distance, it can be seen that the predictions of Equation 11 are surprisingly accurate. For both simulations the ratio rapidly increases as the albedo approaches one as a consequence of the increasing number of scattering events. From the measurements in [Jensen et al. 2001] we can see that all of the materials have an albedo close to one. As an example, red wavelengths in skim milk (assuming $g = 0.9$) have a scattering albedo of $\alpha \approx 0.9998$, which gives a ratio $\ell_d/\ell_s \approx 129$. This means that the average distance traveled by diffuse radiance is 129 times larger than the average distance traveled by unscattered radiance. In effect this means that single scattering is substantially more localized than diffuse scattering.

The importance of multiple scattering increases with the albedo of the material. To further investigate how important multiple scattered light is for translucent materials, we performed another Monte Carlo photon simulation. In this simulation we traced photons from random directions towards the surface scattering medium. At the surface we recorded the radiant exitance from the photons that scattered in the medium. We used an index of refraction of 1.3 for the medium (the results are very similar for other values). Two important parameters for the medium are the scattering anisotropy and the scattering albedo. The right graph in Figure 1 shows the fraction of the radiant exitance from the material due to multiple scattered light as a function of the albedo. Note, how the fraction gets close to 1.0 for the forward scattering material, and close to 0.9 for a material with isotropic scattering.

3 A Two-Pass Technique for Evaluating the Diffusion Approximation

As shown in the previous section, the radiant exitance from highly scattering translucent materials is dominated by photons that have scattered multiple times inside the material. Jensen et al. [2001] compute the contribution from multiple scattering by sampling the irradiance at the material surface and evaluating the diffusion approximation — in effect convolving the reflectance profile predicted by the diffusion approximation with the incident illumination. Even though the diffusion approximation is a very effective way of approximating multiple scattering, this sampling technique becomes expensive for highly translucent materials. The reason for this is that the sampled surface area grows and needs more samples as the material becomes more translucent.

The key idea for making this process faster is to decouple the computation of irradiance from the evaluation of the diffusion approximation. This makes it possible to reuse irradiance samples for different evaluations of the diffusion equation. For this purpose, we use a two-pass approach in which the first pass consists of computing the irradiance at selected points on the surface, and the second pass is evaluating the diffusion approximation using the precomputed irradiance values. For the second pass we exploit the decreasing importance of distant samples and use a rapid hierarchical integration technique.

Pass 1: Sampling the Irradiance

To obtain the sample locations on the surface of a piece of geometry we use Turk's point repulsion algorithm [Turk 1992], which produces a uniform sampling of points on a polygon mesh. We do not change (retile) our mesh as we only need the point locations. To ensure an accurate evaluation of the diffusion approximation we need enough points to account for several factors including the geometry, the variation in the lighting, the scattering properties of the material, and the integration technique. We use the mean-free path, ℓ_u, as the maximum distance between the points on the surface. The approximate number of points that we use for a given object then becomes $A/(\pi\ell_u^2)$, where A is the surface area of the object. This is a conservative estimate, since anything below the mean-free path will be blurred by multiple scattering. However, the sample density should not be much lower since this will result in low-frequency noise in the reconstruction of the diffusion approximation. Note that our reconstruction does not require a uniform sampling since we weight each sample point by the area associated with the point. It would be possible to use other approaches that sample more densely around discontinuities in the irradiance or the geometry.

With each sample point we store the location, the area associated with the point (in the case of uniform sampling, this is simply the surface area divided by the number of points), and a computed irradiance estimate. Since the irradiance is computed at a surface location we can use standard rendering techniques including methods that account for global illumination (such as photon mapping [Jensen 1996] and irradiance caching [Ward et al. 1988]).

Pass 2: Evaluating the Diffusion Approximation

The diffusion approximation can be evaluated directly (using Equation 8) by summing the contribution from all the irradiance samples. However, this approach is costly since most objects have several thousand irradiance samples. Another, strategy would be to only consider nearby "important" points. This approach would work, but it could potentially leave out important illumination, and for accurate evaluations it would still need hundreds of irradiance samples (e.g. our sampling produces roughly 300 samples within a disc with a radius of 10 mean-free paths). Instead we use a hierarchical evaluation technique which takes into account the contribution from all irradiance samples by clustering distant samples to make this evaluation fast. The exponential shaped fall-off in the diffusion approximation makes the hierarchical approach very efficient. The concept is similar to the hierarchical approaches used for N-body problems [Appel 1985].

Several different hierarchical structures can be used for the irradiance samples. We use an octree in our implementation. Each node in the tree stores a representation of illumination in all its child nodes: the total irradiance, E_v, the total area represented by the points, A_v, and the average location (weighted by the irradiance) of the points, $\vec{P}_v$. To increase efficiency we allow up to 8 irradiance samples in a leaf voxel.

The total radiant exitance flux at a location, x, is computed by traversing the octree from the root. For each voxel we check if it is "small enough" or if it is a leaf node that it can be used directly; otherwise all the child nodes of the voxel are evaluated recursively. If the point x is inside the voxel then we always evaluate the child nodes. For all other voxels we need an error criterion that specifies the desired accuracy of the hierarchical evaluation. One option would be to compute the potential contribution from the voxel and decide based on this estimate if the voxel should be subdivided. Unfortunately, this is not trivial — and simply using the center of the points in the voxel is not a good approximation for nearby large voxels. Instead we use a much simpler criterion that is both fast to evaluate and that works well in practice. We base our criteria for subdividing a voxel on an approximation of the maximum solid angle, $\Delta\omega$, spanned by the points in the voxel:

$$\Delta\omega = \frac{A_v}{||\vec{x} - \vec{P}_v||^2}. \tag{12}$$

To decide if a voxel should be subdivided we simply compare $\Delta\omega$ to a value ϵ which controls the error. If $\Delta\omega$ is larger than ϵ then the children of the voxel are evaluated; otherwise the voxel is used directly. Another option would be to check the solid angle of the voxel itself; however, using the area of the points makes the evaluation faster, since we can use the clustered values for large voxels with just a few irradiance samples (e.g. large voxels that just barely intersect the surface).

The radiant exitance due to a voxel is evaluated using the clustered values for the voxel, or if it is a leaf-voxel by summing the contribution from each of the points in the voxel. The radiant exitance, $M_{o,p}(x)$ at x from a given irradiance sample is computed using the dipole diffusion approximation

$$M_{o,p}(x) = F_{dt}(x)\frac{dM_o(x)}{\alpha' d\Phi_i(\vec{P}_p)} E_p A_p, \tag{13}$$

where P_p is the location of the irradiance sample(s), E_p is the irradiance at that location, and A_p is the area of the location. $dM_o(||x - \vec{P}_p||_2)/(\alpha' d\Phi_i(\vec{P}_p))$ is computed using Equation 8. Notice that we scale each irradiance sample by the diffuse Fresnel transmittance, $F_{dt} = 1 - F_{dr}$ (F_{dr} is computed using Equation 10). This is necessary when approximating the irradiance by the dipole source. We could have scaled the contribution from each of the sample rays used to compute the irradiance by the true Fresnel transmittance, but by using the diffuse (Lambertian) assumption we can benefit from fast rendering techniques for diffuse materials (e.g. caching techniques such as photon maps [Jensen 1996] and irradiance caching [Ward et al. 1988]). The dipole approximation for an irradiance sample is illustrated in Figure 2. Note that this approximation has been derived assuming a semi-infinite medium. In the presence of complex geometry (e.g. curved surfaces or thin geometry) we use the same techniques as Jensen et al. [2001] to ensure numerical stability.

The result of traversing and evaluating the voxels is an estimate of the total (diffuse) radiant exitance, M_o at x, which we convert into radiance, L_o:

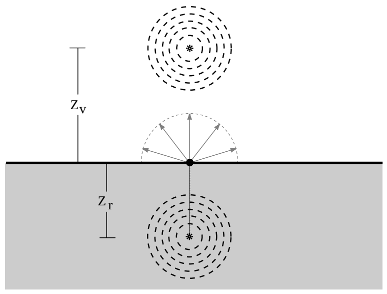


Figure 2: For each point sample we use the dipole diffusion approximation to compute the radiant exitance.

$$L_o(x,\vec{\omega}) = \frac{F_t(x,\vec{\omega})}{F_{dr}(x)}\frac{M_o(x)}{\pi}. \tag{14}$$

We scale the contribution by the Fresnel transmittance, F_t to account for reflection and transmission at the surface. Since, the diffusion approximation already includes a diffuse Fresnel transmittance we divide by F_{dr}. Alternatively, we could omit the Fresnel terms and assume a diffuse radiance.

4 Reparameterizing the BSSRDF

One difficulty in simulating subsurface scattering is that it is difficult to predict the resulting appearance from a given combination of absorption and scattering coefficients (since their effect is highly non-linear). In this section, we will outline a simple technique for reparameterizing the BSSRDF into using intuitive translucency and reflectivity parameters. These parameters are already present in the computations in the form of the diffuse mean free path ℓ_d and the diffuse reflectance of the material, and they are sufficient for computing the scattering and absorption properties of the material.

First, using the diffuse reflection coefficient (see [Jensen et al. 2001]), we solve for the reduced albedo of the material:

$$R_d = \frac{\alpha'}{2}\left(1 + e^{-\frac{4}{3}A\sqrt{3(1-\alpha')}}\right)e^{-\sqrt{3(1-\alpha')}}. \tag{15}$$

This equation is not easily invertible, but it has a simple monotonic shape in the valid region $\alpha' \in [0 : 1]$, and we use a few iterations of a simple secant root finder to compute α'.

We know the effective transport coefficient, $\sigma_{tr} \approx 1/\ell_d$, and given the reduced albedo we can find the reduced extinction coefficient:

$$\frac{\sigma_{tr}}{\sigma'_t} = \sqrt{3(1-\alpha')} \quad \longrightarrow \quad \sigma'_t = \frac{\sigma_{tr}}{\sqrt{3(1-\alpha')}} \tag{16}$$

Finally, this gives us the absorption and the reduced scattering coefficients: $\sigma'_s = \alpha'\sigma'_t$ and $\sigma_a = \sigma'_t - \sigma'_s$. If the scattering anisotropy, g, is given then the real extinction and scattering coefficients can be computed as well.

5 Results

In this section we present several results from our implementation of the rendering technique. We have used two different systems to implement the model: A Monte Carlo ray tracer with support for global illumination, and a modified a-buffer renderer used in production. Our timings were recorded on a dual 800MHz Pentium 3 for images with a width of 1024 pixels and 4 samples per pixel.

The first example include several renderings of a translucent marble teapot as shown in Figure 3. All of these images were rendered with the Monte Carlo ray tracer. The left column shows a comparison with the BSSRDF sampling technique by Jensen et al. [2001], and our hierarchical technique under the same lighting conditions (for this comparison we use the BRDF approximation for the single scattering term). The important thing to notice is that the two images are practically indistinguishable except for a small amount of sampling noise in the BSSRDF image. This shows that the hierarchical approach is capable of matching the output of the BSSRDF for a translucent material. However, the hierarchical technique took just 7 seconds to render (including 1 second to compute the irradiance values at the samples), while the BSSRDF sampling took 18 minutes — a factor of 154 speedup in this case. The speedup will be even more dramatic for objects that are more translucent. The top image in the right column shows a glossy teapot illuminated by a high dynamic range environment map [Debevec 1999]. To enhance the translucency effect we made the environment black behind the camera. The render time for the image without glossy reflection is 7 seconds (the rendering time including glossy reflection is 40 seconds). The precomputation time for the irradiance samples (sampling the environment map) was roughly 1 minute. This scene would be extremely costly to render using the BSSRDF sampling approach, since it would involve picking a point on the light source and then sampling in the direction of the teapot — a process where the probability of generating good samples is very low. Finally, the lower right image shows the 150,000 sample locations on the teapot that we used for all images.

