Interactive Animation of the Eye Region

by

João Afonso Libório Cardoso

B.Sc., University of Coimbra, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

September 2016

Abstract

Humans are extremely sensitive to facial realism and spend a surprisingly amount of time focusing their attention on other people's faces. Thus, believable human character animation requires realistic facial performance. Various techniques have been developed to capture highly detailed actor performance or to help drive facial animation. However, the eye region remains a largely unexplored field and automatic animation of this region is still an open problem. We tackle two different aspects of automatically generating facial features, aiming to recreate the small intricacies of the eye region in real-time.

First, we present a system for real-time animation of eyes that can be interactively controlled using a small number of animation parameters, including gaze. These parameters can be obtained using traditional animation curves, measured from an actors performance using off-the-shelf eye tracking methods, or estimated from the scene observed by the character using behavioral models of human vision. We present a model of eye movement, that includes not only movement of the globes, but also of the eyelids and other soft tissues in the eye region. To our knowledge this is the first system for real-time animation of soft tissue movement around the eyes based on gaze input.

Second, we present a method for real-time generation of distance fields for any mesh in screen space. This method does not depend on object complexity or shape, being only constrained by the intended field resolution. We procedurally generate lacrimal lakes on a human character using the generated distance field as input. We present different sampling algorithms for surface exploration and distance estimation, and compare their performance. To our knowledge this is the first method for real-time or screen space generation of distance fields.

Preface

Versions of Chapter 3 have been published in the following:

- Debanga R. Neog, João L. Cardoso, Anurag Ranjan, and Dinesh K. Pai. Interactive gaze driven animation of the eye region. In Proceedings of the 21st International Conference on Web3D Technology, Web3D '16, 2016
- Debanga Raj Neog, Anurag Ranjan, João L Cardoso, and Dinesh K Pai. Gaze driven animation of eyes. In Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2015

The ideas described in Chapter 3 are part of the *EyeMove* project, started before I joined the U.B.C. Department of Computer Science. The *EyeMove* project was funded in part by grants from NSERC, Peter Wall Institute for Advanced Studies, Canada Foundation for Innovation, and the Canada Research Chairs Program. This project was supervised by Prof. Dinesh K. Pai, and worked on by myself, Anurag Rajan and Debanga Raj Neog. We worked independently on different parts of the project according to our fields of competence. I focused on the real-time aspect of the project, while Anurag Rajan and Debanga Neog focused on capturing real world data and training generative models. Debanga Neog and I collaborated intensively in the intersection of these two realms of the project.

With that in mind, Sections 3.4 and 3.5 contextualize the remainder of the chapter by briefly describing work developed entirely by the remainder of the group. I was responsible for the majority of the implementation, with Section 3.6 being mostly implemented by Debanga Neog. Regarding writing, Sections 3.1, 3.2 and 3.3 are based on the aforementioned publications, where the corresponding writing had been done mainly by Prof. Dinesh Pai.

Chapter 4 describes work that has never been submitted for publication. I alone worked on this chapter, which was reviewed by Prof. Dinesh Pai.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First of all, I would like to thank those who have worked with me during my graduate studies. Special thanks to my supervisor, Dr. Dinesh K. Pai, and Debanga Raj Neog, both of which accompanied me throughout the whole journey. Your constructive criticism was always invaluable. I'd like to thank Dr. Dinesh K. Pai for teaching me about research process and scientific writing. Debanga Neog, for all the times we brainstormed and chatted together. I would also like to thank Anurag Rajan, whom I worked with, even only for a brief period of time, and Paul Debevec from the WikiHuman Project, for providing the Digital Emily model used in our examples.

This goes out to my laboratory: thank you for having me around. To Prashant, for always being open to share his experience. To Cole, for putting up with our hardware requests. To Darcy, for your most valuable comments to my presentations in the laboratory meetings.

To Dr. Paul Kry, for having introduced me to the world of research and computer animation, inviting me to your lab and putting up with me even when I was just starting my undergraduate studies. I would never have pursued graduate studies in Canada if it was not for you. To Dr. Derek Nowrouzezahrai, for introducing me to physically based rendering and letting me study on your lab.

Lastly, I'd like to thank my family and friends. To all my longtime friends back in Europe and to the more recent ones in Vancouver. But especially to my parents, for without them I'd not be the same. Thank you for visiting me so often, even with me on the other side of the world. Thank you for all the guidance and insufferable nagging. Finally, to my sister, for by growing you always remind me of how long I've been gone.

Chapter 1

Introduction

Humans communicate by a mixture of body language, voice and facial expressions. Still, the most important communication visual aid is arguably the eyes: humans spend a surprisingly large amount of time looking at people's eyes, as shown by the work of Alfred Yarbus (1967). The eyes are extremely important because they communicate information regarding a person's mental and physical state (e.g., attention, intention, emotion, health, fatigue). Hence humans are extremely perceptive to the region surrounding the eyes and all the small details it conveys.

Nowadays, computers can generate very realistic environments and highly detailed characters. Yet, human facial animation is still not on par with the remaining character animation. We are very sensitive to human realism, especially faces, and hence even small discrepancies in the appearance are glaringly obvious to us. These small discrepancies are quite worrisome, as they can cause a response of revulsion from observers - a phenomenon known as the uncanny valley, first described by Masahiro Mori et al. (2012) and coined by Jasia Reichardt (1978). This is a problem that the film and interactive media industries frequently struggle with. For example, the film Final Fantasy: The Spirits Within received negative reactions due to its near photo-realistic yet imperfect visual depictions of human characters (Eveleth, 2013). The humans in The Polar Express were heavily criticized by reviewers. Anderson (2004) described them as "creepy and dead-eyed (...) zombies".

These discrepancies are consistently found in the ocular region. We define the ocular region as the combination of the eye globes, the upper and lower eyelids, the eyelashes, the canthus, the conjunctiva, the tear film and the periorbital soft tissues that surround the orbit. Besides the "dead eye" look caused by imperfect shading andor animation of the globes, the ocular region is full of small features which, if not animated properly, will lead into uncanniness.

In the past few years, there have been some cases of exceptional digital doubles created for film: a famous example is Paul Walker's double for the movie Fast and Furious 7, which was made when the actor died before the movie was completed. Letteri (2015) explained that the features and performance were reconstructed by hand from old footage. The realism was further enhanced by blending the character performance with the real footage. Even still, 10% of the shots were made by imposing existing footage of the actor. Ed Ulbrich (2009), talking of the film The Strange Case of Benjamin Button, described the human head as the "holy grail of the film industry". Although his team managed to create a remarkable digital character, it took intensive actor performance recording and facial scan, per expression manual tweaking and 155 people over two years to achieve it. For example, an artist worked exclusively on the ocular region over two years. Automatizing this process remains a challenging problem. This is especially troublesome in interactive media, were the performance limitations are much more severe and actor performance cannot be directly used to animate characters, neither can each single possible shot be handcrafted by an artist.

1.1 Overview of Contributions

As previously discussed, there is a very high time investment associated with animating all the small intricacies of the ocular region by hand. While research on modeling gaze - and hence globe motion - is quite mature, research on automatically animating the other components of the ocular region is almost non-existent. In this dissertation I explore models for interactive animation of some of these components in real-time based on real-world data or biophysical models. In particular, I describe:

- A system for real-time animation of the eyelids, canthus and periorbital soft tissues that can be interactively controlled using a small number of animation parameters, including gaze.
- A system for real-time computation of distance fields in screen-space of any arbitrary geometry that does not depend on scene complexity.

From these two systems, I highlight as my personal major contributions:

- A real-time algorithm for skin sliding over the globes, which is able to recreate corneal displacement and reconstruct fine details from simplified skin representations.

- A model for intuitive interactive control of the factors that affect skin movement in the eye region.
- Three sampling algorithms for real-time generation of distance fields in screen-space.
- A proof of concept algorithm for procedural lacrimal lake generation based on distance to skin.

Chapter 2

Related Work

The synthesis of a realistic ocular region requires an accurate modeling, animation and rendering of its underlying structure. In this section, I discuss background knowledge and relevant related work regarding each of these three topics.

2.1 Anatomy Background

2.1.1 Structure

According to Ruhland et al. (2014), the "eye globe is one of the most complex organs in the human body", with multiple layers, each one devoted to perform a specific task. The transparent cornea, located in front of the eye, is the first refractive surface to the light entering the eye. The tear film moistens and softens the cornea's surface to minimize distortion and scattering of light. The cornea is embedded in the opaque white sclera and separated by the limbus. The sclera tissue preserves the shape of the eye and protects against substances and pathogenic elements. The light passes through the pupil and is focused by the lens behind it onto the retina. The primary function of the iris is to regulate the amount of light that reaches the retina as as a function of the prevailing light conditions. The retina forms the inner, light sensitive part of the eye. The light captured at the retina is processed and transmitted as electrical signals to the brain for processing.

The pupil shape and diameter changes with the contraction of the iris muscles. The lens shape changes with the contraction of the ciliary muscle, thereby increasing the optical power and accommodating the projection of nearby objects on the retina.

External to the globe, the medial canthus and lateral canthus are, respectively, the inner and outer corners of the eye where the upper and lower eyelids meet (Dorland, 1980). The lacrimal apparatus is the physiological system containing the structures for tear production and drainage (Gray,

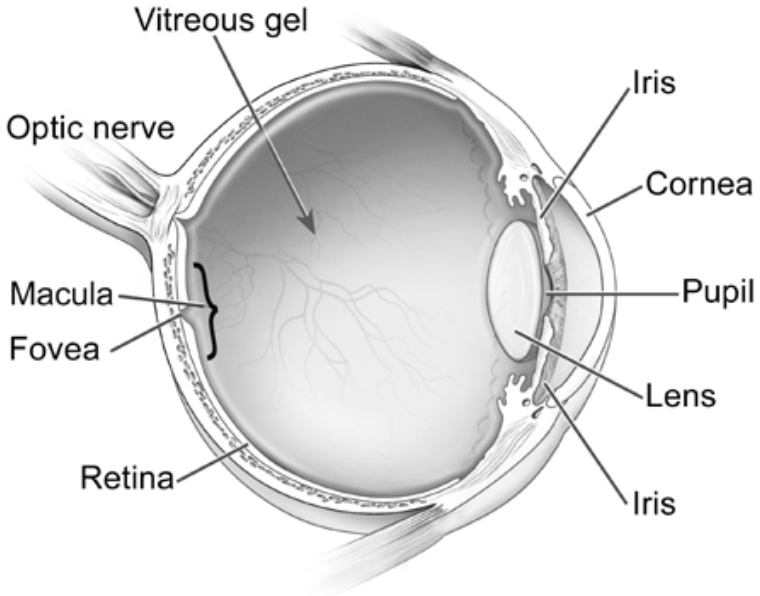


Figure 2.1: Structure of the eye globe. Reproduced from National Eye Institute (2016).

2009).The lacrimal gland, close to the lateral canthus, secretes the tears into the globe. The conjunctiva, located inside of the eyelids and covering the sclera, helps lubricate the eye by producing mucus and a smaller volume of tears. The nasolacrimal duct, near the medial canthus, drains the fluid into the cavity of the nose.

2.1.2 Movement

It is not commonly appreciated that the eyes have full 3 degrees of freedom, even though to point the optical axis at a target only requires 2 degrees of freedom. Rotations about the optical axis, called "torsion", have been known and modeled at least since the 19th century. The six separate muscles that control eye globe motion, the extra-ocular muscles, are a surprisingly complex muscular system that allows the globe to perform a wide repertoire of movements (Leigh and Zee, 2015). While complex, these movements have been extensively studied by neurologists, psychologists and neuroscientists (Ruhland et al., 2014). Therefore, both their characteristics and the condi-

tions in which they occur are well known and extremely well documented.

"Saccades", arguably the most noticeable type of globe movement, are the rapid shifts in globe rotation that focus the gaze on targets of interest. They are characterized by extremely rapid initial acceleration and final deceleration (Becker, 1989). Humans make an average of three saccades every second, and slow movements called "smooth pursuit" to track small moving targets. The "vestibulo-ocular reflex" is responsible for stabilizing the eyes during head motion. It occurs with extremely short latency and hence it can be considered as effectively simultaneous with head movement (Leigh and Zee, 2015). The smooth pursuit system serves to stabilize moving images on the retina. It has an higher latency than the vestibulo-ocular reflex but lower than saccades. Unlike them, smooth pursuit is more situational and consequently not as common. Vergence occurs when a target lies near the visual mid-line. It consists of a convergence of the globes rotations to maintain a single binocular vision. Vergence movements are far slower than saccades.

The two most studied lid movement types are blinks and lid saccades. There are three types of blinks: spontaneous, voluntary, and reflexive. While there is an high variation in blink rates, their frequency has been subject of a wide variety of studies. They have been linked to cognitive state and activity (Skotte et al., 2007; Stern et al., 1984), fatigue (Anderson et al., 2010; Johns et al., 2007), lying (Burgoon et al., 2016), speech production (Nakano and Kitazawa, 2010) and others. Blinks are characterized by a quick down-phase, which causes an almost complete closure of the lids, followed by an approximately twice as slow up-phase. Lid saccades, on the other hand, do not exhibit as much marked asymmetry between down and up-phases as blinks do.

2.2 Geometry Reconstruction

Due to the aforementioned complexity of the ocular region, modeling it manually is no simple task. Consequently the region is commonly grossly simplified. Eyes are traditionally modeled as spherical shapes and high resolution pictures of human eyes are used as texture maps (Itti et al., 2004; Weissenfeld et al., 2010). According to Bérard et al. (2014), these typically generic eye models are insufficient for capturing the individual identity of a real human.

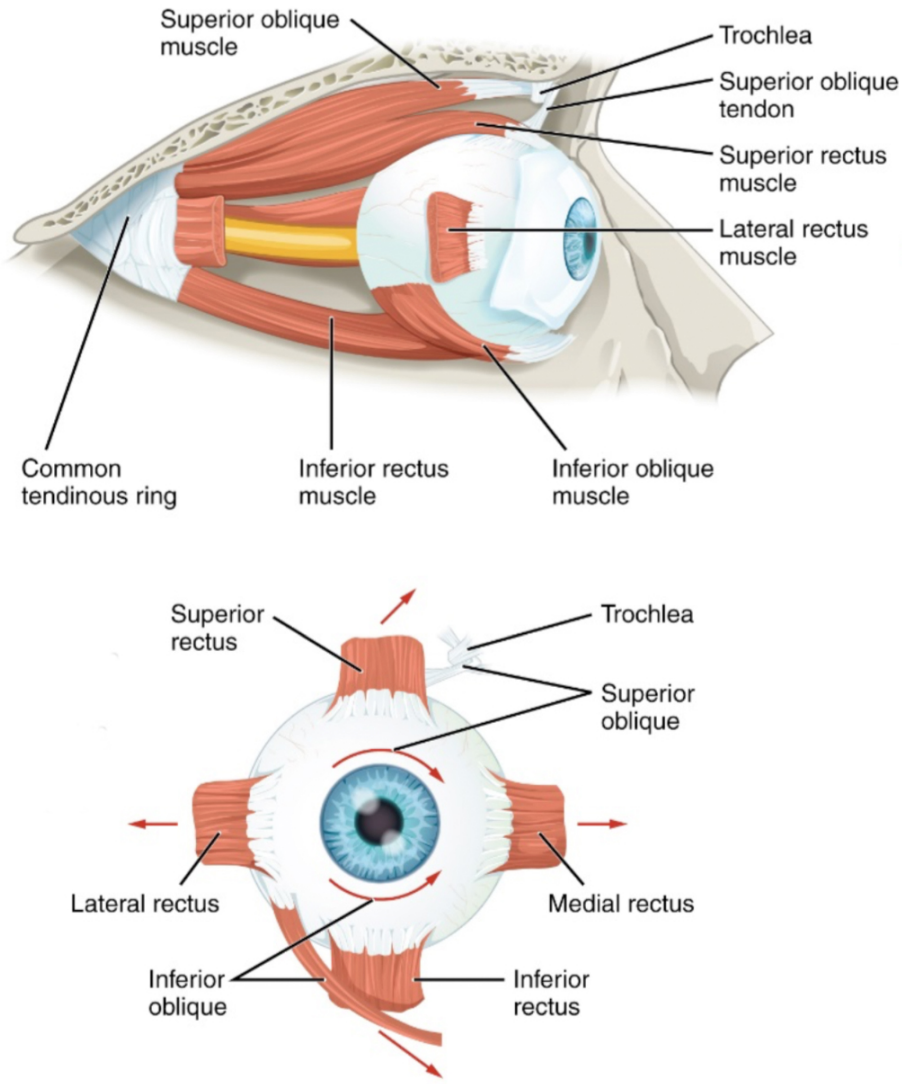


Figure 2.2: Anatomy of the extraocular muscles. Reproduced from OpenStax (2013).

Some methods that produce more anatomically correct models of the eye, particularly the iris, have been proposed. Sagar et al. (1994) presents a model for surgery simulation, where the iris is represented as two layers (ciliary and pupillary) of fibers with opposite curvatures. A Gaussian perturbation is used on the ciliary fibers as the pupil dilates. Retinal blood vessels are generated on a spherical sclera using fractal trees (Oppenheimer, 1986). Lefohn et al. (2003) is able to synthesize human eyes, more noticeably the iris, using knowledge from the field of ocular prosthetics. Eyes are synthesized by stacking multiple layers of dots, radial smears or radial spokes, in a similar fashion to what ocularists do. François et al. (2007) presents a method to recover the iris structure and scattering features from a single photograph, taking into account reflection and refraction at the corneal surface based on the ambient light.

