Art and Technology at Pixar

SIGGRAPH ASIA 2018 Course Notes

Course Organizer

Ryusuke Villemin
Pixar Animation Studios

Presenters

Chia-Chi Hu
Pixar Animation Studios

Sonoko Konishi
Pixar Animation Studios

Hiroaki Narita
Pixar Animation Studios

Magnus Wrenninge
Pixar Animation Studios

David G Yu
Pixar Animation Studios

Course Description

As described by this now famous quote "The art challenges technology, and the technology inspires art", technology has always played an important part in Pixar's movie making process. This course will show how we develop and utilize new technical advancements to tell more appealing stories, using real-world examples from our latest movie *Incredibles 2*. This is a direct refresher of the previous course from Siggraph Asia 2015. Since that time, Pixar's pipeline has been heavily restructured, switching its main rendering algorithm to pathtracing. We will describe how that technology has now matured, and how we were able to reintroduce complete artistic control in this new physically based world. At the same time, the pipeline kept evolving and we introduced a new stage that didn't exist before, in the form of deep compositing. Finally we'll focus on USD, OpenSubdiv and the Hydra render engine, to showcase how the whole pipeline is moving toward real-time feedback, not only for rendering, but also for many other departments such as animation, simulation and crowds.

Level of Difficulty: Intermediate

Intended Audience

Practitioners from CG animation, VFX and videogames, plus researchers interested in the current trends in the industry.

Prerequisites

A basic understanding of computer graphics, modeling, simulation, lighting and compositing.

Course Website

`http://graphics.pixar.com/Library`

Contact

`rvillemin@pixar.com`

About the Presenters

CHIA-CHI HU studied cinematography at Université de Paris - Saint Denis, and worked as as commercials director before moving to the visual effects industry. Before joining Pixar, she worked as a Senior and Lead Compositor at Dubois and Macguff in Paris, Moving Picture Company in London, Digital Domain in Los Angeles and Industrial Light and Magic in San Francisco. Some of her film credits include *The Last Jedi*, *Rogue One*, *Jurassic World* and *Tron Legacy*. She is currently Compositing Technical Director in Pixar's lighting department.

SONOKO KONISHI joined Pixar in 1994 as a Jr. TD after graduating from a fine arts college in the States. She worked on *Toy Story*, *Toy Story 2* and *Toy Story 3* in various departments such as lighting, modeling and character rigging. Over the past 20 years Sonoko has predominantly focused on feature film production but she has also worked on the production of shorts, TV specials and promotional materials. She currently works for the simulation & digital tailoring department.

HIROAKI NARITA is currently an Effects Technical Director at Pixar Animation Studios. He worked as a CG artist for an architecture design company and a medical software company before moving into the film industry. He has been working on both animation and live-action films such as *Oz: The Great and Powerful*, *Edge of Tomorrow*, *The Amazing Spiderman 2*, *Big Hero 6*, *Zootopia*, *Moana* and *Incredibles 2*. He got nominated for Animated Effects in an Animated Production in Annie Awards 2017.

RYUSUKE VILLEMIN began his career at BUF Compagnie in 2001, where he co-developed BUF's in-house raytracing renderer. He later moved to Japan at Square-Enix as a rendering lead to develop a full package of physically based shaders and lights for mental ray. After working freelance for Japanese studios like OLM Digital and Polygon Pictures, he joined Pixar in 2011 as a TD. He currently works in the Research Rendering department.

MAGNUS WRENNINGE is the author of the book Production Volume Rendering and the SciTech awarded open source project Field3D. He started his career writing fluid simulation and environment rendering software at Digital Domain, and in 2005 he joined Sony Imageworks where he worked both as a Senior Software Developer and Lead Technical Director on films such as *Alice in Wonderland* and *Oz: The Great and Powerful*. Since 2013 he has worked as a Principal Engineer at Pixar Animation Studios, where he focuses on all things related to rendering.