Our second example in Figure 4 shows the classic Cornell box global illumination scene with a translucent box. This image was rendered using Monte Carlo ray tracing — we used photon mapping [Jensen 1996] to speed up the computation of global illumination. Note the light scattering through the box, and the color bleeding in the scene. In the precomputation of the indirect illumination for the irradiance samples on the translucent box we used a diffuse assumption in order to account for multiple reflections between the box and the walls. However we do account for translucency when evaluating color bleeding on the wall. For scenes where translucency is important for the indirect illumination of the translucent elements (e.g. light bleeding through a leaf onto a white floor which then reflects back on the leaf) a multi-pass approach could be used where the indirect illumination is computed recursively for the translucent materials. The rendering time for the image was 55 seconds, and the precomputation of the irradiance for 20,000 points on the box was 21 seconds.

Our third example in Figure 6 shows the lower half of a face model rendered using a standard skin shader (on the left) and using a skin shader with support for translucency (on the right). This face model was built before the translucency shader was developed. It uses several textures, which we apply by scaling the 250,000 irradiance samples with filtered texture values (the filter support is equal to the area associated with each irradiance sample). This approach made it possible to replace the previous skin shader with a translucent version. The added translucency vastly increases the realism of the model as it softens the appearance of the skin and naturally simulates effects such as the color bleeding around the nose. The rendering time using the standard skin shader was 16 minutes while the translucent skin shader took 20 minutes (including generating the sample points). A significant advantage of our approach is that it works with all the standard lights used in production such as fill lights, rim lights and key lights. This natural integration of the translucency shader in the production framework made it a natural choice for the main character in “Sprout” (a short animation). Translucency helps depict the small size of the character as shown in Figure 5.

6 Conclusion and Future Work

We have presented an efficient two-pass rendering technique for translucent materials. We combine a point sampling of the irradiance on the surface with a fast hierarchical evaluation of a diffusion approximation. Our approach is particularly efficient for highly translucent materials where the BSSRDF sampling [Jensen et al. 2001] becomes costly, and it integrates seamlessly with both scanline rendering and global illumination methods. Our results demonstrate how the technique is both fast and robust making it suitable for rendering translucent materials in production of computer animations.

Future improvements include extending the approach to translucent materials with a visible internal 3D structure. It would also be useful to investigate the accuracy of the dipole diffusion approximation in the presence of complex geometry.

Another interesting path to explore is interactive rendering of translucent materials. This could be done by further simplifying the evaluation technique so that it can be implemented directly on programmable graphics hardware.

7 Acknowledgments

Thanks to the SIGGRAPH reviewers and to Maryann Simmons, Mike Cammarano, Pat Hanrahan, and Marc Levoy for helpful comments. The first author was supported by NSF ITR (IIS-0085864). The second author was supported by PDI/DreamWorks and by the PDI R&D group and thanks them for supporting the development and publication of these techniques.

References

APPEL, A. 1985. An efficient program for many-body simulations. *SIAM Journal of Scientific Statistical Computing 6*, 85–103.

CHANDRASEKHAR, S. 1960. *Radiative Transfer*. Oxford Univ. Press.

DEBEVEC, P., 1999. St. Peter’s Basilica (www.debevec.org/probes/).

DORSEY, J., EDELMAN, A., JENSEN, H. W., LEGAKIS, J., AND PEDERSEN, H. K. 1999. Modeling and rendering of weathered stone. In *Proceedings of SIGGRAPH’99*, 225–234.

FURUTSO, K. 1980. Diffusion equation derived from space-time transport equation. *J. Opt. Soc. Am 70*, 360.

GEMERT, M., JACQUES, S., STERENBORG, H., AND STAR, W. 1989. Skin optics. *IEEE Trans. on Biomedical Eng. 16*, 1146–1156.

GROENHUIS, R. A., FERWERDA, H. A., AND BOSCH, J. J. T. 1983. Scattering and absorption of turbid materials determined from reflection measurements. 1: Theory. *Applied Optics 22*, 2456–2462.

HANRAHAN, P., AND KRUEGER, W. 1993. Reflection from layered surfaces due to subsurface scattering. In *Computer Graphics (SIGGRAPH’93 Proceedings)*, 165–174.

HENYEY, L., AND GREENSTEIN, J. 1941. Diffuse radiation in the galaxy. *Astrophysics Journal 93*, 70–83.

ISHIMARU, A. 1978. *Wave Propagation and Scattering in Random Media*, vol. 1. Academic Press, New York.

JENSEN, H. W., MARSCHNER, S. R., LEVOY, M., AND HANRAHAN, P. 2001. A practical model for subsurface light transport. In *Proceedings of SIGGRAPH 2001*, 511–518.

JENSEN, H. W. 1996. Global illumination using photon maps. In *Rendering Techniques ’96*, Springer Wien, X. Pueyo and P. Schröder, Eds., 21–30.

KOENDERINK, J., AND VAN DOORN, A. 2001. Shading in the case of translucent objects. In *Proceedings of SPIE*, vol. 4299, 312–320.

NICODEMUS, F. E., RICHMOND, J. C., HSIA, J. J., GINSBERG, I. W., AND LIMPERIS, T. 1977. Geometric considerations and nomenclature for reflectance. Monograph 161, National Bureau of Standards (US), Oct.

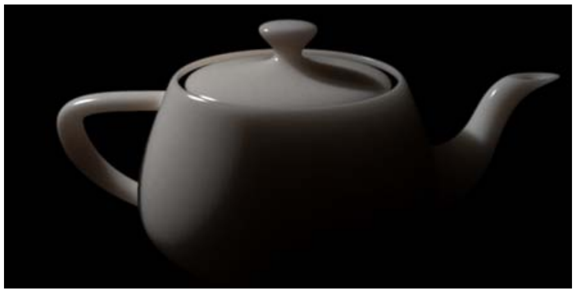
PHARR, M., AND HANRAHAN, P. 2000. Monte carlo evaluation of non-linear scattering equations for subsurface reflection. In *Proceedings of SIGGRAPH 2000*, 75–84.

STAM, J. 1995. Multiple scattering as a diffusion process. In *Eurographics Rendering Workshop 1995*, Eurographics.

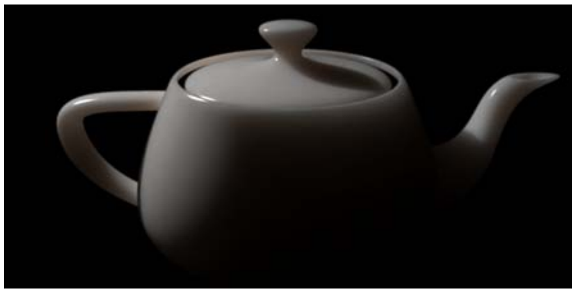
TURK, G. 1992. Re-tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH ’92 Proceedings)*, vol. 26, 55–64.

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. In *Computer Graphics (SIGGRAPH ’88 Proceedings)*, vol. 22, 85–92.

WYMAN, D. R., PATTERSON, M. S., AND WILSON, B. C. 1980. Similarity relations for anisotropic scattering in monte carlo simulations of deeply penetrating neutral particles. *J. Comp. Physics 81*, 137–150.


BSSRDF: sampled evaluation - 18 minutes

Illumination from a HDR environment


BSSRDF: hierarchical evaluation - 7 seconds

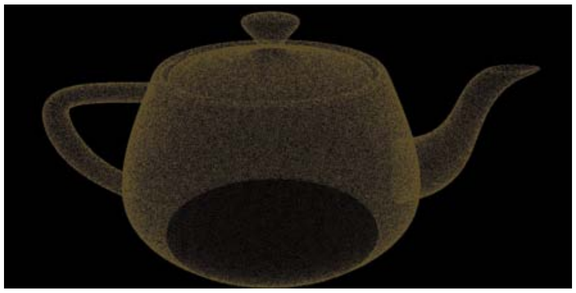

The sample locations on the teapot

Figure 3: A translucent teapot. On the left we compare our hierarchical BSSRDF evaluation (bottom) to a sampled BSSRDF (top). The top right image shows the teapot in a HDR environment, and the bottom right shows the 150,000 sample points on the teapot.

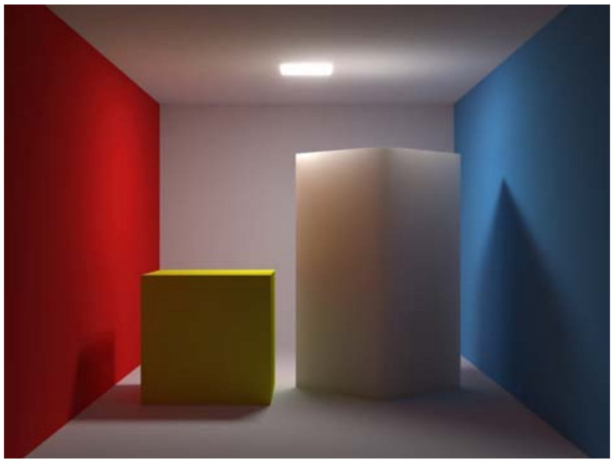


Figure 4: A global illumination scene with a translucent box. Note the light bleeding through the box, and the color bleeding in the model.

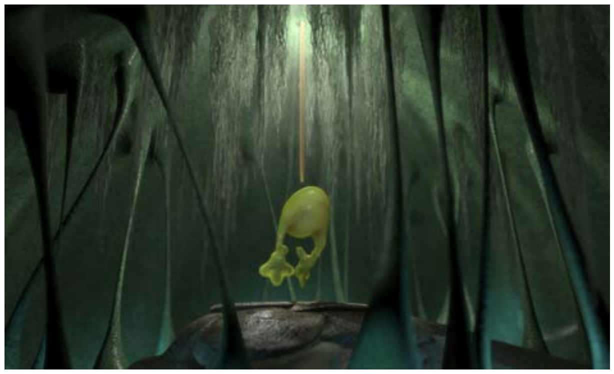


Figure 5: An animation with a translucent character. Translucency helps depict the small size of the character. Image courtesy of Scott Peterson - PDI/DreamWorks.

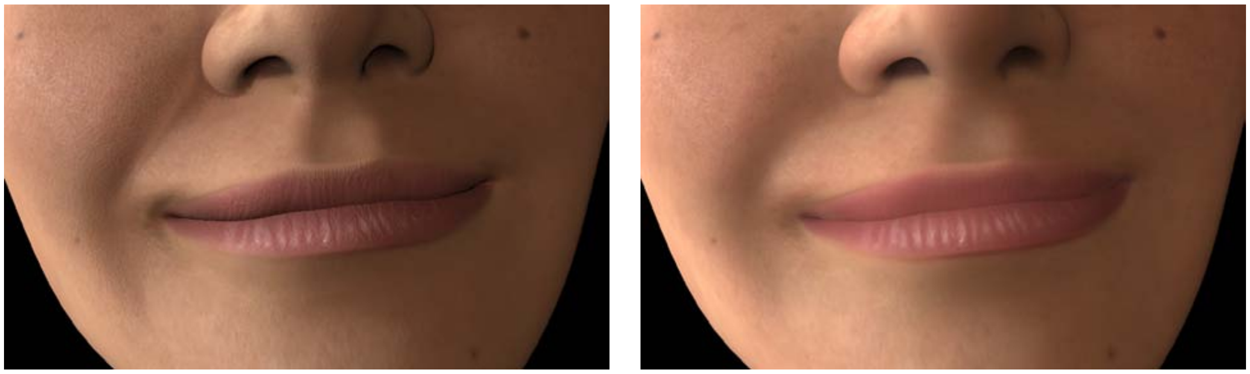


Figure 6: A textured face model lit by three light sources (key, fill, and rim). The left image shows the result using the skin shader that was used in the movie "Shrek", and the right image shows the result after adding our simulation of translucency to this shader.

