Shrek: The Story Behind the Screen

APRIL 2001

Contents

Preface 3

From *Antz* to *Shrek* 5

- **Story and Editorial** 5
- **Visual Development** 6
- **Modeling** 6
- **Surfacing** 6
- **Character Technical Development** 6
- **Layout** 7
- **Character Animation** 8
- **Effects** 8
- **Matte Painting** 8
- **Lighting** 8
- **2D Paint** 9
- **Infrastructure** 9

Production Design for Digital Motion Pictures 11

- **Design Process Breakdown** 11
- **Design Breakdown** 12

Generic Character Variations 15

- **Generic Characters** 15
  First Considerations 16
- **Generic Character Setup** 22
  Body 22
  Face 23
- **Wardrobe** 26
- **Hair** 27
- **Using the System** 28
- **Lighting and FX** 32
  Shading Generic Characters 33
  Crowds 35
- **Acknowledgments** 37

The Use of Simulation 39

- **Simulation?** 39
  Why Simulate 39
  Why Not 40
  The Goal 41
- **Types of Simulations** 41
  Instancing of Pre-computed Simulation Elements 42
  Hero Simulation 44
  Pose Driven Dynamics 47
  Minimally Interactive Dynamic Systems 49
  Turn-key Simulation 51
- **Conclusion** 51

***Shrek* Effects** 53

***Shrek* Effects—Flames and Dragon Fireballs** 55

Introduction 55

The Flame Implementation 56
Features of Flames 56
Particles Technique 56
Modeled Surface Technique 58
Comparisons Between the Particle and Modeled Surface Techniques 58
Our Approach 58

The Dragon Fireball Implementation 59
What Should Dragon Fire Look Like? 59
Features of Fireballs 59
The Dynamics of the Dragon Fireball 60
The Simulation Techniques 60
The Rendering of the Fireball 62

Conclusion 66

Future Work 66

Acknowledgments 66

***Shrek* Effects—Animating Trees** 67

Introduction 67

Creating Trees 68

Representing Trees 68

Deforming Trees 70
Deforming a Branch 71
Maintaining Tree Structure 72

Conclusions 74

Bibliography 74

***Shrek* Effects—Mud and Water** 75

The Flu System 75

Challenges for *Shrek* 76

Mixing Fluids 76

Enhancing Fluid Simulations 77

Getting Objects Wet 79

Rendering 79

Conclusion 80

Suggested Reading 80

***Shrek* Lighting** 81

Art 81
The Swamp 83
Duloc 85
Redwood Forest 85
Sycamore Forest 85
Pine Forest 87
Dragon's Keep 87

Lighting Design Principles 89
1. Keep it Real 89
2. Keep it Rich 90
3. Sweat the Values 90
4. Shape Shape Shape 90

Technology 91
Skin and Hair 91
Complement Lights 93

Pipeline 95
Library File Structure 95
Keep 'em Honest 95

In the Future 96

CG Infrastructure 97

Purpose of the Feature Pipeline Infrastructure 97

Components of the Feature Infrastructure 98
Hardware 98
Software 98

Job Navigation and Structure 98
Resource Management 99
Asset Management 99

Feature Animation Project Cycles 100

Biographies 101

Preface

Andrew Adamson, Director

To produce a feature-length computer-generated film requires the integration of art and technology. This course describes the chronology of producing such a film—the technological infrastructure required to support a production pipeline that starts with the artistic vision, continues with technological and design development and implementation, and concludes with a finished film.

Shrek is the story of a cynical, green, ugly ogre whose swamp is overrun by annoying fairy tale creatures—the elves, mice, pigs, and wolves that are familiar to us from our childhood storybooks. In an attempt to save his home, Shrek sets out to confront Lord Farquaad, ruler of Duloc, who has banished all the fairy tale misfits from Duloc in order to create his own perfect world. Along the way, Shrek is befriended by a wise-cracking donkey. He's told by Lord Farquaad that he can have his swamp back, sans fairy tale creatures, if he saves a beautiful princess from a fire-breathing dragon. He agrees, and sets out with Donkey to rescue the princess, not knowing her deep, dark secret. In the end, he learns to love and be loved.

Unlike previous CG movies, *Shrek* involved a world that wasn't miniaturized. Because the main characters were on a quest, multiple environments had to be created. We wanted those environments to fit within our vision of a fairy tale storybook world, that, although stylized, would be as rich and believable as our own. Once the audience was transported into that world, it had to be interesting and real enough to make them want to stay.

Because we were parodying so many fairy tales, we had to create multiple characters of different species, including some humans. These characters, although not given starring roles in the film, still had to be capable of very subtle acting. They required complex musculatures, believable skin, clothing, fur or hair, and voices to bring them to life. And they had to be recognizable in their fairy tale roles.

As an artistic and technical achievement, we think *Shrek* lived up to our expectations, and we hope you'll find this course lives up to yours.

From *Antz* to *Shrek*

Ken Bielenberg, Visual Effects Supervisor

In order to produce our first computer-generated movie, *Antz*, PDI developed a vast array of tools and a production pipeline that seemed applicable to use in any computer generated feature. *Shrek*, however, introduced a new set of challenges even greater than those that had to be overcome in *Antz*—richer environments, CG humans, and extensive special effects. The production pipeline had to allow for dozens of animators to work on various sequences and shots in a logical progression, but also allow for shots to be reworked or elements added at a later time. This portion of the course provides an overview of the departments with an emphasis on the key artistic and technical challenges specific to the making of *Shrek*.

Story and Editorial

Story Department is the first step for any computer-animated film, and for *Shrek* it was no different. Once the script was ready, it was broken down into parts or numbered sequences. Each sequence was given to a story artist who illustrated the storyboard panels for that entire sequence. Once the sequence was approved by the film's Directors, the sequence's storyboard panels were numbered and sent to Editorial.

Editorial then shot each storyboard onto video to be edited on the Avid. Dissolves, pans, and sound effects were added, along with recordings of scratch voices, to make a story reel. Story and Editorial coordinated to make necessary revisions as necessary to accommodate script and story changes, directorial requests, timing, and dialogue changes. Later, the actors' voices were recorded to replace the scratch dialogue. Once completed, the story reel was used by other departments as a guide for their creative work on the movie.

Visual Development

The art direction and design process for *Shrek* is described in the following chapter, "Production Design for Digital Motion Pictures."

Modeling

Modeling was provided drawings of the environments and props for *Shrek* created by the Art Department. Basing their work on the drawings, Modelers either sculpted characters and then digitized them or built models directly in the computer.

Once the digitized information was captured, it was then converted into PDI's proprietary patch-based format and processed for use in production.

Surfacing

The Surfacing Department defined all the visual components that made up the surfaces of all the rendered models used in *Shrek*. To do this, they created maps and assigned materials for attributes such as color and texture, bump and/or displacement, specularity and transparency, ambience and translucency, fur, and so on. A PDI-proprietary 3D paint program allowed surfacers to paint all the visible parts of a model using the colors and patterns associated with that object. When "unwrapped" from its model, the map was just a rectangular-shaped image.

Surfacers utilized many of the different map and material shaders inside PDI's lighting tool, often combining different shaders to get a desired effect. For example, Shrek's tunic had a velvet material shader that helped it look more like soft cloth and fur growing at it's edges to make it look more tattered and real.

Working with the Lighting, Modeling, and sometimes the Matte Painting departments, Surfacing helped determine the movie's look. In the beginning of its development, *Shrek*, for example, was supposed to be "painterly", but it quickly became more realistic as work progressed.

Character Technical Development

For *Shrek*, Character TDs defined character setup standards, developed character setup technology in conjunction with the R&D Department, set up the characters, and supported the characters throughout the production.

During development, a leading Character TD developer defined the project's technical requirements and worked closely with R&D to develop specific character setup software. After development was well underway, Character TDs began implementing the technical solutions in the character environment.

- Character models were digitized and cleaned up by the Modeling Department under direct supervision from a Character TD
- Directing Character Animators negotiated the requirements for character motion or animation systems with the Character TDs
- Production Engineering worked with Character TDs to establish and maintain animation pipeline standards for character setups
- Character Animators tested the production character setups.

Character Animators were the biggest "clients" of the Character TDs' work. Character TDs maintained a constant review process with Character Animators concerning character functionality and deformation quality. Throughout the production schedule, Character TDs assisted Animators with character setup problems.

For a detailed description of how character technical development was done for non-hero characters in *Shrek*, see "Generic Character Variations."

Layout

The Layout Department for *Shrek* was made of up two groups: Rough Layout and Final Layout. Rough Layout artists were responsible for re-creating the look of the storyboards in the computer. They did the initial camera work and blocked the characters. Their environment consisted of rough models that were simpler than the "real" models (which were still in development in the Modeling Department).

Final Layout then put in the "real" models and swapped out the "rough" characters for "real" characters. Putting in the final models and characters often pmeant that the camera needed to be adjusted slightly. Once a sequence was in the final layout stage, the Art Director would add set dressing such as bushes, trees, tree limbs, rocks, furnishings, etc., and Final Layout artists would place these props into the shots.

Depending on the requirements of the scene, the flow branched in three directions: Character Animation, Effects, or Lighting. Those departments added motion to the characters, any special effects the scene required, and the lighting. However, Final Layout was responsible for watching a sequence as it proceeded through the production pipeline and, if necessary, was responsible for fixing technical problems.

Character Animation

The Character Animation Department used a PDI-proprietary tool to create the movement of the characters in *Shrek*. The Animators worked closely with the Directors as well as the Supervising and Directing Animators to determine the general direction and controlling idea of each shot. They defined a character's attitude, blocked the motion, and established the timing and screen direction for each shot. Each character's basic timing, placement, and expression was then roughed out for the whole sequence so that continuity between shots (and Directing Animators' styles) could be analyzed. And, finally, the detailed motion, such as a raised eyebrow or a smirk, was added to complete the shot.

Simulation and its role in character animation is described in "The Use of Simulation."

Effects

Shrek required a myriad of effects, including dust, water, fire, smoke, and mud, and it required the creation of film-specific shaders, complex prop setups, and the use of other unique systems such as foliage and crowds.

Detailed information on how these effects were achieved can be found in "Shrek Effects—Flames and Dragon Fireballs," "Shrek Effects—Animating Trees," and "Shrek Effects—Mud and Water."

Matte Painting

Matte Paintings, usually background paintings, are everything we see in a computer-generated film that aren't modeled in CG. Used throughout the production pipeline, matte paintings are images hand painted by artists in the computer that are mapped onto geometries. Some sequences require only one matte painting, for example, a background mapped onto a plane or disc-shaped geometry, while other sequences may use several paintings in order to "fill in the blanks" between models and characters after they've been rendered.

Lighting

The Lighting Department was responsible for digitally lighting the film. For each shot in *Shrek*, Lighters would place virtual CG lights, determine the placement of key, fill, and bounce light, and determine the color and intensity of light and shadow. In addition, Lighting was responsible for bringing together the many components and elements that comprised each shot, such as the texture maps and shaders that represented the surfaces of objects, the motion of the characters, effects such as smoke and fire, and the matte paintings.

A detailed description of the lighting techniques used in *Shrek* can be found in “Shrek Lighting.”

2D Paint

Rendering didn’t always process an image as expected. The 2D Paint Department was responsible for correcting problems such as white pixels or other artifacts, and for adding or deepening shadows and fixing collisions of objects.

Infrastructure

The technical infrastructure necessary to produce a movie like *Shrek* is described in “CG Infrastructure.”

Production Design for Digital Motion Pictures

James Hegedus, Art Director

The production design of a digital film like *Shrek* starts before the script is even completed and lasts until approval of the answer print. The art direction and design process for *Shrek* will be described in detail during the course presentation. Please use the next page to take notes.

Design Process Breakdown

I. Script

II. Storyboards for sequence
   - A. Conceptual design
   - B. Prop design
   - C. Texture/ color details
   - D. Blueprint/set design
   - E. Mock-up
   - F. Review and approval by directors

III. After approved story reel and before breakdown
   - A. Conceptual design changes
   - B. Key frames
   - C. Additional (or changes to) props, texture, blueprints, and mock-up
   - D. Review and approval by directors

IV. The evolution of a sequence, from story concept to final film
   - A. How visual decisions get made

B. Dissection of selected sequences

Design Breakdown

Conceptual Design
: Color illustrations for the "look of" the environment (set) with color, lighting, mood, etc.

Prop Design
: All props (separate elements) within a certain set and sequence, such as chairs, tools, weapons, lights, doors, maps, signs, kegs, picnic food, trees, sky, clouds, etc.

  *Modeling needs scale, thickness, front, back, and 3/4 view.

Texture/Color Details
: Texture and color of surfaces (floors, pillars, walls, etc.) and elements (acid, ant guts, etc.) within a set or sequence.

Blueprint (Set Design)
: Technical plan and elevation illustrations of all sets, etc.

  *Modeling needs scale, thickness, front, back, and 3/4 view.

Mock-up
: A rough model (in scale) of a set (or portion thereof).

Key Frames
: Based on the story reel. Accurate illustrations in 1.88 format of those important frames in a sequence that show the Sequence Director what each shot in a sequence will look like (how they're laid out, colored, and lit).

Lighting Design
: The Production Designer designs and directs the lighting of characters and sets throughout the production to final film.

Set Dressing
: Based on layout video, the designer will pick shots that the Layout Department will supply in print form. Designer will sketch in props (from modeling) in the proper positions, using the key frames as a guide. After the Sequence Director has approved the set dressing, the dressed print will go back to Layout.

Matte Painting
: Painting in full color scenery (not built in 3D by Modeling) such as all the skies, background elements, and some major set pieces.

Character Design
: Design of characters (primary, secondary) and their clothing, props, etc.

Generic Character Variations

Lucia Modesto, Character TD Co-Supervisor on Shrek
Jonathan Gibbs, Lead Effects Animator
Vanitha Rangaraju, Lighting Technical Director

Setting up many characters for crowds and secondary roles is a challenge in any production. As the story develops, the number of secondary characters often changes. For *Shrek*, the number of secondary characters in crowd scenes and minor parts increased so much that an easy way of adding new characters had to be developed. Our goal was to find a way of creating many different characters from one model and one setup. In this course we will cover the assumptions we took, the decisions we made and the compromises we had to make in order to accomplish that.

Generic Characters

The task of creating the generic characters was split between the Character TD and FX groups. The Character TD Group was responsible for creating the shape changes which included body and face variations, different wardrobes, and hair styles. The FX Department wrote the shaders to achieve skin color variations, different hair types and color, and different fabric textures and colors.

Figure 1: manA, manB, and manC Variations

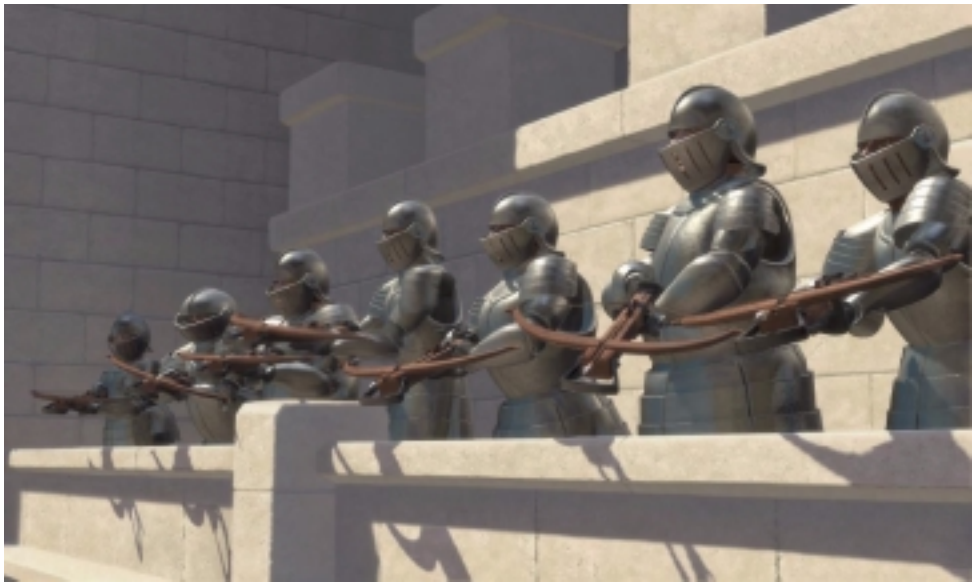

Figure 2: guardA and guardB Variations

First Considerations

Several questions had to be taken into consideration when this approach was first put forth. Why build a generic character instead of creating many different ones? How many different original models are necessary to create a good variety of characters? How many variations of each model are necessary to create a good variety of characters? How many different outfits and hats are needed? How many different hair styles are needed? What are the physical characteristics of a generic character model?

Why Build Generics Characters Instead of Building Many Different Ones?

The reasons we decided to build one big setup where clothes and body/face variations could be added instead of many simple smaller setups were the following:

- We assumed that the setup time for one versatile generic character would be less than setting up many (unknown number of) individual characters
- It was easier and faster to add more clothing and hats to an existing setup than setting up a new character from scratch
- Animation controls would be consistent from character to character. If all characters have the same setup, Character Animators don't have to spend time learning what each control does on each new character, and character testing and debugging time is reduced.

How Many Different Original Models are Necessary to Create a Good Variety of Characters?

As in any production, finding a compromise between the director's "ideal" number of characters and the "production wise" number of characters is a hard job. We initially settled for three different setups, with the caveat that, if needed, we would add more (we didn't have to).

We chose three main body types—fat, normal, and skinny (Figure 3). We expected that by making each of them skinnier and fatter, we would cover a large range of body types.

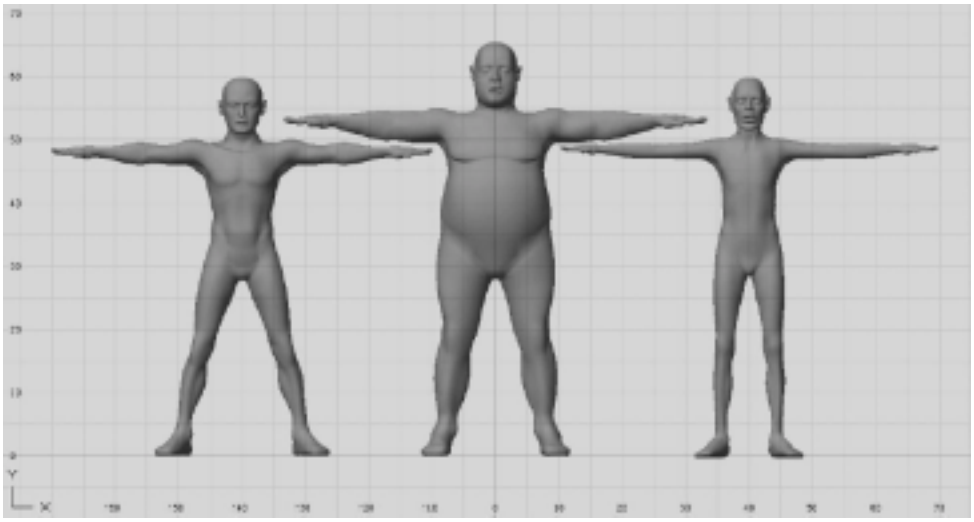

Figure 3: manA, manB, and manC

How Many Variations of Each Model are Necessary to Create a Good Variety of Characters?