Moriyama et al. (2006) presents a different take on the problem. It suggests a parametrization of the individuality of the ocular region as a simplified set of features, such as iris size and eyelid skewness. It is then capable of identifying such parameters from real world images while tracking gaze and eyelid opening across multiple frames. This data can be used to reconstruct simplified geometry.

More recently, there have been major improvements on capture and reconstruction of the ocular region. Notably, Bérard et al. (2014) is able to capture and reproduce all the intricacies of a subject's sclera, cornea and iris. Bermano et al. (2015) captures and outputs time-varying high-resolution eyelids and is able to reproduce the the eyelid fold, even under complex deformation, folding and strong self-occlusion.

Considerable research in the past decade has been focused around facial simulation and performance capture. Physically based deformable models for facial modeling and reconstruction include the seminal work of Terzopoulos and Waters (1990). Synthesis of high definition textures using a generative Bayesian model has been discussed in Tsiminaki et al. (2014).

The majority of recent work has been focused on data driven methods. Some state-of-the-art methods focus on obtaining realism based on multi-view stereo (Beeler et al., 2011; Bickel et al., 2007; Furukawa and Ponce, 2010; Ghosh et al., 2011; Wu et al., 2011). This data can be used to drive blendshapes (Fyffe et al., 2013). Some of the work is based on binocular

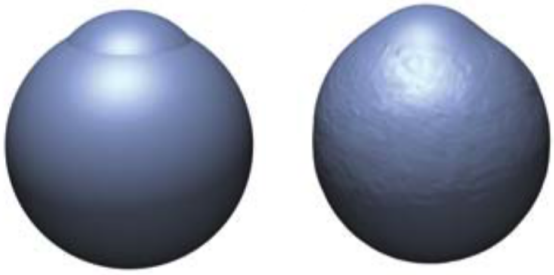


Figure 2.3: Left: generic spherical eye representation. Right: higher order approximation of an individual eye, as presented in Bérard et al. (2014).

(Valgaerts et al., 2012) and monocular (Garrido et al., 2013; Shi et al., 2014) videos. Recent work by Li et al. (2013b) described a system for real-time and calibration free performance driven animation.

Several methods are specifically targeted towards robust tracking of facial movements. Some of the recent work includes using *Active Appearance models* (Koterba et al., 2005; Xiao et al., 2004) and physically based methods (Decarlo and Metaxas, 2000). Much of facial tracking includes sparse tracking of different features across the face in a learning based approach. The sparse tracking methods such as feature based tracking and AAMs fails to capture the detailed and complex motion of skin especially around the eyes.

2.3 Interactive Animation

Most of the attention on eyes has been on modeling gaze, especially the gaze behavior of a character in a virtual environment. Bahill et al. (1975) correlates the magnitude of saccades with some of their proprieties, such as duration and peak velocity, by looking into experimental data. Harwood et al. (1999) presents a model for the shape of saccade trajectories as a function of their duration. Blohm et al. (2006) proposes a model of the smooth pursuit system, which makes use both of saccadic and smooth movements to track moving objects. Yeo et al. (2012) implement a smooth pursuit system to animate human characters performing fast visually guided tasks.

Eye blinks are also a widely studied phenomenon. Flash and Hogan

(1985) describes the web known bell shape profile of muscular propelled motion. Evinger et al. (1991) describes the relation of blink amplitude to blink maximum velocity and phase duration. Trutoiu et al. (2011) tracks inter-eyelid distance on test subjects using an hight-speed video camera. It then studies blink duration and eyelid closure profile variation across different subjects. Ishimaru et al. (2014) analyzes head motion patterns and blink frequency associated with performing different activities. It then is able to recognize activities with some success given the head motion and blink frequency.

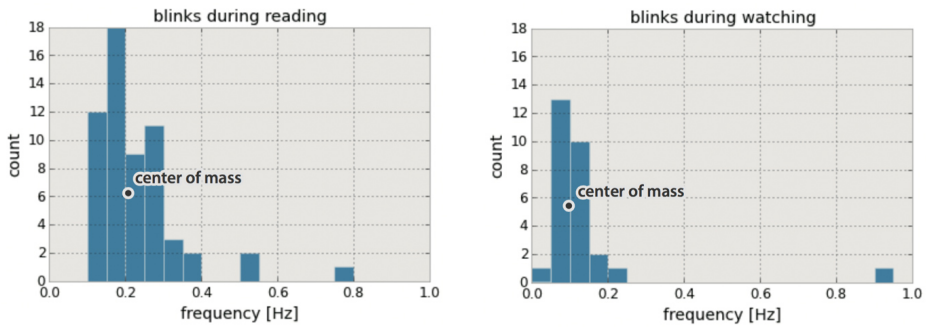


Figure 2.4: Blink frequency when performing different tasks. Reproduced from Ishimaru et al. (2014).

The dynamic behavior of the pupil has also been studied. Pamplona et al. (2009) builds a model of pupil deformation as a function of the lighting of the environment. Agustin et al. (2006) measures pupil brightness in multiple subjects as a function of gaze direction.

On the other hand, very few papers deal with animating specifically the ocular region. Pinskiy and Miller (2009) presents an anatomically motivated approach for the region. It is able to procedurally produce deformations of the skin surrounding the eye due to eyelid closure and gaze direction. The same lack of attention has been given to human tears animation. Most of research on this topic deals almost exclusively with the animation of teardrops flowing on the human face. Hence the great majority of existing work applicable to human tears consists of generic methods for droplet flow on surfaces. For example, Müller et al. (2003); Wang et al. (2005); Zhang et al. (2012) present different physically-based models to simulate small-

scale fluid phenomena in contact with solid surfaces, which could be used to simulate teardrops. Jung and Behr (2009) present an image-space method for real-time simulation of droplet flows on 3D surfaces, optimized for GPU processing, and show its effectiveness on teardrop simulation. Chen et al. (2012) presents a hybrid method between a particle system and image-space model for simulating water droplets on the glass pane on interactive rates. van Tol and Egges (2012) develops a variant of the Smoothed Particle Hydrodynamics method optimized for real-time tear generation and control.

2.4 Real-Time Rendering

One of the biggest challenges of rendering human skin stems from the fact it is composed of multiple different semi-translucent layers: the *epidermis*, the *dermis* and the *hypodermis*. When light reaches the surface of the skin, part of it is scattered by interacting with the translucent materials, transverses the layers, and exits the surface at different points. This effect is known as subsurface scattering.

There are well established methods for physically based rendering of scattering inside materials. A traditional approach is to use the classical dipole scattering model from radiative transport and medical physics, in the two part form proposed by Jensen and Buhler (2002). This model is based on the assumption that light entering a material will go through many internal scattering events. In this situation, the diffusion theory is applicable and an analytical solution to the subsurface scattering profile becomes possible. This allows to simulate the scattering effect without having to explain the large number of individual internal propagation events. However, the dipole model does not address the multilayered structure of the skin or similar materials. For that purpose, Donner and Jensen (2005) proposed a multipole model, which allows the rendering of layered thin translucent slabs.

However, handling multiple scattering events is still too computationally expensive for real-time rendering. Luckily, research on real-time skin rendering is quite mature, and several methods for approximating the effect of skin subsurface scattering on a time constraint have been proposed. Mertens et al. (2005) approximates the dipole scattering model as a Gaussian filter blurring operation on a 2D diffuse irradiance texture. d'Eon and Luebke (2007) enhances this idea by approximating the multi-pole model with a sum of Gaussians. Yet these methods scale poorly with scene com-

plexity, since the subsurface scattering shading needs to be performed on a per-object basis. Jimenez et al. (2009) solves this issue by performing the operation in image space, which also limits computations to the visible parts of objects. Jimenez et al. (2015) further optimized the method, making it implementable in a post-processing stage using two 1D convolutions and a low sample rate. Penner and Borshukov (2011) avoids Gaussians entirely by pre-integrating the illumination effects of subsurface scattering due to curvature into the shading model, assuming screen space object normals have been pre rendered. Curvature is estimated using two mipmap levels of the normal map.

(a) No subsurface scattering (b) Subsurface scattering

Figure 2.5: The effect of subsurface light transport, reproduced from Jimenez et al. (2015).

A major issue with all the previous real-time models is that they do not convey the effect of light transversal from the back of an object to its visible surface. Jimenez et al. (2010b) addresses this issue separately. This presents an approximate model of transverse light ratio given transverse distance and approximates the transverse distance as the distance between the point and its projected location on the light source shadow map.

Recent work takes real-time rendering one step further. For example,Jimenez et al. (2010a) proposes a model of skin rendering taking into account concentrations of two chromophores, melanin and hemoglobin, which allows for dynamic control of skin color. Nagano et al. (2015) synthesizes

the effects of skin micro-structure deformation by anisotropically convolving the micro-structure displacement map.

Eye rendering is not as mature as skin rendering. As described in Section 2.1, the complexity of the eye allows for very different geometrical models of the eye. These, in turn, lead to different rendering approaches, not all of which are usable for interactive rendering. For example, Lefohn et al. (2003) renders the eye by ray tracing through cones with transparent and opaque textures, which is not practical for interactive rendering. In contrast, Vill (2014) models the eye as a single outer surface and ray traces from it an explicit procedural concave surface, which represents both the iris and the pupil.

Figure 2.6: Synthesized iridal chromatic variations, reproduced from Lam and Baranoski (2006).

Lam and Baranoski (2006) presented the first biophysically-based light transport model of the human iris. It takes into account the iridal morphological and optical characteristics to compute the light scattering and absorption processes occurring within the iridal tissues. Biophysical attributes can be controlled to achieve different iris coloration. Yet it does so by divid-

ing the iridal tissues into four layers (aqueous humour, ABL, stromal layer and IPE) and using a Monte Carlo ray-tracing approach.

Chapter 3

Interactive Gaze Driven Animation of the Eye Region

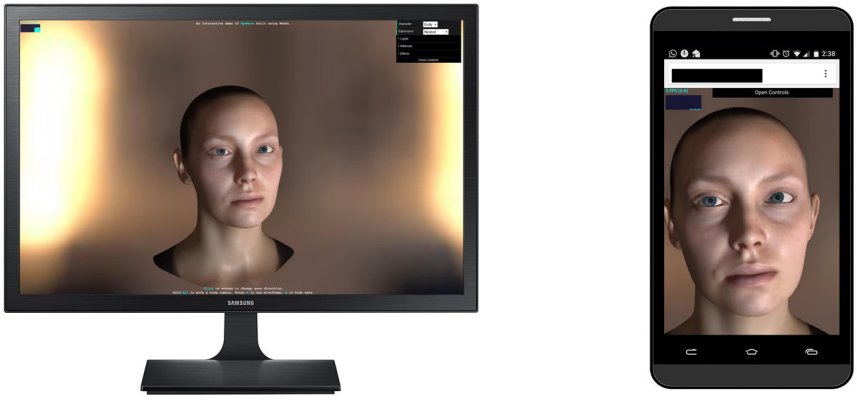
3.1 Introduction

In this chapter I describe our system for real-time animation of the eyelids, canthus and periorbital soft tissues that can be interactively controlled using a small number of animation parameters, including gaze. My first two contributions are part of this system. Our goal is to model the movements of the skin around the eyes, because it is the most important part of the face to convey expression. Therefore it is worthwhile to design a model specifically for this region, while other parts of the face may be modeled by more traditional methods.

Our system has two motivations. First, since there are no articulating bones in the eye region, the skin slides on the skull almost everywhere. Therefore, we would like to efficiently model this skin sliding. The exception is where it slides over the globes, which consists in my first contribution. Second, we would like the model to be easily learned using single camera videos of real human subjects.

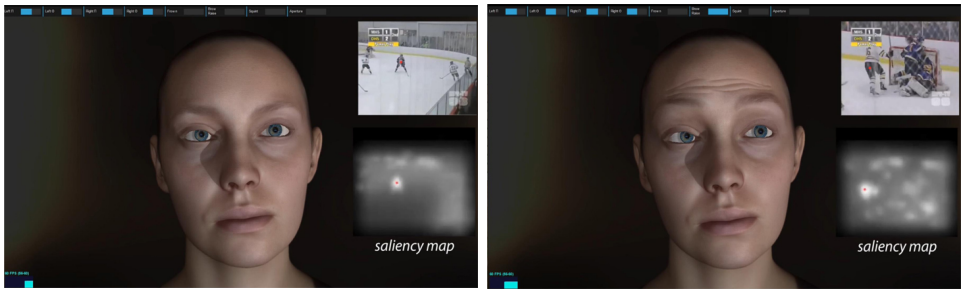
Our system also makes use of my second contribution, a generalized control model for of the factors that affect skin movement. Thus, the animation parameters used by our model can easily be obtained using traditional keyframed animation curves, measured from an actors performance using off-the-shelf eye tracking methods, or estimated from the scene observed by the character, using behavioral models of human vision.

Sections 3.2 and 3.3 clarify the core concepts used in the entire *EyeMove* project. Sections 3.4 and 3.5 contextualize the remainder of the chapter by briefly describing our actor capture system and skin motion generative model. Section 3.6 describes our system for transferring animation from one



(a) (b)

Figure 3.1: My WebGL application renders a character in real-time and runs in most computers (a) and mobile devices (b).



(a) (b)

Figure 3.2: Example of automatic facial animation generation while watching a hockey video. The character starts watching the match with a neutral expression (a) and gets concerned when a goal is scored (b). Eye movements were generated automatically using salient points computed from the hockey video.

character to another. The remainder of the chapter focuses on my main contributions to the *Eyemove* project, which I alone worked on.

3.2 Skin Movement in Reduced Coordinates

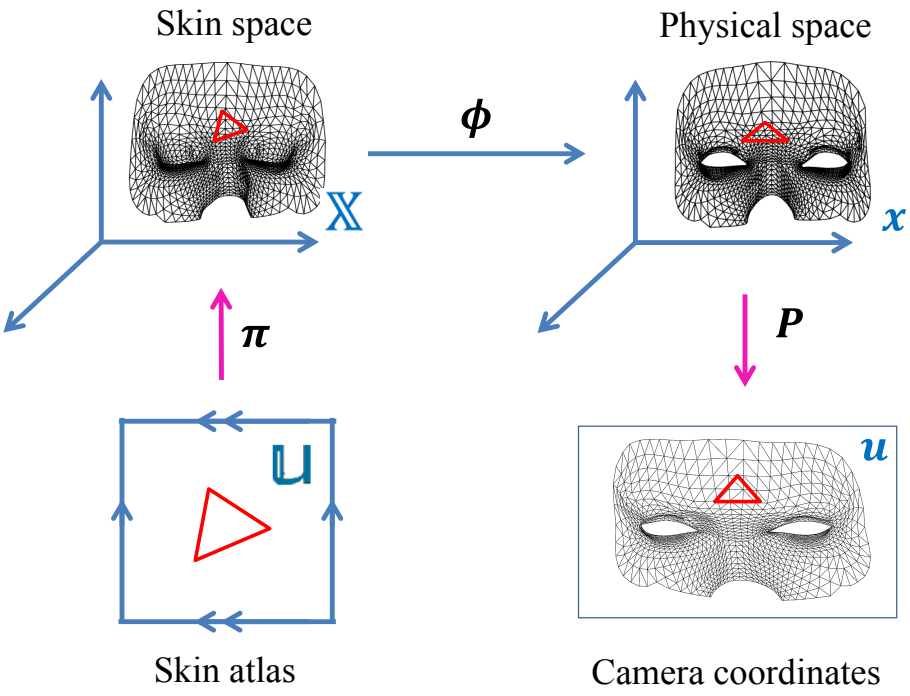


Figure 3.3: Overview of the spaces used for modeling skin movement

To represent the motion of skin, we use the reduced coordinate representation of skin introduced by Li et al. (2013a). This representation constrains the synthesized skin movement to always slide tangentially on the face, even after arbitrary interpolation between different skin poses. This avoids cartoonish bulging and shrinking and other interpolation artifacts. We will see in Section 3.9 that deformation perpendicular to the face can be achieved where needed, for example in the movement of the eyelid. This representation also reduces the data size in most computations.

Skin is represented by its 3D shape in a reference space called *skin space*.

This space is typically called "modeling space" in graphics and "material space" in solid mechanics. The key idea is that since skin is a thin structure, we can also represent it using a 2D parameterization π, using an atlas of coordinate charts. In our case a single chart is sufficient (see Figure 3.3). We call this chart *skin atlas*. It can be thought of as the skin's texture space.

Skin is discretized as a mesh $S = (V, E)$, a graph of n_v vertices V and n_e edges E. In contrast to Li et al. (2013a), this is a Lagrangian mesh, i.e., a point associated with a vertex is fixed in the skin space. Since most face models use a single chart to provide texture coordinates, these coordinates form a convenient parameterization π. In a slight departure from the notation of Li et al. (2013a), a skin material point corresponding to vertex i is denoted $\mathbb{X}_i$ in 3D and $\mathbb{u}_i$ in 2D coordinates. We denote the corresponding stacked arrays of points corresponding to all vertices of the mesh as $\mathbb{X}$ in 3D skin space and $\mathbb{u}$ in the 2D skin atlas.