Realistic Human Face Rendering for “The Matrix Reloaded”

George Borshukov and J.P.Lewis
ESC Entertainment

Introduction

The ultimate challenge in photorealistic computer graphics is rendering believable human faces. We are trained to study the human face since birth, so our brains are intimately familiar with every nuance and detail of what human skin is supposed look like. The challenge of rendering human skin is further complicated by some technical issues such as the fact that skin is a highly detailed surface with noticeable features in the order of ~100 microns and the fact that skin is translucent. On *The Matrix Reloaded* we had to create completely photorealistic renderings for most of the principal actors including Keanu Reeves, Lawrence Fishborne, and Hugo Weaving.

Facial Surface Detail

The geometry used for our rendering was based on a 100-micron resolution scan of a plaster cast mold of the actors’ faces. Arius3d provided the scanning technology. These scans had extremely high polygonal counts (10 million triangles; see Fig. 1). To use these models in production and preserve the detail we deployed the following technique. A low-res ~5K quad model was constructed using Paraform software. The model was given a UV parameterization and then used as a subdivision surface. The high resolution detail was extracted using the lightmapping feature of the mental ray renderer combined with custom shaders that performed ray tracing from the low-res subdivision surface model to the high-detailed 10M triangle raw scan; the distance difference is stored in a displacement map. We applied the low frequency component of this map as displacement; the high frequency component was applied using bump mapping.

Image-based Derivation of Skin BRDF

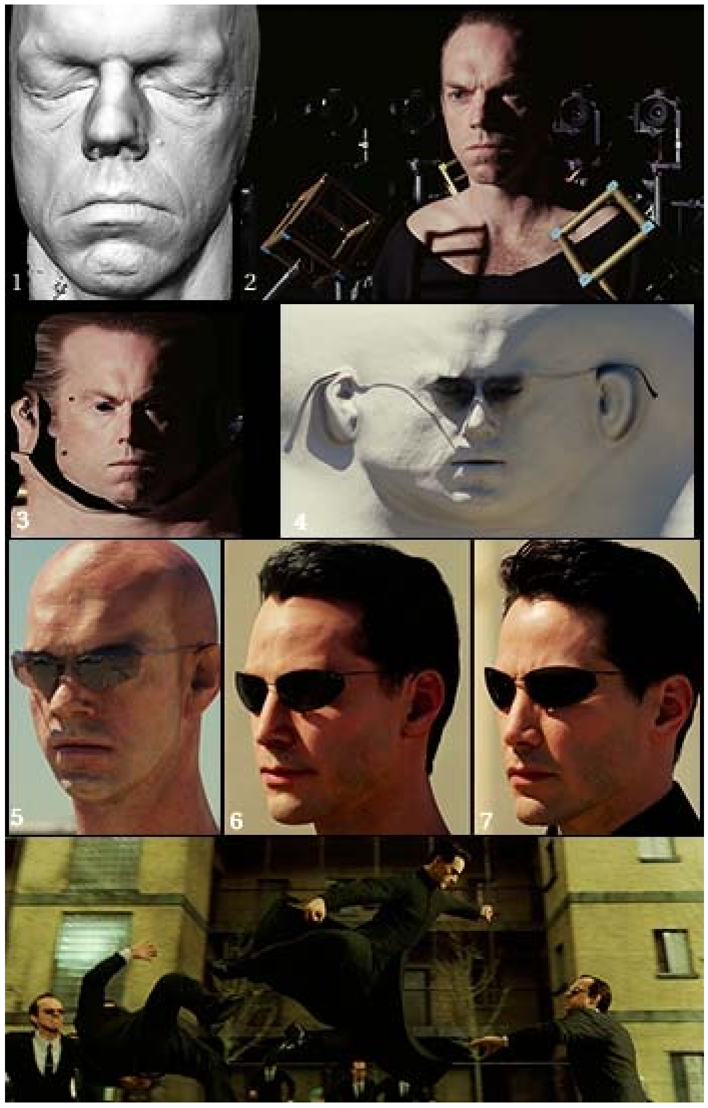
Our skin BRDF was derived using an image-based approach. In Summer 2000 as part of the early stages of *Matrix Reloaded* R&D we had a setup, which consisted of 30 still cameras arranged around the actor’s head. Actors were photographed illuminated with a series of light sources from different directions (see Fig. 2). The setup was carefully color calibrated and photogrammetry was used to precisely reconstruct the camera positions and head placement with respect to each camera for each image. The collected image data from each camera was brought into a common UV space through reprojection using a cyberscan model of the actor. This convenient space (see Fig. 3) allowed us to analyze the skin reflectance properties for many incident and outgoing light directions. We derived parameters for an approximate analytical BRDF that consisted of a Lambertian diffuse component and a modified Phong-like specular component with a Fresnel-like effect. (We would like to acknowledge Matthew Landauer for his contributions to this section).

Subsurface Scattering of Skin

As production progressed it became increasingly clear that realistic skin rendering couldn’t be achieved without subsurface scattering simulation. There are a number of published methods for rendering translucent materials however they are all fairly complex, require large amounts of CPU power and produce somewhat disappointing results. To address this we developed a technique for producing the appearance of subsurface scattering in skin that is computationally inexpensive and fairly easy to implement. The result of the diffuse illumination reflecting off the face in the camera direction is stored in a 2-d light map (see Fig. 4). We then approximately simulate light diffusion in the image domain. To simulate the different mean free path for different light colors we vary the diffusion parameters for each color channel. For animations the lightmap needs to be computed at every frame, so our technique computes an appropriate lightmap resolution depending on the size of the head in frame. For objects like ears where light can pass directly through, we employed a more traditional ray tracing approach to achieve the desired translucency effect.

Results

The above components are combined with our Universal Capture, real world Lighting Reconstruction technologies, and a ray tracer such as mental ray to produce the synthetic images in Fig. 5 and 6. For comparison Fig. 7 shows a photograph of Keanu Reeves (Neo). The bottom image is a fully virtual frame from *The Matrix Reloaded*.



Reflectance Field Rendering of Human Faces for “Spider-Man 2”

Mark Sagar
Sony Pictures Imageworks

The creation of convincing computer generated human faces which can withstand close-up scrutiny under arbitrary lighting has been notoriously difficult to achieve, especially for well known actors. For the film “Spider-Man 2” it was decided to try recent experimental computer graphics research that looked promising but that had never been production tested.

In order to create highly realistic digital versions of the faces for the main characters (played by Tobey Maguire and Alfred Molina) the techniques introduced in the SIGGRAPH 2000 paper “Acquiring the reflectance field of a human face”[1] were extended to support deforming shapes for facial animation and integrated into the Sony Imageworks CG pipeline.

The USC ICT Lightstage 2.0 was used for the capture. Four film cameras were placed at various angles around the subject and synchronized to the strobes for simultaneous image capture. The resulting images were color corrected and projected onto a 3D model of the performer. Colorspace Analysis was used to decompose each dataset into diffuse and specular components. The specular component distribution was transformed to a view independent representation. The decomposed multiview datasets were then combined in UV space weighted by surface normal and visibility criteria to create the final reflectance functions.

In order to apply facial motion capture to create an animated model which could be relighted it is necessary either to capture images of the face in many expressions and interpolate, requiring a huge amount of data, or to modify a single set of neutral expression reflectance functions as the model deforms.
Because we had very limited time with the actors, and to avoid the logistics and processing cost of capturing many separate expressions, only the images captured for the faces in the neutral position (and with teeth bared) were used. In order to approximate the lighting changes as the face deformed (driven by motion capture or animation) into different expressions the reflectance functions were transformed based on changes in surface normal direction and light source visibility. This proved successful, however it is a first order approximation only as it does not account for indirect illumination changes and does not represent visual changes due to blood redistribution as the face compresses.

The eyes had to be treated separately as they move independently from the head, and the captured data for corneal highlights were too sparse to re-synthesize sharp continuous highlights. Obscured regions of the eyes were filled in by mapping normalized reflectance functions of nearby tissue. The surface geometry of the corneas was determined from the reflectance data, to create a bump map for a conventional shader.

A few unexpected problems had to be dealt with. The actors both had haircuts which did not match the final look in the film, and also had some surface blemishes. Certain small areas in the ear were not seen by any camera and so had no capture data. Several approaches used to fix these problems (which had to be addressed on 480 frames per camera) including paint fixes and the creation of a "virtual lightstage" to generate skin texture for missing regions.

One of the main challenges was integration with the conventional CG shading pipeline. To do this the reflectance field shader was implemented to work with both individual lights controlled by the existing lighting tools and with HDR image based environmental lighting. A downsampled dataset was available to speed up rendering for the environmental case. The acquired reflectance field is non-local so the data does not account for partial shadowing and other local lighting effects (e.g. the lighting changes caused by wearing sunglasses). This was approximated by locally modifying the illumination model using conventional shading techniques.

The successful combination of the reflectance field shaded and conventionally shaded CG permitted artists/Technical Directors to use preexisting tools and workflow without special consideration to which shading technique was being employed.

Contributors: (Sony Pictures Imageworks) John Monos, John Schmidt, Dan Ziegler, Sing-choong Foo, Remington Scott, Jeff Stern, Chris Waegner, Peter Nofz, (ICT) Tim Hawkins, Paul Debevec

References

1. Debevec, P., Hawkins, T., Tchou, C., Duiker, H.P., Sarokin, W. and Sagar, M. 2000. Acquiring the reflectance field of a human face. In SIGGRAPH 2000, *Computer Graphics Proceedings*, 145-156, ACM SIGGRAPH

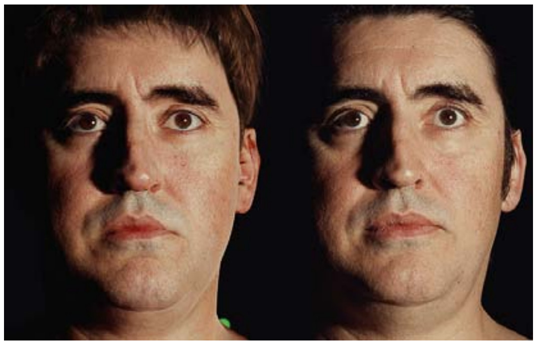


Figure 1 Digital vs Real

Analysis and Synthesis of Facial Expressions with Hand-Generated Muscle Actuation Basis

Byoungwon Choe[*] Hyeong-Seok Ko

School of Electrical Engineering and Computer Science
Seoul National University, Seoul, KOREA

Abstract

We present a performance-driven facial animation system for analyzing captured expressions to find muscle actuation and synthesizing expressions with the actuation values. Significantly different approach of our work is that we let artists sculpt the initial draft of the actuation basis—the basic facial shapes corresponding to the isolated actuation of individual muscles, instead of calculating skin surface deformation entirely relying on the mathematical models such as finite element methods. We synthesize expressions by linear combinations of the basis elements, and analyze expressions by finding the weights for the combinations. Even though the hand-generated actuation basis represents the essence of the subject's characteristic expressions, it is not accurate enough to be used in the subsequent computational procedures. We also describe an iterative algorithm to increase the accuracy of the actuation basis. The experimental results suggest that our artist-in-the-loop method produces more predictable and controllable outcome than pure mathematical models, thus can be a quite useful tool in animation productions.

1. Introduction

Since Williams' pioneering work on performance-driven facial animation [23], applying facial expressions from human faces to computer-generated characters has been widely studied [9, 3, 24, 12, 19]. To control facial movement, facial expressions were analyzed into the position of feature points [9, 3, 24] or the weights for blending pre-modeled expressions [12, 19]. Another fascinating approach, which we took in this work, is finding muscle actuation parameters from facial expressions [21, 6, 1]. Expressions can be easily modified by editing muscle actuation curves [1], and the actuation values can be converted to other control parameters such as the values of Actuation Units in Facial Action Coding System [4] without much effort.