As we expected, each body's range of variations overlap the next one, i.e., the fattest skinny guy should have a body type close to the skinniest normal guy, the fattest normal guy should have a body type close to the skinniest fat guy. We decided to use six body types for guards and five body types for the other characters. We varied height and bulk in order to have half of the variations skinnier than the original model and the other half fatter (Figure 4).

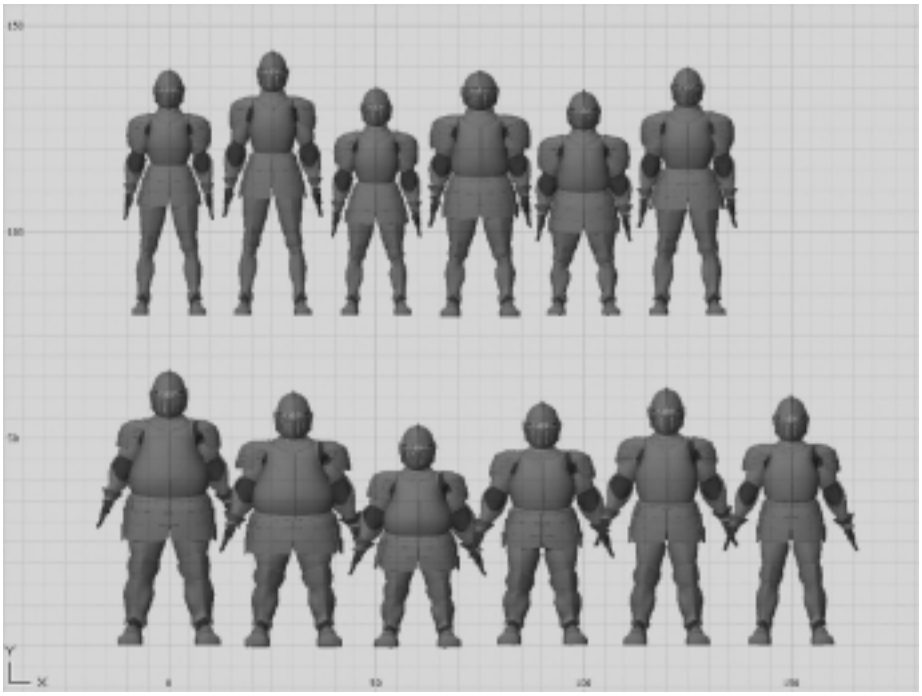

Figure 4: Body Variations for guardA and guardB

Since most of the characters in the movie were men, they had four different head types, and the women had three head types for regular people and two for witches (bigger noses and chins). The majority of generic characters had only three head types.

How Many Different Outfits and Hats Are Needed?

For crowd shots most of the outfit variations were achieved with texture and color (the people of Duloc were supposed to dress alike, almost in uniform), there was an average of three hat types per generic head type (Figure 5).

Figure 5: manC's Head and Hat Variations

For specific characters, different outfits were created (Duloc people, wizards, Merry Men, friar, Robin Hood, priest, ogre hunters, witches, Gepetto, Peter Pan, Snow White). For the guards, two different sized sets of body armor were created to make them look skinnier or fatter. The total number of outfits varied depending on how many different parts the character played.

Figure 6: womanA's Wardrobe

How Many Different Hair Styles Are Needed?

We decided that having three hairstyles for men and four for women, combined with color, texture, and length changes, provided enough variation. One of the hairstyles was specifically built to be used with hats in order to avoid hat and hair crashing problems.

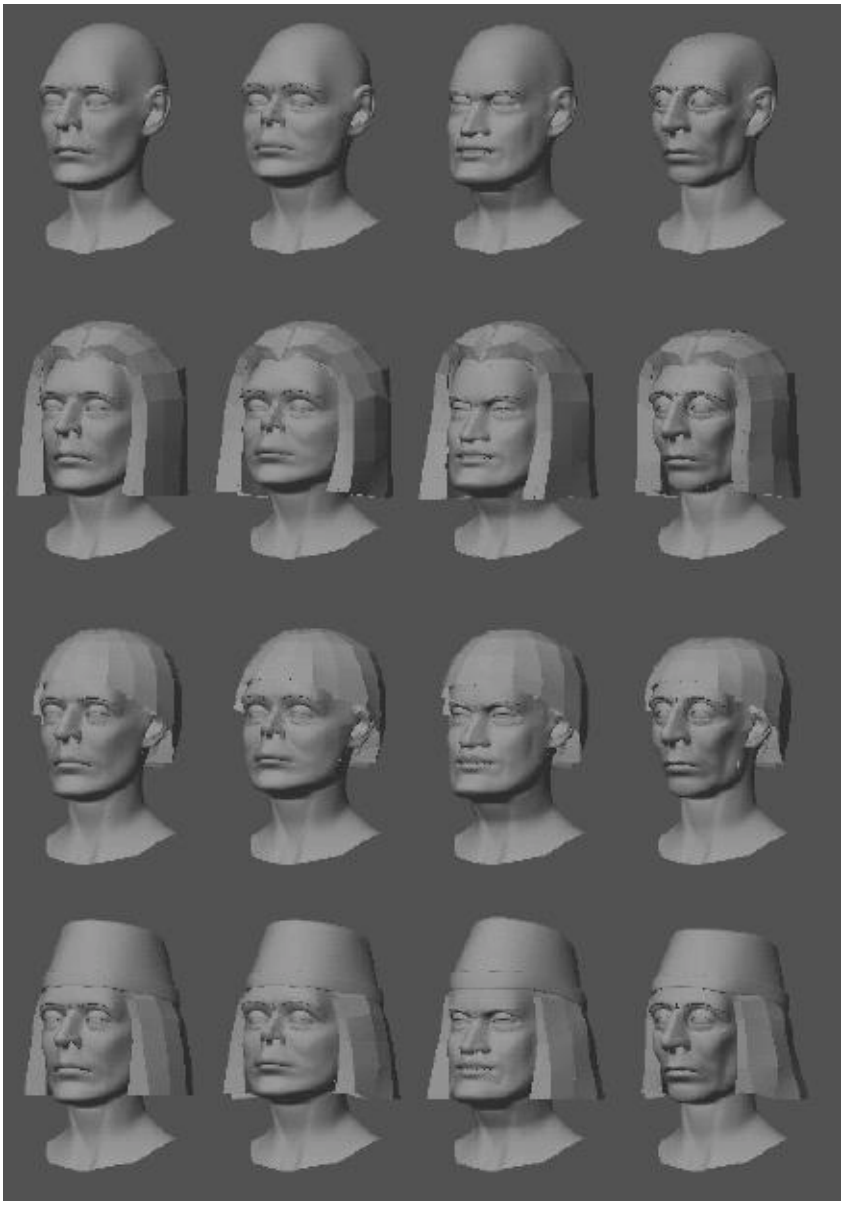

Figure 7: manC's Head and Hair Variations

Physical Characteristics of a Generic Character Model

A generic model has to be as "generic" as possible, i.e., average, with no outstanding features. A very interesting character will stand out in a crowd and we will be able to easily spot him. A good generic character is an inconspicuous one.

Generic Character Setup

Body

All humanlike characters had the same body motion system with slight variations (some generics had hat controls, some had arrow quiver controls, some had long beard controls, etc.).

Achieving Body Variations

We used scales applied to different joints to obtain the variations, so in theory, the number of attainable shapes was unlimited (Figure 8). Some of these shapes were aesthetically unpleasant, however. If the scales were set up by the Character Animators on a shot level, it would make it almost impossible to maintain continuity across shots. In order to avoid these situations, a distinct set of scales was chosen to represent a body type. A lookup table of scales was created so that by choosing one body type, different scales were applied to head, neck, uparms, loarms, hands, chest, shoulders, waist, hips, uplegs, lolegs, feet and toes.

Figure 8: manA's Body Variations

Limitations on Body Variations

There is a limit on how much you can vary a body before it starts to look weird (Figure 9 and Figure 10). We did not want overly outstanding body shapes so that we could repeat them without them being easily spotted.

Figure 9: A Body Variation That Was Too Extreme

Figure 10: Another Body Variation That Was Too Extreme

As the generics were used for crowd shots, motion cycles were created for the original model. Since all the variations used the same setup, these motion curves remained compatible with all body variations (except for inverse kinetics, where, for example, if an arm was too short, a hand couldn't reach its intended goal, so in this case, the motion would have to tweaked).

Deformations

The applied scales had to be easily handled by one deformation setup. Deformations had to minimize localized stretching and pinching.

Face

All main characters and generic humanlike characters had the same facial animation system.

Overview of the Facial Animation System

The facial setup consists of muscles attached to bones and to other muscles. There is a soft tissue or skin layer on top of the muscles. The motion of the face is animated by varying the muscle tension. The muscles then move the soft tissue layer on top. The system has the capability of moving bones, but it is not usually used in motion. The motion animators have access to several layers of animation controls:

- Low-level face controls
  All muscles can be individually animated; rarely used on shots. They are there to handle challenging exceptions that occur once or twice in a film.

- High-level face controls
  The Character TDs combine the muscle tension values to create higher level animation controls such as mad right brow, sad left brow, right mouth smile, left mouth sneer, pucker, cheek puffing, open right eye, tension left eye, etc.

- Library mouth shapes
  A lip sync library is created by the Supervising Character Animator using the high- and low-level controls. Then the Character Animators set lip sync phonemes keyframes by using this library.

Using the Facial System

The facial animation is done by animating separate layers. First, a first pass of the lip sync is taken by the Character Animators using the lip sync library. Next, the whole facial expression is animated on top of the lip sync animation. As the two layers of animation are independent of each other, both can be tweaked without changing each other. An extra layer of individual muscle animation can be used, if necessary, to further deform the face (it is almost never used).

Using the Facial System to Get Different Faces from a Single Model or Setup

The face system was not originally designed to achieve facial variations. It only had the capability of animating the bones and soft tissue. On *Antz*, there was a need for an extra character and not much time to set it up, since the facial system could animate bones, we copied an existing character, changed the bones' positions and a "new" face was created (Figure 11).

Figure 11: Colonel Cutter and Scout Faces: Scout's Face is Derived from Cutter's

On *Shrek*, we didn't want to have to copy the character over and over again for each different face, or keep animating the bones every time the character showed up, so we adapted the system to read in a different set of bone and soft tissue positions for each head type. By varying the bone structure and the soft tissue layer, we achieved the facial variations (Figure 12).

Limitations of the Facial System Design

We started building the generic characters' setups in mid-production so the facial system was already installed and in use for all other characters. As the system was common for all characters, we could not change it (due to time and production constraints). Since the system was not designed for creating variations, but for animating a face, certain controls that would be necessary to further vary the face were non-existent, so we had to work around it. For example, there are animation controls on the tip and on the bridge of the nose but none in-between, making it impossible to change the curvature of the nose.

Character Design

Character designers like to create interesting characters. Getting them to create a face with average neutral features was a big challenge. Bone ridges, hollow cheeks, or character lines are hard to get rid of, and you can easily identify them on the variations. We achieved very extreme variations and had to pull back from them because some of the faces were too memorable, or too striking and easy to spot in a crowd (Figure 12).

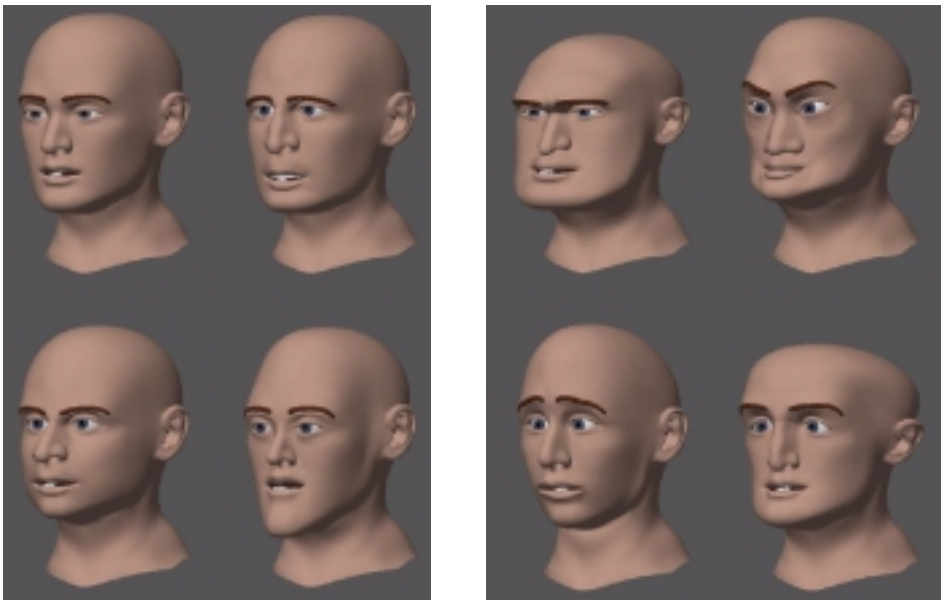

Figure 12: manA's Facial Variations and Too Extreme Facial Variations

Deformations

The big challenge for the deformation system was that it had to hold up for regular deformation caused by animation in addition to the deformations caused by a different head type.

Animation

For crowd shots, we wanted to be able to apply the same animation curves to all face variations, without having to do any tweaking. The facial variation around the mouth was kept to a minimum so that they could all share the same phoneme library.

Wardrobe

Specific outfits were created for certain characters such as Snow White, Gepetto, Peter Pan, Robin Hood, wizards, a priest, etc. All the tight-fitting clothing had a non-dynamic wrinkle system. The wrinkles could have their position, direction, and shape art directed at a specific pose and didn't require any simulation. The background characters had a animatable or "automatic" skirt. The men's kilts and women's skirts would automatically move away from their legs when they walked. The men had a few animation controls to further tweak the kilt motion. A dynamic simulated cloth motion was used for foreground or secondary characters skirts and kilts. Some hats had animation controls that were used to create further variations (twists and bends) (Figure 13).

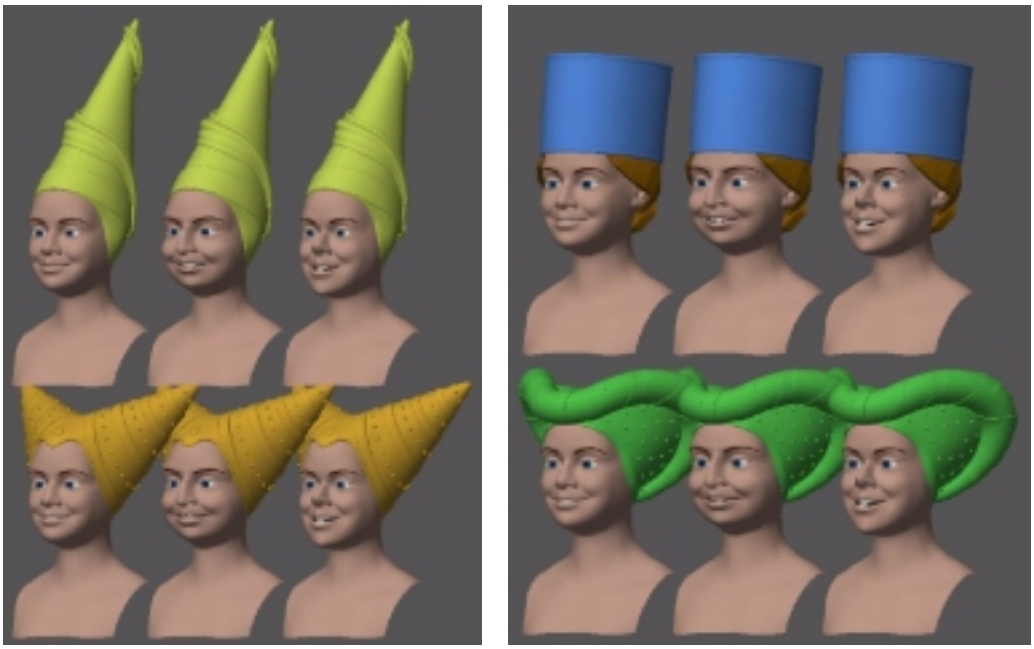

Figure 13: womanA's Head and Hat

Hair

Different hairstyle models were created. We chose hairstyles that presented very different silhouettes from each other in order to achieve an impression of hairstyle variety in a crowd (Figure 14).

Figure 14: womanA's Head and Hair Variations

For the main characters, we developed a high-level system that allowed animators to animate the hair, use dynamics to simulate it, or use the animated keys to drive the

dynamics. This technique was too expensive for the generics, so all generic characters' hair was simulated. Once the character motion was done, the hair was simulated and the hair models were created.

Using the System

In a shot, each generic character was created by specifying its setup variables:

```
generic_char
body = 0
head = 0
hair = 0
generic_clothes
tunic = 0
pants = 0
shoes = 0
hat = 0
skirt = 0
```

For each role a generic character played, a different combination of generic setup variables were specified. So, in one shot using a certain set of generic setup variables, the character would play one role, and in another shot using different values, it played another. By applying different values to these variables, we achieved many different body, head, and wardrobe combinations (Figure 15). Not all variable permutations were allowed. For example, certain hats wouldn't work with some hairstyles, and high boots didn't work with certain pants.