DAVID G YU has been at Pixar since 2000 working on the Presto animation system (and its predecessor Marionette) with a focus on advancing the use of GPU technology. Prior to that David was at Silicon Graphics Inc (SGI) developing workstation hardware and software for OpenGL and visual simulation.

Presentation Schedule

08:30-08:35 **Introduction** *(Villemin)*

08:35-09:40 **Simulation & FX** *(Konishi, Narita)*

09:40-09:50 **Break**

09:50-10:55 **Rendering & Compositing** *(Villemin, Wrenninge, Hu)*

10:55-11:05 **Break**

11:05-12:00 **Real-time Tools** *(Yu)*

12:00-12:15 **Q&A**

The Incredibles 2 Simulation

by Sonoko Konishi

In 2004, Brad Bird joined Pixar to develop the *Incredibles*. Besides a great story, he brought a passion for 2D animation and stylized character design to our studio. Applying this passion to our films brought us to a new level of visual design; one with an increased focus on the nuances of character silhouette and cartoon physics. As a member of the first *Incredibles* character team it was exciting to reunite with the esthetically and technically advanced Parr Family.

In the 14 years since the first *Incredibles*, we have continued to advance our cloth and hair technology resulting in a more visually sophisticated look. In turn, this has made us more willing to pursue technical innovations; the end result is that that the *Incredibles 2* is our most expansive film to date.

1 Department Structure

The simulation department is responsible for shot simulation of hair & cloth for all characters, flesh & skin simulations on selected characters, vegetation and motion dressing, props that interacts with characters such as beverages, pillows, telephone cords and rigid body simulations. The crowds group is also part of our team and is responsible for the tech to generate vehicles and humans with a mixture of procedural setups, clip choreography and motion capture while collaborating with crowds animators. This structure allows us to interchange tasks fluidly and was key to pushing trough the large quantity of shots with layers of character interactions.

At the peak of production, our simulation department was 20 TDs and approximately 40% of the staff members were new to the studio. The broad range of technical backgrounds and experience with in-house & third party tools allowed us to forge visually complex shots in a compressed schedule. In the end, 2200 shots included simulation, which is 25% more than any of our other films, 400 crowds shots, about 20% of film, 300 vegetation shots and 60 prop sims that characters interact with. Some of the shot simulation TDs also were responsible for CG tailoring and grooming during asset creation. Having artists in the shot production group with knowledge of the assets and its simulation setup helped streamline production and ensure that the intended look and feel of garments and hairs was maintained throughout shot production.

2 Approaching the pipeline

But how did we organize the sheer quantity of shots in limited directors review time? We had two simulation supervisors that split the tactical and strategic management. Artists met at simulation dailies to show in progress shots and everybody was welcome to make feedback and share tips for improvements peer-to-peer till they are ready for cross-department review with leads from animation, character and art departments. To put it simply, we were communicating and collaborating with all departments at the peer level as well as on the supervisory level; this was key to keeping everybody on the same page. Once the shots are approved by simulation supervisors we bring them to Phase1 review.

Traditionally, the production pipeline at Pixar was very linear, but as our films become more complex and our timelines tighter we adjusted our pipeline to a more layered and parallelpipeline. This is great for the optimization of work but can make it very difficult for the creatives to get a strong sense of how the pieces are coming together before they get too expensive to change. To address this we broke our review process into three phases. Phase1is an early merge of final animation, sim,

crowds, lighting blocking and FX blocking together in an early but high quality render for preliminary approval. The primary goal of this phase was for Brad to approve final animation, simulation and crowds before the complexity of final lighting and FX kicked in. But it also allowed him to give key lightingand FX notes while it was still easy to change them.

Following this process early and frequently also allowed Brad to get more comfortable with the process and the skills of the teams which made it easier to show work at looser, thumbnail type stages for quick direction checks.

In Phase2 we focused on refinement and polish. Because we had the phase one reviews and he could see the animation and simulation was going to work with the final work he didn't give as many notes that would have required us to back to the head of the pipelineand we could focus on getting to The Final Phase.