Terzopoulos *et al.* [21] and Essa *et al.* [6] analyzed the expressions recorded in video footage into muscle actuation values for *facial expression recognition*. They also synthesized facial expressions with the actuation values. The synthetic expressions, however, showed only conspicuous ones such as 'opening mouth' or 'raising eyebrows', which were not yet to be used in high quality animation production. Recently, Choe *et al.* [1] proposed an algorithm to find muscle actuation values from the trajectory of feature points generated by an optical capture system. They could reproduce delicate facial movements by extracting complicated set of muscle actuation values with a linear finite element model, and showed the possibility of practical use in character animation. Still, the heuristic muscle model could misinterpret the original expressions, and simplified finite element model occasionally produced unnatural artifacts in skin deformation.

In this work, instead of relying entirely on the mathematical models to compute the 3D facial shape, we include the artists' modeling capability as an integral part of the method. According to our previous tests, the result of pure mathematical modeling was usually distant from what was *expected*.[1] Such expectation cannot be quantitatively stated; we thought that an artist may be able to form the expected (or desired) facial shape. Thus we made the artists sculpt manually a set of expressions called the *muscle actuation basis*, and let the computer program synthesize expressions based on the basis elements. Each element of the actuation basis corresponds to the facial shape when a single expression muscle is fully actuated and the rest are left relaxed.

We can synthesize a facial expression by the linear combination of the basis elements on the same principle as the linear muscle model [1]. Then our algorithm is basically re-

[*] 133-316, Seoul National University, Shillim-dong, Gwanak-gu, Seoul, 151-742, KOREA. drei@graphics.snu.ac.kr

[1] Some of the reasons might be that we could not calibrate the muscle size and layout of the computer model with those of the subject being captured, and we made too many simplifying assumptions to use mathematical models.

duced to the methods that synthesize expressions by blending pre-modeled expressions, which was experimented by Kouadio *et al.* [12] and Pighin *et al.* [18, 19]. Our method is different from theirs in the pre-modeled expression set: we use an artificial but functional set of expressions instead of using real human expression samples such as 'happiness' or 'sadness'. Using the actuation basis rather than real human expression samples has an important consequence. The elements in the actuation basis are *orthogonal* to each other, and form a meaningful basis for the facial expression space—the actuation basis can produce (or in mathematical terms, *span*) the complete set of human expressions. When real human expressions are used, on the other hand, the linear combinations of them cannot generally guarantee to produce the complete set of expressions.[2]

We can summarize our facial animation process into two major steps: **modeling** to set up the neutral face and the actuation basis of a subject and **analysis** to find muscle contractions from the subject's performance using the actuation basis. We can synthesize expressions by applying the analyzed muscle contractions to any computer model with an equivalent muscle structure.

In order to model the actuation basis, we first obtain the neutral face of the subject using a 3D scanning device. Starting from the neutral face, we let an artist sculpt the basis elements considering the human facial anatomy [2]. The work of Faigin [7], which illustrates the facial shape corresponding to the actuation of each individual muscle, serves a good guide for the job. It could be expected that the first hand-generated draft would not give a satisfactory result. Moreover, considering that the accuracy of the actuation basis greatly affects the result of the **analysis**, we need to develop a procedure for improving the basis. The improvement procedure (described in Section 4.2), in turn, refers to the result of the **analysis** on some trial data; the procedure takes the form of fixed point iteration between **modeling** and **analysis**.

Once the actuation basis of a subject is ready, we can start analyzing the expressions captured from the subject. We approximate each frame of the facial performance by a linear combination of the basis elements. Finding the best approximation can be formulated as a *constrained quadratic programming*, and the coefficients in the resulting solution are interpreted as the muscle contraction values.

The rest of this paper is organized as follows. Section 2 reviews related work in facial animation. Section 3 and Section 4 present the **modeling** and **analysis** procedures respectively. Section 5 shows the experimental results of our method, and Section 6 concludes the paper.

[2]There have been efforts to resolve the correlation among human expression samples and map the expressions into an orthogonal domain [14]. A popular domain studied first in psychology was a two-dimensional space represented by *pleasure* and *arousal* axes [8]. However, the quantitative use of the parameters (e.g., for expression synthesis) does not seem suitable since the dimension is quite limited and assigning the coordinate values is done in a subjective manner.

2. Background

This section reviews the state-of-the-art techniques on performance-driven facial animation and muscle-based facial modeling. More topics on facial modeling and animation can be found in [17].

Williams [23] introduced a performance-driven facial animation system which synthesized expressions by changing texture coordinates calculated from the position of feature points on the face. Guenter *et al.* [9] captured both the 3D geometry and shading information of a human face, and reproduced photorealistic expressions. Eisert and Girod [3] modeled a face with a triangular B-spline surface, and analyzed facial expressions by estimating the facial animation parameters of MPEG-4 standard. Pighin *et al.* [18] reconstructed the geometry and texture of an individual face from five photo images of the subject. With this method, they modeled basic expressions such as 'joy' or 'surprise', and synthesized novel expressions by blending them. The result was photo-realistic, showing detailed wrinkles and creases. Later, they proposed an algorithm to find the blending weights from the video recording of a performance [19]. Kouadio *et al.* [12] animated a synthetic character by the linear combination of previously modeled 3D facial expressions by extracting the interpolation weights from the feature points traced by an optical capture device.

Waters [22] introduced an anatomically based muscle model which was kinematically formulated. Terzopoulos *et al.* [20, 13] represented the mesh of the skin surface by a mass-spring model, and calculated skin deformation due to muscle actuation. Koch *et al.* predicted the geometry of skin surface due to the skull shape change using a finite-element model [11], and synthesized expressions by embedding expression muscles [10].

Terzopoulos and Waters [21] developed a method to extract muscle contractions from the expressions recorded in video footage based on a dynamic muscle model. Essa *et al.* [5, 6] developed a system to estimate muscle actuation corresponding to a given expression using feedback control theory. Choe *et al.* [1] calculated muscle actuation values based on the finite element skin model and linear muscle model.

3. Modeling Muscle Actuation Basis

Muscle actuation basis is a set of expressions $\{\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_m\}$, each of which represents the 3D facial shape when a single expression muscle is fully actuated and the others are relaxed. Figure 1 shows an example of the actuation basis.

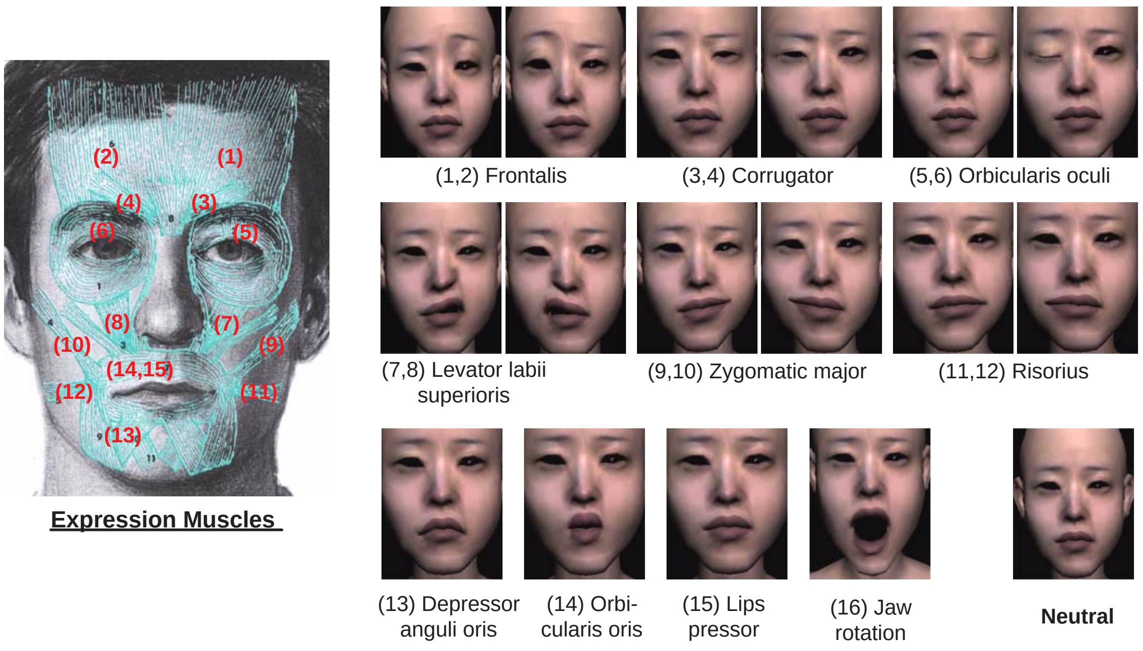


Figure 1. Expression muscles and the corresponding basis elements in the actuation basis.

Once we have the actuation basis, we can synthesize facial expressions by linear combinations of the basis elements if we assume the linear muscle model [1]. Let $\mathbf{E}_0$ denote the neutral expression—the position of about 1,500 vertices that constitute the facial surface. Let $\mathbf{e}_i = \mathbf{E}_i - \mathbf{E}_0$ ($i = 1, 2, \ldots, m$) be the difference between the basis element and the neutral expression, where m is the number of the basis elements. When the muscle contractions $x_1, x_2, \ldots, x_m$ are given, we synthesize an expression $\mathbf{E}$ by

$$\mathbf{E} = \mathbf{E}_0 + \sum_{i=1}^{m} x_i \mathbf{e}_i. \tag{1}$$

We normally expect the muscle contractions have the value in $[0, 1]$ since each basis element embodies the full actuation of an expression muscle.

In this work, we used an actuation basis of 16 elements as shown in Figure 1: *six* for the muscles in the upper region around the eyebrows (Figure 1 (1)$\sim$(6)), *ten* for the muscles in the lower region around the mouth (Figure 1 (7)$\sim$(16)). We get the reduced set of 16 basis elements to represent the operation of not less than 26 expression muscles in the human face. The operation of several muscles can be merged into a single basis element if they are dependent on each other, and the operation of a single muscle should be represented by multiple basis elements if the actuation of the muscle can produce multiple distinct shapes:

- **Merging:** We merge the operation of the muscles into a single basis element if they usually actuate simultaneously. For example, we merge the three muscles *Levator labii superioris alaeque nasi*, *Levator labii superioris*, and *Zygomatic minor* which are known as the sneering muscles (see Figure 1 (7, 8)) into the single basis element *Levator labii superioris*. The basis elements *Corrugator* (Figure 1 (3, 4)), *Risorius* (Figure 1 (11, 12)), and *Depressor anguli oris* (Figure 1 (13)) are also the results of merging the operation of two or three muscles.

- **Mouth:** The operation of *Orbicularis oris* around the mouth is very complicated, and the full actuation of the muscle can generate many different shapes. In this work, we created two basis elements to represent the operation of the muscle: normal *Orbicularis oris* which corresponds to the mouth shape when pronouncing /u/ sound (Figure 1 (14)), and the *Lips pressor* which corresponds to the protruded (upset) mouth (Figure 1 (15)).[3] Gentle closing of the mouth is covered by the neutral expression $\mathbf{E}_0$.

- **Eyes:** *Orbicularis oculi*, the sphincter muscle at the eyes, consists of the palpebral and orbital parts. In this

[3]*Orbicularis oris* was an obvious choice for the basis, but the inclusion of *Lips pressor* was based upon our experiences: without the *Lips pressor*, we observed the elements *Risorius* and *Orbicularis oris* had to combine frequently to produce the shape of *Lips pressor*, which was quite unnatural in the operation of human expression muscles.

work, we implemented only the operation of the palpebral part (gentle closing of the eyes) as a basis element (Figure 1 (5, 6)). Therefore emphatic closing of the eyes cannot be generated.

We let artists model the basis elements considering the size and location of expression muscles [2]. Faigin [7] illustrated the facial expressions resulting from the actuation of a single or multiple expression muscles, which served an excellent guide to the modeling job. The actuation basis only used for expression *synthesis* does not need to come from a human subject. However, the actuation basis for expression *analysis* should accurately reflect the operation of expression muscles of the subject because it affects drastically the result of expression analysis (Section 4.1). Therefore artists were asked to watch carefully the video recording of the subject (or the *training data* in Section 4.2) where the subject was asked to make all kinds of expressions including the extreme actuation of each muscle.