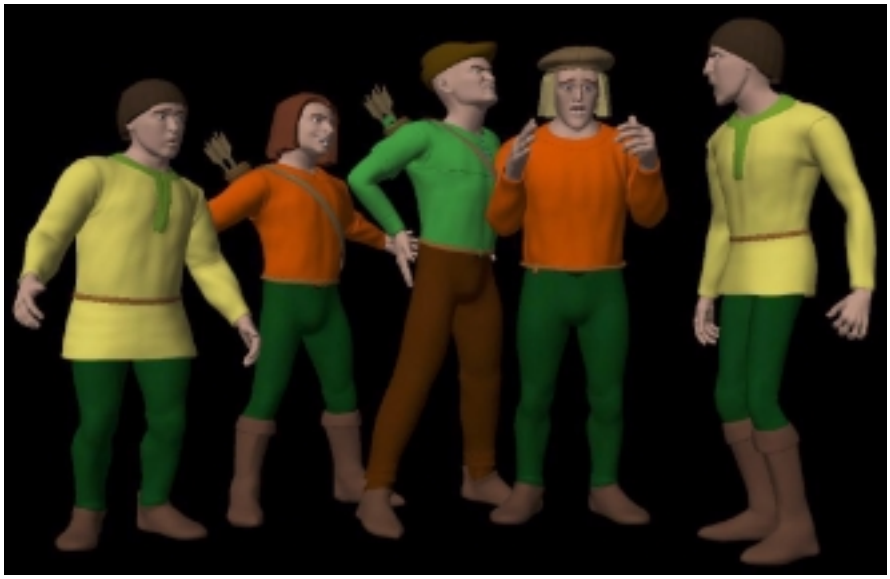

Figure 15: Five manAs with Different Body and Head Types, Hair, and Wardrobe

By combining these body, face, wardrobe, and hair shape variations with skin, facial hair, hair, and wardrobe color and texture variations, we were able to have one of the unique generic character setups play nine different parts (Robin Hood, four of the

Merry Men, and four of the ogre hunters) and many background characters. Another unique setup played eight roles (the priest, Gepetto, wizards, one of the Merry Men, and four of the ogre hunters) and many background characters. We saved a good deal of time by using this generic setup technique. By the end of production, we had built 13 setups (two guards, three men, two women, two children, one dwarf, one gnome, one elf, and one fairy godmother) that combined, played more than 60 roles in the movie.

Figure 16: Guard Variations

Figure 17: womanA Variation

Figure 18: manC as Gepetto

Figure 19: manC as a Merry Man

Figure 20: manC (Forefront Character) as an Ogre Hunter

Figure 21: manC as a Priest

Figure 22: manC as a Wizard

Lighting and FX

As described above, the Character TDs go through a very complicated process to deliver final models for lighting and rendering. However, everything up until now has focused on the shape of the model. The lighting stage of the pipeline is responsible for adding color and texture to the model, as well as additional render-time shape modifications (usually in the form of displacement maps). This process adds quite a bit more variation to ensure every character looks unique. Using these tools we can take the same shirt model and make it look like it's made of different materials and make it look brand new or very old. We can change the skin tone to make the character more pale or use displacement maps to add wrinkles and make character look old.

In addition, when more than 10 to 20 characters are in one scene (Figure 23), they officially become a "crowd", and the FX Department is in charge of automating this process as much as possible.

Figure 23: womanA in a Crowd

There are many parts to this pipeline, but this section of the course will cover two interesting parts. The first is the shader system for coloring and texturing generic characters, and the second is one way in which we automate that process for large crowds.

Shading Generic Characters

The PDI renderer allows us to write custom shaders to describe materials. These materials can be completely custom to the character, or they can call other shaders to do part of the work for them. Because of this ability for one shader to call another, we have written a large collection of general purpose shaders which can be attached together to form a shade tree. This allows our surfacing department to put together custom shade trees without needing to be programmers. It also allows our shader writers to focus on the unique aspects of new characters and write specific shaders to solve those problems.

For a normal lead character, an animator from the surfacing department would be responsible for putting together the custom shader tree. This shader tree is typically comprised of several painted texture maps, a few procedural elements and the appropriate lighting model. This is either done strictly by connecting and adjusting the parameters of existing shaders or by working with the shaders writers to write new custom shaders.

However, for the generic characters, we realized we wouldn’t have the time or resources to put together a custom shader tree for each one. Instead, we decided to put together a shader tree which would work for a broad selection of generic characters. Then, when casting a particular shot, simpler high-level controls can be adjusted by those lighting the shots to realize a unique individual for that role.

For instance, we had one big shade tree which would work for all the generic men. Actually, there were several such trees for different components of the men (clothing, skin, hair, eyes, etc), and together they formed a sophisticated shading model. We had similar setups for the women, children, and several classes of common fairy-tale creatures such as gnomes, elves, dwarfs, wizards, and witches.

When building such a shade tree, we wanted to keep the underlying components as generic as possible. So, instead of painting the textures as we usually do, all the textures were painted in grayscale. This allowed us to color them procedurally in the shader, and allowed lighters to change the colors on the characters simply by adjusting one color pot in the lighting tool. It also made it easy to ensure that no two people were wearing exactly the same colors.

For a generic man's shirt we might paint several textures representing several types of materials the characters might wear: cotton, burlap, velvet, etc. The colors come from an art director-approved palette which is comprised of a set of color gradients. To choose the color for a particular shirt, one would first pick which gradient to use, and then choose some point along the gradient. This system let us build a simple color-choosing interface and then ensured we would always pick colors which were suitable to the scene.

These generic shaders also included a wide variety of other high-level controls. For instance, there is a control on the generic women called "skin tone". This was a simple value from 0 to 1 which picks from a blend of three different types of skin: neutral, olive, and pale. An age control allowed us to age the face by adding wrinkles and changing the shape of certain features. It would also age the hair by adding in the appropriate amount of grey.

A wide variety of hair styles was also included. Even though an overall hair guide geometry was chosen with the model, procedural alterations to the actual hairs at render-time could take the same basic guide geometry and grow from it many different types of hair from straight to wavy to curly. Figure 24 shows the simple guide geometry for a clump of hair (on the left) and two different types of hair generated from it. The "virtual haircut" system helped ensure that everyone didn't look like they went to the barber on the same day.

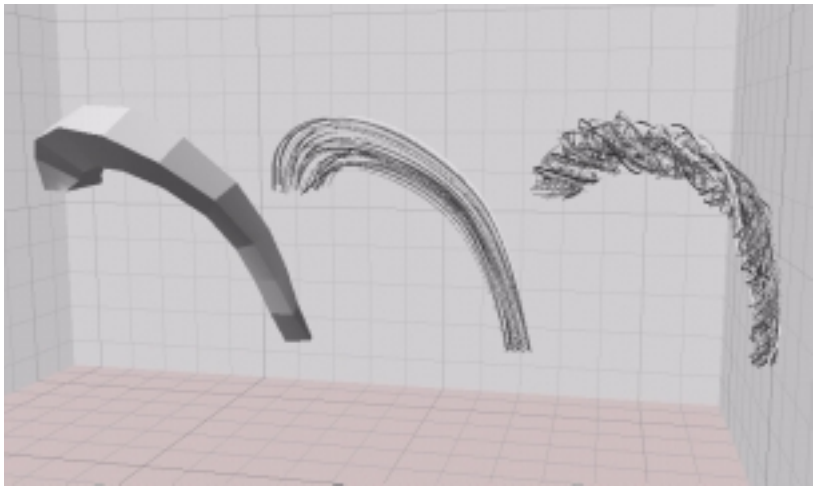

Figure 24: Simple Guide Geometry for a Clump of Hair

Crowds

When the above system is used on generic characters in a shot, the high-level controls are set by hand. However, when you have to fill a cathedral or a stadium full of thousands of people, that approach becomes less appealing. Also, we really don't want the renderer to have to load a custom shader for each character in the crowd.

To solve the latter problem, the generic shaders get their high-level controls from a simple database. Each generic character is assigned an ID number which is stored inside the model. At shading time, the shaders can access this ID number and query the database for the appropriate high-level parameters. To change these parameters, the animator just updates the database and re-shades their image, and they can see the changes in near real-time.

Once this system is in place, the whole crowd problem simply boils down to creating the database procedurally when the crowds are generated. And at first we thought this was going to be an easy problem to solve... we should have known better.

The first thing we tried was simply picking random values in the appropriate range for each parameter. We quickly realized that the problem with this approach was that the characters didn't have any sense of fashion. It was like they were picking random items from their closet. So, we had to smarten up the system by teaching it about fashion. To start with, we looked at the outfits the Art Directors had chosen by hand up to that point and talked to them about what sort of things they wanted to see. We also rendered sets of characters wearing different sorts of color combinations to see which ones the directors tended to like the best.

As mentioned before, all the color choices were color gradients which gave us appropriate dark and light version of the base color. We divided these gradients into two groups of about ten gradients each. The gradients in the first group had a nicely contrasting gradient in the second group. The basic fashion rule we used was that we had to either pick matching colors or contrasting colors. Figure 25 shows the palette of three-color gradients used for the generic man's pants in the tournament sequence.

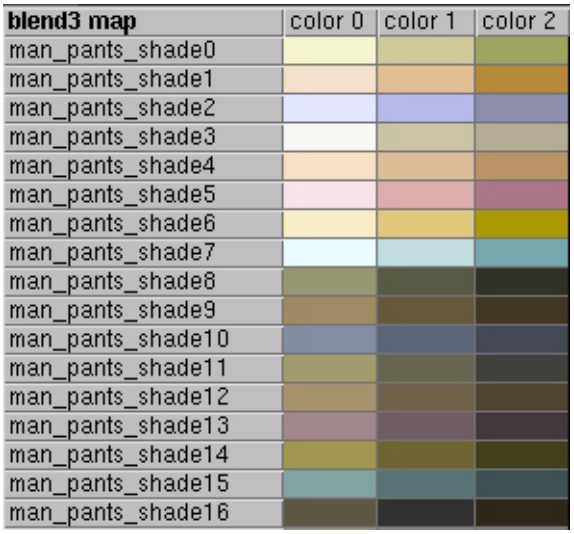

Figure 25: Palette of Three-color Gradients

So to start off, the color of one item of clothing would be chosen at random. For instance, we would decide to pick blue pants. Then the rest of the clothing was chosen to either match or contrast the first item. So, when picking the color for the shirt, we first had to choose if we wanted to pick a matching shirt or a contrasting shirt. If we chose matching, then another (slightly different) blue was picked for the shirt. If we chose contrasting, then we had to select a color which had been determined to nicely contrast with blue, perhaps yellow.

Other items of clothing were more tightly constrained. For instance, the belt had to always either match the pants or the shirt, and the socks always had to match the belt.

We would assign other high-level parameters, such as fabric type, the same way. All the choices were assigned different probabilities depending on the directors preferences for that sequence.

Of course, we still had to be able to edit these automatic choices by hand in the event that the director just didn't like one particular outfit. Once the database was generated, the Lighters could edit it just as they could edit the setups for the other generic characters. If the crowds were ever changed and the procedural database regenerated, we would intelligently merge the Lighters changes into the new database so no work was lost.

While this system proved to be much more complicated than we originally anticipated, once it was in place, it ran without modification for all the crowd sequences in the film.

Acknowledgments

Thanks to: Luca Prasso, Character TD Co-Supervisor
Jane Hartwell, for all the support throughout production and for trusting that we could make this generic setup work
Milana Huang, Rob Vogt, Matt Authement, and Vanitha Rangaraju, for writing and supporting all generic character scripts
Jeff Hayes and the Modeling Department, for modeling all clothing, hats, wigs, etc.
Dick Walsh, for writing such a versatile facial system
Andrea Stoops, for keeping track of all variations and possible wardrobe combinations
Character TD Group, for setting up all these characters.

The Use of Simulation

Rob Vogt, Senior Character Technical Director
Scott Singer, Senior Effects Animator

Simulation can be a powerful tool, but when is it appropriate to use in a feature production? This portion of the course describes the use of simulation in feature production using examples from PDI/DreamWorks features *Shrek* and *Antz*. Based on the production requirement and level of commitment toward a completely simulated solution, we will categorize the types of simulations PDI/DreamWorks has used in its computer-generated feature films, and use examples to show how they are integrated into the production.

Simulation?

Simulation can be a double-edged sword; although it can be a cost-effective way of producing large amounts of motion, it can also produce undesirable results that are difficult and expensive to alter. The animated feature film, *Shrek*, incorporates the extensive use of simulation. In this course, we will categorize the types of simulations based upon production requirements, siting simulation methods used on *Shrek*, while we hope to shed light on how to determine "why" or even more importantly "why not" simulate.

Why Simulate

In general, the decision to simulate is made because either a certain look or a certain savings is desired. Often the motion we want is tied to the look of a particular technique. Also, simulation is usually the best way to get the kind of naturalistic incidental details to make the motion more physically convincing. When we hand over the outcome of motion to simulation, we open the door to accident and complex inter-reactive feedback that is impossible to get with manual animation. Simulation is also the best way to make animated objects respond to a dynamic environment (Figure 26). If the rules of the simulation are robust enough, then any changes to the environment and surrounding animation should induce equally convincing motion.

Figure 26: Mud Shower

Why Not

The reasons not to simulate are a sort of strange doppelganger of the reasons to simulate. What are the trade-offs between complexity, cost, and directability? Simulation, by its very nature, can be very expensive to compute and difficult to control. We liken animating with dynamics to moving a marble over an obstacle course by blowing on it through a straw. Though simulation can respond to changing dynamic environments, changes to those environments necessitate re-simulation. Even when the rules are robust, an equally convincing naturalistic motion may simply not work for the director; and usually, because of this, no one spends the time even trying to make them that robust. Another more philosophical reason is that our goal in an animation is not a simulation, but rather the successful completion of a sequence of images. A simulation is generally used to find the outcome of a system given an environment and starting condition, either to test the model upon which the simulation is built, or to test the starting condition against the environment. Either way, the outcome, no matter what it is, is the goal. In animation, we pretty much know what the outcome needs to be. We know the past. We know the future of the environment. We have the script, and, therefore, we should know about 90% of what we want to see. What we want from a simulation is that last ten percent of accident, detail, and specific arbitrariness that lends the final motion its physical credibility.

Figure 27: Software Changes Caused Lord Farquaad to Experience a Bad Hair Day

The Goal

Ideally, we want all of the benefits without any of the costs, but the adage holds true: good, fast, cheap—pick any two. So the problem becomes how to devise solutions which can balance dynamic simulation with hand animation; how do we balance control and accident? Some of the techniques we have found involve data reuse, exploiting our omniscient knowledge of the scene and events, the crucial importance of robust simulation and animation environments, and, not to be underestimated, really fast computers. We hope that by shedding light on our experiences with *Shrek* and *Antz*, we can start a larger discussion of the economics of simulation as well as the roles of dynamic and directed simulation in entertainment computer animation.

Types of Simulations

One can think of simulation in an animation as a continuum from entirely hand-animated on one end to entirely dynamic on the other end. This puts animating a joint chain with forward kinematics at one pole and a Jet Propulsion Laboratory “bow shock” simulation at the other. Along the way we pass through the world of animation production simulations—somewhere in the area of dynamically-assisted animations and dynamically-driven animations. In general, the cost benefits break down as follows: the more the simulation tends toward the dynamically-assisted, the more control the animator gets over the outcome; however, it makes the hand of the individual animator that much more apparent. The more dynamically-driven a simulation is, the more “physical” the animation becomes at the cost of controllability.

Instancing of Pre-computed Simulation Elements

The instancing of pre-computed simulations into an animation can be a very powerful way of bringing the "look" of simulation into what is actually a non-dynamic or semi-dynamic animation. Along with the benefits come some pretty strict limitations that need to be figured into whatever planning stage is taken. Most notable of these are the limits on how well the effects hold up to close scrutiny, and how inflexible the methods are at accomplishing effects outside of their original purpose. To illustrate these points we will discuss the pros and cons of two *Shrek* systems which relied heavily on pre-computed simulations: the Impact Dust[1] and Torch[2] systems. Both underwent serious re-design as the production progressed, and neither weathered the growing pains particularly well.

Impact Dust

Impact dust was a system designed for *Antz* based on the limitations of the day and the visual requirements of the movie. At its heart, it is a mechanism for instancing pre-computed dust puffs in time and space.[3] Its primary benefit is that it can compactly describe many dust puffs, handle their rendering, and simplify their placement. Its major drawback was that, being non-dynamic, it offered very little means for adding shot-specific interaction. As *Shrek* was originally envisioned, dust would play a very minor role in the visual look of characters interacting with the environment, and so we decided that the limitations of the Impact Dust system did not outweigh the cost savings, both in animator time and disk space. Doing a custom simulation for each insignificant dust puff was seen as too wasteful of available resources. As the production progressed, however, the lack of direct interaction became a burden. At the same time, disk drive prices were plummeting and we were developing some plug-ins and tools and a new rendering pipeline that significantly simplified the process of doing hero dust simulations. The upshot of this was that by the end of the production, the Impact Dust system, an integral part of the *Antz* effects production ethos, was all but cast aside in favor of custom simulations. In this case, the advance of technology had obviated an entire technique, making what was cheapest and most efficient actually more cumbersome than what it was supposed to augment.

Torches

Torches were another system that was based upon a dynamic instancing model, however, in this case, an extra level of dynamics was placed on the flame geometry and rendering in the shot. Originally designed as a system to do the "oil torch" flames in the Dragon's Keep sequences, it also had to be used to animate the torches carried by the Ogre Hunters in the opening sequence (Figure 28). Unfortunately, the model designs of the Ogre Hunter torches, as well as the differences in the desired look of these torches,

1. Contributors: Scott Peterson, Terran Boylan, Scott Singer
2. Contributors: Erdem Taylan, Terran Boylan, Mark Edwards, Scott Singer
3. See the SIGGRAPH course notes from *1999 Visual Effect Galaxy* for more description of the Impact Dust System.

meant that the old system wouldn't work "out of the box". So we had to redesign the effect to handle multiple libraries of torches, as well as a slightly different rendering technique. As the new torches were being animated in shots, their importance to the movie began to increase, and being in the opening sequence of the movie, everything including the torches began to fall under far more scrutiny. The technique kept having to be shoehorned and added to, re-purposed and hacked until probably more time was spent forcing the technique to the shot than it would have taken to treat each torch as a one-off effect. In this case, the need for a catch-all panacea system failed as it met the reality of production. That isn't to say that torches or the torch system failed—just that a system was mistaken for a tool set. As an example of this, the same torch systems, when used in their original roles, performed flawlessly in putting hundreds of static torches, oil lamps, and medium-to-long-range flames all over the movie.

Figure 28: Torch

Foliage

Shrek's world is comprised of lush greenery—trees, bushes, sunflowers, cat tails—the list goes on. To bring this world to life, a system for animating tree branch to leaf, petal, and stalk was developed utilizing PDI's fluid simulation tools[1]. Fluid simulations are the driving force of the moving environment, providing the low-frequency motion that changes slowly over time and moves in broad strokes. Coarse fluid simulations were used to provide this quality. The main controls are the speed at which the simulation is played back through the environment, and a wind scale value. A layer of oscillating motion was added on top of the motion, due to the wind. Oscillations represent the internal forces of a plant which cause it to spring back and forth against the force of

1. Contributors: Scott Peterson, Foliage System Developer and Nick Foster, Fluid Simulation System Developer

wind. Without oscillations, everything appears to move as if underwater. The controls for oscillations are frequency and magnitude as a percentage of the wind vector.