The skin moves on a fixed 3D *body* corresponding to the shape of the head around the eyes. Instead of modeling the body as an arbitrary deformable object as in Li et al. (2013a), we account for the specific structure of the hard parts of the eye region. We model the body as the union of two rigid parts:

- The *skull*, a closed mesh corresponding to the anatomical skull with the eye sockets closed by a smooth surface.
- The mobile *globes*, two closed meshes corresponding to the outer surface of the eyeballs.

This representation allows us to efficiently parameterize changes in the shape of the body using the rotation angles of the globe and train the model independently of globe geometry. It is also useful for synthesizing gaze drive as described in Section 3.8.

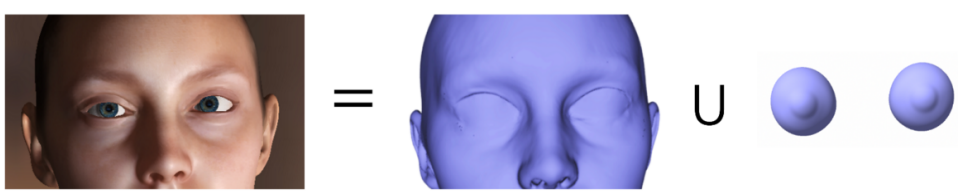


Figure 3.4: Body is the union of "skull" and globes

The skin and body move in the *physical space*, which is the familiar space in which we can observe the movements of the face, for instance, with a camera. For modeling, we assume there is a head-fixed camera with projective transformation matrix P that projects a 3D point corresponding to vertex i (denoted $\boldsymbol{x}_i$) into 2D camera coordinates $\boldsymbol{u}_i$, plus a depth value d_i. This *modeling camera* can be either a real camera used to acquire video, as described in Section 3.4, or a virtual camera used as a convenient way to parameterize the skin in physical space. We note that P is invertible, since P is a full projective transformation, and not a projection. We denote the stacked arrays of points corresponding to all vertices of the mesh as $\boldsymbol{x}$ in 3D physical space and $\boldsymbol{u}$ in the 2D camera coordinates.

During movements of the eyes, the skin in the eye region slides over the body. It is this sliding we are looking to model. Following the standard notation in solid mechanics, the motion of the skin from 3D skin space to 3D physical space is denoted ϕ (see Figure 3.3). Therefore, we can write $\boldsymbol{x} = \phi(\mathbb{X})$. However, directly modeling ϕ is not desirable, as it does not take into account the constraint that skin can only slide over the body, and not move arbitrarily in 3D. Instead, the key to our reduced coordinate model is that we represent skin movement in modeling camera coordinates. In other words, we model the 2D transformation:

$$\boldsymbol{u} = P(\phi(\pi(\mathbb{u}))) \stackrel{\text{def}}{=} f_g(\mathbb{u}) \tag{3.1}$$

Our goal is to directly model the function f_g as a function of input parameters, g, such as gaze and other factors that affect skin movement around the eyes. This representation has the dual advantage of both enforcing the sliding constraint and being easy to acquire video data from which to learn how the skin moves, as shown in Section 3.4.

3.3 Factors Affecting Skin Movement

We now examine the different input variables g that determine skin movement in the eye region. The most important and dynamic cause is eye movements that change gaze, in other words, that change what the character is looking at. However, other parameters, such as eyelid aperture and expressions, also affect the skin. We combine these into the "generalized" gaze vector g.

In the rest of this chapter, we assume g is a $n_i \times 1$ column matrix, where

n_i is the total number of possible inputs. Submatrices are extracted using Matlab-style indexing, e.g., $g_{1:3}$ is the submatrix comprised of rows 1 to 3, and $g_{[4,5]}$ is a submatrix with just the fourth and fifth elements.

Gaze We represent eye motion as a 3D rotation around the center of the globe. Any parameterization of 3D rotations could be used, but we use the coordinates from Fick (1854), which are widely used in the eye movement literature to describe the 3D rotation of the eye, since it factors the torsion in a convenient form. These are a sequence of rotations: first horizontal (g_1), then vertical (g_2), finally torsion (g_3).

Eyelid Aperture Eyelid movements are affected by both gaze and other factors. When our gaze shifts, eyelids, especially the upper eyelids, move to avoid occluding vision. We also move our eyelids to blink, and when expressing mental state such as arousal, surprise, fatigue, and skepticism. The upper and lower eyelids move in subtly different ways. Therefore, we use two additional input parameters to define aperture. One is the displacement of the midpoint of the upper eyelid above a reference horizontal plane with respect to the head (g_4); the plane is chosen to correspond to the position of the eyelid when closed. The other input is the displacement of the midpoint of the lower eyelid below this plane (g_5).

Expressions The skin in the eye region is also affected by facial expressions, such as surprise, anger, and squint. We can optionally extend the input parameters g to include additional parameters to control complex facial expressions. Expressions may be constructed using action units (AUs), defined by the Facial Action Coding System (FACS), first proposed by Ekman and Friesen (1977). In our implementation, action units are used in a similar way as blend shapes: they may be learned from using 'sample poses' that a subject is asked to perform or could also be specified by an artist. The strength of the ith action unit used in the model contributes an additional input parameter, $g_{i+10} \in [0, 1]$. Note that we defined five parameters per eye (3 gaze and 2 aperture), which together contribute the first 10 inputs.

3.4 Movement Capture and Tracking

To train a model, first we need to capture real world data. I briefly describe the system we used for tracking skin motion and gaze of human subjects, just for the purpose of contextualization.

Our setup (shown in Figure 3.5) only requires a single high frame rate RGB camera. We sit subjects with their chins firmly resting in front of the camera and ask them to look at markers set up around the environment. We then ask them to follow a point on a screen placed between them and the camera, and finally to perform some expressions. This process takes about 40 seconds to execute.

We also generate a 3D mesh of the subject using a Kinect camera and the commercial program Faceshift. By manually selecting a few landmarks on the recorded video and the mesh, we are able to project the mesh into the camera coordinates. We then track some of the mesh vertices along the frames while, at the same time, also tracking the gaze direction. Additionally, we have a system capable of tracking wrinkle lines and reconstructing their shape for any video frame.

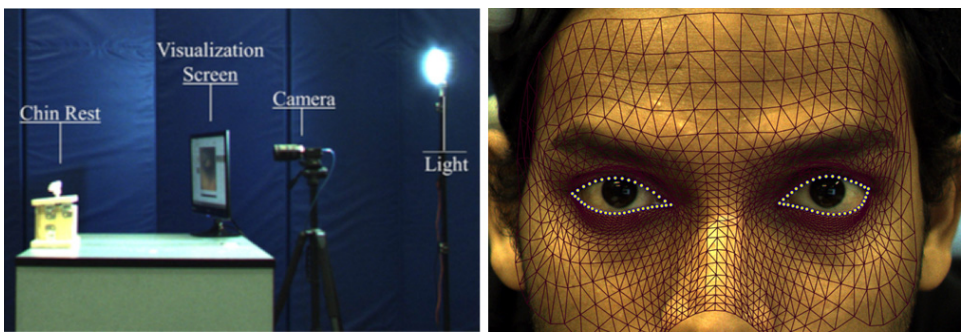


Figure 3.5: Left: gaze and skin tracking setup. Right: tracked vertices displayed on top of a recorded video.

3.5 Generative Model of Skin Movement

I now describe the learned skin motion model from the training data, just for the purpose of contextualization.

Learning the model directly from gaze parameters resulted in over-fitting and did not perform well for wide ranges of gaze parameters. We observed that the deformation of skin in the eye region is well correlated with the shape of the eyelid margin. This makes biomechanical sense, since the soft tissues around the eye move primarily due to the activation of muscles surrounding the eyelids, namely *orbicularis oculi* and *levator palpebrae* mus-

cles. Following these observations, we factored the generative model into two parts: *eyelid shape model*, and *skin motion model*. Each of these two models can be constructed or learned separately. A schematic diagram of the implementation is shown in Figure 3.6.

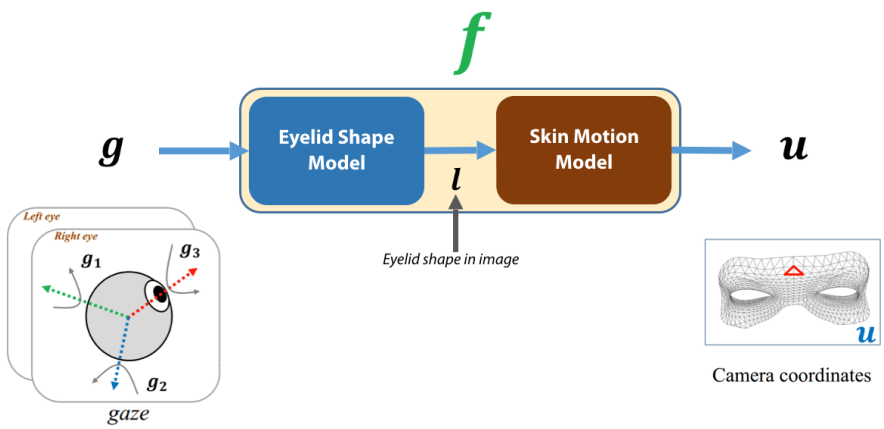
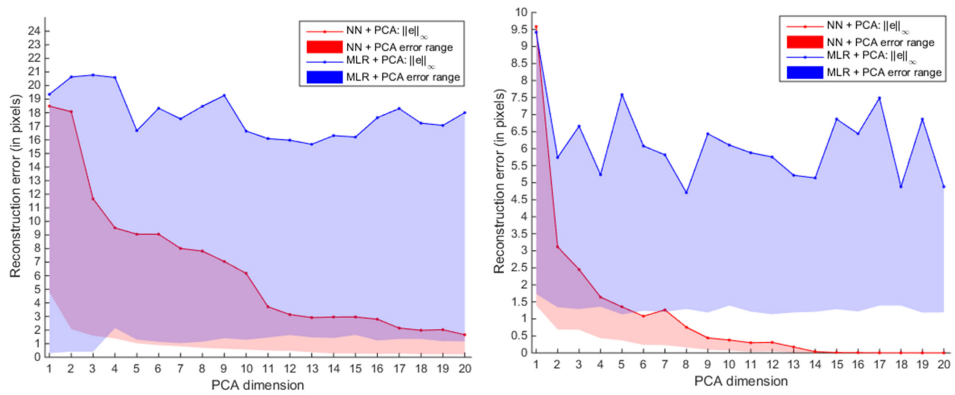


Figure 3.6: We predict the eyelids shape from gaze and aperture. We then predict skin motion using it and expression affects

We explored two different modeling approaches: *neural networks* of *radial basis functions* and *multivariate linear regression*. Although the neural networks achieved a lower normalized reconstruction error with just one radial basis function layer (see Figure 3.7), the difference did not justify the significant additional computational cost, as the reconstruction error from the linear approach is already quite low. Errors were computed using cross validation with randomly picked data points from the training data. Hence, to achieve real-time performance, we choose multivariate linear regression to exploit GPU computation and keep evaluation cost low. The advantages of MLR are shown in greater detail in Section 3.7.

We also reduced the dimension of the training data before training the models. We tested a variety of sophisticated dimensionality reduction methods, including *probabilistic principal component analysis*, *neighborhood component analysis* and *maximally collapsing metric learning*. Yet simple *principal component analysis* provided the best results.



(a) Eyelid shape model

(b) Skin motion model

Figure 3.7: Reconstruction error for the two models depending on the number of used principal components.

Eyelid Shape Model We define eyelid shape for each eyelid as piecewise cubic spline curves. We found that using between 17 and 22 control points for each spline faithfully captures the shape of the eyelids. The eyelid shape depends on both gaze and aperture. The general form of the model is:

$$l = K\,\ddot{l} = K\;L\;g_{[1-10]} \tag{3.2}$$

where l is the column matrix of coordinates of all control points for the eyelid and $\ddot{l}$ is its reduced principal component representation. We found that the results are well approximated using either 5 or 13 principal components, as additional components do not improve considerably the results (see Figure 3.7).

Skin Motion Model The skin of the eye region is modeled at high resolution (using about a thousand vertices in our examples) and is deformed based on the movement of the eyelids and expression data. Note that the skin motion depends on all four eyelids. The stacked vector of coordinates of all eyelids and expression action units is denoted $\boldsymbol{l}$. To generate expression data for training, we manually mark sample poses for each expression and the preceding neutral poses (see Section 3.4). We linearly vary each action unit from 0 to 1 between the two corresponding marked poses. The resulting model is:

$$\boldsymbol{u} = \boldsymbol{u}_0 + J\ \boldsymbol{e} = J\ M\ \boldsymbol{l} \tag{3.3}$$

where $\boldsymbol{e}$ is the reduced principal component representation of $\boldsymbol{u}$ and $\boldsymbol{u}_0$ is the position of the skin at the origin of the principal component space. We found that the results are well approximated using only 4 principal components, as additional components do not improve considerably the results (see Figure 3.7).

3.6 Transferring Animations

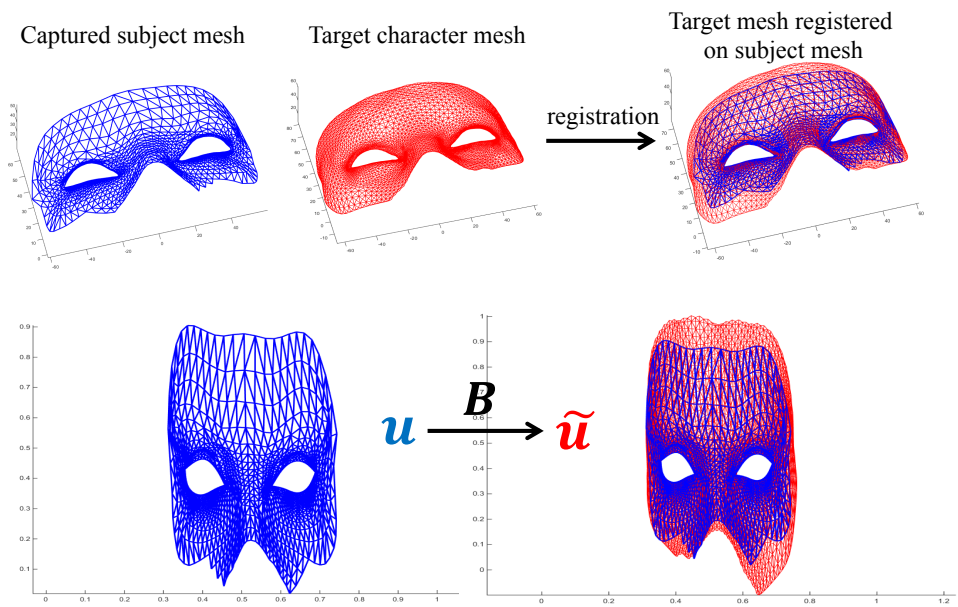


Figure 3.8: Overview of mesh registration. The target character mesh (red) is registered non-rigidly on the capture subject mesh (blue) shown in the top row. Image coordinates of target mesh are computed from the image coordinates of the model output using barycentric mapping computed during registration.

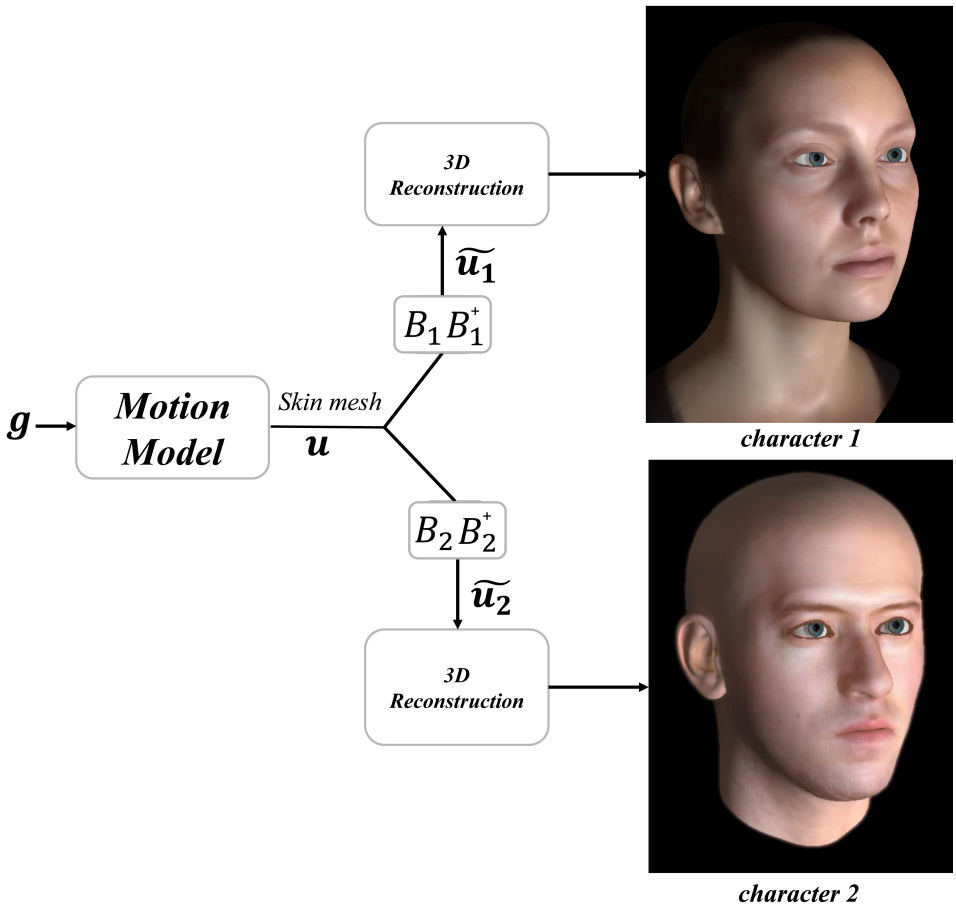


Figure 3.9: Models trained on one subject can be used to generate animations on arbitrary character meshes of any topology.