Our last stage is Digital Dailies. In Digital Dailies we are focused on the final quality control check of all elements. At this stage, any performances changes would be extremely expensive since it would require that all departments revisit and potentially redo their work. By being very clear with our director and producers up front about where changes would become costly and carefully and regularly reviewing in context during phase 1 & 2 we had a record low number of performance fixes called out during digital dailies.

3 Cloth & Hair Tools that helped shot simulation

In the early stages of production we focused on setting up cloth and hair rigs at the asset level so that shot production would be very streamlined. In the *Incredibles 2* we had a much larger number of characters wearing multiple layered garments with a wide range of hairstyles interacting with themselves and their environments in extreme conditions and we needed to minimize the amount or per shot setup required. In the first film we had 150 unique garments for all the characters but in the *Incredibles 2* we had over 130 unique garments for the background characters alone. In 2004, simulation on super suits was simulated through a comprehensive training set to capture wrinkles and baked garments and today their super suits still have landmark details such as iconic logo on the chest, belts and briefs, geometric printed patterns and strong tailored outlines. This time our tailoring technology provided a natural way to form wrinkles on stylized character by a technique called Dynamic Alterations (Kutt et al. (2018)). The tool set dynamically adjusts by region, the fit of the pattern where character deforms the most. As an optimization, the dynamic alteration was driven by a lower density cage mesh that deforms the higher density body mesh. Having a cloth rig installed as a component of the asset allowed us to achieve a consistent style throughout the film. The cloth rig exposed many simulation parameters which would then allow us to create per shot adjustments so that we could easily tweak the behavior of the cloth on a shot by shot level. To keep the form of core design elements , we used patch-based surface relaxation (de Goes et al. (2018)) which, especially visible on chest logos, removes undesirable wrinkles without distorting texture and removes unappealing folds resulting in clean lines.

In the *Incredibles*, amplified and stylized animation was the rule and we were able to emphasis this in our simulations by keying the primary animation and using simulation to accentuate the secondary motion. In this way, simulation became an extension of the visual language of the characters and the story telling. Understanding cartoon physics between both TDs and animators was crucial to making this work so we kept close communication and planning about how to approach the shots coming into production. Cloth and hair controls were exposed to animators and TDs to author simulation setup on top of keyframed animation.

For instance, Helens parachute transformation has an animated parabola shape including the waving edges timed to the action and silhouette. Her belt and logo stretches but dynamic alterations allow us to specify stretch and compression by region. Keeping the logo from stretching by only stretching the surrounding region and restricting the belt to scale width but not height are examples of how we

supported dynamic motion while staying true to design. The simulation on hair emphasizes the force of the action and simulation with dynamic alteration region preserve the shape and design of detail elements such as the the belt, logo and textures during extreme actions (Figure1)(Singleton et al. (2018)).