It would be impractical to assume that the hand-generated actuation basis is accurate enough. Fortunately, there is a way to evaluate the given basis: we simply run the expression analysis procedure on the training data, then we can infer that the basis is not accurate when the resulting muscle contractions go far beyond the expected range $[0, 1]$. In such a case, we ask the artists to re-model the basis elements. We repeat the step until a reasonable basis is obtained. However, it would be still impractical to assume that the resulting basis is accurate enough to start our computational steps of expression analysis. We present an algorithm that improves further the actuation basis (at this time without the help of artists) by taking iterations between the expression analysis and basis modification procedures. The algorithm cannot be fully described until the expression analysis procedure is understood, so the description is deferred to the end of the next section.

4. Analysis of Facial Expressions

This section presents the procedures to extract muscle contractions from facial performances, and shows how the procedure can be used for improving the hand-generated actuation basis.

4.1. Extracting Muscle Contractions

We analyze the facial expressions by finding muscle contractions to reproduce optimally the marker trajectories generated by optical capture systems. We improved the algorithm proposed by Choe *et al.* [1].

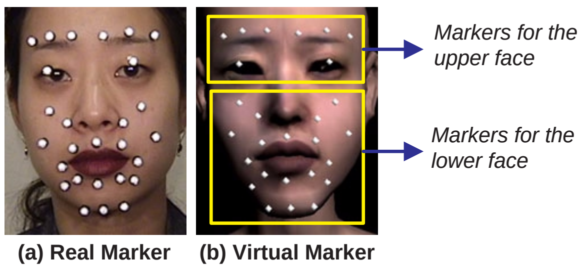


Figure 2. Real markers and corresponding virtual markers on the synthetic model.

4.1.1 Coordinate Alignment

While the 3D geometry of the synthetic face is resolved in its own local coordinate system $\{M\}$ (*model coordinate system*), the marker points in the performance data are resolved in another coordinate system $\{P\}$ (*performance coordinate system*). Before calculating the muscle contractions, we first have to transform the marker points from the performance coordinate system to the model coordinate system. We assume the transform from $\{P\}$ to $\{M\}$ is an affine (similarity) transform with scale s, rotation $\mathbf{R}$, and translation $\mathbf{t}$. We calculate the transform only once at the first frame of the performance data, and apply the same transform to all the other frames. In the following, we use the notation ${}^{Z}\mathbf{p}$ to denote the 3D coordinate of a point $\mathbf{p}$ resolved in the coordinate system $\{Z\}$

Let the position of the j-th marker be ${}^{P}\mathbf{p}_j$. We define the *virtual marker* ${}^{M}\mathbf{p}_j$ to be the corresponding point in $\{M\}$. Figure 2 shows the real and virtual markers. We want to find s, $\mathbf{R}$, and $\mathbf{t}$ that satisfy the following equations,

$$ {}^{M}\mathbf{p}_j = s\mathbf{R}\,{}^{P}\mathbf{p}_j + \mathbf{t}, \quad (j = 1, 2, \ldots, n) $$

where n is the number of markers. To initiate the computation, we first manually mark the virtual marker positions looking at both the first frame of the performance and the 3D model of the face. Then we solve the linear least square problems to find s, $\mathbf{t}$, and $\mathbf{R}$ sequentially, and repeat the procedure until the least square error is saturated.

The accuracy of s, $\mathbf{R}$, and $\mathbf{t}$ solved from the above procedure is at best limited to the accuracy of hand-marked position of the virtual markers. We note that, once we have s, $\mathbf{R}$, and $\mathbf{t}$, then the real markers can now be transformed to the model coordinate system. But the resulting points may not lie exactly on the surface of the 3D face model. By slightly adjusting the points along the normal directions, we can make the points lie on the surface, and get the next estimation of the (virtual) markers. We can further improve the accuracy of s, $\mathbf{R}$, and $\mathbf{t}$ by repeating the least square procedure with the new position of the markers.

The above assumes that the facial shape at the first frame of the performance is the same with the pre-modeled neutral face. Therefore we asked the actors to make a consistent neutral expression at the beginning of each performance capture.

4.1.2 Calculating Muscle Contractions

The final position of the virtual marker j in the above procedure will be a point within one of the triangles that constitute the facial geometry. Thus the marker point $\mathbf{p}_j$ can be encoded by the triangle index and the relative position within the triangle (barycentric coordinates), which do not depend on the subsequent deformation of the face. But the marker point will have different 3D position depending on the current shape of the face.

Let $\mathbf{d}_{ij}$ be the displacement of $\mathbf{p}_j$ at basis element $\mathbf{E}_i$ from $\mathbf{p}_j$ at neutral expression $\mathbf{E}_0$. From the synthesis equation (1), if muscle contractions x_i are given, the total displacement $\mathbf{d}_j$ of $\mathbf{p}_j$ is given by

$$\mathbf{d}_j = \sum_{i=1}^{m} x_i \mathbf{d}_{ij} \ .$$

We find the muscle contractions so that $\mathbf{d}_j$ is closest to the observed displacement $\hat{\mathbf{d}}_j$ from the performance by minimizing

$$\sum_{j=1}^{n} |\hat{\mathbf{d}}_j - \mathbf{d}_j|^2 = \sum_{j=1}^{n} |\hat{\mathbf{d}}_j - \sum_{i=1}^{m} x_i \mathbf{d}_{ij}|^2 .$$

Because the muscle contractions should be lie in $[0, 1]$, we can find the contractions by solving the following optimization problem:

$$\begin{array}{ll} \textbf{minimize} & \sum_{j=1}^{n} |\hat{\mathbf{d}}_j - \sum_{i=1}^{m} x_i \mathbf{d}_{ij}|^2 \\ \textbf{subject to} & 0 \le x_i \le 1 \ (i = 1, 2, \ldots, m) \end{array} . \qquad (2)$$

The muscle contraction vector $\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$ can be obtained by solving the constrained quadratic programming

$$\begin{array}{ll} \textbf{minimize} & \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{x}^T\mathbf{c} \\ \textbf{subject to} & 0 \le x_i \le 1 \ (i = 1, 2, \ldots, m) \end{array} , \qquad (3)$$

where

$$\mathbf{Q} = 2 \begin{pmatrix} \sum_{j=1}^{n} |\mathbf{d}_{1j}|^2 & \sum_{j=1}^{n} \mathbf{d}_{1j} \cdot \mathbf{d}_{2j} & \cdots & \sum_{j=1}^{n} \mathbf{d}_{1j} \cdot \mathbf{d}_{mj} \\ \sum_{j=1}^{n} \mathbf{d}_{2j} \cdot \mathbf{d}_{1j} & \sum_{j=1}^{n} |\mathbf{d}_{2j}|^2 & \cdots & \sum_{j=1}^{n} \mathbf{d}_{2j} \cdot \mathbf{d}_{mj} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{j=1}^{n} \mathbf{d}_{mj} \cdot \mathbf{d}_{1j} & \sum_{j=1}^{n} \mathbf{d}_{mj} \cdot \mathbf{d}_{2j} & \cdots & \sum_{j=1}^{n} |\mathbf{d}_{mj}|^2 \end{pmatrix} ,$$

$$\mathbf{c} = 2 \begin{pmatrix} \sum_{j=1}^{n} \hat{\mathbf{d}}_j \cdot \mathbf{d}_{1j} \\ \sum_{j=1}^{n} \hat{\mathbf{d}}_j \cdot \mathbf{d}_{2j} \\ \vdots \\ \sum_{j=1}^{n} \hat{\mathbf{d}}_j \cdot \mathbf{d}_{mj} \end{pmatrix} .$$

We solve this problem using the active set method, and apply Lagrange method for the sub-problems derived from the active sets [15]. To make the optimization procedure more robust, we divided the face into the upper and lower regions. The contractions of *Frontalis*, *Corrugator*, and *Orbicularis oculi* were calculated using only the markers in the upper region, and contractions of the other muscles were calculated using only the markers in the lower region (Figure 2). A muscle contraction value larger than one can be thought of as an exaggerated expression. So, we set only $x_i \ge 0$ as the constraints if we need to allow the exaggeration.

4.2. Improving the Actuation Basis

The actuation basis only used for expression synthesis can be entirely depend on the craftsmanship of the artist. However, the actuation basis for the subject being captured needs to have a certain level of accuracy to get reliable expression analysis results. It is not likely that hand-generated basis has such an accuracy. Therefore we develop an iterative algorithm that increases the accuracy of an actuation basis.

The algorithm takes the form of a fixed point iteration between **modeling** and **analysis**—the result of **modeling** is used for the **analysis**, and the result of **analysis** is in turn used for improving the actuation basis. For the iteration, we collect a performance data called *training data* in which the actor is asked to make all sorts of expressions. We let the actor fully contract each individual muscles. Even though ordinary people cannot make isolated muscle actuation, the facial expressions generated in the process of trying to use only a single muscle contain important information about the operation of the muscles, and helps to find more optimal basis elements. The training data also includes a significant amount of ordinary expressions that involve compound actuation of multiple muscles.

We first calculate muscle contractions at all frames of the training data by solving (3). Then the following equations should be satisfied in ideal cases for the marker point $\mathbf{p}_j$:

$$\begin{array}{c} x_1^{(1)}\mathbf{d}_{1j} + x_2^{(1)}\mathbf{d}_{2j} + x_3^{(1)}\mathbf{d}_{3j} + \cdots + x_m^{(1)}\mathbf{d}_{mj} = \hat{\mathbf{d}}_j^{(1)} \\ x_1^{(2)}\mathbf{d}_{1j} + x_2^{(2)}\mathbf{d}_{2j} + x_3^{(2)}\mathbf{d}_{3j} + \cdots + x_m^{(2)}\mathbf{d}_{mj} = \hat{\mathbf{d}}_j^{(2)} \\ \vdots \\ x_1^{(N)}\mathbf{d}_{1j} + x_2^{(N)}\mathbf{d}_{2j} + x_3^{(N)}\mathbf{d}_{3j} + \cdots + x_m^{(N)}\mathbf{d}_{mj} = \hat{\mathbf{d}}_j^{(N)}, \end{array}$$

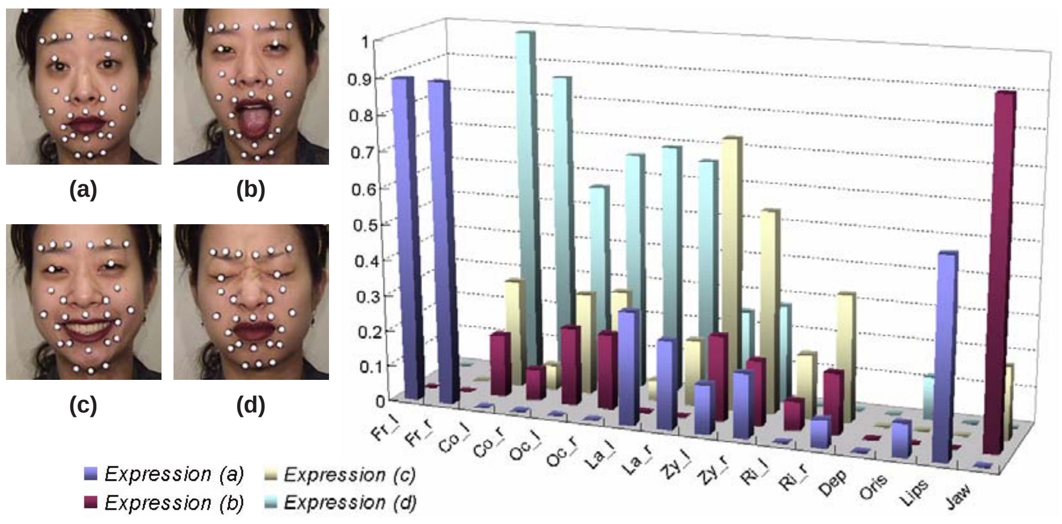


Figure 3. The four snapshots and plotting of the corresponding muscle contraction vectors

where $x_1^{(t)}, \ldots, x_m^{(t)}$ are analyzed muscle contractions at frame t, $\hat{\mathbf{d}}_j^{(t)}$ is the observed displacements at frame t, and N is the total number of frames in the training data. In reality, however, the equalities do not hold. But, if we solve the equations for $(\mathbf{d}_{1j}, \ldots, \mathbf{d}_{mj})$, the least square solution can provide us the improved position of the marker point $\mathbf{p}_j$ in each of the basis elements $\mathbf{E}_1, \ldots, \mathbf{E}_m$. If we perform the above steps for all the marker points $\mathbf{p}_j$ ($j = 1, \ldots, n$), we can get a new (improved) actuation basis.