The skin motion model of Section 3.5 is constructed for the captured subject and a specific facial scanning of his or hers face. While facial scanning is now becoming widely available on commodity hardware Weise et al. (2011), training a skin motion model still requires our more complex capture setup (see Section 3.4). Animating artist made fictional characters is also an issue. Finally, the skin model may not include some parts of a given character, such as the inner eyelid margins, which are difficult to see and track in video (see Figures 3.5 and 3.8).

Here we discuss how the information in the generative model can be transferred to other target characters and to untracked parts of ocular region. The majority of this section has been done by Debanga Raj Neog, with me contributing ideas mostly towards the movement extrapolation process.

Character Transfer Given a new target character mesh with topology different from the *captured subject mesh* (3D face mesh of the subject for whom the model was constructed), we have to map the model output $\boldsymbol{u}$ to new image coordinates $\tilde{\boldsymbol{u}}$ representing the motion of the new mesh in image coordinates. The map is computed as follows: we first use a non-rigid ICP method (Li et al., 2008) to register the target mesh to the captured subject mesh in 3D. The resulting mesh is called the *registered mesh*. The vertices of the registered mesh are then snapped to the nearest faces of the captured subject mesh. We compute the barycentric weights of the registered mesh vertices with respect to the captured subject mesh, and construct a sparse matrix B of barycentric coordinates that can transform $\boldsymbol{u}$ to $\tilde{\boldsymbol{u}}$ such that:

$$\tilde{\boldsymbol{u}} = B\ \boldsymbol{u} \tag{3.4}$$

Finally, the registered mesh is projected to the image space using the projection matrix P to obtain $\tilde{\mathbf{S}}$.

Movement Extrapolation Some vertices of the captured subject mesh, particularly those of the inner eyelid margin, are not included in the skin movement model computed in Equation 3.3. This occurs because such vertices are difficult to be tracked in video. As such, a data-driven model cannot be built for them. Instead, we compute the skin coordinates of untracked vertices as a weighted sum of nearby tracked vertices. The general form is:

$$\tilde{\boldsymbol{u}}^{+} = B^{+}\ \tilde{\boldsymbol{u}} \tag{3.5}$$

Where $\tilde{\boldsymbol{u}}^{+}$ is a vector containing both the image coordinates of the tracked vertices in the new mesh, $\tilde{\boldsymbol{u}}$, and of the newly added untracked vertices. For extrapolation, we use normalized weights proportional to the inverse distance to the neighboring points in the starting frame.

3.7 Interactive Model Control

One issue with our model, as defined in Section 3.5, is the input parameters. While powerful, they are not consistent nor easy to control: first, aperture is defined in the capture camera space, which is not consistent between different subjects or data capture sessions; second, gaze and aperture need to be controlled simultaneously to achieve realistic skin movements. Additionally, the skin motion model is not optimized for evaluation performance, making it unusable in real-time situations. The addition of the animation transfer operations, defined in Section 3.6, creates additional overhead.

In this section I describe my second contribution, which presents solutions to these issues.

Aperture Normalization I want to specify aperture in a format $\tilde{g}$ that is independent of capture conditions and training subject performance. I define aperture $\in [0, 1]$ as we did for expressions. That is, an aperture input of 0 must correspond to having the eye closed, while an aperture input of 1 to having the eye wide open.

Remember that we asked subjects to close and open their eyes during capture (see Section 3.4) and that eyelid aperture is defined by the four input parameters $g_{[4,5,9,10]}$ (see Section 3.3). I define the following heuristic for eye closure:

$$\iota = |g_4 - g_5| + |g_9 - g_{10}| \tag{3.6}$$

I find the frames with the highest and the lowest ι scores on the training data. These two correspond to the moments when the eyes were closed and open, respectively. Let g^c and g^o be the input parameters associated with these two frames. Conversion from the general input parameters $\tilde{g}$ into the parameters used by the eyelid shape model g becomes, for the left eye:

$$g_{[4,5]} = g^c_{[4,5]} + (g^o_{[4,5]} - g^c_{[4,5]})\, \tilde{g}_{[4,5]} \tag{3.7}$$

Optimized Motion Model Notice that all of the operations described so far are linear, and were represented as vector additions or matrix multiplications. Consider g to be a point in a $10 + \mathbb{i}$ order euclidean space, where $\mathbb{i}$ is the number of auction units used. One can then use homogeneous coordinates representation to replace vector addition operations with matrix multiplications as well. If I were to pre-multiply all the aforementioned matrices, I would obtain a new matrix U such that:

$$\tilde{\boldsymbol{u}}^{+} = U\ \tilde{g} \tag{3.8}$$

This implies U is a $\tilde{n}_v$ by $10+\mathbb{i}$ matrix, where $\tilde{n}_v$ is the number of vertices in the target character mesh. Now the entire pipeline can be run in parallel on a vertex per vertex basis, which is ideal for exploiting GPU computational power. Yet remember we represented vertex motion in 4 principal directions, as described in Section 3.5. Thus I can further optimize computations. I define instead the general model:

$$\tilde{\boldsymbol{u}}^{+} = \tilde{U}\ W\ \tilde{g} \tag{3.9}$$

where

$$\boldsymbol{e} = W\ \tilde{g} \tag{3.10}$$

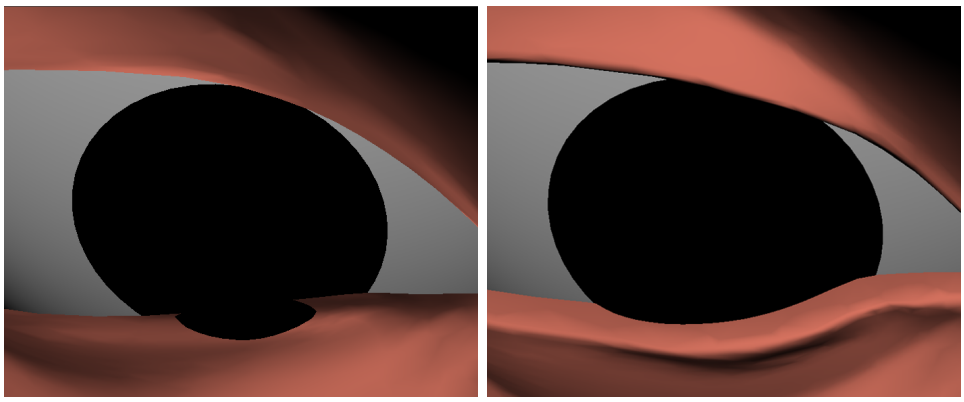
This results in a small 4 by $10+\mathbb{i}$ matrix W and a tall n_v by 4 matrix $\tilde{U}$, which reduces per vertex data to a 4-dimensional vector, perfect for GPU computations.

Baseline Aperture Model As discussed previously, the eyelid aperture is influenced by gaze direction. Testing the system found that maintaining a static aperture produces unrealistic results, but manually controlling aperture to match gaze movements is impractical. I want aperture to depend on gaze, but still be able to control eyelid opening and closing to produce voluntary actions such as partial eye closure or wide eye opening.

We used multivariate linear regression to train a linear baseline aperture model, A, for predicting the aperture due to gaze. Since the torsion angle of the globe does not have a significant effect on aperture, I only use the first two components of gaze. As a consequence of the aforementioned normalization of the aperture in the range $[0, 1]$, the baseline aperture can then scaled by an *eye closing* factor $c \geq 0$ to simulate blinks, frowning, arousal, etc. The resulting model for the left eye is:

$$\tilde{g}_{[4,5]} = c\ A\ \tilde{g}_{[1,2]}. \tag{3.11}$$

3.8 Deformation by Globe Movement



(a) (b)

Figure 3.10: Visualization of the effects of globe mesh fitting. Without deforming (a), intersections between the globe and skin geometry are glaringly obvious. Eyelid depth and corneal subsurface deformations are not present.

Our skin motion model predicts motion of the skin in 2D modeling camera coordinates $\tilde{u}^+$, which correspond to the 3D coordinates $\tilde{x}^+$ of the character skull mesh. However, remember that we model the body as the union of the skull and the globes (see Section 3.2). Our skin motion model does not take into account the globes geometry. The reason behind this separation of the body is that, as described in Section 2.2, the globe is not spherical and has a prominent cornea. Thus, rotations of the globe should produce subsurface deformations of the skin. I want to recreate these effects in a computationally efficient manner, while also preventing geometry intersections between the skin and the globes. However, computing geometry intersections would be far too computationally intensive for this task.

To address this challenge, I present my first contribution: an efficient method that relies on computing the distance from any point to the globe's surface.

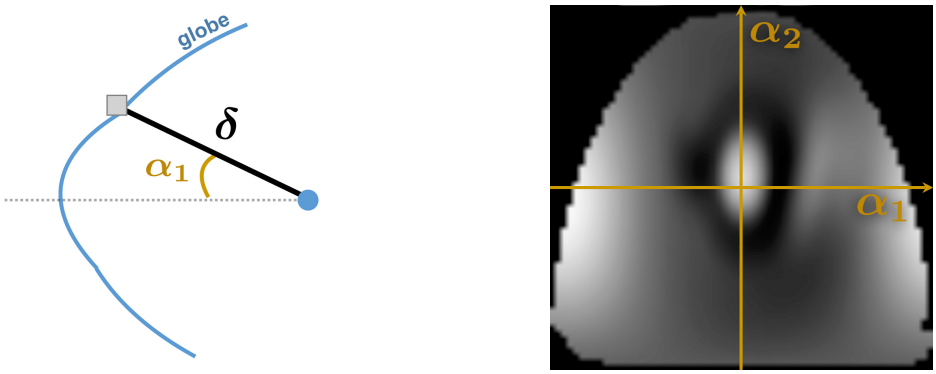
Deformation in Polar Space As the globe is still similar to a sphere, I represent the shape of the globe in polar coordinates in the globe's reference frame. Define a radial distance map D of the globe's outer surface to the globe center in polar coordinates. This map can be sampled to obtain the globe's radial distance δ as a function of polar coordinates α:

$$\delta \approx D(\alpha) \tag{3.12}$$

Then, for any point with polar coordinates α and cartesian coordinates r in the globe's reference frame, the distance to the eye surface can be approximated as:

$$|r| - \delta + t \tag{3.13}$$

A negative value means the point is inside the globe. t a user defined skin thickness on the point. For each frame, radially displace vertices inside the globe at its current orientation, which allows the skin to slide on the globe and prevents geometry intersections. Additionally, I am able to dynamically control skin thickness.



(a) (b)

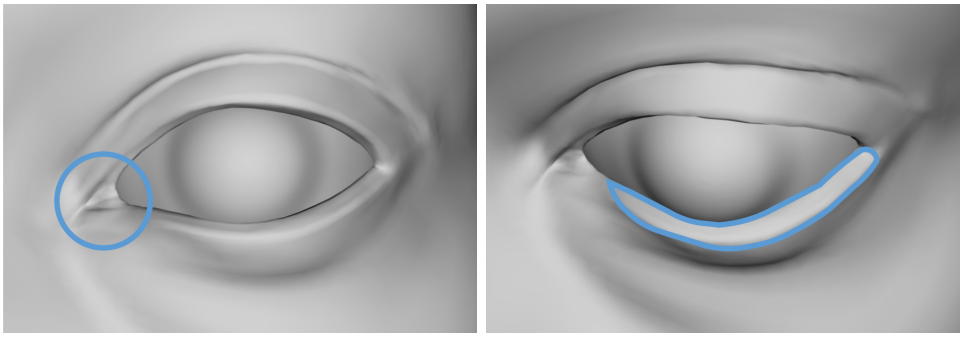
Figure 3.11: Spherical coordinates are computed for each globe vertex (a). Distance to the globe center is mapped according to the remaining coordinates (b).

Preserving Fine Details Some regions of the face should not slide directly on the skull or globe geometry. For example, the canthus is located further deep in the skull than the surrounding regions, while the eyelid margins must be at a distance from the globes. Luckily, all these details are

located in the extrapolated regions. As such, I handle vertices in these regions differently. For each extrapolated vertex, I precompute an heuristic δ_0, which is the radial distance between the original vertex position and the globe surface:

$$\delta_0 = |r_0| - D(\alpha_0) \tag{3.14}$$

r_0 is the original position of the vertex in the globe's reference frame and α_0 in polar coordinates. I then radially offset the skin vertex position by δ_0, which allows me to approximately reconstruct the eyelid margin thickness and canthus depressions.



(a) Canthus (b) Eyelid margin

Figure 3.12: Examples of details that would be lost if all the skin slided directly on the skull and globes.

3.9 Interactive Motion Synthesis

Our framework is well suited for real-time synthesis using GPUs. I now discuss how our model can be used in a real-time rendering pipeline to generate realistic skin motion and shading.

As a proof of concept, I implemented two different WebGL browser applications. Both synthesize animations of the ocular region in real time using our system. I used the *Digital Emily* data provided by the WikiHuman project (Ghosh et al., 2011; WikiHuman), with permission, to create our main virtual character. The applications start by downloading all the

required mesh, texture, and optimized model data. Then they run offline and perform all computations without communicating with a server.

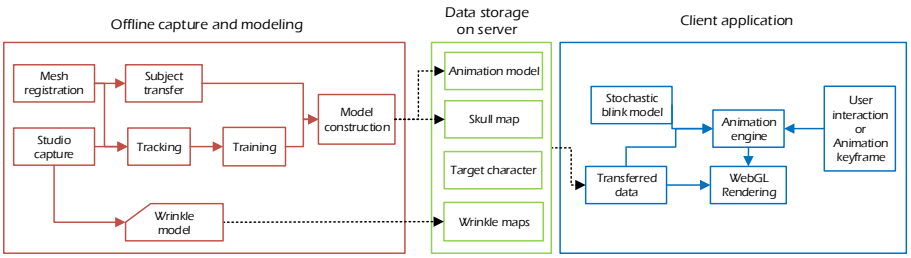


Figure 3.13: Overall architecture of my applications.

The two applications differ only on the model input sources: one is a fully interactive user controlled application, while the other plays a cut-scene defined using keyframed animation curves. Head movements can be added for realism. Different expressions, such as surprise and anger, can also be added at key moments.

Reconstructing 3D Geometry Recall that we represent the 3D coordinates $\tilde{\boldsymbol{x}}^+$ of the character skull mesh using the 2D skin coordinates $\tilde{\boldsymbol{u}}^+$ corresponding to the neutral pose. Yet, due to animation transfer, this operation might not correspond to a projective nor affine transformation. Hence, to determine 3D vertex positions $\tilde{\boldsymbol{x}}^+$ given $\tilde{\boldsymbol{u}}^+$, I pre-render a texture of $\tilde{\boldsymbol{x}}^+$ in 2D skin space using natural neighbor interpolation on the vertices values. This texture T, which I name the *skull map*, can be sampled to obtain $\tilde{\boldsymbol{x}}^+$ as a function of $\tilde{\boldsymbol{u}}^+$:

$$\tilde{\boldsymbol{x}}^+ \approx T(\tilde{\boldsymbol{u}}^+) \tag{3.15}$$

Following the same procedure, I pre-render a texture Υ to sample the skull surface normals $\tilde{\boldsymbol{\eta}}^+$ given $\tilde{\boldsymbol{u}}^+$, which we name the *skull normal map*:

$$\tilde{\boldsymbol{\eta}}^+ \approx \Upsilon(\tilde{\boldsymbol{u}}^+) \tag{3.16}$$

The aperture model and the PCA-reduced motion model W are performed on the CPU. The PCA reconstruction using U, the more computationally expensive portion of the model, is performed on the GPU on a per vertex basis, as a matrix-vector multiplication in the vertex shader (see Figure 3.14).