The main advantage in animating Shrek's environment in this manner was the artistic quality and realism of the motion (Figure 29). All of the environment's motion was driven by portions of three separate fluid simulations, so there was no requirement to simulate on a shot basis. The simulation data was reusable, and since they were small and coarse, they occupied very little disk space. A possible disadvantage is that this method requires a stable fluid simulation system, which can be difficult to develop. Additionally, the final motion for the environments were not completely procedural, requiring some information to be saved to disk for each shot.

Figure 29: Foliage

Hero Simulation

When the budget is thrown to the wind, the job becomes too large in scope, or when only the "accidental" look of pure simulation will do, we turn to the hero simulation. At PDI these are simulations which are essentially modeled as unique simulations developed and designed to meet only the needs of their shot. There is, of course, a lot of tool and code reuse, and concepts travel from implementation to implementation. Very often when these effects move from the conceptual stage to actually being in the shot, the approach and techniques have to change thus strict adherence to a rigid implementation becomes impractical.

Fluid Simulations

Shrek included numerous examples of moving fluid—beer, mud, and swamp water[1]. The key to its success was going to be how well it meshed into the world of *Shrek*. In the tournament sequence, Shrek needed to battle a group of knights in a beer-soaked mud hole (Figure 30). In order to pull this off visually, only a physical simulation could do the job, and in order to achieve the effect on a practical level, a good deal of very "non-physical" simulation management techniques had to be used. For example, there was no way that the entire field of mud could be described in a single simulation at the resolution needed to capture the detail which the visual style demanded. Therefore, the field needed to be broken down into a number of sub-simulations and a technique was developed for merging those together to create the illusion of an unbroken puddle of slop.

Figure 30: Beer Mud

Window Destruction

In the transformation sequence at the end of the movie, rows of stained glass windows[2] had to explode outward (Figure 31). Achieving this effect required a different level of control than just simply turning the windows into a random bunch of triangles and then propelling them off camera (ah, the old days). The initial breaks in the windows had to be in groupings that more or less followed the lines of the leading in the window

1. Contributors: Scott Peterson, Juan Buhler, Jonathan Gibbs, Nick Foster, Mahesh Ramasubramanian, Erdem Taylan

2. Contributors: Matt Bear, Andrew Harris, Scott Singer

textures. At first we thought of using some kind of algorithm to shatter the windows but abandoned that in favor of painting maps to indicate breakage. Additional maps were painted to control how the shards were then exploded out. By controlling the variables with maps we were able to very quickly address director comments in a very direct way.

Figure 31: Window Destruction

Ropes and Chains

An interesting mix of simulation and non-simulation techniques came into play with the various ropes[1] and chains[2] animated in the movie. The most apparent use of dynamic ropes was in the WWF section of the tournament sequence. The problem here was to design a system of springy ropes that could respond to arbitrary character interaction and still give enough user control to use all of the kludges and hacks that actually make a simulation work in a shot. The ropes were modeled as cloth, and character interaction was handled as user-defined collision models and constraints. The actual rendered elements were derived from this deforming cloth. It is key to abstract the technique from the effect. Often the detail visible in a shot is not necessary to have in the simulation. A good case in point is the chain animation in *Shrek* (Figure 32). The chains are actually several different approaches applied to a central concept of instancing links in space. Most of the chain shots were entirely non-dynamic, and the control points of the animation structures were essentially hand-animated. Only when the chain needed to interact with the environment (or when we thought that it would) was a dynamic system used to get a believable sense of recoil from the chain slapping against the floor or bouncing off an object. The fact that most of the chain animation was done by hand

1. Contributors: David Allen
2. Contributors: Juan Buhler, Rhett Collier, Scott Singer

is, we think, proof that in skilled hands, simulation is often not the best way to get convincing physical action.

Figure 32: Chain

Pose Driven Dynamics

Hero Hair

Except for Shrek himself, *Shrek's* characters have hair, or at least the option of having hair. Early in the development phase of *Shrek*, the decision was made to have simulation provide the motion for all characters' hair. Unlike a live-action shoot where no attempts are made to control the actors' hair aside from preparation, in an animated feature the hair may be expected to contribute to the character, or be used to help amplify a story point. As a reference to the desired controllability of the hair, we looked at Belle, from *Beauty and the Beast*. The tuft of Belle's hair the intermittently fell from its groomed position and was pushed back into place suggested the level of control that was needed for our characters. In the case of Belle, there was only one tuft of hair which needed explicit controllability, where as the remaining hair needed to just look natural in its motion. The hair would also be used to fall into position to better frame the face, helping the viewers to focus on the dialogue. With these references in mind, we decided to develop two hair simulation systems, one system which would provide hero hair motion while also providing explicit control and directability (Strand), and a second system designed to provide motion to the bulk of hair not requiring explicit control (Coif).

A general dynamics solution, “strand”, is integrated in portions of *Shrek’s* main characters: Shrek, Fiona, Donkey, and Farquaad[1] [2]. The system provides motion for long clumps and flowing hair (Figure 33), and is additionally utilized for simpler applications like Shrek’s ears, and the donkey tail and ears. Controllability and proposed widespread use of the system would suggest that a formalized system be developed to deliver believable hair motion with little to no interaction from the animators. The formalized system also needs to have the ability to utilize hair poses as inputs to guide, or direct, the simulation when more specific hair motion was required. The resulting Strand system incorporates pose-influence, which defines how closely the dynamic motion adheres to the kinematic, or animator-defined, motion. The dynamics are constrained to stay close to the posed strand by increasing its influence, or allowed to have a looser dynamic quality by “dialing down” the influence. By this method, we are able to control the dynamic behavior where needed, allowing the animators to satisfy directors’ requests.

Figure 33: Hero Hair

The disadvantages are the development cost and setup time required to tune the strand setup parameters for each character. The advantage of this system is its ability to simulate a reasonable solution while maintaining the ability to keyframe animate as well, and further utilize the keyed animation to drive the simulation. The strand system utilizes traditional animation controls as the main input, making it an intuitive tool for animators. Portability between characters and feature productions, as well as a reduction in the scope or volume of keyframed animation required, ensured this system to recoup any development expense.

---

1. Contributors: Rob Vogt and Sumit Das, *Strand System*

2. See the SIGGRAPH 2000 sketch, “Integrated Pose-driven Character Dynamics,” by Rob Vogt and Sumit Das for a detailed description of the *Strand System*.

Minimally Interactive Dynamic Systems

Simulations in this category approach more the “typical” simulation where the controls to alter the simulation are sparse. With minimally interactive dynamics, you get what you get, and if you want something different you adjust the input parameters and re-simulate. Other than adjusting spring-mass parameters, another typical control is a blending feature used to apply a percentage of the dynamics motion. With this in mind, these types of systems should be used conservatively, resisting the urge to rely on them beyond their designed use or stability range. A big advantage of these types of systems are that they can be relatively inexpensive to develop, depending on the application.

Generic Hair

The “Coif” general hair simulation system[1] was designed for use with the short or constrained hairstyles, where range of motion is limited. Longer hairstyles would utilize pose-driven dynamics, described in the earlier section, “Pose Driven Dynamics”. Because keyframe animation would not be feasible for the high number of characters involved, a procedural dynamic system was implemented. The system allowed for multiple hairstyles to be applied to the same character, which could be toggled on a shot basis. To avoid the look of a mass-spring system, additional weighting attributes were given to each control vertex. The system would then interpret the dynamic motion based on the control vertices’ weighting attributes (Figure 34).

Figure 34: Generic Hair

1. Contributors: JJ Blumenkranz and Sumit Das, *Coif System*

Setup time was primarily a function of the time required to model the hairstyle, and the time to implement the new hair model with the coif system was on the order of minutes. The only interactive setup was positioning collision objects. Collision objects had to be set up only once per character; multiple hairstyles used the same collision set. During setup, the system would automatically establish weighting based on the new hairstyle geometry and the head which was wearing it. The setup Technical Director also had the option of setting motion parameters, which were also used in establishing weighting. In practice, this functionality was rarely used.

In production, the animator did not have to be bothered with hair motion. However, the system did allow for "overriding" the default motion. If so inclined, the animator could tone down motion for the entire shot, or only over a specific range. The same is true for overall mass, damping, and spring stiffness. Again, in practice this functionality was rarely used. The result was a uniform system which could accommodate literally hundreds of different hairstyles with minimum setup time. The biggest drawback was the range of motion. If the generic hair was required to deform too much, it started to look "stretchy." For this reason, we restricted use of the Coif System to hairstyles which were shorter, or to hair that was tied up. Since the Coif System was so robust and a snap to implement into new characters or hairstyles, it was difficult to resist the urge to use it beyond its design limitations. In these cases, we turned to the more expensive Strand System.

It might also be noted that simulation data was saved as a deformed hair model, requiring a simulation to be run to produce a hair model. Due to the performance of the system, this was not considered a disadvantage, merely a data storage choice.

Clothing

One of the design fundamentals for the clothing pipeline[1] was that the character animators were not to interact with the clothing animation system. One of the design fundamentals of the universe is that you can't always get what you want. Another design fundamental of the clothing pipeline was that there was not enough time for the clothing team to tweak settings for every shot; this brings us the production universe design corollary – sometimes you get what you need. Through a lot of hard work, a pipeline was designed which shielded the motion animators from most of the cloth implementation such as garment creation, cloth attributes, creating scene files, launching them on the renderfarm, etc. Still, the animators had to be aware of, and work within, the constraints of a dynamic simulation environment. There were strict limitations on how a character could be posed at the beginning of a frame. Parts of the character that were invisible to the camera, but not to the simulation, had to be animated. The characters had to be animated, not just to the camera, but so they would make sense physically. The way in which animators were shielded from having to tweak settings themselves and the way that the Clothing Team was saved from the same fate was by isolating and simulating only those parts of garments that required it. Much of what appears to be dynamic cloth—wrinkles in sleeves, and so forth—was actually

1. Contributors: Marty Usiak, Randy Hammond, Andrew Harris, Matt Baer, Scott Singer

non-dynamic. For the parts that were simulated, a lot of time was spent up-front testing parameters and garment design to arrive at some robust settings or sets thereof which could get the cloth through most situations (Figure 35).

Figure 35: Cloth

Turn-key Simulation

A turn-key, or ideal, system would provide completely reproducible simulations which react to an arbitrary number of contributing environmental influences to produce desirable results automatically without animator interaction, or without the need to adjust input parameters. For a number of odd reasons, this system should never be built within a CG feature production framework. Development resources and costs for such a complicated system would be forbidding. In a CG feature production, a physically accurate simulation is merely a good starting point, and to achieve the desired artistic vision, the laws of physics will usually need to be compromised. For this reason, development resources are better utilized maximizing controllability.

Conclusion

The most important lesson learned about simulations in an animation environment is that taking the time to analyze the needs of the shot to find the most appropriate technique is time well spent. Nothing wastes more time in a production than pounding a square peg of a system into a round hole. Take the time to implement the system that fits, and when the parameters of the effect are unknown and the time budget is tight, make sure that there is a robust set of tools with which an animator can quickly fashion a solution. And because an accurate cloth simulation of Princess Fiona wearing a full-length velvet dress and running faster than Carl Lewis in the 100 meter dash would

have to have her falling down and breaking her neck, you must remember that simulation is not animation.

Figure 36: Achieving Artistic Vision via Multiple Simulation Techniques

Shrek Effects

A computer-generated feature film requires a large number of visual effects. Some of these effects are the same type that you would find in a live-action film, such as fire or large crowd scenes. In addition to these traditional effects, a CG film requires a large number of effects which involve replicating the less dangerous parts of the real world, such as trees, water and mud.

In *Shrek*, we were not trying to match the real world as much as we were trying to create the world inside the heads of our Directors and Art Department. And while we have lots of science explaining the real world, the mind of an artist can be quite a bit more difficult to figure out.

The *Shrek* Effects Group was responsible for hundreds of effects in literally every frame of the movie. The ability to create a film like *Shrek* relies on animators being able to work quickly and fluidly, to make radical departures from current assumptions and mix sometimes disparate parts into a visually contiguous “effect”. At the same time, the sheer number of shots demanded that we create reusable systems so development time could be leveraged over as many shots as possible.

The following sections discuss three of the many types of effects we had to create for *Shrek*: fire, trees, and fluids.

Shrek Effects—Flames and Dragon Fireballs

Arnauld Lamorlette, Senior Effects Animator

Creating visually convincing fire is one of the most difficult problems in computer graphics. Up to now, every time fire has been an important part of the image in a movie, the most efficient approach has been to composite live-action footage of fire. It is cheaper to composite live-action and, so far, no one has been able to find the physical equations that would entirely describe the dynamics and appearance of flames or fireballs. Fire is a complex chemical reaction between fuels and oxidizers, and to simulate it realistically, one would have to be able to describe precisely the chemical reactions involved. Such reactions would include the creation of smoke, soot, fluid dynamics of viscous gas, heat exchange, and, most importantly, the lighting model, because we perceive fire through the light that it emits.

Introduction

For the making of *Shrek* we had to create many different type of fires. Shrek lives in the medieval age, and the only source of light besides the sun and pixie dust is fire (fireplace, torches, and candles). In addition, *Shrek* is a fairy tale, and a good fairy tale has to have a fire-breathing dragon that burns down everything in its path. We are aware that the different fires we've created are not yet totally photo-realistic, but their stylized look is consistent with the "stylized reality" of the *Shrek* universe. This section of the course notes will briefly describe techniques used to create CG flames and in more detail, explain the practical steps to implement CG fireballs.

The Flame Implementation

Features of Flames

If we look carefully at images of flames, most of the time we will see a complex shape flickering and breaking. We identify flames mainly because of their outline and the way they flicker. This flickering is so unique to flames that some sophisticated fire alarms are based on the variation of the flickering light produced by a flame. Flames have somewhat sharp contours, evolving upward and breaking at the top.

Particles Technique

A flame is a gaseous shape evolving along time. The easiest way to simulate its variation of shape is to use a set of particles, with each particle having a specific position, color, velocity, and opacity. Instead of trying to compute a physically correct simulation of the flame phenomena, we created a set of rules in a particle system that try to mimic the behavior of flame as we perceive it, and more precisely, the flickering and breaking of the top of the flame.

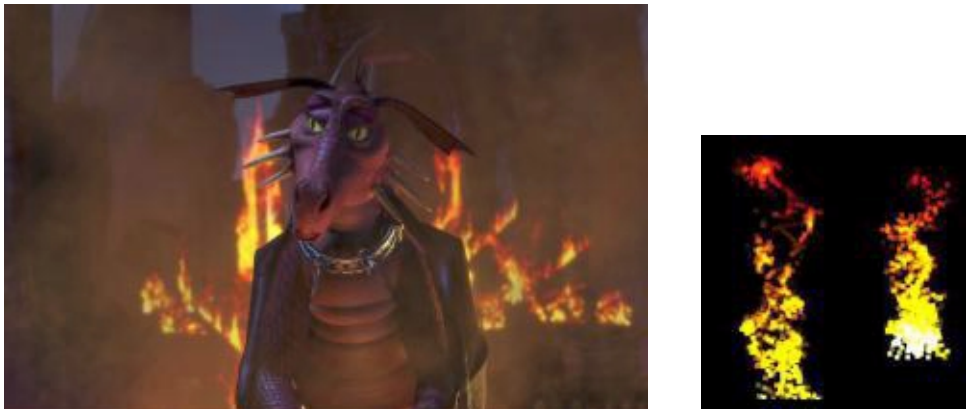

Figure 37: Using Particles to Create Flames

Using a particle approach gives overall dynamic qualities that look satisfying, but when it comes to rendering the flames, the discreet nature of particles makes it very difficult to render a continuous, well-defined, textured flame. The motif we got from the particle simulation didn't provide a satisfying result. Even with a high number of particles, a texture was needed to increase the complexity of the flame. The problem was how to add a continuous texture that followed the dynamic motion of the particles.

Surface Derived from Particles

The easiest method to texture a flame continuously is to create a surface. Naturally, the best way to derive a continuous surface from a set of particles is to generate either an iso-surface from the particles or the enveloping surface around the particles. We can then use the intersection of the surface with a solid noise. But then the shapes tends to look “blobby” and smooth—it doesn’t really look like a flame. And worse, the flame looks like an empty shell.

Volumes Shaders Derived from Particles

Rather than render the surface derived from the particles directly, we use a ray-marching solution which samples the density function derived from the particles. This approach yields a flame which has the illusion of thickness.

Problems with Particle-based Techniques

The main problem with the particle-derived techniques (iso-surface, volume shader) is in texturing the flame. We tried unsuccessfully to generate a continuous texture out of a set of particles that would follow the dynamic of the particles and wouldn’t shear or stretch. We ended up using a static solid noise which intersects our resulting surface and matches the particles vertical speed. Unfortunately, when using this technique, the dynamic motion of each particle doesn’t control the dynamic of the noise.

Figure 38: Simplified Flame Using a Volume Shader

Figure 39: Final Flames Using a Volume Shader

Modeled Surface Technique

A deformed, twisted model of a cylinder describes very well the volume and continuous shape of a flame. It is also very easy to texture a surface and to add a flame-like texture that perfectly follows whatever dynamic motion is applied on the surface.

Figure 40: Surface Modeled by Hand

The Problems of the Modeled Surface Technique

Unfortunately, a surface doesn't look very gaseous. The upper part doesn't break apart like a real flame would. We lose the "fluid" qualities of fire.

Comparisons Between the Particle and Modeled Surface Techniques

Particle Approach

The particle approach provides a very nice dynamic, good breaking, and good volume, but it is difficult to obtain a continuous textured surface and it can only be used on flames which move in one direction. The noise space must be translated to approximate the flame speed and direction.

Modeling Approach

The modeling approach gives a flame which is easy to texture, easy to create flame-like patterns, and it's continuous, but it looks solid since the surface cannot break and have the dynamic motion of a real flame.

Our Approach

We generated several render layers with different techniques and then combined them to keep the qualities of each approach.

Figure 41: Combining Multiple Techniques Provided the Best Flames

The Dragon Fireball Implementation

What Should Dragon Fire Look Like?

Since there is not a known reference on the subject, we had to imagine what dragon fire should look like. We wanted a very threatening and powerful effect, and initially thought about using big, long flames. But some shots in the movie called for a dragon fire that could develop over a distance of about 800 feet, which didn't seem possible with a flame style. We then decided to go for a fireball style like the one we see in "Hollywood" explosions.

Features of Fireballs

When we look at a live-action fire ball, we can identify several features that we need to mimic: an explosion gives a convective, rolling, turbulent cauliflower-like surface that expands upward from the center and dissipates in flame-like, transparent, gaseous structures, soot, and smoke.