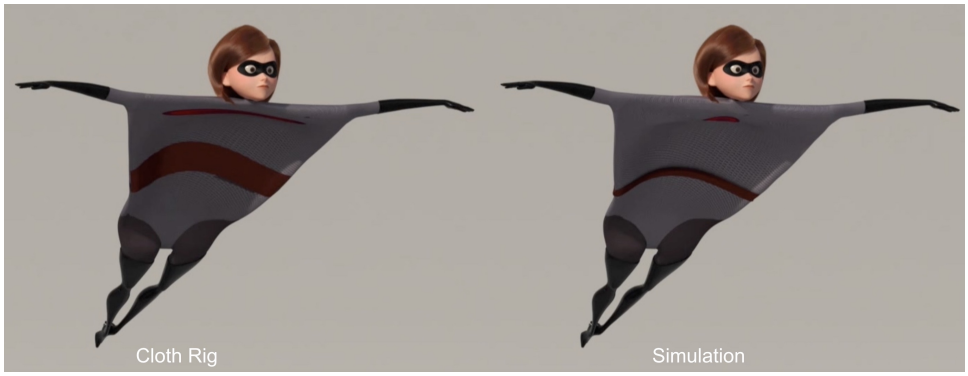


Figure 1: Example of Dynamic Alterations on supersuit

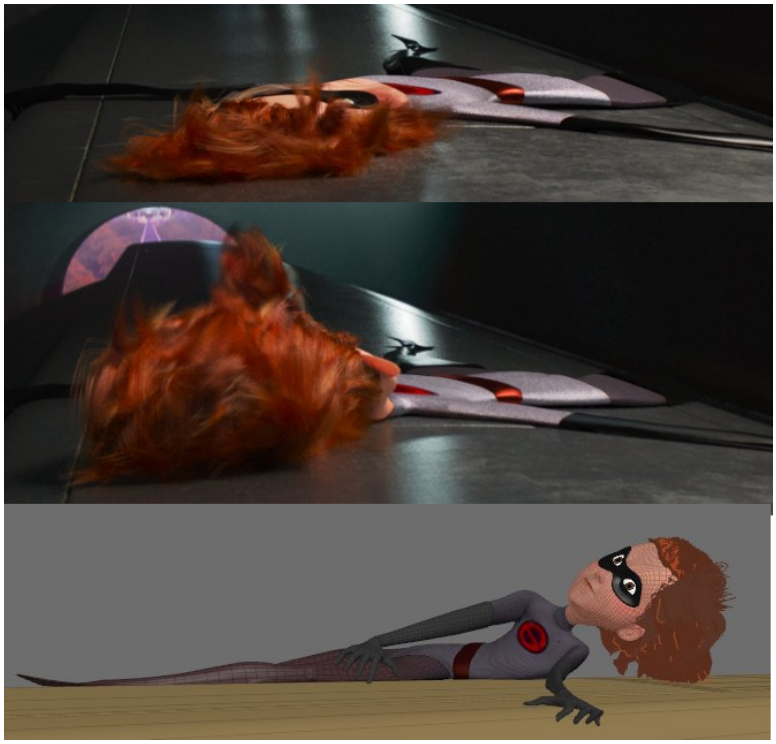


Figure 2: Example of multiple hair pose blending

Another example of extreme body deformation is Helen flattening on the top of the train and her transition back to normal shape (Figure2). Hair simulation had specific shape notes from the director when her scalp was flat to the roof and separate hair simulation was blended as her head returns to normal volume. In previous shows, we would handle action sequence like whipping hair by sculpting key hairs and rendering offline to see the precise results, This time tools provided us full-fidelity hair representation of hair (Butts et al. (2018)) that we can manipulate in real time resulting in an increase of artist productivity. Hair parameters are set by the groomers for different hairstyles, environment and actions and those parameters are exposed to TDs for further tweaking. With new technology we

were able to use hair operators such as length, scraggle, clump and curl in real time and some shots took advantage of changing the look of the hair just to heighten the character reaction (Figure3).

Figure 3: Screen shot from full-fidelity hair manipulation

Our simulation philosophy is to enhance storytelling and we have total control how we reorganize the world in Presto. Many characters were riding on vehicles and to support the cinematic effect they are moving faster than real world speed which would have resulted in unstable simulations. Instead, we simulated the characters stationary or only transferred a percentage of the actual movement to the simulator. In fact most of vehicles are moving so fast that character simulations would have been impossible. For example, with Helen on the motorcycle we often removed all translation from her animation prior to simulation and instead keyed wind speeds to match the perceived motion. This resulted in a controllable but effective impression of hair moving at high speeds. The hydro liner peak speed is 450 mph and simply activating cloth and hair simulation on characters under this force is unstable. Also the action spans a large area from inside of the car to the ocean so we had to be careful crafting the world space vs. local space relationships to prevent unexpected precision errors during simulation. Usually those artifacts appear as a form of cloth and hair jittering when characters are still.