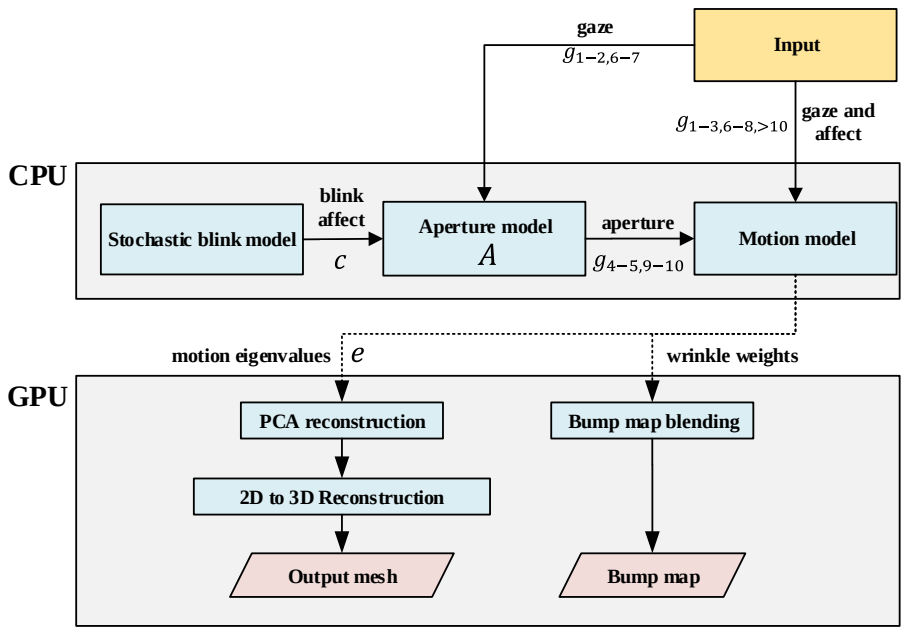


Figure 3.14: Flow chart of how inputs are handled in the applications and where computations are performed.

Surface Normal Generation So far, I have described how to compute the vertices positions in real-time. To perform surface shading, one must also compute the surface normals $\tilde{\boldsymbol{\eta}}^{+}$. I compute normals on a per-vertex basis. As described in Equation 3.16, the surface normals can be easily retrieved for the portions of the skin that are sliding in the skull mesh using the skull normal map Υ. However, the same is not true for vertices that have been radially displaced, whether because they are sliding on a globe, or for being extrapolated vertices with a δ_0 heuristic.

For any vertex sliding on a globe, I take the surface normal of the globe τ where the vertex is sliding on, which I name the *globe sliding normal*:

$$\tilde{\boldsymbol{\eta}}^{+} = \hat{\tau} \tag{3.17}$$

τ could be sampled by precomputing a surface normal map in polar coordinates, in a similar fashion to the map D defined for globe mesh fitting. Yet, it is enough to approximate the surface of the globe to a sphere in this case. The surface normal in head space then becomes:

$$\tau = \tilde{\boldsymbol{x}}^{+} - \emptyset \tag{3.18}$$

where $\emptyset$ is the globe's center in head space.

For points that are not sliding neither on the skull nor on a globe, that is, for vertices with a δ_0 heuristic, I propose a different method. Remember that extrapolated vertices are always radially displaced. As in Equation 3.18, I compute the current globe sliding normal τ, but also compute the globe sliding normal τ_0 at the vertex original position. I then compute the smallest rotation $\boldsymbol{\tau}$ that aligns these two normals:

$$\boldsymbol{\tau} = \hat{\tau_0} \times \hat{\tau} \tag{3.19}$$

This rotation serves as an estimation of how the surface around the vertex has been rotated from it's original position. Finally, I rotate the original vertex surface normal η_0 using $\boldsymbol{\tau}$.

Input: Motion eigenvalues $\boldsymbol{e}$
Output: Head space 3D vertex coordinates $\tilde{\boldsymbol{x}}^{+}{}_i$ and normal $\tilde{\boldsymbol{\eta}}^{+}{}_i$

Compute skin coordinates: $\tilde{\boldsymbol{u}}^{+}{}_i = \tilde{U}_i \cdot \boldsymbol{e}$
Sample skull map: $\tilde{\boldsymbol{x}}^{+}{}_i = T(\tilde{\boldsymbol{u}}^{+}{}_i)$
Sample skull normal map: $\tilde{\boldsymbol{\eta}}^{+}{}_i = \Upsilon(\tilde{\boldsymbol{u}}^{+}{}_i)$

Compute globe sliding normal: $\tau = \tilde{\boldsymbol{x}}^{+}{}_i - \emptyset$
Compute position in globe space: $r = \emptyset \cdot \tilde{\boldsymbol{x}}^{+}{}_i$

Compute polar angle: $\alpha = (tan^{-1}(\frac{r_x}{r_z}),\ cos^{-1}(\frac{r_y}{|r|}))$
Compute globe radial distance: $\delta = D(\alpha) + t + \delta_0$
Compute distance ratio: $\hat{\delta} = \delta / |r|$

If $(\hat{\delta} > 1 \mid\mid \delta_0)$ then radially displace: $\tilde{\boldsymbol{x}}^{+}{}_i = \emptyset^{-1}(r \cdot \hat{\delta})$
If $(\hat{\delta} > 1)$ then $\tilde{\boldsymbol{\eta}}^{+}{}_i = \hat{\tau}$
If (δ_0) then $\tilde{\boldsymbol{\eta}}^{+}{}_i = \eta_0$ rotated by $\hat{\tau_0} \times \hat{\tau}$

Algorithm 1: My vertex animation algorithm for a single globe executed per frame.

Wrinkle Reconstruction As mentioned in Section 3.4, we have a system that is able to reconstruct wrinkle geometry from any captured pose, represented as spline geometry. We can also synthesize bump map textures that

represent these wrinkles for any given pose, and project it into the target character's texture space. To generate facial expression wrinkles interactively, we synthesize a bump map texture for each of the expression sample poses marked in Section 3.5. I then combine these bump maps using as the blending weights the corresponding expression affect parameters. The blending operation is performed on the fragment shader.

Realistic Input Synthesis In the user controlled application, blink input is handled by a stochastic blink model that generates spontaneous human blinking. Our model determines blinking intervals and blink amplitude according to a normal distribution, as this is often used to represent natural occurrences. We estimated blink frequency distribution based on data collected by Ishimaru et al. (2014) and blink amplitude distribution on the data by Trutoiu et al. (2011) and Evinger et al. (1991). See table 3.1. The relation of amplitude to phase duration was based on Evinger et al. (1991) and the blink shape profile was based of Flash and Hogan (1985).

	μ	σ
Blink frequency	0.1339 Hz	0.1605 Hz
Blink amplitude	34.5895°	3.805°

Table 3.1: Normal distribution of blink frequency and amplitude while performing a watching activity.

Meanwhile, eye movement is modeled by a biologically based saccade model, which controls gaze shifts due to user requests to changing the visual target. It was based on the work of Bahill et al. (1975) and Harwood et al. (1999). We also implemented a target tracking model, which makes use of both smooth pursuit and saccadic movements based on the work of Blohm et al. (2006). Yet we never used it, as a smooth pursuit system did not complement well our application user interface.

3.10 Results

To our knowledge, this is the first system for data-driven real-time animation of soft tissue movement around the eyes based on gaze input. As described in Section 2.3, almost all previous work in animation of eyes has been on animating gaze, with some recent attention paid to the kinematics of blinks and the appearance of the globe and iris. For instance, Ruhland et al.

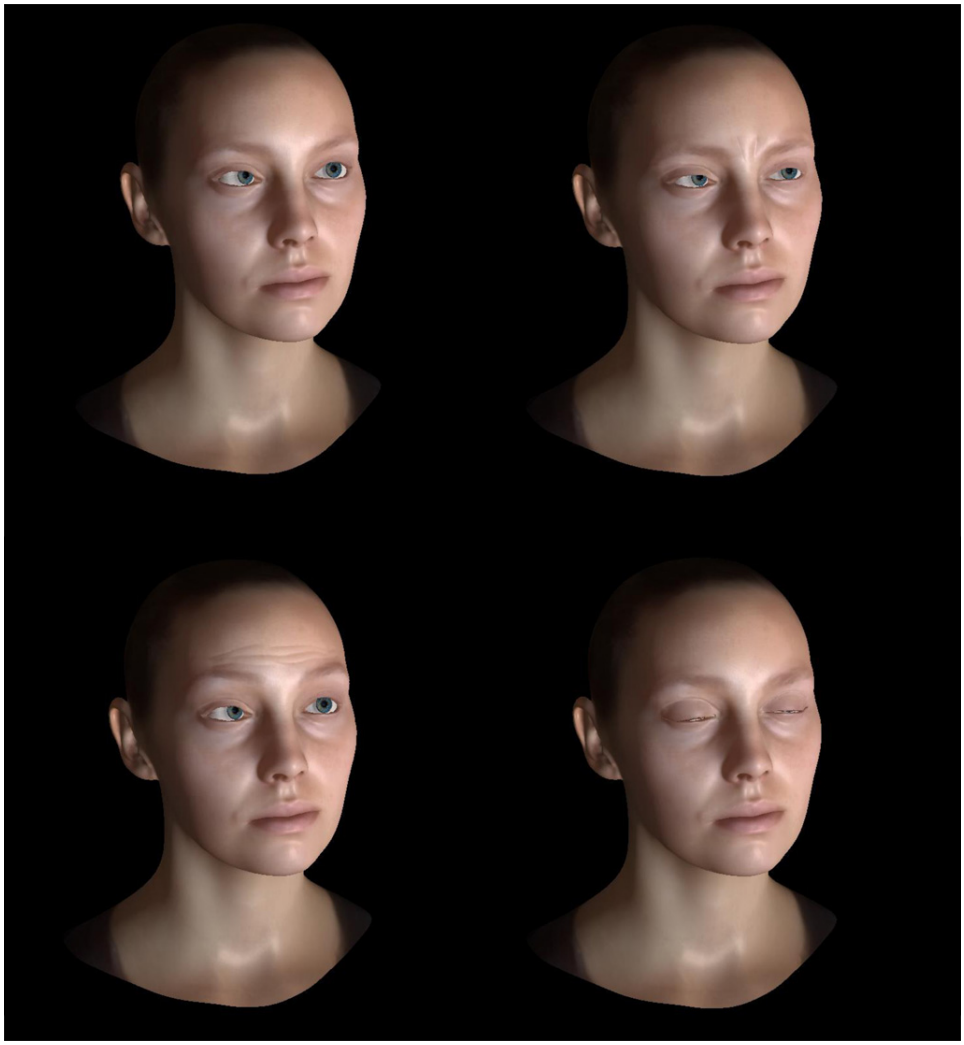


Figure 3.15: Facial expressions wrinkles generated interactively.

(2014), an excellent recent survey of eye modeling in animation does not even mention the soft tissues or wrinkles surrounding the eyes.

Performance The applications run in any modern browser, at 60fps on a desktop with an Intel Core i7 processor and an NVIDIA GeForce GTX 780 graphics card, and at 24fps on an ultraportable laptop with an Intel Core i7 processor and integrated Intel HD 5500 Graphics, and at 6fps on a Nexus 5 android phone with Adreno 330 GPU (see Figure 3.1). The majority of workload is for rendering; the model itself is very inexpensive (see table 3.2).

	Static	Animated	Overhead
File download (MB)	3.7	5.3	1.6
Memory usage (MB)	240	370	130
GPU memory (MB)	390	417	27
Runtime per frame (ms)	0.5450 ± 0.1553	0.6717 ± 0.1564	0.1267 ± 0.0011

Table 3.2: Performance overview of the applications. Note the animation framework is run twice per frame as described in Section 3.11.

Eyelid Deformation during Blink We can generate realistic skin deformation in a blink sequence using my stochastic blink model described in Section 3.9. A blink sequence is shown in Figure 3.16.

Saliency Map Controlled Movement When we observe object motion in real life or in a video, our eyes produce characteristic saccades. We computed saliency maps, a representation of visual attention, using the method proposed by Itti et al. (1998). Points in the image that are most salient are used as gaze targets to produce skin movements around the eyes. We show an example of skin movement controlled by gaze, using salient points detected in a video of a hockey match in Figure 3.2.

Static Scene Observation Our generative gaze model can be controlled by gaze data obtained from any eye tracking system. We used gaze data of a subject observing a painting to drive our system. This produces very realistic movements of eyelid and skin around the eyes as can be seen in Figure 3.17.

Figure 3.16: Skin deformation in eye closing during a blink.

3.11 Legacy Hardware Considerations

One of the major changes of rendering paradigms in the interactive media industry over the past few years was the adoption of *multi-pass rendering*. This feature is used for less traditional rendering methods, such as *deferred shading*, or to achieve post-processing effects, such as *depth of field* (for example, Demers, 2004), *screen-space reflections* (e.g. Tatarchuk, 2009) or, more importantly for my case, subsurface scattering methods (e.g. Jimenez et al., 2015).

One of the most important requirements for efficient multi-pass rendering is support for multiple render targets. That is, to be able to render into multiple draw buffers in a single pass. While *multi-target rendering* has become prevalent on desktop and laptop devices, this is still not true on mobile devices. For example, one of the main limitations of the WebGL specification is that multi-target rendering is not part of the core specification. While the majority of computers support the *multi-target rendering* extension, the majority of mobile devices still does not.

As described in Section 2.4, to achieve realistic real-time skin rendering it is standard to apply an approximation of the subsurface scattering proprieties of the skin. While some do not require multi-pass rendering, the

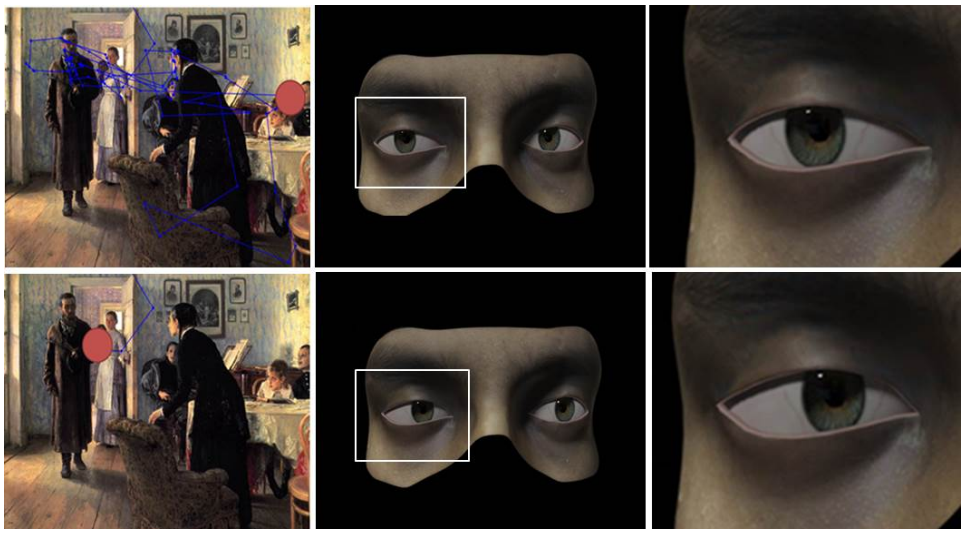


Figure 3.17: Skin movement driven by gaze during static scene observation. The red circle in the left represents the image point subject is looking at.

majority do. In our demo application, I implement the method proposed by Jimenez et al. (2015). It requires 3 rendering passes: the first projects the geometry to generate screen space diffuse, specular and depth maps of the skin. This is followed by two screen space passes that perform a Gaussian blur on the diffuse illumination map taking into account the depth map. The last of these passes will also sum the diffuse and specular illuminations to achieve the final result.

In my implementation, while I still only perform 3 passes, I need to perform two geometric passes (instead of one): the first rendering pass generates the diffuse illumination and depth maps (depth is stored in the alpha channel). The second one is a screen space Gaussian blur. The third, although it performs the screen space blur, also computes specular illumination and sums it to the diffuse for the final result. Hence, my optimized subsurface scattering system for WebGL requires the vertex shader program to run twice per frame.

Chapter 4

Screen Space Distance Fields

4.1 Motivation

As explained in Section 2.3, while there is some work regarding interactive animation of tear drops flowing on human faces, there is no work regarding the interactive animation of the tear film. I hypothesize that small scale details, such as the tear film, might be reproducible procedurally at the post-rendering stage. In other words, I hypothesize that the visual effects of the tear film might be reproducible in screen-space. Alexander et al. (2013) suggests that the main noticeable visual effects of the lacrimal lake are:

- Darkening due to light absorption during light transversal
- Specular reflections

The first effect can be parameterized by the amount of fluid visible at each pixel. That is, the depth of the lacrimal lake from the camera perspective (screen space). The second requires the positions and normals of the film surface to be computed per pixel. Given a screen-space depth map of the scene and the aforementioned depth of the fluid at each screen point, the film surface position can be reconstructed using the sum of the two and the inverse projection matrix. Similarly, the surface normal can be computed using a screen-space normal map and the derivative of the fluid depth.

Hence, I want to define a function of fluid depth per pixel. A possible strategy would be to parameterize this function according to distance from the surface point to the nearest skin point. This distance could be efficiently computed if a volumetric distance field of the skin was available. This is a common strategy in real-time lighting computations, as volumetric distance fields can be precomputed for rigid objects. However, the skin is not a rigid object, and computing a volumetric distance field of it in real time is not a viable solution, as seen in Erleben and Dohlmann (2008) and Sanchez et al. (2012).

(a) volumetric distance fields

(b) computation time

Figure 4.1: Volumetric signed distance fields and corresponding computation times. Reproduced from Erleben and Dohlmann (2008).

To solve this problem, I explored computing distance fields of an object (in this case, the skin of a human face) in screen space. In this chapter I describe my third contribution: three different algorithms for real-time procedural generation of distance fields in screen-space, and a study of their comparative performance and quality. I show that it is a viable strategy, both in terms of performance and quality.