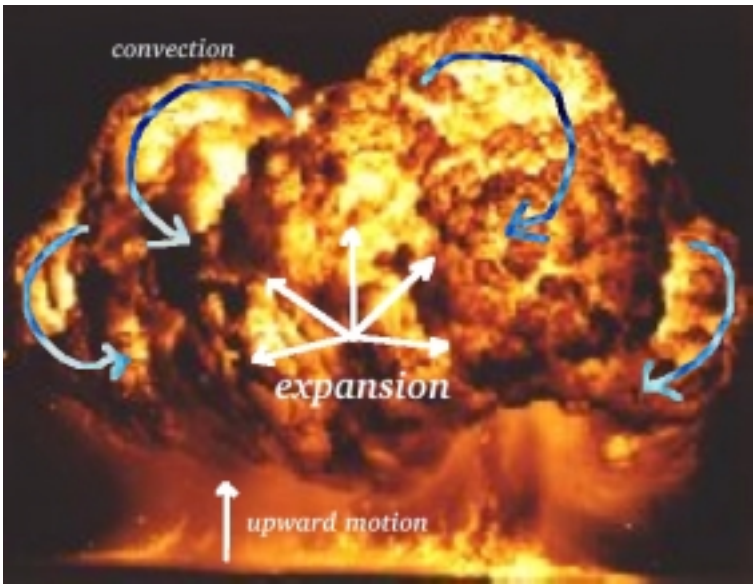

Figure 42: What Makes a Fireball Look Like a Fireball

The Dynamics of the Dragon Fireball

Since we knew how the dragon fire should look, we wondered how it should move. The directors wanted to have the flexibility to decide the direction and overall behavior of the fire, essentially making it a character that would interact with the environment and the other characters. A live-action fireball doesn't progress along a path—it mainly expands from a center point. For *Shrek*, we needed to have a fire which had the "look and feel" of a live-action explosion but was still controllable in its overall motion.

The Simulation Techniques

Our goal was to find an interactive system in which we could define where the fire would be generated and in which overall direction it would go. A simple way of animating fireballs was to use spheres for visualization (Figure 43).

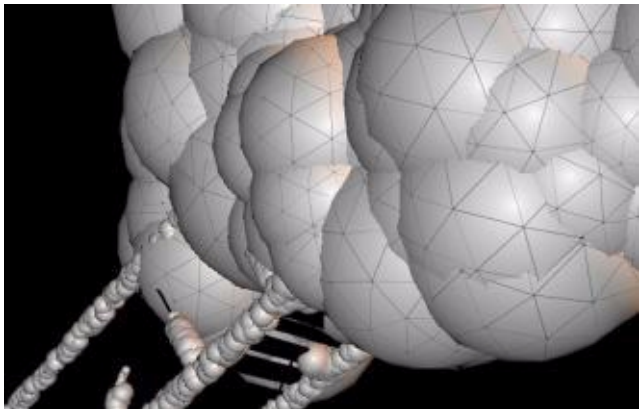

Figure 43: Using Spheres to Interactively Visualize a Fireball

The Overall Motion

The biggest challenge we faced with respect to the overall motion of the fireball was to create a forward moving, growing volume, while still keeping the rolling, convective qualities of an explosion.

Our early simulation tests were basically particles moving along a path. It quickly became apparent that such a simple approach would not suffice. The result looked solid; like the fire itself was translating through space. We decided that what our first attempt at the overall motion lacked was the convective nature of an explosion with the gasses slowing down.

It is the phenomenon of convection and expansion of gasses that we perceive when we see real explosions. The rapidly expanding gasses slow down because of friction with the atmosphere, while new, hotter gasses are fed from inside. Convection causes older, cooler material to fall, while the hotter gasses are rising. All these forces at work together produce the billowing effect that we see in the real world.

To actually compute this complex set of forces would obviously be impractical, so we set out to model the effect with rotating, expanding spheres. This proved to be a very practical, interactive way to represent the main visual qualities of a fireball. The basic approach was to start with a single expanding sphere moving in the direction we wanted the fire to travel. Hidden within that sphere, we emit another sphere which is also expanding and moving forward. By constantly emitting new spheres just behind existing ones, we were able to avoid the unnatural looking effect of the fireball simply moving forward as a whole. This approach successfully produced the illusion that the volume was actually growing from within. We then rotate the spheres in such a way as to mimic the effect of convection using a simple simulation rule based on the cross product of the direction of travel and the position relative to the center of the fireball (Figure 44).

While this approach allowed us to create the illusion of billowing and expansion, the real trick was to balance the positions, sizes, and rotations of all the spheres to form a coherent and esthetic fireball.

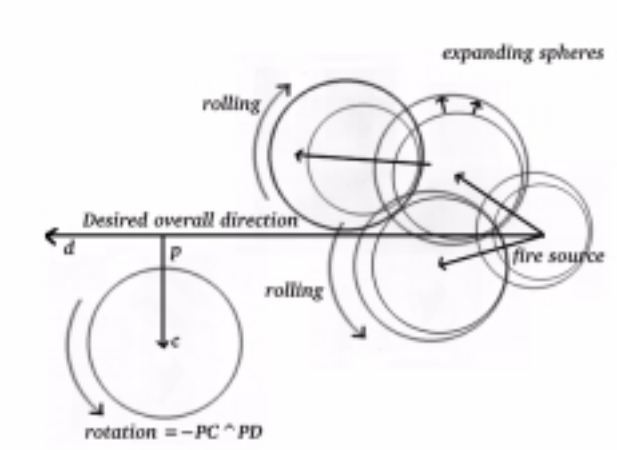

Figure 44: Rules for Expanding and Rotating the Spheres

The Secondary Motion

Now that we have a pretty good way of doing the overall motion, how do we generate the internal “boiling” of the fire ball? We decided that the secondary motion would be addressed at render time.

The Rendering of the Fireball

Creating the Surface of the Fireball

To merge all these rotating spheres, the obvious technique was to create iso-surfaces.

Figure 45: Iso-Surfaces Merge the Spheres

Texturing the Fireball

From our initial render tests of the *Shrek* fireballs, we discovered that two kinds of textures were needed.

High-frequency Texture

A high-frequency turbulent noise provided the “boiling” aspect to the fireball surface. A high-frequency solid texture was used in the computation of the iso-surface. Each sphere had its own UVW space associated with it. When the sphere was rotated or scaled, we applied the same scaling, or rotation to the associated noise space. We generated a turbulent noise that was changing with time. The variation of the noise along time created the illusion of boiling. We then introduced the noise in the computation of the iso-surface.

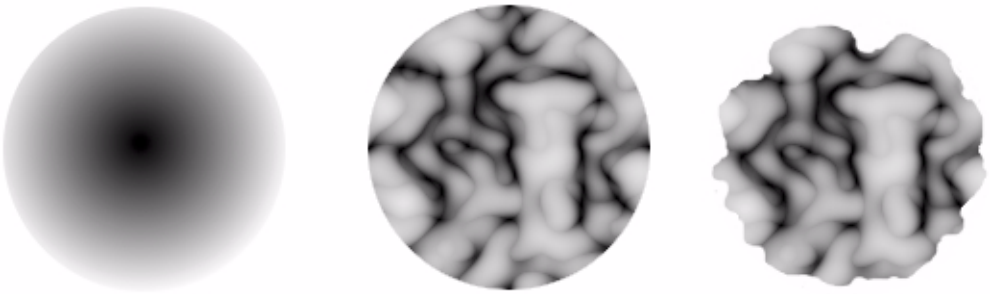

Figure 46: Noise Spheres

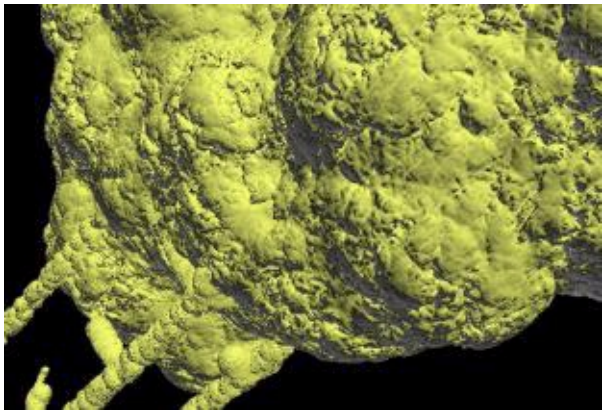

Figure 47: High-frequency Turbulent Noise on the Fireball

Low-frequency Texture

A low-frequency texture was used for coloring the fireball. As it spreads over several spheres, the texture defines hot spots and darker areas. This part was pretty tricky because we had to create a texture that would propagate continuously over independent spheres, and the noise had to move with each sphere. We couldn't use a static solid noise like we did on the flames, because each sphere was moving in different directions.

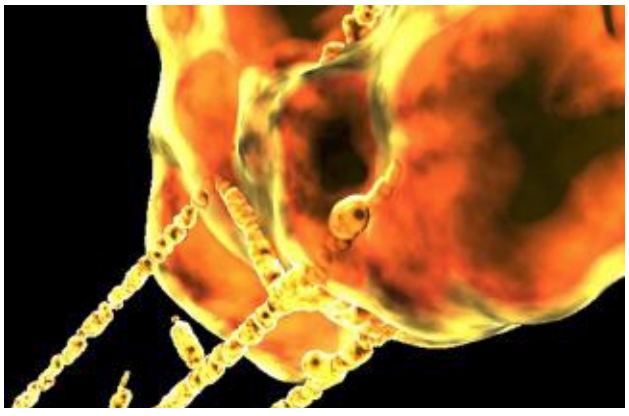

Figure 48: Low-frequency Texture Colors the Fireball

Here is the way we addressed the problem. When the iso-surface was computed, for a given part of the surface, each sphere contributes with a certain weight to the field intensity. We compute the UVW textures for each sphere and then blend each resulting color with the same weight we used for the iso-surface. In fact we blended all the other parameters of the simulation the same way—intensities, opacities, velocity (for motion blur), and various textures.

Additional Coloring Technique

One important rendering technique used to improve the realism of the fireball was to modify the color of the fireball based on the dot product of the normal of the surface with the direction of the ray casted by the camera. Using colors we chose from real fireballs, we created a gradient that is applied on the surface following the rules in Figure 49.

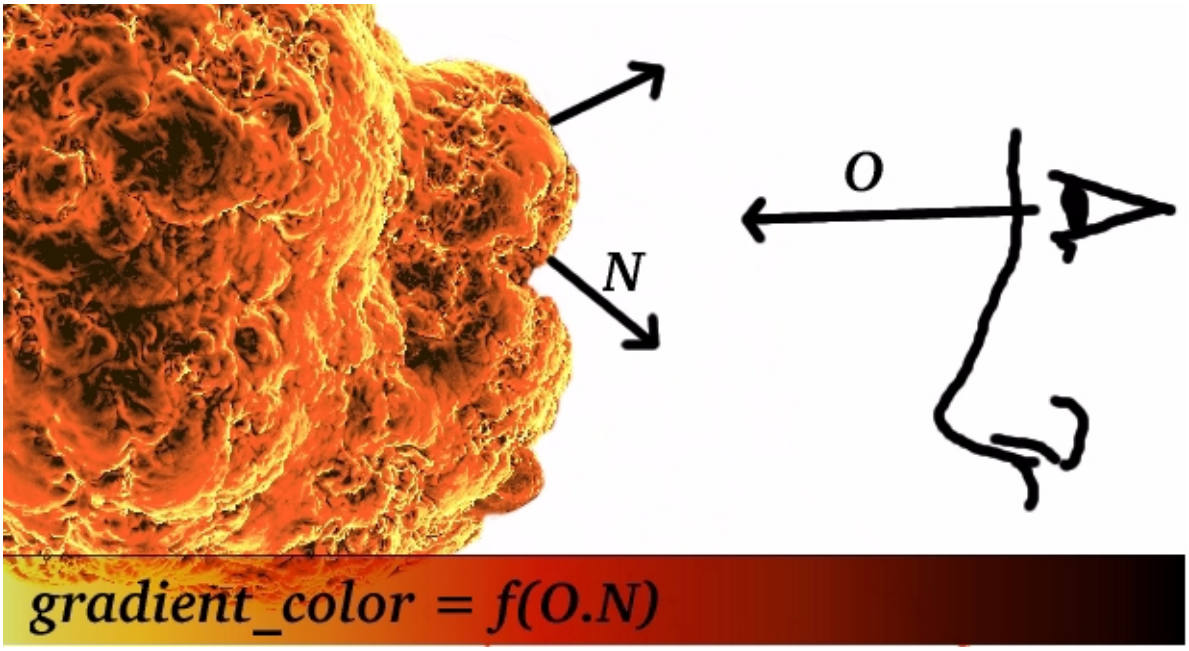

Figure 49: How to Apply Color Gradient

How to Simulate Dissipation and Improve Depth

So far, we have been able to render a hard surface with hard contours (Figure 50). But we still need to give a more voluminous and gaseous aspect to that surface. We have seen that when a fireball dissipates, it slows down, becomes transparent, flame-like, and gaseous. We used a pretty smart trick to give the illusion of that dissipation. Using the same spheres we used for the overall motion, we computed another simulation. But this time, we slowed down the rotations and increased the transparencies. We rendered the sphere with a ray-marching volume shader. The result was the “flame” equivalent of the iso-surface (Figure 51). Then we composited the iso-surface rendering through the density of the “flame” image. Each time an area becomes transparent, the fire seems to slow down because it reveals the shapes of the slowed down simulation (Figure 52).

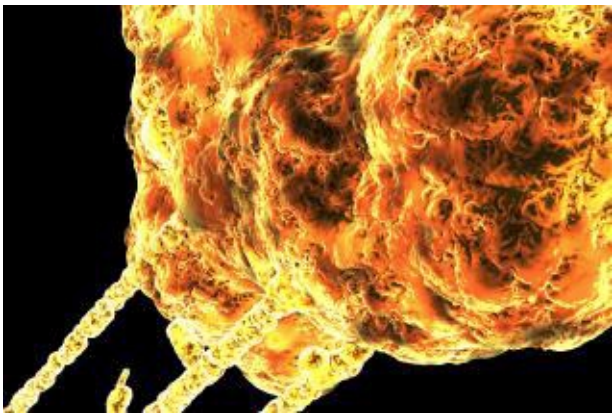

Figure 50: A Hard Surface with Hard Contours

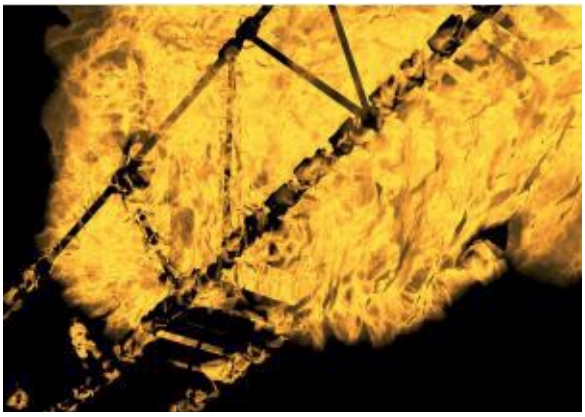

Figure 51: Slowing Rotations and Increased Transparencies on the “Flame” Add Depth

Figure 52: Final Composited Fireball

Conclusion

To create fire for *Shrek*, we used a combination of the particles/volume and textured surface/iso-surface approaches. This dual approach was necessary to solve both the dynamic and rendering problems associated with fire. It is possible to create realistic fire with different techniques, but we recommend the use of the following simple rules:

- Use a particle approach for the dynamic (physically correct or not). It will provide nice breaking and will easily interact with the environment (forces and obstacles). Rendering the particles with a volume shader will provide a sense of depth, a gaseous look, and a nice contour.
- Use a surface (model, iso-surface) approach to define UV or UVW textures. The rendering will provide a continuous texture.
- Composite the two layers using the particle layer as an alpha channel.

Of course, the two-layers trick is an oversimplification of what we really did on *Shrek*, but that was the underlying principle used.

Future Work

Years from now, physically correct simulation will be able to describe not only the dynamic, but the complex shading properties of all different kinds of fire. Before such solutions are available, however, the entertainment industry has the need to create realistic digital fire. Some research should be done to animate a continuous solid noise based on a particle simulation. The main problem with that technique is the need to avoid excessive stretching of the texture. The overall patterns should move with the particles but not stretch.

Acknowledgments

We would like to thank all the people who worked on the various fires for *Shrek*: Erdem Taylan, Marty Usiak, Alain De Hoe, Philippe Gluckman, Ken Bielenberg, Scott Singer, David Allen, Apurvah Shah, Mark Edwards, and Terran Boylan.

Shrek Effects—Animating Trees

Scott Peterson, Senior Effects Animator

Introduction

One of the visual themes in *Shrek* is to see objects as large, simple, geometric shapes from a distance, but to still see them as natural and complex when close up. When designing trees, we start with complex natural branching patterns and then prune them down to more simple forms. We animate the trees by applying wind forces and character-driven interaction forces to affect the shape of each branch.

Our challenge was to create a system that could allow us to render entire forests of animating trees. We divided the problem up into several categories:

1. Creating tree models
2. Representing trees using curves, and
3. Deforming tree models for animation.

Creating Trees

Most algorithms for creating realistic trees, including ours, use recursion in order to mimic the biological process of branches growing and splitting. One of the most difficult challenges in developing a tree-growing algorithm is deciding on the rules it uses in order to achieve a desired appearance. We chose to implement a technique which assigns a unique set of attributes to discreet branch levels based on the depth of each branch in a hierarchy. *Split_angle*, *number_of_branches*, *attract_up*, are the names of a few of the attributes which allow animators to control various aspects of a tree at one level. Each attribute is simple in name and behavior so that animators can predict how changing a value might change the appearance of the tree. By breaking a tree down into levels, it allows an animator a fine degree of control over the entire tree.

Our system can either generate an entire tree procedurally or it can grow branches on a hand-modeled trunk. The procedural growth system was partially inspired by a curve-based system discussed by Weber and Penn [1]. Once a tree is grown, the animator can prune it by painting red areas on an image of the tree where branches should be cut.

Representing Trees

Many researchers and production studios have already addressed ways to create realistic trees using L-systems [2] and other procedural approaches. We decided to use a curve-based approach because curves can be readily adapted to represent tubes of geometry, they are not limited to branching on discreet segments as is a simple transformation hierarchy approach, they are compact data structures, and we already have a flexible curve file-format which is a good foundation for passing data between separate tools. Other information such as connectivity and leaf distribution is stored in a separate file.

Each branch of a tree, including the trunk, is represented by a tube. Each tube is a single curve where each control vertex has a radius attribute in addition to a position attribute. This data describes a circle extruded along a curve as seen in Figure 53.