Thus we get an improved actuation basis from the initial draft: (1) calculating muscle contractions from the initial draft, (2) finding new $\mathbf{d}_{ij}$ ($i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$) with the muscle contractions. We repeat the cycle until the *total analysis error* $\sum_t \sum_j |\hat{\mathbf{d}}_j^{(t)} - \mathbf{d}_j^{(t)}|^2$ is saturated, and finally get the optimized displacements $\bar{\mathbf{d}}_{ij}$ ($i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$). Finally, using the scattered data interpolation method with a radial basis function [16], we can form an optimized actuation basis from $\bar{\mathbf{d}}_{ij}$ and the initial draft of actuation basis.

5. Experiments

We implemented our algorithms on a PC platform. For the experiment, we also developed an optical capture system with five video cameras [1], which generated 3D trajectory of the markers and gross motion of the head at 30 frames per second. The experimental results (demo clips) described in this Section is available at `http://graphics.snu.ac.kr/research/basis/`.

First, we 3D-scanned the face of an actor, let an artist model the actuation basis of it, ran the improvement algorithm described in Section 4.2, and got the final actuation basis. Then we captured performances and analyzed them into muscle actuation values. Figure 3 shows four expression snapshots during a performance: (a) raising eyebrows, (b) jaw rotation, (c) smiling, and (d) frowning. The graph in the figure plots the muscle contractions at the snapshots which were analyzed by the algorithm described in Section 4.1. The result agrees well with our anticipation:

- **Expression (a):** The contractions (about 0.9) of the left and right *Frontalis* are dominant in raising eyebrows.
- **Expression (b):** We can see the jaw rotation is conspicuous.
- **Expression (c):** The two dominant contractions in the middle correspond to the left and right *Zygomatic majors*, which matches well with the muscle actuation in real smiling.
- **Expression (d):** We can see the six muscles are dominant in the last row: the pairs of *Corrugator*, *Orbicularis oculi*, and *Levator labii superioris*. The contraction of *Corrugator* and *Levator labii superioris* matches well with the muscle actuation in real frowning. *Orbicularis oculi* resulted from the close of the eyes are not directly related to this expression.

Figure 4 shows the result of applying the contractions of the expressions to the computer model of the actor and other two cartoon-like characters.

Figure 5 plots the contractions of left *Frontalis* and *Jaw rotation* during the performance of “Tulip Season” contained in the demo clip. The x-axis represents the frame

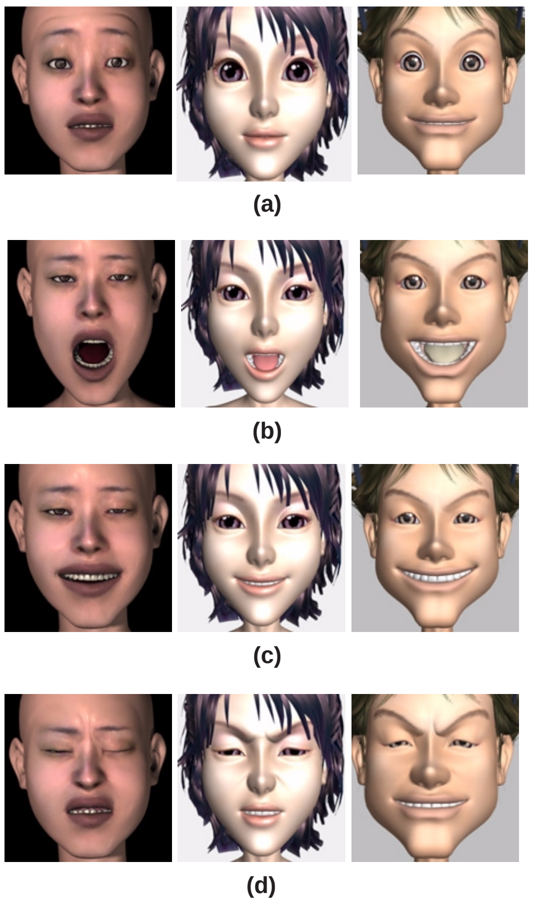


Figure 4. Results of applying the muscle contractions in Figure 3 to different 3D models.

number and the y-axis represents the muscle contractions in $[0, 1]$. The figure also shows *per-frame marker error* $(\sum_{j=1}^{n} |\hat{\mathbf{d}}_j - \mathbf{d}_j|)/n$, which is measured in centimeters. The error was computed separately for the upper and lower regions of the face. The error is bigger in the lower region due to the nonlinear and irregular movement around the mouth, which is mainly caused by *Orbicularis oris* muscle.

6. Conclusion

In this paper, we presented a new muscle-based facial animation technique that uses the actuation basis, a set of 3D facial shapes corresponding to the full actuation of individual muscles. Instead of completely relying on a mathematical method, we let artists manually sculpt (the initial draft of) the basis elements so that we could get more predictable deformation of the face. To increase the accuracy of the actuation basis, we developed an iterative algorithm that refined the actuation basis. Once an actuation basis was ready, a performance could be analyzed quite accurately, and the result could be applied to any 3D models with equivalent muscle structures.

An interesting contribution of this paper is that it proposed a technique that includes the artists' modeling capability as an integral part of the algorithm. The manual shaping of the basis elements complemented the pure mathematical approach which produced unexpected results occasionally. The proposed method is robust, and we believe that this artist-in-the-loop method can be quite useful in animation productions until the mathematical models can accurately simulate the operation of the muscles and concomitant movements in the facial tissue and skin.

Acknowledgment

This work was supported by the Brain Korea 21 Project. This work was also supported by ASRI (The Automation and Systems Research Institute) at Seoul National University. We thank Eunjeong Kang very much for modeling all the computer models and basis elements of them used in the experiments.

References

[1] B. Choe, H. Lee, and H.-S. Ko. Performance-driven muscle-based facial animation. *The Journal of Visualization and Computer Animation*, 12(2):67–79, May 2001.

[2] C. D. Clemente. *Anatomy: A Regional Atlas of the Human Body, 2nd edition*. Urban and Schwarzenberg, 1981.

[3] P. Eisert and B. Girod. Analyzing facial expression for virtual conferencing. *IEEE Computer Graphics & Applications*, 18(5):70–78, September - October 1998. ISSN 0272-1716.

[4] P. Ekman and W. V. Friesen. *Facial Action Coding System*. Consulting Psychologists Press, Inc., 1978.

[5] I. Essa, S. Basu, T. Darrell, and A. Pentland. Modeling, tracking and interactive animation of faces and heads using input from video. In *Proceedings of Computer Animation '96 Conference*, June 1996. Geneva, Switzerland.

[6] I. A. Essa and A. P. Pentland. Coding, analysis, interpretation and recognition of facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):757–763, July 1997.

[7] G. Faigin. *The Artist's Complete Guide to Facial Expression*. Watson-Guptill Publications, 1990.

[8] M. K. Greenwald, E. W. C. III, and P. J. Lang. Affective judgment and psychophysiological response: dimensional covariation in the evaluation of pictorial stimuli. *Journal of Pyschophysiology*, 3:51–64, 1989.

[9] B. Guenter, C. Grimm, D. Wood, H. Malvar, and F. Pighin. Making faces. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 55–66. ACM SIGGRAPH, Addison Wesley, July 1998.

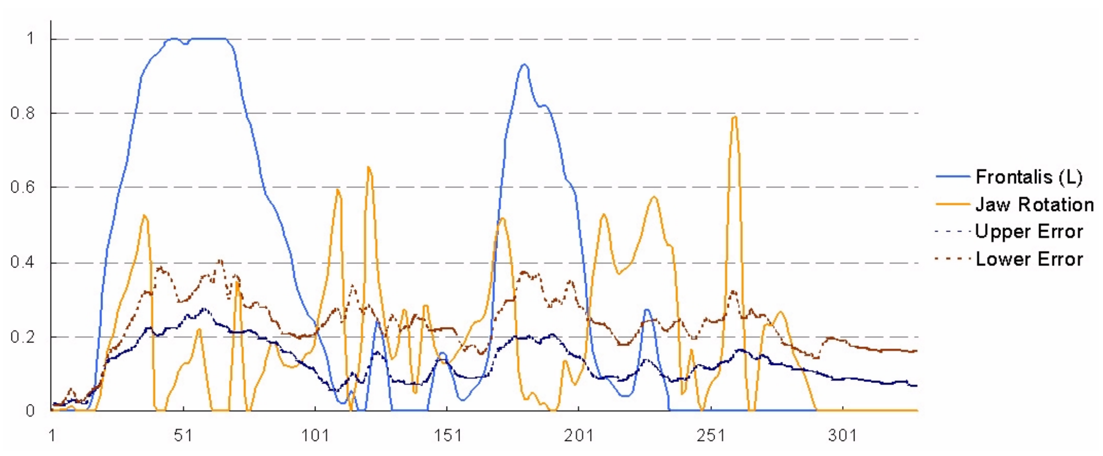


Figure 5. Contractions of ‘left frontalis’ / ‘jaw rotate’ and marker errors.

[10] R. M. Koch, M. H. Gross, and A. Bosshard. Emotion editing using finite elements. *Computer Graphics Forum*, 17(3):295–302, 1998. ISSN 1067-7055.

[11] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Büren, G. Fankhauser, and Y. Parish. Simulating facial surgery using finite element methods. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 421–428. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[12] C. Kouadio, P. Poulin, and P. Lachapelle. Real-time facial animation based upon a bank of 3D facial expressions. In *Proceedings of Computer Animation ‘98 Conference*. IEEE Computer Society Press, 1998.

[13] Y. Lee, D. Terzopoulos, and K. Waters. Realistic face modeling for animation. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 55–62. ACM SIGGRAPH, Addison Wesley, Aug. 1995.

[14] J. P. Lewis, M. Cordner, and N. Fong. Pose space deformations: A unified approach to shape interpolation a nd skeleton-driven deformation. *Proceedings of SIGGRAPH 2000*, pages 165–172, July 2000. ISBN 1-58113-208-5.

[15] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 2nd edition, 1984.

[16] G. M. Nielson. Scattered data modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, Jan. 1993.

[17] F. I. Parke and K. Waters. *Computer Facial Animation*. A K Peters, 1996. ISBN 1-56881-014-8.

[18] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. H. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 75–84. ACM SIGGRAPH, Addison Wesley, July 1998.

[19] F. Pighin, R. Szeliski, and D. H. Salesin. Resynthesizing facial animation through 3D model-based tracking. In *Seventh International Conference on Computer Vision (ICCV ’99) Conference Proceedings*, pages 143–150, September 1999. Corfu, Greece.

[20] D. Terzopoulos and K. Waters. Physically-based facial modelling, analysis, and animation. *The Journal of Visualization and Computer Animation*, 1:73–80, 1990.

[21] D. Terzopoulos and K. Waters. Analysis and synthesis of facial image sequences using physical and anatomical models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):569–579, June 1993.

[22] K. Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):17–24, July 1987. Held in Anaheim, California.