These three different sampling strategies for estimating distance fields are described in detail in Section 4.4. To elucidate how and why I designed these algorithms, Section 4.2 provides some background on unrelated but similar problems. Section 4.5 provides a comparison of these three algorithms and shows them being used to procedurally generate lacrimal lakes in real-time.

4.2 Background

It is common in games to use *feature outline rendering* to draw contours around objects. This is generally done either to highlight objects on the scene or for artistic purposes. One traditional method to achieve outline rendering is the one proposed by Rossignac and van Emmerik (1992), which consists in rendering the objects first in the stencil buffer, and then rendering thick wireframes of outlined objects. One could potentially compute distance fields by rendering multiple overlapping outlines for each skin object, with each outline having a different scale and color intensity corresponding to the distance they represent. Yet, this would scale poorly with screen resolution and, more importantly, scene complexity.

I choose to explore computing distance fields without resorting to additional geometry passes. Instead, I resort to searching the screen-space neighborhood. This strategy shares a premise with all screen space ambient occlusion methods, which is that the properties on a surface point can be approximated by the visible surrounding geometry. Hence, while the purpose of this method is very different and the final sampling methods presented diverge much from the techniques used for computing ambient occlusion, it is still relevant to understand these existing screen space methods.

4.2.1 Traditional Screen Space Ambient Occlusion

(a) (b)

Figure 4.2: Screen sampling distribution as in Mittring (2007) (a). Black points are identified as occluders. Using the surface normal (b) allows to reduce the sampling space to a more relevant hemisphere. Reproduced from Chapman (2011).

Screen space ambient occlusion (Mittring, 2007) is generally accepted as the first method of this kind. Its essential concept is to approximate an occlusion factor for each point on a surface by sampling points in a sphere centered on the point and estimate occluders.

Let P be the camera projection matrix, as in chapter 3, and x a visible surface point in the camera view space. Randomly choose n_s samples in view space inside of a sphere of user defined radius μ centered around x. Thus, the set of all possible samples Ψ is defined as:

$$\Psi = \{\mathbb{s}_i \mid \mu > |x - \mathbb{s}_i|,\ 1 \leq i \leq n_s\} \tag{4.1}$$

where $\mathbb{s}_i$ is the ith random sample. The goal is to determine whether each sample $\mathbb{s}_i$ lies behind the visible geometry. This can be achieved by casting

a ray from the camera to each sample $\mathbb{s}_i$ to find the intersecting visible geometry position s_i along the ray. If $\mathbb{s}_{i_{\vec{z}}} > s_{i_{\vec{z}}}$, then $\mathbb{s}_i$ is not visible and contributes to the occlusion of x. The ambient occlusion of x is estimated by the number of occluded samples found.

To efficiently find the surface point s_i corresponding to a sample point $\mathbb{s}_i$, Mittring (2007) avoids performing ray-to-geometry intersections. Instead, the sample $\mathbb{s}_i$ is first projected to screen image space:

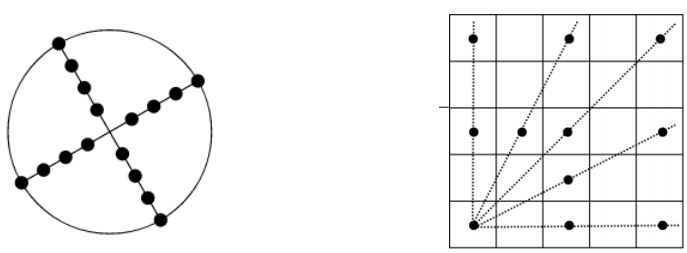
$$u_i = P\ \mathbb{s}_i \tag{4.2}$$

The screen space depth buffer, $D(u)$, is then used to reconstruct the surface position rendered at u_i:

$$s_i = P^{-1}\ [u_i, D(u_i), 1] \tag{4.3}$$

Later methods also make use of the surface normal to sample within a hemisphere oriented along the surface normal at that pixel (Bavoil et al., 2008; Chapman, 2011). This improves the relevance of the sampling space, increasing fidelity of the results. It has the disadvantage of requiring a per-fragment normal map, but this is already available when performing deferred shading.

4.2.2 Image-Space Horizon-Based Ambient Occlusion



(a) (b)

Figure 4.3: Theoretically, Bavoil et al. (2008) raymarches the depth buffer in a number of equiangular directions across a circle in image space (a). In reality, sampling is made at texel centers (b).

Horizon-based ambient occlusion (Bavoil et al., 2008) is an example of a more recent influential method for screen space ambient occlusion. One

of the improvements in comparison with the traditional method described before is that it splits the unit sphere by a horizon line, defined by computing a signed horizon angle at every surface point. Yet, for the purposes of my problem, I am only concerned with the sampling strategy used, and hence I will not cover the above problem.

Instead of sampling points in a sphere on view space, Bavoil et al. (2008) picks n_s directions in the image space around the current pixel, which corresponds to directions around the z axis in eye space, and raymarches along those directions. The raymarching is used to keep track of the elevation angle at each direction. Each time the angle is larger than the previous maximum, a new chunk of occluding geometry has been found. A static step size is used for raymarching, but sampling is always made on texel centers to avoid depth discontinuity artifacts.

4.3 Problem Definition

My goal is to compute, for any visible surface point x, the shortest distance λ from x to the skin object. In other words, I want to produce a screen space image of λ, which will be the screen space distance field.

My method is intended to be used on the second stage of a deferred shading pipeline. This means that one can assume information such as the surface albedo, normal and view depth have been rendered and stored into screen-space texture buffers. I compute λ in screen space by making use of some of these textures.

Distance Metric To efficiently compute λ and not depend on scene complexity, I approximate it by searching the visible geometry. I estimate λ by finding the closest visible skin surface point to x and computing the distance between the two points in camera view space:

$$\lambda \approx |\boldsymbol{s} - x| \tag{4.4}$$

where $\boldsymbol{s}$ is the ideal skin surface position I would like to find. Let n_s be the number of skin surface samples, and s_i the ith sample. Then intuitively the distance λ can be approximated as:

$$\lambda \approx \min_i \{|s_i - x|,\ 1 \leq i \leq n_s\} \tag{4.5}$$

Material Detection How to determine whether a given sample s_i lies on the skin surface or some other material must also be addressed. Let u_i be the screen space coordinates of s_i. Define a screen space binary function, $M(u)$, which returns whether a given sample u_i corresponds to a skin point. Similar functions can be defined to identify other materials.

This function can be implemented as a screen-space material mask texture, interactively rendered alongside the depth buffer. This is quite efficient, as a single material mask texture can support up to 256 different materials using a single 8-bit channel. That is, it can implement up to 256 different binary functions.

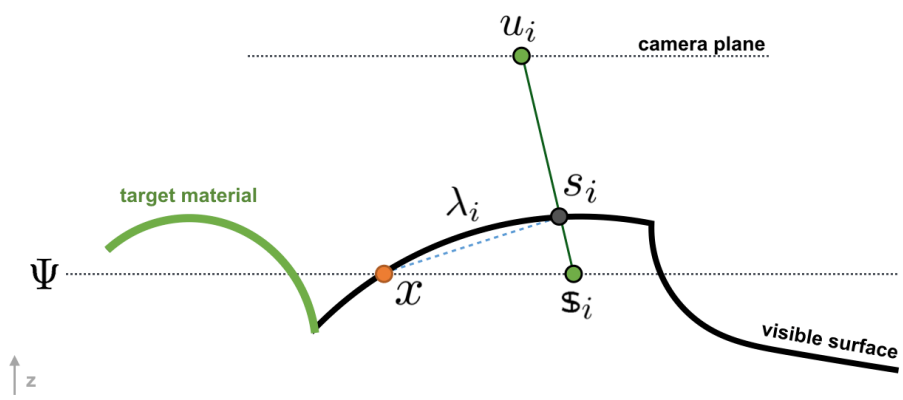


Figure 4.4: Overview of the spaces used for sampling. All sample points lie in the plane Ψ, defined by the current point x, and have corresponding screen space and visible surface positions.

Sampling Space I am only interested in sampling points along the visible surface. Hence, sampling on a sphere in view space as in Mittring (2007) is not appropriate. On the other hand, one of the major differences between this distance problem and ambient occlusion is that the scale of the lacrimal lake I want to recreate is known. Thus the maximum distance at which the lacrimal lake can be from the eyelids, Λ, is known as well. However, sampling in image space, as in Bavoil et al. (2008), does not take into account the distance of the camera to the eyeball. This can lead to inconsistent results at different distances and does not allow to take advantage of prior knowledge.

So, instead, I the set of potential sample points Ψ as:

$$\Psi = \{\mathbb{s}_i \mid \mathbb{s}_{i,z} = x_z,\ 1 \leq i \leq n_s\} \tag{4.6}$$

where x_z is the z coordinate of x in view camera coordinates. That is, the samples are picked in a plane that passes through x, parallel to the camera projection plane. I then cast, as in Mittring (2007), a ray from the camera to each sample $\mathbb{s}_i$ in the Ψ plane to find the intersecting visible geometry position s_i along the ray. Equations 4.2 and 4.3 describe how s_i can be efficiently computed.

The biggest advantage of this approach versus the image space sampling approach of Bavoil et al. (2008) is that the process becomes independent of the distance from the camera to the eyeball, while still avoiding the redundancy of the spherical sampling space used by Mittring (2007).

4.4 Sampling Methods

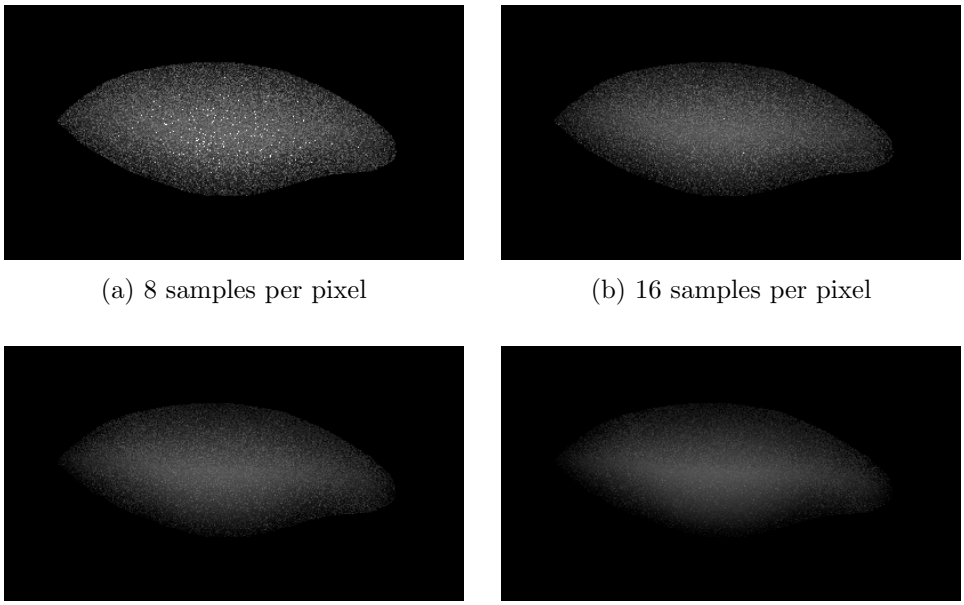
The major bottleneck of my approach is sampling, as randomly accessing textures on the GPU is computationally costly. Sampling across the entire screen is not an option. Thus, the sampling strategy chosen is of extreme importance.

In this section I describe three different sampling algorithms I developed, which vary greatly in the strategies used. However, they share the premise of searching in the plane Ψ, resorting to the same distance metric and detecting skin points as described in the previous Section 4.3.

4.4.1 Equiangular Sampling on a Circle

This first algorithm represents a naive approach heavily inspired by ambient occlusion methods. While it differs from previous work, due to the nature of the sampling space and the problem, I tried to keep it as close as possible to related work. It is presented as an example of why the sampling strategies used for ambient occlusion methods cannot be transfered to a distance computation problem.

Sampling Directions Pick n_s equiangular directions in view space along the Ψ plane and sample once in each direction around x. This results in the general formula:



(a) 8 samples per pixel (b) 16 samples per pixel

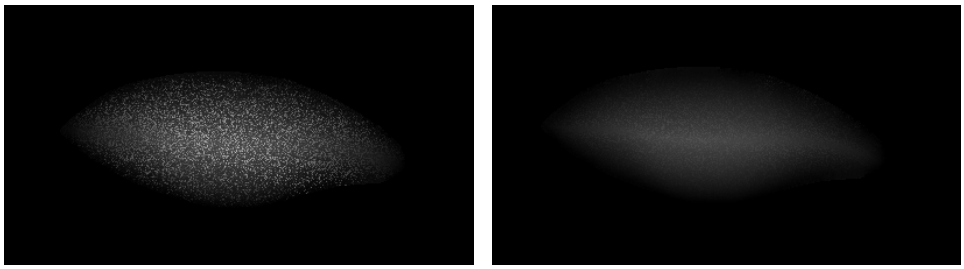
(c) 32 samples per pixel (d) 64 samples per pixel

Figure 4.5: Distance field of skin on a right eye region using different quality levels of equiangular sampling. As expected, results are not ideal.

$$\mathtt{s}_{i,[x,y]} = x_{[x,y]} + h_i \; \mu_i \tag{4.7}$$

where $\mathtt{s}_{i,[x,y]}$ are the x and y axis coordinates of $\mathtt{s}_i$ in view camera space, h is a set of 2D unit vectors defined equiangularly around the origin, and $\mu_i \in]0, \Lambda]$ is a scalar that indicates the sampling distance from x.

Sampling Randomization Sampling in the same n_s directions on every pixel will produce biased results for a low number of directions, which in turn can lead to artifacts. This is a very common problem in ambient occlusion algorithms. As increasing sampling per pixel is costly, a common strategy to solve the problem is randomizing sampling on a per pixel basis (Bavoil et al., 2008; Chapman, 2011; Mittring, 2007). This results in trading bias for high frequency noise, which is not problematic, as high frequency noise can be greatly reduced by a post process blurring step (see Figure 4.5).



(a) initial vertical pass (b) final horizontal pass

Figure 4.6: Distance field of skin on a right eye region using exponential line search with 8 samples per axis direction. Quality is superior for the same number of samples than the previous method.

I rotate the n_s directions at each surface position x using a randomly rotated kernel. I also randomize the sampling distances μ from x per sample, making the probability the same for all distances. All random values were obtained using a pseudo-random function that appears random in the screen image space.

4.4.2 Exponential Line Search

The main problem with the equiangular sampling approach, described in Section 4.4.1, is that it is better suited for estimating overall properties of the surrounding geometry, like ambient occlusion. However, I am looking for a local minimum, which is a fundamentally different problem. One could take advantage of previous samples of a pixel for the subsequent samples.

In this section I describe the use of line search to find the minimum along the view space axis directions. It provides less noisy results and much better distance metric precision than the previous algorithm.

Search Algorithm To perform line search I make one assumption: that a pixel closer to another in screen image space is likely to correspond to closer positions in view camera space. While this does not apply to all possible surfaces, the regions I am working with are reasonably smooth and orthogonal to the view direction, and as such it is a rational heuristic.

Let $h_{i,0}$ be a 2D vector in the Ψ plane, with some user defined length. My

objective is to find the point, in direction $h_{i,0}$, on which the other objects end and skin starts by searching along that direction. Let n_m be a user defined number of iteration steps, and $m \in [1, n_m]$. Define the first sample $\mathbb{s}_{i,0}$ as:

$$\mathbb{s}_{i,0,[x,y]} = x_{[x,y]} + h_{i,0} \tag{4.8}$$

Then, take up to n_m exponentially increasing steps until a skin point is sampled. That is, until $M(u_{i,m})$ holds true:

$$h_{i,m} = h_{i,m-1}\ \mu \tag{4.9}$$

$$\mathbb{s}_{i,m,[x,y]} = \mathbb{s}_{i,m-1,[x,y]} + h_{i,m} \tag{4.10}$$

where $\mu > 1$ is a user defined constant and $u_{i,m}$ are the screen space coordinates of $\mathbb{s}_{i,m}$. Once a skin point is found, it implies that the boundary of the skin is between $\mathbb{s}_{i,m}$ and $\mathbb{s}_{i,m-1}$. At that stage, invert the tracing direction and take exponentially decreasing steps:

$$h_{i,m} = \frac{h_{i,m-1}}{\mu} \tag{4.11}$$

$$\mathbb{s}_{i,m,[x,y]} = \mathbb{s}_{i,m-1,[x,y]} - h_{i,m} \tag{4.12}$$

Once a non-skin point is found (in other words, $M(u_{i,m})$ no longer holds true), invert the direction once again but keep taking decreasing steps. This direction inversion process continues until n_m samples have been made.

Axis Directions One could search in equiangular directions, as in Section 4.4.1. However, I am now making n_m samples in each direction, and thus I need to be more conservative about the number of sampled directions.

Thankfully, one can take advantage of the search directions by dividing the algorithm into two passes. On a first pass, I sample along one of the view space axis and store the result, which I call $\tilde{\lambda}$. Notice that any point in $\tilde{\lambda}$ contains not only information about that point, but also pertaining to the neighboring samples, which were located along the view space axis chosen. I exploit this fact: on a second pass, I sample perpendicularly to the previous sampling direction and use $\tilde{\lambda}$ for my line search boundary condition.