The connectivity as well as the hierarchy of curves is also stored so that when the trunk of a tree is deformed, all of the branches of the tree can be transformed as well in order to stay connected to the trunk. This is shown in Figure 54.

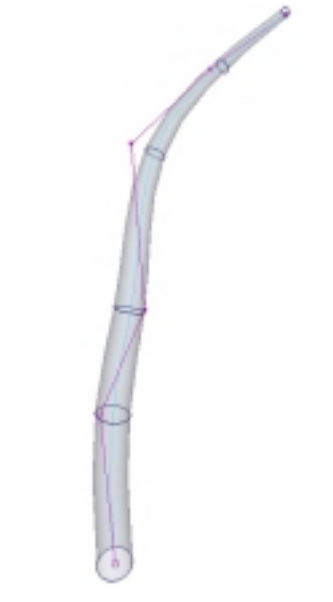

Figure 53: Circle Extruded Along a Curve

Figure 54: Curve Connectivity

A hierarchy of curves simply implies that each curve, with the exception of the trunk, has a parent curve. Connectivity is the parameter value along the parent curve where the child curve begins.

This compact representation of a tree allows us to write separate tools for modeling, animating, and rendering trees.

Trees are converted into geometry by creating tubes for each curve. Tubes that are connected to the endpoints of other tubes are stitched together in order to hide seams. Other branch tubes simply penetrate through the sides of their parent branches. In extreme cases where this penetration is obvious, we use constructive solid geometry to create the union of several branches, and then relax the resulting polygonal geometry in order to create smooth fillets as seen in Figure 55.

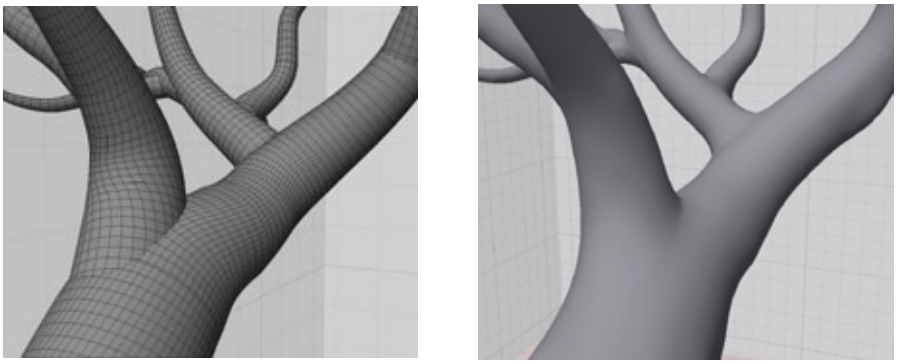

Figure 55: Creating Smooth Fillets

Leaves are created procedurally at render-time. Each leaf-level branch stores the number of leaves that should be attached to that branch. Leaves are positioned using a small set of leaf-growing parameters such as angle of deflection from the parent branch, scale, random scale variation, scale variation as it relates to the length of a branch, rotation about the parent branch, etc. Leaf parameters apply to an entire tree, and are therefore stored only once per tree.

Deforming Trees

There is a wide variety of animation that the tree system must be able to deal with. Most outdoor environments in *Shrek* have a gently blowing breeze which causes tree branches to sway. Sometimes branches interact with characters. In one shot, Donkey pulls a flower off of a tree, and the connecting branch reacts by stretching quickly and damping back down to a slow oscillating motion.

Almost all of the tree animation is expressed in terms of force fields in space. Force fields can be derived from character motions, by large fluid simulations, or by expressions written in a scripting language.

Deforming a Branch

A force is sampled only at the tip of each curve in a tree. Force is then converted into a bend factor. The bend factor is represented as a vector which is the same direction as the force but with a different magnitude. The bend factor has an additional oscillating motion which is computed based on the length and radius of the curve, and it is scaled based on hierarchical level of the branch that the force is affecting.

Equation 1 shows how to prepare a vertex-level bend factor.

$$\mathbf{b}_i = \mathbf{b} \cdot \left(\frac{i}{n-1} \right)^2$$

Equation 1

b is a vector representing the bend factor of a branch. $\mathbf{b}_i$ is a vector representing the bend factor of control vertex *i* within a branch. *n* is the number of control points in the curve.

Notice that you can change the characteristic of the bend along the branch by changing the power of two to some other power, or a smooth curve interpolation. Equation 1 has been simplified to a square function for easier reading.

Equation 2 shows how $\mathbf{b}_i$ is used to change the shape of a curve at point *i*. Figure 56 illustrates the function.

$$\mathbf{c}_i' = \mathbf{c}_{i-1}' + \frac{(\mathbf{c}_i - \mathbf{c}_{i-1}) + \mathbf{b}_i}{|\mathbf{c}_i - \mathbf{c}_{i-1}|}$$

Equation 2

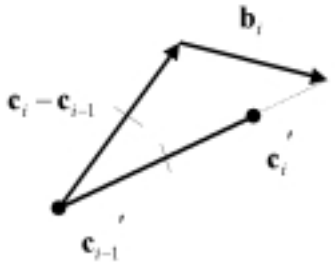

Figure 56: Bending One Point in a Curve

$\mathbf{c}_i$ is the position of control point *i*. $\mathbf{c}_i'$ is the resulting bent position of $\mathbf{c}_i$.

Maintaining Tree Structure

All the points of every curve are transformed by a 3×3 matrix containing orientation. The rotation of a branch always occurs about the point where it connects to its parent branch. Since the trunk curve has no parent, its rotation matrix is set to identity and is not transformed.

Branch deformations are applied to the tree in a depth-first traversal. Rotations accumulate up the hierarchy of branches so that a bend in the trunk propagates a change in direction all the way up to the highest-level branches of a tree. Because we have not stored an actual transformation hierarchy for the tree, we must derive one on the fly as we deform each branch. A copy of the undeformed tree is used as a reference for maintaining branch angles for the deformed tree. Figure 57 shows the effects of rotating a tree with and without accumulating orientation from level to level.

Figure 57: Accumulation of Rotation Versus No Accumulation of Rotation

We must create a rotation matrix that adjusts for any bend that has been applied to the parent branch (Figure 58). We compute this rotation by looking at the difference between the tangent of the reference tree parent curve and the tangent of the deformed tree parent curve at the point of connection (Figure 59). In order to be able to accumulate rotations up the hierarchy of the tree, we first transform the direction of the parent curve by the parent's orientation matrix (Figure 60). This puts the two directions in the same local space.

Figure 58: Finding the Orientation of the Dotted Twig on the Right Branch

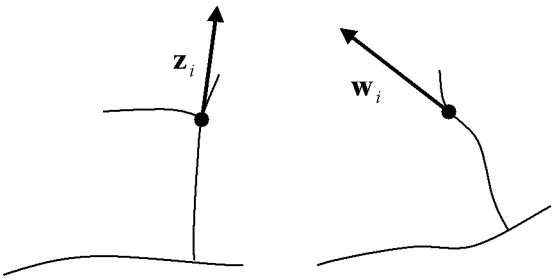

Figure 59: Consider the Two Parent Branch Tangents

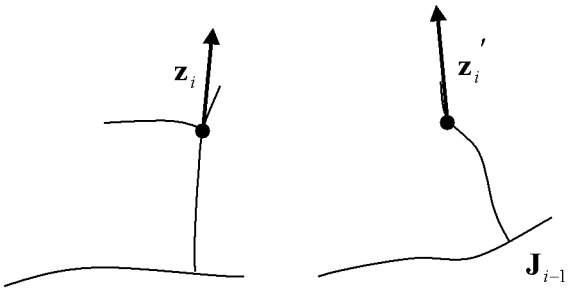

Figure 60: Transform Parent Curve Direction into Local Space of Deformed Branch as Defined by Matrix J_{i-1}

$$\mathbf{z}_i' = \mathbf{J}_{i-1}\mathbf{z}_i$$

Equation 3: The Transformation of the Reference Parent Tangent

$\mathbf{z}_i$ is the reference tree tangent. $\mathbf{J}_{i-1}$ is the parent orientation matrix.

We create a rotation about a vector perpendicular to the two tangents such that the first tangent transformed by this rotation is a vector that has the same direction as the second tangent (Figure 61). We multiply this rotation by the parent rotation matrix in order to create a new local space for the branch. Each control vertex of the reference tree branch is multiplied by this rotation in order to create the newly deformed branch (Figure 62).

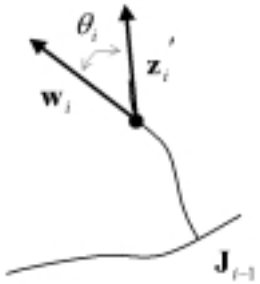

Figure 61: Create a Rotation Which Will Transform z_i' to w_i

$$\mathbf{J}_i = \mathbf{J}_{i-1}\,\mathrm{R}(\mathbf{z}_i' \times \mathbf{w}_i, \theta_i)$$

Equation 4: Create a Branch Rotation and Then Accumulate it with a Parent Orientation

R(**v**,*a*) returns a matrix that defines a rotation of *a* radians about a vector **v**.

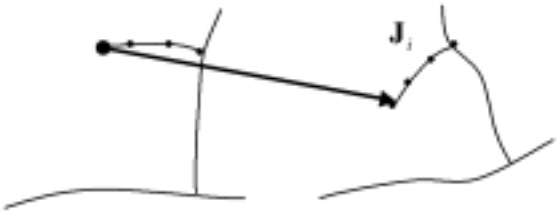

Figure 62: Transform Control Points of Reference Tree by J

Conclusions

We use a compact file format which stores curves and their connectivity in order to write separate tools to create, animate, and render trees. In order to deform trees for animation, we devised a technique to deform individual branches and a technique to accumulate branch orientations.

Rather than store a transformation hierarchy on disk for each tree, we derive one from a hierarchy of curves.

Bibliography

1. Jason Weber, Joseph Penn, “Creation and Rendering of Realistic Trees”, SIGGRAPH 1995 Conference Proceedings.
2. David Prescott, “A Visual Effects Galaxy”, “Creating a Digital Tree and Using L-systems in Production for *What Dreams May Come*”, Siggraph ‘99 Course Notes.

Shrek Effects—Mud and Water

Juan Buhler, Senior Effects Animator

The final sequence in *Antz*, PDI's first computer-animated feature film, involved a flood in the anthill. For the development of the tools necessary for this endeavor, PDI recurred to the help of Nick Foster, whose Doctoral thesis involved the animation of fluids [1]. Nick developed the "flu" tools, which are now the backbone of PDI's fluid simulations. They allow animators to set up an environment, determine initial liquid distributions and obstacles, and obtain realistic fluid motion. Our second feature film, *Shrek*, involved some new challenges on top of what we had for *Antz*. These notes attempt to describe the process of developing the tools and systems used in the movie.

The Flu System

Although the Navier-Stokes equations to describe the behavior of a fluid have been known for 150 years, their complexity makes it very expensive to simulate fluid motion at a resolution suitable for visual effects. Nick Foster's main contribution was the use of "marker particles" on a low-resolution grid where the Navier-Stokes equations have been solved. These particles introduce an additional level of visual detail, allowing us to solve the fluid equations on a low-resolution grid (Figure 63).

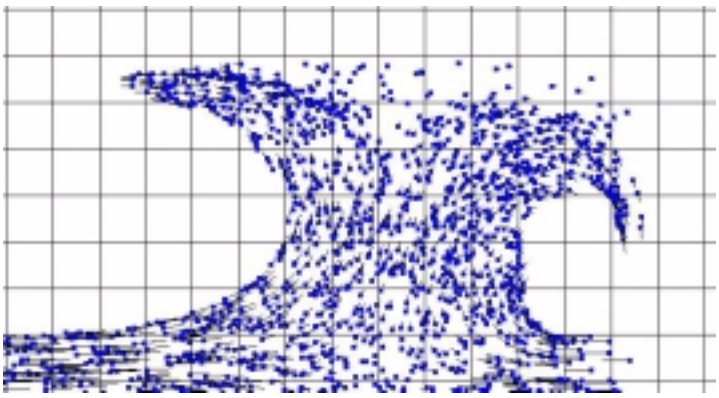

Figure 63: Marker Particles Introduce Extra Visual Detail to an Otherwise Low-resolution Simulation

Challenges for *Shrek*

For *Shrek*, we knew that we were going to have to depict an overall better interaction between objects and fluid, that we'd have to make thick fluids, like mud, and that some scenes included two fluids of different viscosities mixing together. The environments in *Shrek* are very complex and alive, and they are at a human scale. This means that we didn't have the latitude that we had in *Antz* regarding scale of the various effects.

Mixing Fluids

Our first attempt was the simplest one: Make two passes, simulating the thicker fluid first, and then use the data from this simulation as an obstacle for the second one. This approach is good when the thicker fluid is not splashing too much, and can be considered a height field. If the thicker fluid is separating, splashing, and merging onto itself, the second simulation will see an infinitely massive obstacle moving around, and this would cause instabilities. For example, imagine two walls of mud closing into each other. When the first (mud) simulation runs, everything is smooth, and the mud doesn't find any resistance. When the second (water) simulation runs and if some water goes into the space between the two walls of mud, then when these close, the water will have nowhere to go. This will cause a big differential in the local pressures which will lead to the water "boiling," or jumping around in a non desired way (Figure 64).

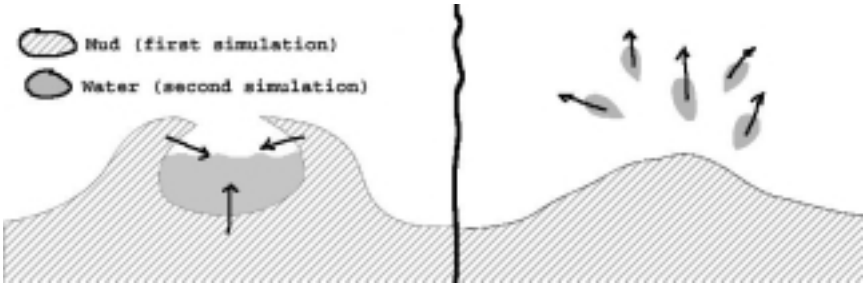

Figure 64: If the mud and water simulations are run independently, the first simulation acts like an infinitely massive barrier for the second one, causing instabilities

As mentioned before, this technique works in cases where the bottom fluid moves only slightly, and can be considered a height field. We used it in the opening title, where the mud was not splashing around and where we actually needed to carve the letters that form the *Shrek* logo into the mud. For the rest of the cases, Nick Foster added variable density and viscosity to his system to allow for one simulation with two kinds of fluid.

In order to integrate the flu tools with the shot structure, a system in PDI's *script* language was developed. This system standardized the way animators worked in shots, the places where data was saved, the utilities used to process and visualize simulations, etc. It also allowed animators to run multiple simulations within a bigger "pool" of particles in order to optimize the simulation time in cases where the environment was big yet only certain locations needed complex fluid motion.

Figure 65 shows two rectangles, one around Shrek and the other around the background guard. The flu simulation only happens inside these, since the rest of the particles don't need fluid motion.

Figure 65: Optimizing the Simulation Space

Enhancing Fluid Simulations

In many cases, the flu simulation was enhanced with traditional, dynamic particles. This allowed us a fine degree of control on little splashes and water dripping off objects—in cases where the interaction was not as important. In the scene where Shrek breaks open the beer barrels, the flu tools were used for the big flow of beer. But the beer dripping around the edges of the barrel, necessary to integrate the simulation with the rest of the environment, are particles. If rendered alone, the beer gush looks static, almost as if it is frozen in mid-air. The extra particles, in the form of foam, makes the beer motion apparent (Figure 66).

Figure 66: Additional Particle Systems Add Detail and Complexity to the Raw Output of the Flu System

Getting Objects Wet

The wet map generator was another modules written for *Shrek*. This system takes a fluid simulation as input (which could be flu, traditional particles, or a combination of both) and generates the data necessary to combine dry and wet versions of the materials touched by the fluid.

The system also uses a polygonal representation of the object that we want to get wet. For each frame every particle of the fluid is tested against the objects. When a particle gets closer than a certain distance to a polygon, its position is represented in a space local to that polygon, and saved in a parallel database. The newly saved particle will from now on move with the object. The resulting cloud of particles, all stuck to some part of the object, is used as the input for a map module within PDI's rendering pipeline. This specific map uses the cloud of particles to generate a scalar field in 3D space, based on the distance to each particle. Material shaders can use this map to mix between dry and wet versions of themselves.

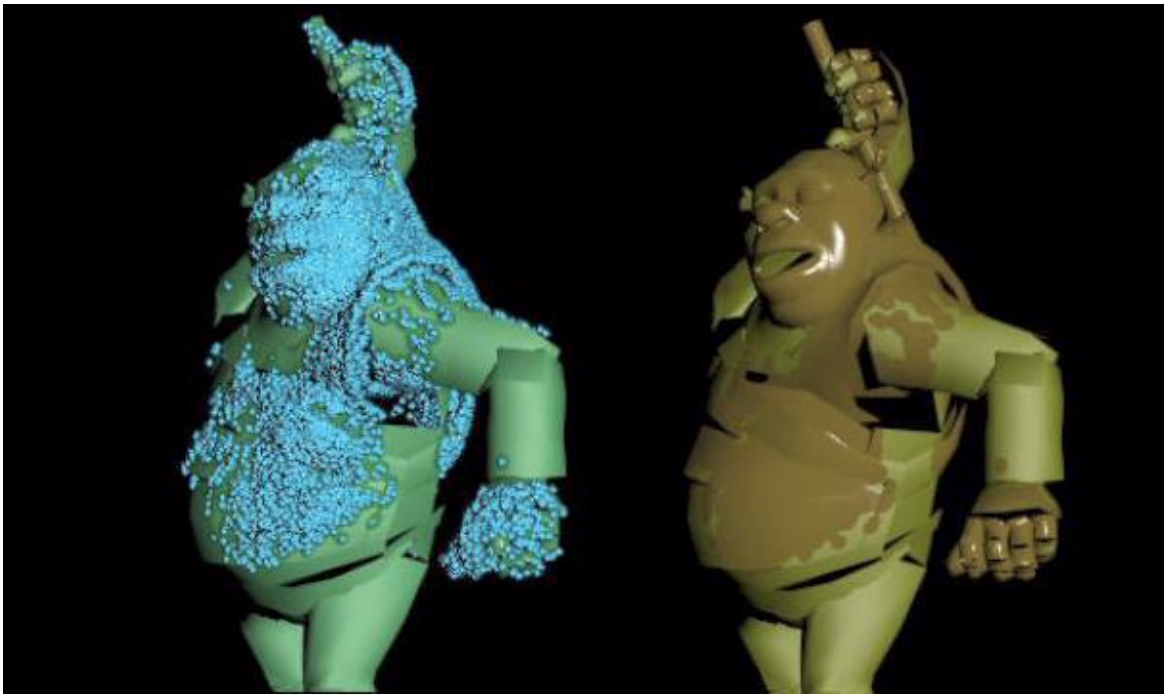

Figure 67: Particles Stick to Objects, and are Later Used to Mix Between Dry and Wet Material Properties

Rendering

The particles that make the output of the water and mud system were rendered using various techniques. Our particle renderer, initially developed for *Antz*, was improved to use the same lights as our normal renderer in order to ease the integration of particles into the scene. This renderer is extremely fast, making it easy to render over a million particles in just a few minutes.