[23] L. Williams. Performance-driven facial animation. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):235–242, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas.

[24] J. yong Noh and U. Neumann. Expression cloning. *Proceedings of SIGGRAPH 2001*, pages 277–288, August 2001. ISBN 1-58113-292-1.

Universal Capture – Image-based Facial Animation for “The Matrix Reloaded”

George Borshukov, Dan Piponi, Oystein Larsen, J.P.Lewis, Christina Tempelaar-Lietz
ESC Entertainment

Introduction

The VFX R&D stage for *The Matrix Reloaded* was kicked off in January 2000 with the challenge to create realistic human faces. We believed that traditional facial animation approaches like muscle deformers or blend shapes would simply never work, both because of the richness of facial movement and because of the human viewer’s extreme sensitivity to facial nuances. Our task was further complicated as we had to recreate familiar actors such as Keanu Reeves and Lawrence Fishburne. Our team had been very successful at applying image-based techniques for photorealistic film set/location rendering, so we decided to approach the problem from the image-based side again. We wanted to produce a 3-d recording of the real actor's performance and be able to play it back from different angles and under different lighting conditions. Just as we can extract geometry, texture, or light from images, we are now able to extract movement. Universal Capture combines two powerful computer vision techniques: optical flow and photogrammetry.

HiDef Capture Setup

We used a carefully placed array of five synchronized cameras that captured the actor's performance in ambient lighting. For the best image quality we deployed a sophisticated arrangement of Sony/Panavision HDW-F900 cameras and computer workstations that captured the images in uncompressed digital format straight to hard disks at data rates close to 1G/sec.

Optical Flow + Photogrammetry

We use optical flow to track each pixel's motion over time in each camera view. The result of this process is then combined with a cyberscan model of a neutral expression of the actor and with photogrammetric reconstruction of the camera positions. The algorithm works by projecting a vertex of the model into each of the cameras and then tracking the motion of that vertex in 2-d using the optical flow where at each frame the 3-d position is estimated using triangulation. The result is an accurate reconstruction of the path of each vertex though 3-d space over time.

Keyshaping, Adapt, Removing Global Motion

Optical flow errors can accumulate over time, causing an undesirable drift in the 3-d reconstruction. To minimize the drift we make use of reverse optical flow. On this production the problem was eliminated by introducing a manual keyshaping step: when the flow error becomes unacceptably large the geometry is manually corrected and the correction is then algorithmically propagated to previous frames.

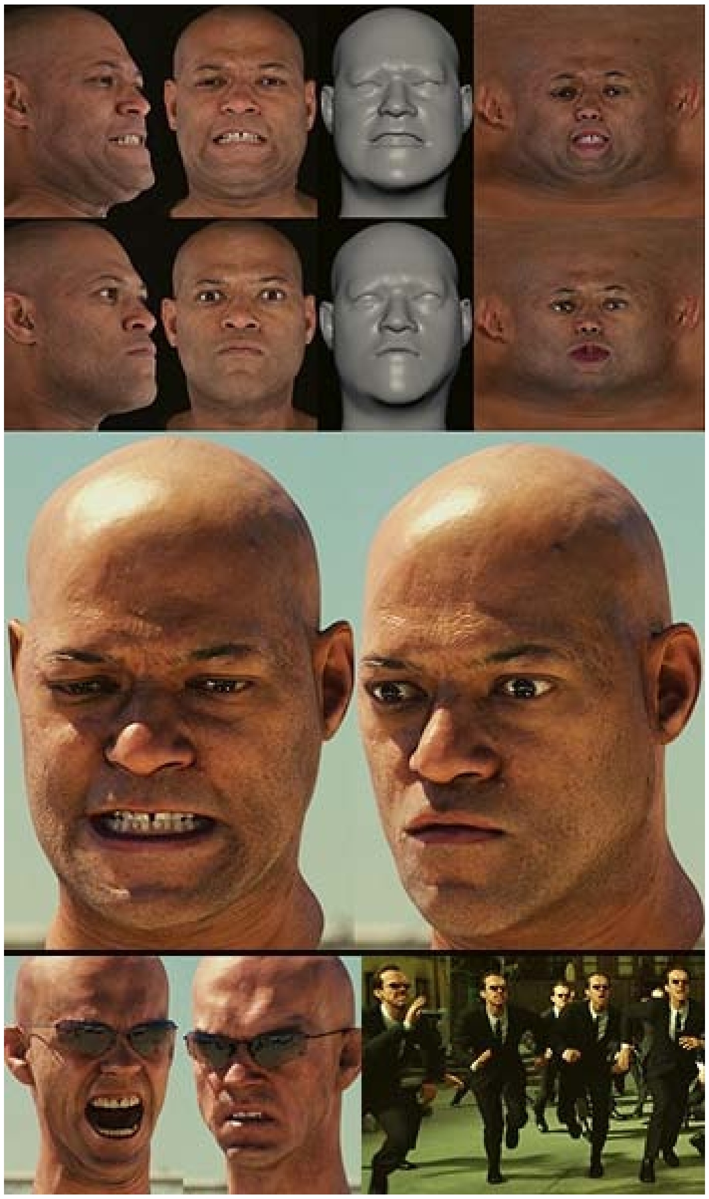
The reconstructed motion contains the global "rigid" head movement. In order to attach facial performances to CG bodies or blend between different performances this movement must be removed. We estimate the rigid transformation using a least squares fit of a neutral face and then subtract this motion to obtain the non-rigid deformation.

Texture Map Extraction

No believable facial rendering can be done without varying the face texture over time. The fact that we did not use any markers on the face to assist feature tracking gave us the important advantage that we could combine the images from the multiple camera views over time to produce animated seamless UV color maps capturing important textural variation across the face, such as the forming of fine wrinkles or changes in color due to strain, in high-res detail on each side of the face.

Rendering

Although the extracted facial animation had most of the motion nuances it lacked the small-scale surface detail like pores and wrinkles. We obtained that by using a highly detailed 100-micron scan of the actor’s face. The detail is then extracted in a bump (displacement) map. Dynamic wrinkles were identified by image processing on the texture maps; these are then isolated and layered over the static bump map. We then combine these with image-based skin BRDF estimation, subsurface scattering approximation, and real-world lighting reconstruction for the highly photorealistic human face renderings below.



Acknowledgments: Steve Avoujageli, Ken Faiman, Steve Rembuskos, Mike Dalzell, John Llewellyn, Ryan Schnizlein, Paul Ryan, John Jack, Kim Libreri, and John Gaeta

Generating Realistic Human Hair for "The Matrix Reloaded"

Tadao Mihashi, Christina Tempelaar-Lietz, George Borshukov
ESC Entertainment

Introduction

In recent years, there have been a few successful applications of realistic computer generated hair in the entertainment industry. For *The Matrix Reloaded* we had to face new challenges. First, we had to handle both close ups and scenes with hundreds of human characters with hair. The task was further complicated by the fact that the CG hair needed to exactly match the styling of familiar heroes from the film: Agent Smith and Neo. Also because of the rendering methods we chose for our virtual actors the hair solution needed to work in a ray tracing context with many lights. Our in-house hair styling tool together with some key rendering techniques made it all possible.

Styling Hair

During the early stages of hair R&D for *The Matrix Reloaded* we tried to use Maya Fur as our hair styling tool. Eventually we decided against it, as it did not provide us with enough freedom to match the detailed hairstyles of the principal actors. Alternatively, we developed an in-house hair styling tool called Gossamer.

Gossamer is a set of plugins and scripts for Alias|Wavefront Maya. It has all the parameters of Maya Fur, but also provides functionality that allows the user to directly move control points on "guide" hairs which are then used to generate many interpolated hairs. The interpolated hair is rendered in real-time using OpenGL in Maya (see Fig. 1). This allows artists to see the hairstyle as it would appear in the final rendered image while they are working. Gossamer also provides the ability to clump areas of hair for further hairstyle control. (We would like to acknowledge Eric Soulvie for his early development work on the Gossamer tool.)

Rendering Hair

Hair objects are generated procedurally during rendering time from a hairstyle description generated by Gossamer. In the beginning, each hair was generated as a very thin NURBS patch because originally mental ray, our renderer of choice, did not support curve rendering. This approach worked well only up to a certain level of complexity. We collaborated with mental images to implement a new fast and memory efficient "hair" primitive in the renderer.

A couple of more techniques are used to reduce the memory usage. One is to subdivide a hair object into partitions. Each partition is treated as a separate object with its own bounding box. If a partition is in the back of character's head and its bounding box is completely occluded by the head object, hairs in that partition will never get generated. The other one is using mental ray's geometry shader object instancing feature. It was very effective especially when rendering many Agent Smiths. Rendering hair for ten Smiths was virtually the same as rendering hair for one Smith.

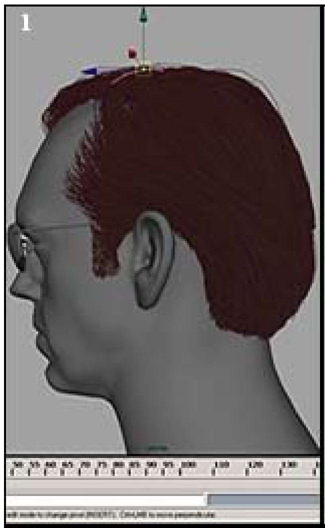
Hair Shadow and Lighting Reconstruction

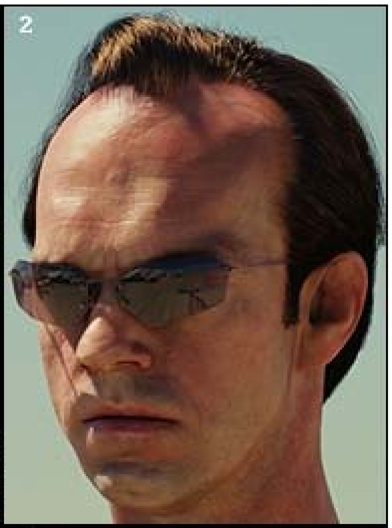
With either of two commonly used shadowing methods, shadow maps and ray traced shadows, it is difficult to produce nice, soft hair shadows. One proven solution to this problem is the deep shadow map, but our real world Lighting Reconstruction setups consisted of many light sources. Generating and maintaining a deep shadow map for each light would have been a pipeline nightmare. Instead, we use a volumetric approach which we call a shadow tree.

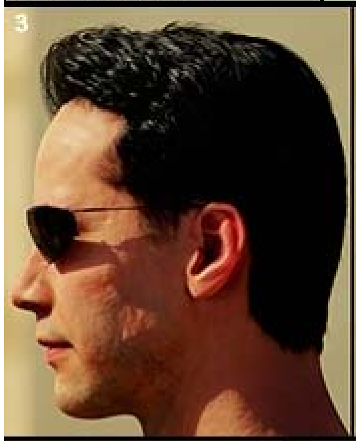
The shadow tree is an octree representation of the hair's density which is constructed during the initialization stage of the render. During rendering, the shadow tree is only visible to shadow rays. When a shadow ray intersects the shadow tree's bounding volume, it reads the density value of each intersected voxel and uses these values to attenuate the light accordingly.

Results

Below we show the results of our techniques. Fig. 2 and 3 show full CG renderings for Agent Smith and Neo. Fig. 4 is a photograph, which demonstrates how close we can match the real hairstyle and look. The image on the bottom is a frame from the film. All hair but the one in the middle is computer generated for this 3-d head replacement shot.







Lighting Reconstruction for “The Matrix Reloaded” (**sketches_0314)**

Haarm-Pieter Duiker*
ESC Entertainment

Introduction

The demands of photo-realism required of the effects for *The Matrix Reloaded* and *The Matrix Revolutions* led us to create a system for directly and accurately reconstructing real world lighting environments. The Lighting Reconstruction Toolkit builds on research in the area of Image-based Lighting and extends current techniques to enable the reconstruction of lighting that more closely matches the real world.