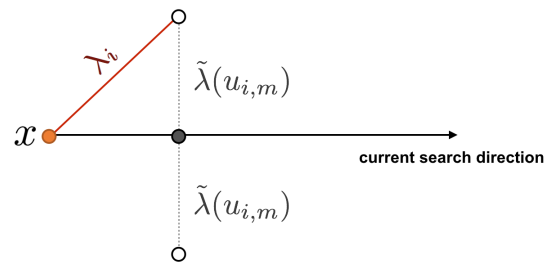


Figure 4.7: 2D schematic of the modified distance metric for a two-pass implementation.

To make use of $\tilde{\lambda}$, I define a modified version of the distance metric specified in Equation 4.4. $\tilde{\lambda}(u_{i,m})$ constitutes the distance from $\mathbf{s}_{i,m}$ to a near skin point along the sampled axis. I approximate the distance between x and that unknown point by assuming the three points make a 90° angle, as this angle is efficient to compute:

$$\lambda_i \approx \sqrt{|\boldsymbol{s}_{i,m} - x|^2 + \tilde{\lambda}(u_{i,m})^2} \tag{4.13}$$

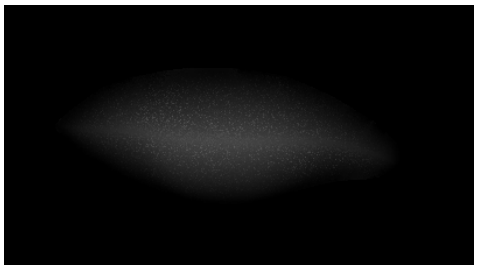
My objective is to find the local minimum of $\tilde{\lambda}_i$. Thus, I perform a line search on $\tilde{\lambda}_i$: I sample in exponentially increasing steps, and then decreasing steps as described for the first pass; however, I choose the sampling direction based on the variation of $\tilde{\lambda}_i$.

Direction Randomization Similarly to the method described in Section 4.4.1, sampling on the image axis can cause aliasing along those axis. Because I implemented $M(u)$ as a material mask texture, it further intensifies the problem, due to the aliasing already present on the texture.

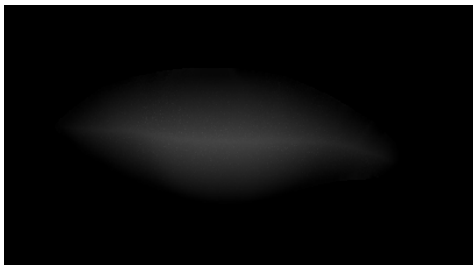
Hence, I randomly rotate the search directions per pixel, in the same manner I did for individual samples in Section 4.4.1. This produces manageable noise, which can be decreased with a blurring step, while still strongly reducing aliasing effects.

4.4.3 Backtracking Line Search

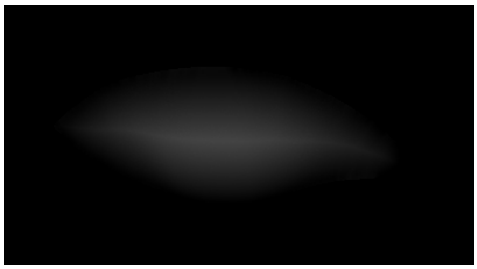
The major problem with the line search algorithm described in Section 4.4.2 is that it presents us with a strict compromise between three factors: max-



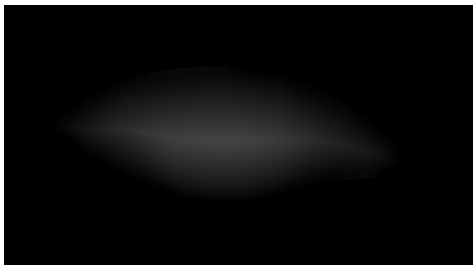
(a) 8 directions x 8 samples



(b) 16 directions x 16 samples



(c) 32 directions x 32 samples



(d) 64 directions x 64 samples

Figure 4.8: Distance field using backtracking line search. The indicated number of samples denotes the maximum number of samples for each search direction, and not the actual number of samples performed.

imum sampled distance Λ, metric precision μ and the number of samples per direction n_m. For example, when keeping n_m low, a low value of Λ will allow for a good distance metric precision, but will limit the maximum size of the lacrimal lake. Increasing the value of Λ will increase search space, but reduce precision. Furthermore, the exponential sampling does treat search space equally, as points nearby to the target material will always present higher distance precision than points farther away.

In this section, I improve upon the line search algorithm from Section 4.4.2. I discuss the use of backtracking to treat search space equally and makes a better use of samples by taking into account Λ, preemptively ignoring unpromising sampling directions and using strategies to reduce overall texture access per sample.

Screen Space Notice that the material detection is performed in screen space (see Section 4.3). Thus, it would be more efficient to line search in screen space, as computing sample positions in view space requires accessing the depth buffer (see Equation 4.3). However, the sampling needs to be independent of the distance between the camera and the skin object, and that information is not present in screen space.

Similarly to the method described in Section 4.4.1, let h be a set of 2D unit vectors defined equiangularly around the origin. For each vector h_i, representing a sampling direction in the Ψ plane, transverse from x by a length of Λ and project into screen space:

$$\mathbb{s}_{i,[x,y]} = x_{[x,y]} + h_i \, \Lambda \tag{4.14}$$

$$u_{i,0} = P \, \mathbb{s}_i \tag{4.15}$$

If $M(u_{i,0})$ is false, meaning the initial sample $\mathbb{s}_i$ is not a skin point, assume that there is not skin between x and $\mathbb{s}_i$, and thus do not search in that direction. Otherwise, define our search region as a line in screen space between x and $\mathbb{s}_i$. As this line always has a length of Λ in view space, the searching process is independent of the distance between the camera and the skin object even in screen space.

Search Algorithm To treat search space equally, I perform line search using a "divide and conquer" approach. Let the extremities of this initial search space be $a_{i,0}$ and $b_{i,0}$:

$$a_{i,0} = P \, x \ , \ b_{i,0} = u_{i,0} \tag{4.16}$$

At each $m \in [1, n_m]$ iteration step, sample the center of the search region:

$$u_{i,m} = \frac{a_{i,m-1} + b_{i,m-1}}{2} \tag{4.17}$$

Then, divide the search space in half. If $u_{i,m}$ is a skin point, take the first half of the region as our new search region:

$$a_{i,m} = a_{i,m-1} \ , \ b_{i,m} = u_{i,m} \tag{4.18}$$

Otherwise, take the second half:

$$a_{i,m} = u_{i,m} \ , \ b_{i,m} = b_{i,m-1} \tag{4.19}$$

Having completed this process, the closest skin sample to x in screen space is b_{i,n_m}. I make the assumption that this is also true in the view space. Thus, proceed to reconstruct the surface position rendered at b_{i,n_m} and compute the distance metric:

$$s_{i,n_m} = P^{-1} \left[b_{i,n_m}, D(b_{i,n_m}), 1 \right] \tag{4.20}$$

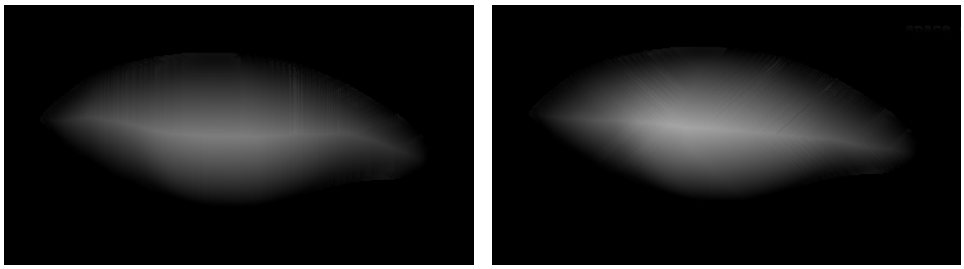
$$\lambda_i = |s_{i,n_m} - x| \tag{4.21}$$

4.5 Results

Overall, the three sampling algorithms behaved as expected. Equiangular sampling produced unsatisfactory distance fields, confirming that existing screen space sampling techniques were not appropriate to our problem. Backtracking line search proved to be an improvement over the simple line search along the axis.

The quality of the results of backtracking line search on a single pass were favorable enough that dividing the algorithm into two passes as in Section 4.4.2 was not worth the overhead. In fact, blurring was no longer very effective in reducing noise. It proved to be much more efficient to use an anti-aliasing method such as Fast Approximate Anti-Aliasing (Lottes, 2011). This suggests that the constraint on the quality of the sampling field might no longer so much the sampling strategy, but the aliasing already present in the material mask.

Performance For each of the three sampling methods, I analyzed the computation time per frame, the number of texture accesses performed per texel in the area of relevancy of the field and computed the per texel error of the obtained distance field. For the ground truth field, I used backtrack sampling with 128 samples per 128 directions as an approximation. The tests were ran on an early 2011 15-inch *MacBook Pro*, sporting a *AMD Radeon HD* 6490M GPU. I used the *Digital Emily* data provided by the



(a) 4 sampling directions (b) 6 sampling directions

Figure 4.9: Distance field using backtracking line search without direction randomization. Aliasing is visible along the search lines.

WikiHuman project (Ghosh et al., 2011; WikiHuman) for my skin mesh, and these metrics were computed from multiple camera perspectives. Their average is shown in Table 4.1.

Testing the method on meshes with different number of faces or vertices is not relevant, as my method does not depend on object complexity. As I only took into account texels in the area of relevancy, distance to the character is also not relevant.

	Equiangular 64	Exponential 8x8	Backtracking 8x8
Runtime (ms)	0.0601 ± 0.0413	0.0215 ± 0.0206	0.0192 ± 0.0159
Accesses	94.0233 ± 6.9828	101.8739 ± 4.0530	46.0344 ± 7.1043
Visual Error	3.3858%	0.4597%	0.3681%

Table 4.1: Average performance comparison of the three sampling algorithms running on a single render pass. Sampling algorithm settings chosen to allow for a maximum of 64 samples per texel.

Interestingly, exponential line search presented an higher texture access count than equiangular sampling. This implies that exponential line search is finding more relevant samples (skin points) and, as such, computing the distance metric more frequently than exponential sampling does.

Meanwhile, as expected, backwards line search presented the best performance, both in computation time and memory access. This is justified

(a) (b)

Figure 4.10: Distance field at two slightly different view angles. Our method can handle sharp angles of view (a). However, if not enough geometry is visible for the method to sample, artifacts will become visible (b).

by two reasons: first, it can preemptively ignore sampling directions, while the other two methods always perform same number of samples regardless. Second, it searches in screen space, thus avoiding accessing the depth buffer on each sample. The other two methods require two texture accesses per sample; once on the mask texture, to check for skin, and once on the depth buffer, to compute the surface position.

Lacrimal Lake Generation As an use case and proof of concept, I present my fourth contribution: a method for procedurally creating lacrimal lakes using screen space distance fields. Figure 4.11 shows lacrimal lakes rendered on the *Digital Emily* character (Ghosh et al., 2011; WikiHuman) using each of the sampling methods.

I naively defined the lacrimal lake depth as a first order linear function of the distance to the skin. Surface normals where computed by deriving the visible surface view space positions in the fragment shader. As an added efficiency bonus, when rendering lacrimal lakes one only needs to compute the distance fields on the surface of the globes.

Limitations As mentioned before, the major limitation of my method comes from the aliasing already present on the material mask M. This can create slight value changes along the sampling directions (see Figure 4.9, for example). While these are very minor, they can be problematic when computing the derivatives of distance fields. Thankfully, I was able to overcome this problem on all the three sampling algorithms using randomization.

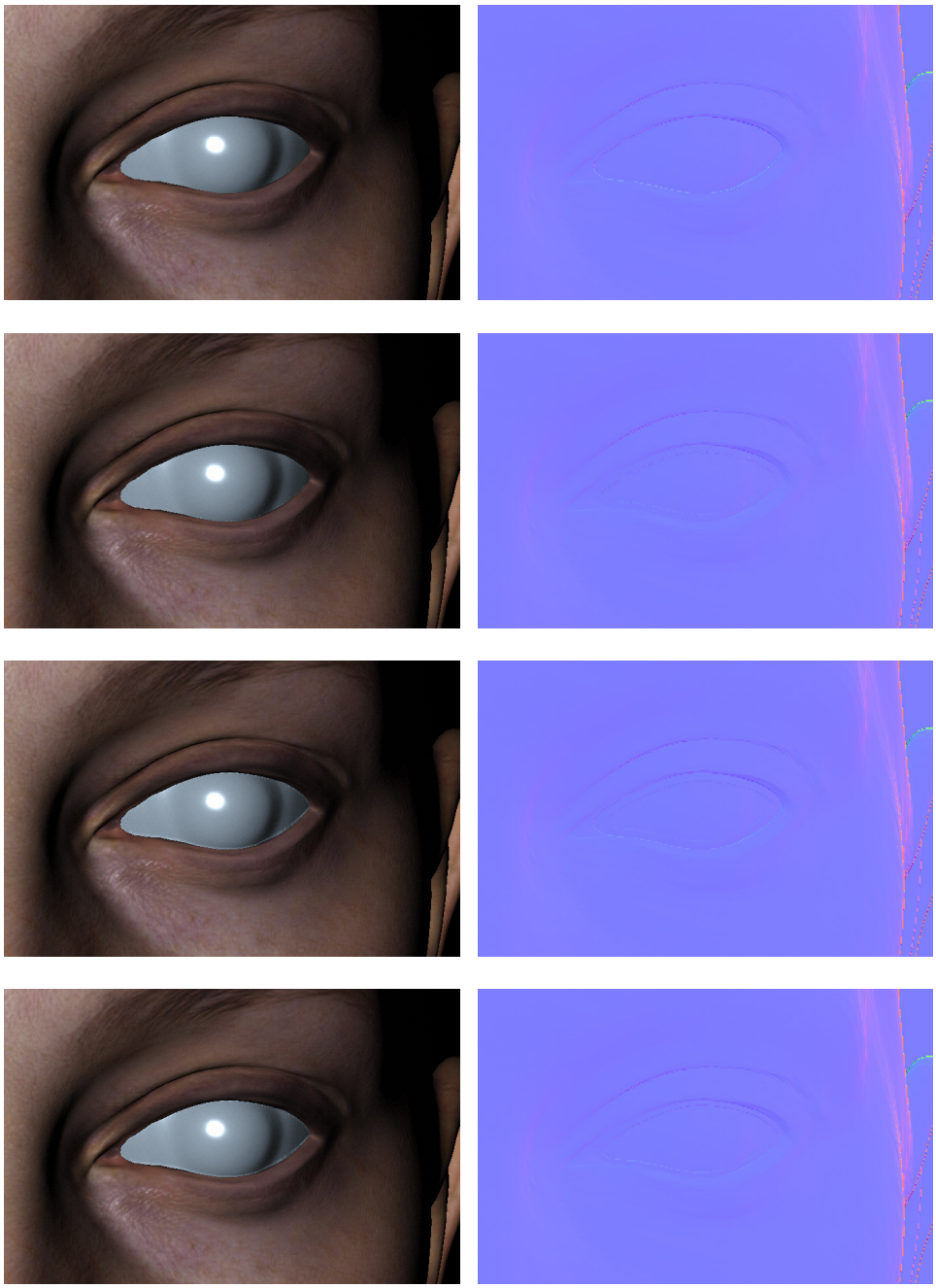


Figure 4.11: Lacrimal lakes generated procedurally using the three methods. From top to bottom: no method, equiangular sampling, exponential line search, backtracking line search.

Another issue is that our method only takes into account the visible geometry. Sharp angles with the skin surface can create visible artifacts that flicker as view perspective changes (see Figure 4.10). This happens when not enough skin geometry is visible, and thus being sampled, to generate informed results.

Chapter 5

Conclusion

In this thesis I have presented work for automatically animating facial details which, while small in scale, play a crucial role in character realism. I have shown that our methods can be used in real-time scenarios and interactively affected by user control. Generalization to different digital characters or globes has also been explored.

The major advantage of our methods is that they improve current interactive human facial animation without additional artist input. The work described in Chapter 3 automatically ensures that skin in the eye region and gaze direction always match. My contributions allow for our system to automatically animate all the smaller features of the region and to be controlled with little to no artist intervention. As recorded actor performance is not viable for animation of interactive characters, recreating these intricacies would otherwise require an artist to meticulously build blendshapes to represent them, with various degrees of success. The same is true for the work described in Chapter 4, which can be easily applied on top of existing rendering frameworks to generate the lacrimal lakes of all characters in scene. As performing fluid simulation is too computationally costly for real-time performance, providing a character with a lacrimal lake would normally require an artist to manually animate a liquid mesh, which would have to react to all the possible character facial poses.

5.1 Discussion and Future Directions

I now make a number of general observations about possible future work in the topics of this thesis.

One of the limitations in my analysis of the system described in Chapter 3 is the one-camera capture system mentioned in Section 3.4, that is unable to track the skin vertices close to the edges of the eyelids or in the canthus. These vertices had to be extrapolated, as described in Section 3.6.

Thus, it would be of some significance to see how the rest of the system, in particular my contributions, would scale to more costly capture setups, which could track those vertices or provide higher precision. On the same note, the major limitation of our reduced coordinate representation of skin motion, described in Section 3.2, is that it can't represent the finer details of the human face. These had to be reconstructed using my heuristic model, described in Section 3.8. Possible research paths could be to explore different heuristics and compare their respective effectiveness in the model, or find alternative representations that do not suffer from the same problem.