We also generated surfaces from these particle sets. This is vitally important when doing fluids like water which don't have a particle look, especially when they are flowing slowly. In the *Shrek* title shot, for example, the mud geometry was generated by projecting a surface grid onto the particles. In order to get the textures to move correctly, particles would carry UV coordinates with themselves. These UV values were inherited by the projected surface, allowing us to texture the mud.

Conclusion

One of the most important things to keep in mind when setting up physical effects in a production environment is control. No matter how realistic your simulator is, chances are that you will have to tweak the simulation to achieve the vision of directors and art directors. Systems that are open and clean in terms of data and workflow make this much easier to achieve.

Suggested Reading

[1] Foster, N. and Metaxas, D., "Realistic Animation of Liquids," Graphical Models and Image Processing, 58(5), 1996, pp. 471-483.

Shrek Lighting

Mark Wendell, Sequence Supervisor

Within a period of just over two years of production, PDI's Lighting Department managed to light, render, and composite over 119,000 frames of extremely detailed, complex, and beautiful animation for the feature animated film *Shrek*. This represents a huge challenge from artistic, technical, and resource management standpoints. This portion of the course is a brief overview of how PDI met these challenges, and is divided into sections on art, technology, and pipeline.

Art

The key to producing a well-lit film is to start with good design. One of the initial tasks for the production designer is to plot the use of various visual elements as they relate to story, the goal being to set up a visual vocabulary that enhances the telling of the story. Audiences naturally respond emotionally to the visual elements of a film, such as color, depth, shape, line, and so forth. Those elements are always on screen, whether you design them or not, thus it is critical to control these components. For example, Figure 68 shows an early color "script" for *Shrek*. The emphasis in this example is not on shape or detail, but on color harmony, value relationships, and mood.

Figure 68: Color Script. Simple color studies showing very early color concepts and their accompanying palettes.

From such studies, we begin to establish visual "rules" for the film. Note that although there are stereotypical ways of using visual elements (e.g., red for passion), there is no need to stick to any tired standards. The important point is that you create associations early on in the film between visual elements (such as color), and the various characters or emotional states in the story. Once these associations are created, you can use them to emphasize or de-emphasize story points as the film proceeds. A classic example of such a stylistic rule is the music associated with the shark early on in *Jaws*. Later in the film, all you have to do is hear the music to know that the shark is lurking nearby. Likewise, in Kubrick's *The Shining*, circular shapes are associated with good, while hard-edged triangles and quadrangles are associated with evil. Once the rule is established, throwing some square shapes into the frame increases the audience's apprehension that evil is nearby. Note that these rules need not be overt or obvious, and in fact can be more powerful emotionally when they work on a subconscious level.

In *Shrek*, we made specific use of color, line, and other visual elements to emphasize many aspects of the story. Following is a brief survey of some of the major environments seen in *Shrek*, their relevant contribution to the story, and how the design and lighting served the story points. Since *Shrek* is a "quest" film, we ended up with a large number of different environments. Without some general rules and simplifying themes, we ran the risk of over-complicating and thus muddling the storytelling.

The Swamp

Although swamps are traditionally seen as creepy and murky places, in *Shrek*, the swamp needed to say "home sweet home" and be fun, like a kid's treehouse. The design challenge was to appoint Shrek's swamp environment realistically with hanging moss, creeping vines, dense foliage, and mucky ground, and yet make it feel cozy, welcoming, and serene (see Figure 69). To this end, we made use of rich warm lighting, volumetric shafts of sun, inviting atmospheric lightness in the distance, soft breezes blowing foliage and pollen magically about, and green everywhere. Trees were modeled and placed to create lots of horizontals, emphasizing serenity, quiet, and balance. Every edge and surface is soft and curved, and shadows are generally soft and warm. We originally tried making the swamp darker and more "realistic", but quickly learned that this was too oppressive and threatening, counter to our story goals (Figure 70). By treating mid-ground trees as darker silhouettes against a distant filtered light, we were able to simplify the swamp compositionally as well as save on mid-ground tree geometric complexity. Examples of the final rendered results can be seen in Figure 71 and Figure 72.

The concept of unifying large areas of complex geometry such as foliage into organized zones was critical for us in many instances. While we wanted the movie to be richly detailed and natural, we also wanted to be able to arrange each frame as a painter would: focusing the eye on the area of interest and creating global harmony. If we scattered trees haphazardly and lit an entire forest with one key light, then we would quickly run the risk of creating visual clutter, muddling the story point of any particular scene. Thus many of our lighting techniques were geared towards simplifying the complexity: dividing foreground from midground from background, unifying areas of visual complexity by controlling color and contrast, and using shadow to force the eye to the desired area. Rarely was a scene "accurate" in the sense that the same illumination was falling everywhere on the scene in the same way and that shadows precisely matched all the geometry. Visual storytelling, not accuracy, was our goal—and we had few compunctions about artistically cheating just about everything to serve that end.

Figure 69: Production Design for Shrek's Swamp Home

Figure 70: Early Production Design Concept for Shrek's Swamp

Figure 71: Establishing Shot of Shrek's Swamp Home

Figure 72: Shrek's Fishing Pond in the Swamp

Duloc

In contrast to the homey swamp, we wanted everything about the evil Lord Farquaad to be totalitarian, cold, impersonal, even fascist. He is obsessed with creating “the perfect kingdom”, thus his environment is full of strong exaggerated vertical elements and precisely repeating linear motifs. His architecture is always in contrasty blues and whites (Figure 73). We light his world with harsh sunlight creating crispy diagonal purple shadows. We minimize soft organic curves in exchange for straight lines. Interestingly, we still needed to break up straight lines with lots of small architectural and brick-n-mortar details to keep them from feeling too computer generated. We also added more “weathering” to the surfaces, just to keep the feeling of organic realism, even though we imagined Farquaad would keep every wall of his “perfect kingdom” spotless!

To cover a large amount of wall space with stone texture, we wrote a brick shader that allows for the random scattering of any number of hand-painted stone maps. We were able to save on texture memory and painter time by using only six basic stone color and bump maps, as the shader randomly applied these maps with a user-definable palette of hue and value variations. Separate maps were then painted globally over the resulting bricks to simulate water staining, sun bleaching, and other weathering influences. We also painted dirt maps on the base of the walls where they contacted the ground plane to enhance the ground-wall integration.

Redwood Forest

Another important environment in the film is the redwood forest. In terms of story, this location acts as a barrier—both physical and psychological—separating Shrek’s swamp from the rest of the world. It is also the initial meeting place for Donkey and Shrek, where they begin to reveal their personalities to the other. Thus the environment needs to convey a sense of safety and isolation, where the characters can feel comfortable enough to begin letting down their own barriers. Here we lit their interactions in warm pools of sunlight, letting the world around them drop off into shadow (Figure 74). An impenetrable wall of huge vertical redwoods always surrounds the action, emphasizing privacy. We didn’t want the forest to feel too harsh and imposing, though, so we treated the local environment with lots of soft lights, rounded boulder shapes, and plenty of fur shader-based moss and grass.

Sycamore Forest

In contrast to the “male bonding” goal to the design and lighting of the redwood forest, the sycamore forest is a place where Shrek and Fiona begin to open up to each other. Here we wanted a more soft organic forest environment, appropriate to the introduction of the movie’s love story. The trees all have curvy feminine forms, and the light is soft, indirect, and canopy-filtered (Figure 75). When the characters do fall into direct sun light, it is usually as a bright rim which blooms romantically around them. The shadowing for the forest, always soft and warm, was projected through animated foliage maps so that scenes were never entirely static. Volumetric beams of light proved

useful as simplifying elements, since they allowed us to lower the contrast on busy sections of the background in an aesthetically pleasing and natural way.

Figure 73: Farquaad's Tournament Arena with the Tower of Duloc in the Background

Figure 74: Shrek and Donkey Get to Know Each Other in the Redwood Forest

Figure 75: The Sycamore Forest Environment.

Pine Forest

By the time we get to the "love montage" sequence, Shrek and Fiona have clearly fallen for each other and we wanted our environment to convey this. We chose a bright high-altitude pine forest, representing freshness and emotional clarity. Even though they are in high meadows, we never created long vistas of mountains because this scene is all about the growing love between Shrek and Fiona, and we wanted to maintain intimacy and simplicity. The sunlight is bright and clear and crisp, the flowers are abundant, and the grassy green hills are alive with the sound of music (Figure 76). Keeping horizon lines high kept the focus on the characters and their growing intimacy.

Dragon's Keep

Here we created the most stylized lighting environment in the entire movie. We wanted the keep to be scary, dramatic, almost theatrical (Figure 77). To that end, we limited our lighting to a simple graphic two-color palette—lava red from below, and diffuse blue/purple from above. This let us play with lots of fun horror-movie cliches with hellish red underlighting, deep mysterious shadows, high contrast shading. We avoided global lighting sources, opting instead for lots of localized spot sources. This resulted in the ability to alter the shapes of everyday objects, making them mysterious and allowing the viewer's imagination to fill in the dark spaces.

Within the Dragon's keep, the one place where we broke this rule was in the Dragon's Lair, where she dotes over the captured Donkey. Here we had fun playing up an overtly sexualized "bedroom" theme, with bright saturated pink, violet, and gold lighting—this is the dragon's love pad and she's trying to seduce donkey (Figure 78).

Figure 76: The Alpine Meadow Environment

Figure 77: Donkey is Chased Through the Dragon's Keep

Figure 78: Dragon Doting Over Donkey

Lighting Design Principles

1. Keep it Real

The overall goal for the film was to create a natural yet stylized look appropriate to a fractured fairy tale. We wanted to find a balance between realism and fantasy, between natural and unnatural. Character designs were stylized, yet their skin textures, clothing, motion, and deformations were kept realistic. Environments were often fantastic in proportion and composition (Farquaad's tower in Duloc is so large that it would easily fit the entire population of San Francisco as an office building), yet the local colors of materials were realistic and richly detailed. Likewise, lighting was allowed to be stylized and artfully exaggerated, but could never be cartoony or wholly unrealistic. We learned this lesson by taking things too far several times and having to back off.

For example, in Farquaad's dungeon, the original design called for an almost entirely blue room with harsh underlighting through floor grates (Figure 79). We tried actually painting the brick walls blue, and quickly learned that no amount of lighting could make them appear correct. So we went back and repainted almost the entire dungeon in more natural tones, and let the lighting be the stylized aspect of the scene. This allowed us to maintain some of the overall cool cast to the scene, and yet when we needed areas or light or warmth for balance and realism, we could get them without cheating (Figure 80).

Figure 79: The Original Production Design for Farquaad's Dungeon

Figure 80: The Final Dungeon Look

2. Keep it Rich

Related to the above, the idea of "keeping it rich" is concerned with not allowing scenes to be too stylized and graphic. Thus, even if we wanted an environment to be dark and spooky and blue (such as the dragon's keep or the dungeon), we found that we always had to provide some compositional balance by introducing some warmth somewhere in the frame. This form of counter-weighting kept us honest, in a sense—we were able to take the stylization only so far. Without balancing compositions and keeping the lighting rich, shots no longer felt natural and real.

3. Sweat the Values

Another guiding principle for the lighters was to always sweat the value relationships. To keep things looking real, we made sure that different materials always maintained their relative local-color brightness relationships in all lighting conditions. This may seem like an obvious point, but it turned out to be a valuable touchstone in solving difficult lighting situations. In fact, whenever a shot wasn't working, we'd look at the luminance alone and in many instances, recognize right away what was wrong. In fact, lighters would often solve the value relationships first before really even bothering with hue and saturation; if a shot doesn't work value-wise, no amount of fiddling with different colors is going to get you closer. Occasionally the production designer would provide lighters with simple four-tone black and white studies as lighting guides: these represented how dark versus light materials should appear in light and in shadow. It is only a slight exaggeration to say that once a lighter nailed these tonal relationships in a shot, they were ninety-percent done with the shot. This was particularly true in organically complex environments such as the various forests and swamp, where the technique of separating foreground, midground, and background planes from each other by tonal contrast was critical for visually organizing the images.

4. Shape Shape Shape

By constantly emphasizing shape through shading, lighters were able to enhance the film's sense of hyper-reality. This was accomplished by installing standard portrait-lighting setups (key/core/fill) individually for each character and major environmental element in a shot. With separate setups, each character's lighting could be cheated and tuned separately, often quite differently from each other. The goal here was to enhance the dimensionality of the characters and environments as much as possible. Ironically, by often exaggerating the shading way beyond what would happen naturally, we were able to achieve what appeared as a quite realistic look.

Technology

Skin and Hair

Among the technical challenges faced by PDI's lighting department was the realistic rendering of human skin and hair. For skin, we wrote a custom shader that simulated several key aspects of real skin—translucency, oily highlights, softness, and selective back-lighting. The skin shader employs a physically-inspired layered model. There are three important layers: the oily layer on top and the two layers of the skin itself called the epidermis and the dermis. The epidermis is the top layer of skin and contains most of the skin's pigments that give it it's color, including surface features such as blemishes and freckles. Underneath is the much-thicker dermis layer which is where all the blood-flow happens. Controlling the thickness of the epidermis layer via maps allows more of less of the dermis to be visible in different regions (such as the lips). (Figure 81). This layered model was shaded according to an implicit model of sub-surface scattering, giving the warm "flush," softness, and tonal variation so critical to representing realistic skin. Another great feature of the skin shader is "bone mapping", which allowed for the painting of areas which do not contain actual bone and allow backlighting to glow through, such as the ears (Figure 82).

PDI's hair shader, like many of our other geometry shaders like the foliage shaders, generates lots (LOTS!) of geometry at render-time. This allows the controlling "clump" models to be geometrically light. A variety of methods are available to drive the motion of these clumps, including simple spring dynamics, more complex collision dynamics, and of course individual keyframe control based on a simple internal skeleton. Once clump models are bound to the hair geometry shader, thin ribbons normal to the camera are generated within each clump (Figure 83 and Figure 84). Shader attributes give considerable control over placement, density, curl, waviness, and a myriad other parameters of the ribbons within each clump.

Lighters generally use depth-map or ray-traced shadows for most scene geometry; however, for hair we had another option: volumetric shadows. These worked through a pre-process phase in which the clumps were divided up into voxels, and then a ray was shot from each light to each voxel to determine that voxel's visibility. Each voxel's position in the clump, relative to the light, is then stored in a file. At render time, the renderer imposes shadowing on the ribbons according to their position within the clump voxel space. This results in fast, smooth shadowing of extremely complex hair geometry. The millions of polygons making up the ribbons are not involved in the shadowing operations whatsoever—they simply receive the same shadowing that the voxel which contains them would get. While lighters also have access to depth-map shadows for hair, and in some instances this is appropriate (such as in the case of distant crowds of humans, where the hair resolution was kept relatively low), volume shadows proved very effective for all hero characters in the mid- and foreground.

Figure 81: Close-up Example of the Skin and Hair Shader's Results

Figure 82: Translucent Backlighting to Shrek's Ears

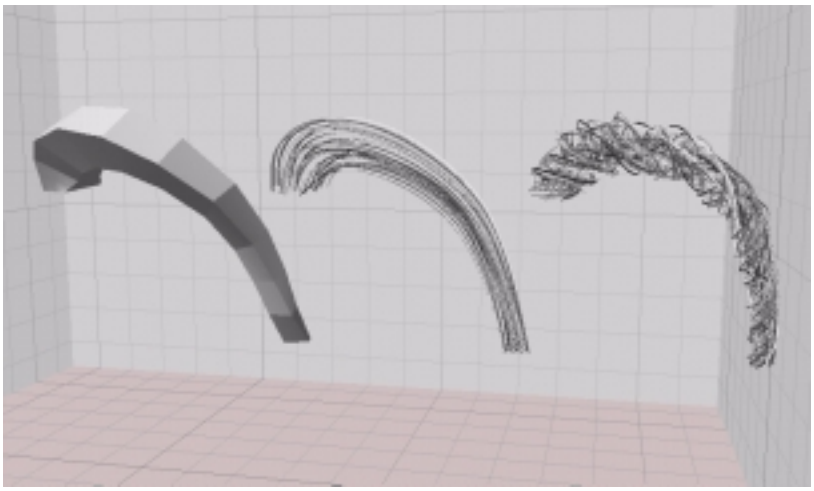

Figure 83: Hair Clumps

Figure 84: Fiona's Clumps

Complement Lights

Another shader developed for *Shrek* was the Complement Light Shader. This shader, simple in concept, gave lighters the ability to infuse a considerable amount of rich color variation in their scenes without adding a large number of lights. Complement lights allow the lighters to pick a gradient of colors which the shader applies variably according to the intensity of the light. Compare the images in Figure 85.

Figure 85: Tree Trunk with Normal Light and Tree Trunk with Complement Lighting

Notice the rich purple falloff as the keylight fades off to shadow in the image on the right. In the campfire sequence, complement lights fell off from a yellow, through orange, and finally into a rich purple (Figure 86). This falloff through a color gradient occurs not only with distance from the light, but also according to normal angle to the light and to shadowing intensity.