Lighting Reconstruction

Lighting Reconstruction has two main stages: **Data Acquisition,** and **Lighting Reconstruction**.

On-set photography is the primary means of real world lighting **Data Acquisition**. Of the on-set photography, multiply exposed images of chrome balls play a central roll. Kodak 20-step grey cards and Macbeth color charts play a not as obvious but also central roll. They are photographed for film response and color neutrality reference. Color temperature, exposure settings, photogrammetric reference images, and set diagrams round out the lot of information gathered.

The images and data are used to create from each group of chrome ball exposures an extended, or high, dynamic range panorama, i.e. an image representing the true color and intensity of all the lighting at a certain point in the real world. The chrome ball images serve as the raw data. The Kodak 20-step grey scale images serve as the basis for solving for the response function of the film, the key to turning film exposure measurements back in to true measurements of intensity.

Given a high dynamic range panorama, the **Lighting Reconstruction**, of which there are currently three main modes, can begin.

In the first and simplest mode, a panorama directly drives the creation of a set of directional lights. The lights represent the integrated color and intensity of either key or fill areas of the panorama. The key lights are separated out from the fill light portion of the panorama. Roughly equal solid angle sub-regions are integrated over to create the fill lights to a user-specified granularity. These lights will not change in any way with respect to the location of the object.

In the second mode, multiple panoramas, their real world locations, and a model of the environment with light source locations allow us to create lighting that varies as the key and fill lighting varied in the real world. The input key light locations and sizes are verified by triangulating their position from the panoramas. The intensity, color, cone angle, and other parameters of a spot light model are then fit to the data in each panorama. The intensity of the fill lighting is established by creating a triangulation of the panorama locations. At render time, for each object, an interpolation weight for each of the panoramas is established based on the proximity of the object to each panorama as well as the proximity of each panorama to the others. An object specific panorama is generated using the weights. From this panorama an arbitrary number of directional lights may be generated.

The third mode of Lighting Reconstruction is based on a new light primitive whose area is dynamically subdivided for each object it will light. The light fulfills the criteria that none of its sub-regions may be bigger than a solid angle threshold relative to an object being lit. The primitive when used in the creation of a simple model of the real world geometry and textured using the panoramas projected on to the model, replaces the panorama-interpolating scheme presented in the previous mode. The primitive allows for better allocation of a limited number of light samples and a better reconstruction of the object specific lighting environments.

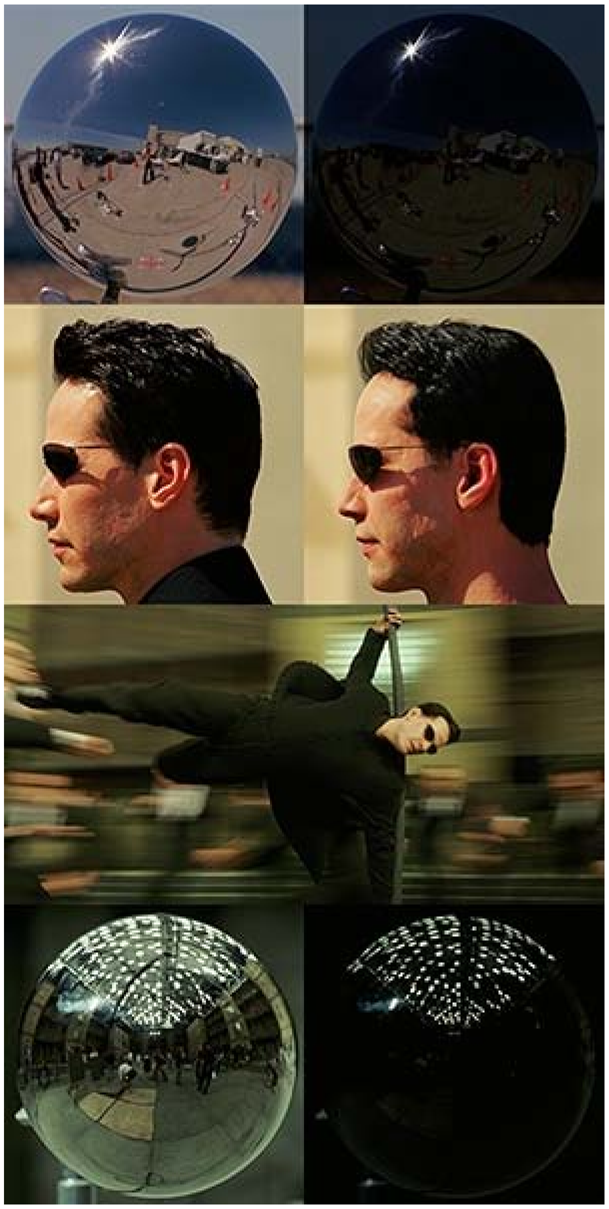


Figure 1: Row 1 shows two differently exposed images of the same chrome ball. Row 2 shows a real image of Keanu Reeves (left) and a version of Keanu Reeves (right) rendered using the lighting from above chrome ball. Row 3 shows an image from the Burly Brawl sequence. Row 4 shows two differently exposed images of one of the chrome balls used to generate the lighting for that sequence.

Conclusion

The Lighting Reconstruction Toolkit has survived a true production environment and was combined with the Universal Capture, Skin, BRDF, Subsurface Scattering, Hair, and Cloth Simulation development efforts at ESC to create many of the realistic images seen in the movie.

----
*e-mail: hp@escfx.com

Making of *The Superpunch*

Presented at Imagina'04 by
George Borshukov, ESC Entertainment

Introduction

The Superpunch is the final punch Neo that delivers to Agent Smith in the final installment of *The Matrix* trilogy during the film's last face-off. It was the first shot that the directors Larry and Andy Wachowski and their conceptual designer storyboarded for the sequels. (The shot's storyboards were also the first storyboards to leak out on the Internet back in Spring 2000). *The Superpunch* was meant to show, in familiar Bullet Time, the event of Neo's super-powerful last punch that occurs over a fraction of a second and lands on Smith's face deforming it in a surreal, anime-like fashion. As a Bullet Time shot, it was to feature an impossible virtual camera yet it had to look real. The nature of the push-in camera move and the requirement of punching someone with an inhumanly strong force meant that the original multiple still camera rig approach could not be deployed. The shot went through many different stages of visualizations, breakdowns, and considerations. *The Superpunch* became even more challenging when we learned that it had to happen under heavy rain. We explored creating the shot based on live action elements and augmenting it with CG elements. Tests revealed that this approach would compromise the required fluid camera. The camera was meant to glide through space showing us an exciting event from previously unseen perspectives. Around the Spring of 2003 when we had gained confidence that our 3 year-long R&D effort in realistic human face rendering technology could pull off a full-frame, slow-motion close up of a familiar actor such as Hugo Weaving, we decided to create the shot entirely in the CG world.

Study of Target Facial Deformation

One of the main points of discussion and study was what Agent Smith's face was supposed to look like during the impact. The storyboards depicted a surreal, exaggerated, caricature-like extreme facial deformation. Our team collaborated with the directors to study the desired target shape by "abusing" Hugo Weaving's face – pushing fists into it, blowing high speed air-nozzle streams at it, etc. These studies resulted in the construction of a practical maquette representing the directors' desired extreme facial deformation. This shape was well more distorted than a real actor's face could get even under the most extreme punch.

Facial Animation

We briefly considered doing a blend-shape like approach to go from a pre-punch pose to the maquette extreme, but we felt that this would not give us realistic results. We had spent 3 years developing a powerful and successful image-based facial animation technique called Universal Capture. This technique allowed us to process a marker-less multiple angle HD footage of an actor's performance and produce a 3-D recording which could be displayed from different angles and in new lighting conditions. By re-projecting the original images, the process allowed us to vary the color texture on the face over time which was critical for the realism of our results. In the case of *The Superpunch* we felt that although we couldn't capture directly what we wanted, we could still make use of a powerful performance from Hugo Weaving and manipulate it to resemble the maquette. We selected a performance, processed it and then had an incredibly gifted artist create the additional facial deformations by hand using a set of cluster, wire and proprietary deformers in Maya. It was the ultimate marriage of technology and artistry. The directors were closely involved in the process of layering the additional facial deformations and ripples. In the end, they slightly revised their vision of the target expression to be less surreal.

Water Elements

The creation of water elements was a very challenging task due to the large amount of water and its required interaction with the virtual actors. We deployed a variety of techniques to construct the various elements. For the static rod-like water drops suspended in the air during the shot (inspired by the storyboards and having no basis in physical reality) we used a set of 30 hand-modeled basic shapes randomly instanced throughout space. After the fists impact in the beginning of the shot, the water splashes out creating a barn door like effect. This effect was achieved with a combination of hand modeled, placed, and animated shapes and instanced raindrop-like objects with motion derived from particle simulations. For the water on and around Neo's fist and sleeve, we used particle simulations and a custom implicit blobby surface plug-in to construct meshes from the particles. The same approach was also used for the water on Smith's face that flies off at fist impact. The spit coming out of Smith's mouth was modeled and animated by hand. Once again the simulation (movement and timing) of these elements was not physically correct, but rather entirely driven by artistic vision and direction.

Realistic Appearance and Rendering

Even though the shot's animation was not based on physics, the appearance had to be - the shot didn't have to feel real, but needed to look real! This required us to rely on a full ray-tracing solution for shadows, reflections, refractions, caustics, including accurate simulation of 3-D depth of field. For the realistic skin surface detail, we had access to 100 micron scans of plaster casts of the actors' faces and fists. We converted this information into displacement and bump maps. For the realistic skin appearance, we relied on modifying our image-based skin reflectance estimation by adding a wet layer in the shaders. We also had an efficient and realistic subsurface-scattering approximation without which the skin would have looked like granite. For the clothing, we added a wet layer on top of the reflectance data measured from dry costume samples. We also extended our in-house hair tools to support wet hair styling and appearance which were matched to live-action reference. For added realism of the skin around the impact areas, we created animated vein and knuckle imprint maps that allowed us to manipulate the color and normals of the skin surfaces. The beauty lighting was reconstructed from high dynamic range chrome ball images of the on-stage lighting set-up for the surrounding live action shots. The lightning timing and direction was designed and tweaked to highlight and complement important moments within the shot. The background, which responds to the lighting and is also being reflected and refracted by all water elements, was constructed using our image-based virtual background pipeline from a lidar scan and photographs of the actual crater built on-stage for live action photography. For photorealism and physically accurate appearance, all elements were rendered together in mental ray with full ray-tracing including proper 3-D depth of field simulation.

Conclusion

The Superpunch was considered the most difficult shot in *The Matrix* sequels for many reasons, most notably because of the challenge of showing a full frame computer-generated face of a known human actor. We believe that we met this challenge and also had an wonderful experience putting the shot together with an approach that combined state-of-the-art technology with exceptionally fine artistry and attention to detail. Most importantly, we felt that were able to realize the directors' fantastic vision and assert the storytelling power of visual effects.

Credits

George Borshukov - Shot Supervisor / Technology Supervisor
Kody Sabourin - Shot Design/Animation Lead, Facial Deformation
Masuo Suzuki - Wet Appearance and Water Look Development
Oystein Larsen - Water Effects Animation
Tadao Mihashi - Rendering Lead
Ken Faiman - Universal Capture Performance Processing
Scot Schinderman - Water Effects Software
Oliver James - 3-D Depth of Field Shader
Rene Garcia - Texture Paint
Christina Tempelaar-Lietz - Wet Hair Software
Matt McDonald - Compositing
Carina Ohlund - Virtual Background Construction
Tristan Ikuta, Nathan Fok - Cloth Simulation and Modeling
Brian Freisinger - Head Model Surfacing
John Jack - Producer
Kim Libreri - VFX Supervisor
John Gaeta - Senior VFX Supervisor
Geofrey Darrow, Steve Skroce – Conceptual Design
Larry & Andy Wachowski – Directors

Results

See next page.