Regarding Chapter 4, distance field generation in screen space opens many unexplored possibilities. A clear research potential is to explore other use cases for these fields, besides lacrimal lake procedural generation. Another is to improve upon the distance field generation itself: for example, I have yet to analyze which are the effects of using 2D screen space distance as the distance metric. This would reduce the number texture accesses. On the same note, one could explore rendering distance fields at half resolution. Combining our screen space approach with the geometry based approaches for mesh border generation, mentioned in Section 4.2, has also never been explored; even though this would make field generation dependent on mesh complexity.

Finally, considering the case of lacrimal lake generation, my results were only a proof of concept, and defining an effective procedural function for this effect is still an open problem.

Bibliography

Javier San Agustin, Arantxa Villanueva, and Rafael Cabeza. Pupil brightness variation as a function of gaze direction. In *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, ETRA '06, pages 49–49, New York, NY, USA, 2006. ACM. ISBN 1-59593-305-0.

Oleg Alexander, Graham Fyffe, Jay Busch, Xueming Yu, Ryosuke Ichikari, Andrew Jones, Paul Debevec, Jorge Jimenez, Etienne Danvoye, Bernardo Antionazzi, et al. Digital ira: Creating a real-time photoreal digital actor. In *ACM SIGGRAPH 2013 Posters*, page 1. ACM, 2013.

Clare Anderson, AW Wales, and James A Horne. Pvt lapses differ according to eyes open, closed, or looking away. *Sleep*, 33(2):197–204, 2010.

John Anderson. 'polar express' derails in zombie land. Newsday, 2004.

A Terry Bahill, Michael R Clark, and Lawrence Stark. The main sequence, a tool for studying human eye movements. *Mathematical Biosciences*, 24 (3):191–204, 1975.

Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks*, SIGGRAPH '08, pages 22:1–22:1, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-343-3.

Wolfgang Becker. The neurobiology of saccadic eye movements. metrics. *Reviews of oculomotor research*, 3:13, 1989.

T. Beeler, F. Hahn, D. Bradley, B. Bickel, P. Beardsley, C. Gotsman, R.W. Sumner, and M. Gross. High-quality passive facial performance capture using anchor frames. *ACM Trans. Graph.*, 30(4):75:1–75:10, July 2011.

Pascal Bérard, Derek Bradley, Maurizio Nitti, Thabo Beeler, and Markus Gross. High-quality capture of eyes. *ACM Trans. Graph.*, 33(6):223:1–223:12, November 2014. ISSN 0730-0301.

Amit Bermano, Thabo Beeler, Yeara Kozlov, Derek Bradley, Bernd Bickel, and Markus Gross. Detailed spatio-temporal reconstruction of eyelids. *ACM Trans. Graph.*, 34(4):44:1–44:11, July 2015. ISSN 0730-0301.

Bernd Bickel, Mario Botsch, Roland Angst, Wojciech Matusik, Miguel Otaduy, Hanspeter Pfister, and Markus Gross. Multi-scale capture of facial geometry and motion. *ACM Transactions on Graphics (TOG)*, 26 (3):33, 2007.

Gunnar Blohm, Lance M Optican, and Philippe Lefevre. A model that integrates eye velocity commands to keep track of smooth eye displacements. *Journal of computational neuroscience*, 21(1):51–70, 2006.

Judee K Burgoon, Laura K Guerrero, and Kory Floyd. *Nonverbal communication*. Routledge, 2016.

John Chapman. Screen space ambient occlusion tutorial. http://john-chapman-graphics.blogspot.ca/2013/01/ssao-tutorial.html, 2011.

Kai-Chun Chen, Pei-Shan Chen, and Sai-Keung Wong. A hybrid method for water droplet simulation. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, VRCAI '12, pages 341–344, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1825-9.

Douglas Decarlo and Dimitris Metaxas. Optical flow constraints on deformable models with applications to face tracking. *International Journal of Computer Vision*, 38(2):99–127, 2000.

Joe Demers. Depth of field: A survey of techniques. *GPU Gems*, 1(375): U390, 2004.

Eugene d'Eon and David Luebke. Advanced techniques for realistic real-time skin rendering. *GPU Gems*, 3(3):293–347, 2007.

Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. *ACM Trans. Graph.*, 24(3):1032–1039, July 2005. ISSN 0730-0301. doi: 10.1145/1073204.1073308. URL `http://doi.acm.org/10.1145/1073204.1073308`.

William Alexander Newman Dorland. *Dorland's medical dictionary*. Saunders Press Philadelphia, PA, 1980.

Paul Ekman and Wallace V Friesen. *Facial action coding system*. Consulting Psychologists Press, Stanford University, Palo Alto, 1977.

Kenny Erleben and Henrik Dohlmann. Signed distance fields using single-pass gpu scan conversion of tetrahedra. In *Gpu Gems 3*. Addison-Wesley, 2008.

Rose Eveleth. Robots: Is the uncanny valley real? BBC Future, 2013.

Craig Evinger, Karen A Manning, and Patrick A Sibony. Eyelid movements. mechanisms and normal data. *Investigative ophthalmology & visual science*, 32(2):387–400, 1991.

A. Fick. Die bewegung des menschlichen augapfels. *Z. Rationelle Med.*, (4): 101–128, 1854.

Tamar Flash and Neville Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *The journal of Neuroscience*, 5(7):1688–1703, 1985.

Guillaume François, Pascal Gautron, Gaspard Breton, and Kadi Bouatouch. Anatomically accurate modeling and rendering of the human eye. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

Yasutaka Furukawa and Jean Ponce. Dense 3d motion capture from synchronized video streams. In *Image and Geometry Processing for 3-D Cinematography*, pages 193–211. Springer, 2010.

Graham Fyffe, Andrew Jones, Oleg Alexander, Ryosuke Ichikari, Paul Graham, Koki Nagano, Jay Busch, and Paul Debevec. Driving high-resolution facial blendshapes with video performance capture. In *ACM SIGGRAPH 2013 Talks*, page 33. ACM, 2013.

Pablo Garrido, Levi Valgaerts, Chenglei Wu, and Christian Theobalt. Reconstructing detailed dynamic face geometry from monocular video. *ACM Trans. Graph.*, 32(6):158, 2013.

Abhijeet Ghosh, Graham Fyffe, Borom Tunwattanapong, Jay Busch, Xueming Yu, and Paul Debevec. Multiview face capture using polarized spherical gradient illumination. *ACM Transactions on Graphics (TOG)*, 30(6): 129, 2011.

Henry Gray. *Gray's Anatomy: With original illustrations by Henry Carter*. Arcturus Publishing, 2009.

Mark R Harwood, Laura E Mezey, and Christopher M Harris. The spectral main sequence of human saccades. *The Journal of neuroscience*, 19(20): 9098–9106, 1999.

Shoya Ishimaru, Kai Kunze, Koichi Kise, Jens Weppner, Andreas Dengel, Paul Lukowicz, and Andreas Bulling. In the blink of an eye: combining head motion and eye blink frequency for activity recognition with google glass. In *Proceedings of the 5th augmented human international conference*, page 15. ACM, 2014.

Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

Laurent Itti, Nitin Dhavale, and Frederic Pighin. Realistic avatar eye and head animation using a neurobiological model of visual attention. In *Optical science and technology, SPIE's 48th annual meeting*, pages 64–78. International Society for Optics and Photonics, 2004.

Henrik Wann Jensen and Juan Buhler. A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph.*, 21(3):576–581, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566619. URL `http://doi.acm.org/10.1145/566654.566619`.

Jorge Jimenez, Veronica Sundstedt, and Diego Gutierrez. Screen-space perceptual rendering of human skin. *ACM Transactions on Applied Perception (TAP)*, 6(4):23, 2009.

Jorge Jimenez, Timothy Scully, Nuno Barbosa, Craig Donner, Xenxo Alvarez, Teresa Vieira, Paul Matts, Verónica Orvalho, Diego Gutierrez, and Tim Weyrich. A practical appearance model for dynamic facial color. *ACM Transactions on Graphics (TOG)*, 29(6):141, 2010a.

Jorge Jimenez, David Whelan, Veronica Sundstedt, and Diego Gutierrez. Real-time realistic skin translucency. *IEEE Computer Graphics and Applications*, 30(4):32–41, 2010b.

Jorge Jimenez, Károly Zsolnai, Adrian Jarabo, Christian Freude, Thomas Auzinger, Xian-Chun Wu, Javier von der Pahlen, Michael Wimmer, and Diego Gutierrez. Separable subsurface scattering. *Computer Graphics Forum*, 2015.

Murray W Johns, Andrew Tucker, Robert Chapman, Kate Crowley, and Natalie Michael. Monitoring eye and eyelid movements by infrared reflectance oculography to measure drowsiness in drivers. *Somnologie-Schlafforschung und Schlafmedizin*, 11(4):234–242, 2007.

Yvonne Jung and Johannes Behr. Gpu-based real-time on-surface droplet flow in x3d. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 51–54, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-432-4.

Seth Koterba, Simon Baker, Iain Matthews, Changbo Hu, Jing Xiao, Jeffrey Cohn, and Takeo Kanade. Multi-view aam fitting and camera calibration. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 511–518. IEEE, 2005.

Michael Lam and Gladimir Baranoski. A predictive light transport model for the human iris. In *Computer Graphics Forum*, volume 25, pages 359–368. Wiley Online Library, 2006.

Aaron Lefohn, Brian Budge, Peter Shirley, Richard Caruso, and Erik Reinhard. An ocularist's approach to human iris synthesis. *Computer Graphics and Applications, IEEE*, 23(6):70–75, 2003.

R John Leigh and David S Zee. *The neurology of eye movements*. Oxford University Press, USA, 2015.

Joe Letteri. Weta digital on their digital paul walker. FXGuide. Retrieved from `https://www.fxguide.com/fxpodcasts/fxpodcast-298-weta-digital-on-their-digital-paul-walker/`, Dec. 2015.

Duo Li, Shinjiro Sueda, Debanga R Neog, and Dinesh K Pai. Thin skin elastodynamics. *ACM Transactions on Graphics (TOG)*, 32(4):49, 2013a.

Hao Li, Robert W Sumner, and Mark Pauly. Global correspondence optimization for non-rigid registration of depth scans. *Computer graphics forum*, 27(5):1421–1430, 2008.

Hao Li, Jihun Yu, Yuting Ye, and Chris Bregler. Realtime facial animation with on-the-fly correctives. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2013)*, 32(4), July 2013b.

Timothy Lottes. Fxaa. *NVIDIA white paper. Retrieved from `http://goo.gl/fr3plh`*, 2011.

Tom Mertens, Jan Kautz, Philippe Bekaert, Frank Van Reeth, and Hans-Peter Seidel. Efficient rendering of local subsurface scattering. In *Computer Graphics Forum*, volume 24, pages 41–49. Wiley Online Library, 2005.

Martin Mittring. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM. ISBN 978-1-4503-1823-5.

Masahiro Mori, Karl F MacDorman, and Norri Kageki. The uncanny valley [from the field]. *Robotics & Automation Magazine, IEEE*, 19(2):98–100, 2012.

Tsuyoshi Moriyama, Takeo Kanade, Jing Xiao, and Jeffrey F Cohn. Meticulously detailed eye region model and its application to analysis of facial images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(5):738–752, 2006.

Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-659-5.

Koki Nagano, Graham Fyffe, Oleg Alexander, Jernej Barbic, Hao Li, Abhijeet Ghosh, and Paul Debevec. Skin microstructure deformation with displacement map convolution. *ACM Transactions on Graphics (TOG)*, 34(4):109, 2015.

Tamami Nakano and Shigeru Kitazawa. Eyeblink entrainment at breakpoints of speech. *Experimental brain research*, 205(4):577–581, 2010.

National Eye Institute. Diagram of eye, 2016.

OpenStax. Anatomy & physiology, June 2013.

Peter E Oppenheimer. Real time design and animation of fractal plants and trees. In *ACM SiGGRAPH Computer Graphics*, volume 20, pages 55–64. ACM, 1986.

Vitor F. Pamplona, Manuel M. Oliveira, and Gladimir V. G. Baranoski. Photorealistic models for pupil light reflex and iridal pattern deformation. *ACM Trans. Graph.*, 28(4):106:1–106:12, September 2009. ISSN 0730-0301.

Eric Penner and George Borshukov. Pre-integrated skin shading. *Gpu Pro*, 2:41–54, 2011.

Dmitriy Pinskiy and Erick Miller. Realistic eye motion using procedural geometric methods. *SIGGRAPH 2009: Talks*, page 75, 2009.

Jasia Reichardt. *Robots: fact, fiction+ prediction*. Thames & Hudson Ltd., London, 1978.

Jareck Rossignac and Maarten van Emmerik. Hidden contours on a framebuffer. In *Proceedings of the 7th workshop on computer graphics hardware*, 1992.

K Ruhland, S Andrist, JB Badler, CE Peters, NI Badler, M Gleicher, B Mutlu, and R McDonnell. Look me in the eyes: A survey of eye and gaze animation for virtual agents and artificial systems. In *Eurographics 2014-State of the Art Reports*, pages 69–91. The Eurographics Association, 2014.

Mark A Sagar, David Bullivant, Gordon D Mallinson, and Peter J Hunter. A virtual environment and model of the eye for surgical simulation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 205–212. ACM, 1994.

Mathieu Sanchez, Oleg Fryazinov, and Alexander Pasko. Efficient evaluation of continuous signed distance to a polygonal mesh. In *Proceedings of the 28th Spring Conference on Computer Graphics*, SCCG '12, pages 101–108, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1977-5. doi: 10.1145/2448531.2448544. URL `http://doi.acm.org/10.1145/2448531.2448544`.

Fuhao Shi, Hsiang-Tao Wu, Xin Tong, and Jinxiang Chai. Automatic acquisition of high-fidelity facial performances using monocular videos. *ACM Trans. Graph.*, 33(6):222:1–222:13, November 2014. ISSN 0730-0301.

JH Skotte, Jacob Klenø Nøjgaard, LV Jørgensen, KB Christensen, and G Sjøgaard. Eye blink frequency during different computer tasks quantified by electrooculography. *European journal of applied physiology*, 99(2): 113–119, 2007.

John A Stern, Larry C Walrath, and Robert Goldstein. The endogenous eyeblink. *Psychophysiology*, 21(1):22–33, 1984.

Natalya Tatarchuk. Advances in real-time rendering in 3d graphics and games i. In *ACM SIGGRAPH 2009 Courses*, page 4. ACM, 2009.

D. Terzopoulos and K. Waters. Physically-based facial modelling, analysis, and animation. *The Journal of Visualization and Computer Animation*, 1(2):73–80, December 1990. ISSN 1049-8907.

Laura C Trutoiu, Elizabeth J Carter, Iain Matthews, and Jessica K Hodgins. Modeling and animating eye blinks. *ACM Transactions on Applied Perception (TAP)*, 8(3):17, 2011.

Vagia Tsiminaki, Jean-Sébastien Franco, Edmond Boyer, et al. High resolution 3d shape texture from multiple videos. In *International Conference on Computer Vision and Pattern Recognition*, 2014.

Ed Ulbrich. How benjamin button got his face. *TED talk. Retrieved from* `https://www.ted.com/talks/ed_ulbrich_shows_how_benjamin_button_got_his_face`, 2009.

Levi Valgaerts, Chenglei Wu, Andrés Bruhn, Hans-Peter Seidel, and Christian Theobalt. Lightweight binocular facial performance capture under uncontrolled lighting. *ACM Trans. Graph.*, 31(6):187, 2012.

Wijnand van Tol and Arjan Egges. Real-time crying simulation for 3d characters. 2012.

Artur Vill. Eye texture raytracer. `https://www.chromeexperiments.com/experiment/eye-texture-raytracer`, Feb. 2014.

Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929, July 2005. ISSN 0730-0301.

Thibaut Weise, Sofien Bouaziz, Hao Li, and Mark Pauly. Realtime performance-based facial animation. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2011)*, 30(4), July 2011.

Axel Weissenfeld, Kang Liu, and Jörn Ostermann. Video-realistic image-based eye animation via statistically driven state machines. *The Visual Computer*, 26(9):1201–1216, 2010.

WikiHuman. `http://gl.ict.usc.edu/Research/DigitalEmily2/`, 2015.

Chenglei Wu, Bennett Wilburn, Yasuyuki Matsushita, and Christian Theobalt. High-quality shape from multi-view stereo and shading under general illumination. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 969–976. IEEE, 2011.

Jing Xiao, Simon Baker, Iain Matthews, and Takeo Kanade. Real-time combined 2d+ 3d active appearance models. In *CVPR (2)*, pages 535–542, 2004.

Alfred L Yarbus. *Eye movements during perception of complex objects*. Springer, 1967.

Sang Hoon Yeo, Martin Lesmana, Debanga R Neog, and Dinesh K Pai. Eyecatch: simulating visuomotor coordination for object interception. *ACM Transactions on Graphics (TOG)*, 31(4):42, 2012.

Yizhong Zhang, Huamin Wang, Shuai Wang, Yiying Tong, and Kun Zhou. A deformable surface model for real-time water drop animation. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1281–1289, August 2012. ISSN 1077-2626.