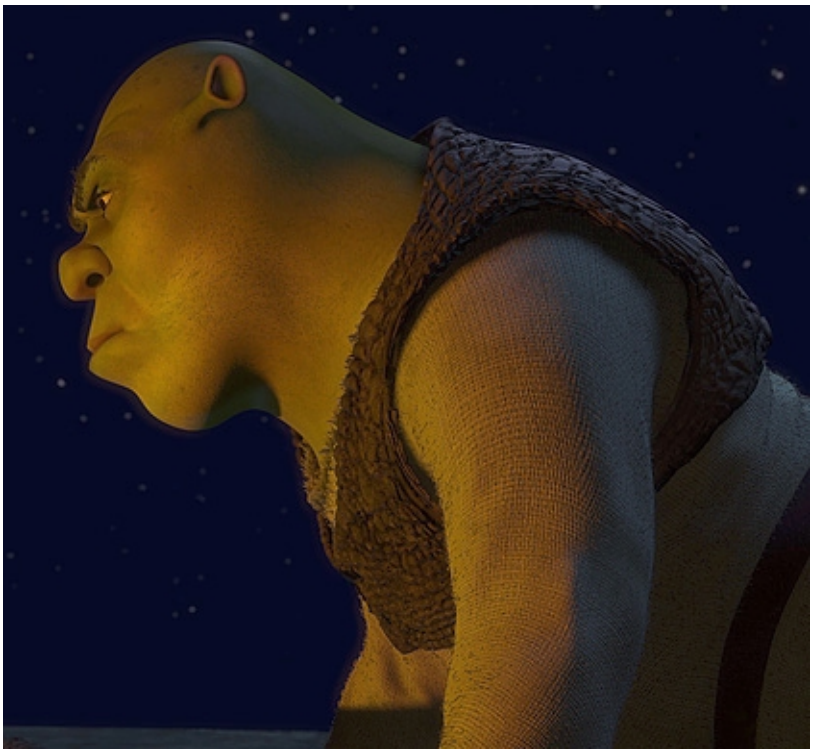

Figure 86: Shrek with Complement Lights for the Campfire Sequence

These “painterly” lights, although useful in a variety of situations, were used sparingly. They were used mostly on environments rather than characters, since the color variation with shading for surfaces in motion had the potential for being a little too psychedelic. Nevertheless, this type of artistic control is at the heart of our approach on *Shrek*: keep the materials real and let the lighting be stylized.

Pipeline

In the process of lighting *Shrek* (as well as *Antz* before it), we learned quite a few things that made our lives easier as we fine-tuned or implemented the various pipeline components.

Library File Structure

PDI uses a flexible library structure to organize much of its feature animation files. Thus, character and environment models, deformation systems, materials and maps, fx control systems, and many other aspects of the pipeline were available to all shots at a library level. For lighters, this meant that they could rely on consistency of maps and materials from shot to shot and sequence to sequence. However, situations inevitably arose where slightly altered library data was needed at the shot or sequence level. The file structure makes this form of “localization” quite simple, and it wasn’t uncommon to have a different version of a character or a map tuned for the needs of specific sequences. Note that only the desired parts were localized; the rest of a character setup could still exist in the library.

For example, a common difficulty arose when taking a character from a daylight to a night lighting situation. Value relationship in clothing and skin that worked in the daylight would no longer work under moon light. Shrek’s tunic is a light burlap, and his vest is dark leather. This worked in day shots, but under cool dim night lighting, the contrast was too great and his vest dropped to black. Since we wanted to simulate the visual acclimation that occurs at night, we needed to reduce contrast between his tunic and vest in a “night Shrek” version of the character. Any lighters with night shots could easily call this version, rather than the day Shrek. Again, the only thing different between the two versions was the material settings for the tunic and vest; all the rest of the Shrek material settings still existed in a single setup at the library level.

Keep ‘em Honest

A potential pitfall for the Lighting Department, since it lives at the “end” of the feature pipeline, is that it inherits all the sins of the departments that come before it. Layout may decide that a tree is in the way for a certain shot and cheat it over, Character Animation may not know where the keylight will be located to make the characters “act to the shadow”, and so forth. These kinds of decisions may cause lighters to have to alter their setups just to get back to an approved look that “broke” because of the upstream change. This is time spent fixing, and not lighting. Therefore, the Layout Department has a

general policy to “keep it real” from shot to shot with respect to the physical layout of props and environmental elements. Whenever they need to cheat a camera or prop position significantly with respect to the other shots, the Sequence Supervisor works with Layout to find less invasive solutions.

In the Future

An important aspect of the pipeline is to involve lighting decisions in as many places in the pipeline as possible; Layout and Character Animation Departments particularly benefit from having keylight and key shadow positions determined early. It is critical for Character Animators to “act to the light,” or even to know if the character is in light or shadow. Likewise, shadows can be a very important aspect of the composition of a scene, so layout must be able to frame shots for these shadows rather than wait for the shot to get into the Lighting Department. The goal is to minimize time spent returning shots to Character Animation and Layout to fix issues related to the light positions. The more steps in place that let lighting decisions happen early on in the pipeline, the stronger the film will be from a cinematography standpoint, since all other departments will have that lighting information for reference.

An important point to remember is that 3D feature animation is much more like live-action than 2D cel animation. Animators have the ability to set up scenes, characters, lights, and cameras, and then go in and “explore.” The ability to react to and adjust most aspects of a scene make the experience more like that of a traditional Director of Photography who has direct access to sets, actors, lights, and cameras on-site. 2D animation requires animators to design and draw every view and character angle, which leaves very little room for exploring. However, 3D animation does differ from live-action in that animators don’t have access to all the props, characters, lighting, and cameras at the same time. This is a consequence of the complexity and nearly linear nature of the CG feature pipeline (Models > Layout > Motion > Lighting). This hybrid nature of the 3D filmmaking process creates the ongoing challenge for us to make the best films we possible can from a traditional cinematography standpoint, yet also take advantage of the unique aspects of the medium.

CG Infrastructure

George Bruder, Supervising Technical Director

The technical infrastructure necessary to produce a computer-generated film such as *Shrek* includes an integrated high-speed network of workstations, high availability fileservers, ample processing power, and terabytes of storage. When combined with groundbreaking graphics software, fine-tuned core pipeline technology, and of course talent and passion, the possibilities are unlimited.

Purpose of the Feature Pipeline Infrastructure

Our approach to creating a feature animation pipeline requires consideration that the project is a giant software project used to build an assembly line within a movie studio. Hardware and software resources are designed, tested, implemented, and maintained to give the creative resources maximum flexibility and control of the creative process. Feature Pipeline Infrastructure is the organization of the technical process of the feature film. Our animated feature films require organization on a large scale to accommodate over 150 production artists and software developers working on over 1200 shots over a two-year production schedule.

The infrastructure is capable of handling any type of character setup, such as realistic human facial systems, and effects, such as fire, wind, water, and dust. Crowd systems are also easily integrated. These setups are integrated via generic “entry points” within the PDI animation language. This allows production developers to integrate their effects and character setups in a modular fashion, whether they be written in the PDI animation language or developed in third-party packages.

Components of the Feature Infrastructure

Hardware

The local area network infrastructure is comprised of redundant wire-speed core routers with Gigabit Ethernet interswitch links to edge switches. UNIX desktop and renderfarm machines are connected via switched 100Mb/s Ethernet. Performance-critical servers are attached via Gigabit Ethernet directly to the network core. The modular architecture of the network allows easy expansion and considerable flexibility.

NFS file sharing service is provided by UNIX fileservers and clustered network-attached storage. Production data and other critical data is stored on RAID-protected volumes. Other network services are provided by UNIX and Linux servers. Automated backup and archival services are provided by a DLT tape library, backup server, and management system.

There are six Network Appliances, one SGI Origin 2000, 1100 processors in the render cluster, 200 workstations networked to over 6TB of disk space.

Production filesystems are accessed via the UNIX automounter with all automount maps served via NIS. All job trees conform to standard naming conventions which are tightly integrated with the job navigation environment and revision control system. The flexible, straight-forward nature of the system allows production filesystems to be added, destroyed, or relocated with minimal impact to the job.

Software

The core animation tools, including the animation scripting language, interactive animation system, renderer, interactive lighting tool, custom tools, and commercial packages, are all "glued" together using Perl scripts. A release schedule is closely coordinated with production to provide timely features, enhancements, and bug fixes. The R&D group provides development and support of the core animation tools. The Production Engineering group is responsible for the Perl pipeline or "glue" tools. Production also provides project-specific support with departmental technical directors. These three groups work very closely to keep the pipeline running smoothly.

Job Navigation and Structure

Feature animation, including *Shrek*, resides on multiple file servers. Multiple copies of the directory structure or "job trees" are created for each department to work within. These trees can be "pruned" to the requirements of each specific department. Other types of trees include picture trees for storing rendered images, a data tree for storing large recreatable datasets, and a model tree for storing pre-rendered animated geometry. These trees are shared among departments.

A feature animation at PDI/DreamWorks typically involves several job trees that house various kinds of information, from job data to rendered pictures. However, because there are so many job trees associated with a production and each tree can be extremely large, a simple way to move among and within the job trees is required.

The job environment is a collection of tcsh shell aliases that provide the user with an easy method of navigating between and within various job trees.

Once a job environment is activated, the user can then use the various aliases to navigate around the job trees. One fairly common alias is ‘cds’. This alias can take the user to a sequence/shot directory in a shadow tree. Once the user has navigated to the sequence/shot directory, they can then use other aliases, such as ‘cdp’ or ‘cdlogs’, to location rendered images or log files.

Resource Management

Render farm and workstation processors and software licenses are managed through a batch queuing system. This allows departments within a project to hierarchically share resources. These resources can be shared across projects as well as within the project. An API between render submission applications and a commercial queuing system allows flexibility to write a custom queuing system in the future. A Web-based resource utilization toolset provides feedback for optimizing resource allocation.

Asset Management

The production of a CG movie can result in the creation of millions of digital assets. These assets can include models, painted textures, motion curves, scripts, and other ancillary data. The management of these assets brings various challenges. Changes made to the assets need to be tracked, and that tracking data needs to remain with the asset. The altering of an asset should be limited to one person at a time to avoid conflicts. The changed assets need to be efficiently moved around the network to allow their use in simulations and rendering. The performance of these tasks, as well as others that deal with managing collections of assets, are handled by a Digital Asset Management System. The scope and abilities of such a system can have a positive impact on the efficiency of a production, and is, therefore, an integral part of the production.

Feature Animation Project Cycles

The feature animation infrastructure is a process of ongoing improvement. Analysis of user and developer feedback are invaluable to improving the technical and creative process of future projects. The opportunity to evaluate and overhaul systems and processes within the pipeline occurs when departments are finished with their production tasks. Enhancements and feature requests or outright overhauls of subsystems occur during a department's downtime. This provides an opportunity for the department to be involved in the work by providing valuable testing and feedback which makes the process of 3D animated movie making more creative and cost-effective.

Biographies

Andrew Adamson
Co-Director, *Shrek*
PDI/DreamWorks

With PDI/DreamWorks since 1991, Andrew Adamson is a co-director on *Shrek*, the second animated feature film to come out of PDI/DreamWorks. Prior to his work on *Shrek*, Adamson worked on visual effects for feature films including *True Lies*, *Hearts & Souls* and *Toys*. His credits as Visual Effects Supervisor for PDI/DreamWorks include *Angles in the Outfield* and *Double Dragon*.

Adamson also served as a key member of PDI/DreamWorks' Commercial Division, taking the lead on numerous award-winning spots, including Converse's "Planet Kevin"; Dow Scrubbing Bubbles "Greatest Show"; and Miller Genuine Draft's "Juke Box".

As an independent visual effects supervisor, Adamson oversaw work on *Batman Forever*, *A Time to Kill*, and *Batman & Robin*. His visual effects contributions have been short listed twice by the Academy of Motion Pictures Arts & Sciences, for *Batman Forever* and *Batman & Robin*.

Andrew Adamson began his career in computer graphics nearly 15 years ago in Auckland, New Zealand as a computer animator at The Mouse That Roared. In 1986 he moved on to Design Director/Senior Animator at Video Images Ltd. Where he worked on a variety of broadcast logos and television commercials.

Ken Bielenberg
Visual Effects Supervisor, *Shrek*
PDI/DreamWorks

Ken Bielenberg just completed work as the Visual Effects Supervisor on *Shrek*, the second computer-animated feature film co-produced by DreamWorks and PDI and released in May 2001. Prior to this role, Ken served as the Effects Supervisor on the *Antz* animation team responsible for creating the extensive and innovative water simulation, dust and dirt effects of the film.

Bielenberg joined PDI in 1990 and has been involved in a leadership capacity on television and film FX projects such as The Simpsons "Treehouse of Horror VI" Homer 3D episode, *Angels in the Outfield*, *Eraser*, and for numerous commercials for clients including Halls (the Cleo-award winning "Penguins" spot), Matchlight, Listerine, and Legos. He graduated from the Rochester Institute of Technology with a degree in Computer Science and Animation.

At previous SIGGRAPH conferences, Bielenberg delivered an Animator Sketch on *The Simpsons*, participated in the "First Look At *Shrek*" sketch, and moderated the Panel "Function and Form of Visual Effects in Animated Films".

James Hegedus
Art Director
PDI/DreamWorks

James Hegedus, a member of the Art Directors Guild, was the Production Designer on *Shrek*. Prior to that, he was the Art Director for *Puppet Masters*, *Jumanji*, *Batman Forever*, *Batman and Robin*, and *Mars Attacks*. In addition, he was Art Director for Apogee Productions, a company founded by John Dykstra which specialized in creating and producing visual effects for motion pictures, commercials, and entertainment ventures.

Hegedus served as Production Illustrator for the films *The Milagro Beanfield War*, *Rainman*, *Always*, *Defending Your Life*, *Forest Gump*, and *The Rocketeer*, and on the recreated television series "The Twilight Zone" based on Rod Serling's original series. He was also a background and layout artist with Hannah Barbera Animation Studio and self-employed as an artist/illustrator.

In 1991, Hegedus was associated with the UCLA Film Extension program as guest speaker and subsequently as an instructor.

Born in the Texas Panhandle town of Amarillo, Hegedus is married to BJ Nathan and has two sons and a daughter. His Bachelor of Fine Arts degree is from California College of Arts and Crafts in Oakland, California.

Lucia Modesto
Character TD Co-Supervisor on *Shrek*
PDI/DreamWorks

Lucia Modesto was a Character TD Co-Supervisor on S*hrek* and retains the role for PDI's third computer-generated movie, *Tusker.* She worked on the motion and deformation systems for faces and bodies on both *Antz* and *Shrek* and is currently doing the same on *Tusker.* She combines an M.S. in Quantum Electronics and Applied Optics with 20 years of experience in computer graphics production for broadcast, commercials, and film.

Jonathan Gibbs
Senior Effects Animator
PDI/DreamWorks

Jonathan Gibbs served as a Lead Effects Animator on the PDI/DreamWorks feature, *Shrek*. For *Shrek*, he led shader development, worked on fur and hair rendering, and continues to develop the crowd system which he co-developed for PDI/DreamWorks' first feature, *Antz*. A native of St. Louis, Jonathan has a B.S. in Computer Science from Principia College. He holds a Masters degree in Computer Science from UC-Santa Cruz.

Vanitha Rangaraju
Lighting Technical Director
PDI/DreamWorks

Vanitha Rangaraju was a Lighting TD on *Shrek*, working to solve various rendering and shot optimization issues. On *Shrek*, she led the pipeline development for generic characters and crowds in Lighting and Surfacing Departments. She is currently on the visual development team for PDI/DreamWorks' next CG feature, *Tusker.* Vanitha holds a Bachelor's degree in Architecture from India and is currently pursuing her Master's in Computation and Simulation at the University of Texas, Austin.

Rob Vogt
Senior Character Technical Director
PDI/DreamWorks

Rob Vogt joined PDI in 1997, and as Character TD has focused on character-related simulation and deformations for *Antz* and *Shrek.* He is currently working on elephant trunk motion system design for PDI's third animated feature *Tusker.* Prior to his work at PDI, Rob was animating and conducting simulations for accident reconstruction at Failure Analysis Associates. Prior to that he worked on electronic warfare simulations at the Naval Research Laboratory. Rob received an Aerospace Engineering degree from the University of Maryland in 1990.

Scott Singer
Senior Effects Animator
PDI/DreamWorks

Scott Singer is a Lead Effects animator at PDI, and has worked on both *Antz* and *Shrek*. He has created methods for animating and rendering the simulation of natural phenomena such as dust, debris, cloth, water and fire and lead efforts to integrate simulation techniques into PDI's feature animation pipeline. Before working at PDI he worked at the Center for Visual Computing at the University of California, Riverside. He holds a Masters of Fine Arts in Painting and Critical Theory from Northwestern University, and a Bachelor's of Fine Arts in Painting and Art History from the University of Colorado.

Arnauld Lamorlette
Senior Effects Animator
PDI/DreamWorks

Arnauld Lamorlette was a Lead Effects Animator on *Shrek* concentrating on fire effects, computer-generated human characters, and organic environment development. Before working at PDI, he was vice president and director of research and development at Buf Compagnie, a Paris-based studio where he was involved in the production of all the company's film projects including *The City of Lost Children* and *Batman and Robin* as well as many commercials and music videos.

Scott Peterson
Senior Effects Animator
PDI/DreamWorks

Scott Peterson was a Lead Effects Animator on *Shrek* concentrating on foliage development. For *Shrek*, he created a system for modeling and animating trees, developing several foliage distribution systems. He keeps his hands dirty by simulating and rendering mud. A native of California, Peterson has a B.S. in Computer Science and a minor in Fine Art from California Polytechnic, San Luis Obispo.

Juan Buhler
Senior Effects Animator
PDI/DreamWorks

Juan Buhler is a Lead Effects Animator at PDI and currently working on visual development for PDI's third computer-generated movie, *Tusker.* Buhler joined PDI in 1996 to work on Antz. His main contribution was the design of a system for crowd simulation and rendering. For *Shrek*, he developed techniques for simulating and rendering mud and beer using Nick Foster's fluid simulation tools. Buhler has also been a student volunteer in Siggraph 93 and 94, a Computer Animation Festival contributor in Siggraph 95, and he organized a panel on crowd simulation for feature films for Siggraph 99.

Mark Wendell
Lighting Supervisor
PDI/DreamWorks

Mark Wendell joined PDI at the beginning of production on *Shrek*. He worked as a Senior Effects/Lighting Animator before stepping into the role of Sequence Supervisor. Prior to PDI, Mark worked at Santa Barbara Studios as lead Effects TD and CG Supervisor on such projects as *Star Trek Insurrection*, *Star Trek Generations*, *Paulie*, *An American Werewolf in Paris*, *500 Nations*, *Parasite Eve*, and *Cosmic Voyage*. He combines an academic background in the biological sciences and scientific visualization with over ten years of experience in visual effects production for film, broadcast, games, and special-format theater. He has taught FX and particle dynamics classes for Silicon Studio, lectured at numerous venues including SIGGRAPH, and in his spare time, builds furniture and wood sculpture.

George Bruder
Supervising Technical Director
PDI/DreamWorks

George Bruder has been at PDI for over 10 years is currently the Supervising Technical Director for Feature Animation in the Production Engineering Group at PDI/ Dreamworks. He has worked on music videos with David Byrne and Michael Jackson, and on commercials, including those featuring the Pillsbury Doughboy. His feature film contributions include *Toys*, *Eraser*, and *Angels in the Outfield*.