Because our director is an animator, we get many motion notes based on animation assisted simulation (Figure4). For example, in a fit of teenage angst Violet throws her super suit into the food disposer and onto the wall. This comical sequence was achieved by animation providing key shapes on the costume that emphasis her emotional state. When the gloves swirl in the food disposer or the dangling cloth peels back from from the wall the key poses carry the classic cartoon style of exaggerated poses and timing, Simulation was then used to enhance this with with dynamic secondary motion. The result was animation that simultaneously is very stylized and controlled but also feels physical and natural. But not all simulation is complex; sometimes it only appears to be. A good example of this is when Violet stretched her super suit arms in full force (Figure5). The cartoony timing and shapes were achieved by threading her arm meshes into torus collision objects to compress the mesh as her hands travels over the fabric. The resulting simulation looks like complex dynamic reactions to the motion of her hands implying force applied to the super suit but the reality behind the scenes is a very simple collision operation. (Cameron et al. (2007))

To support this type of stylized simulation our system allows us to blend fully animated poses with hand keyed poses and to dynamically shift between hand keyed animation driving the positions of objects and simulation. When Violet and Dash land on the high speed hydroliner the precise path and

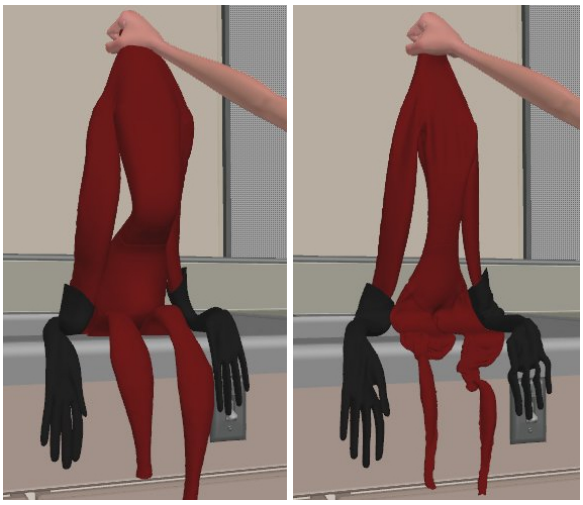


Figure 4: Animation keyframed pose (left) and simulation applied (right)

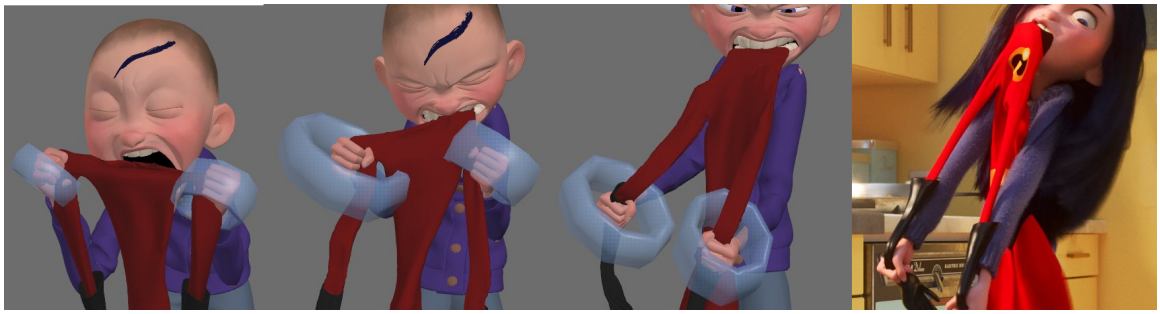


Figure 5: Use of simple primitive collision object for effective comic action

timing of the backpack was important to the impact of the shot. At the head of the shot, animation hand keyed the path and timing of the backpack and simulation provided secondary motion, then as Dash grabbed the strap and swings the backpack onto his back we blend the primary driving force from keyframed animation to simulation driving the primary motion of the backpack as well as providing the secondary motion. Even with a robust system such as ours it can be complex to manage the simulation of calculated and animated poses but the results are a stylized reality that we feel gives a strong sense of design to our film.

Volumetric simulation: Today, you can see volumetric shifts and movement in everything from the action of high impact punches (De Goes and James (2018)) through ordinary movement as Bob jostles down the stairs (Irving et al. (2008), Kautzman et al. (2016)). We did this to increase the sense of naturalism but also because combining natural physical volumetric movement with exaggerated character design enhances their sense of super reality. When a giant Jack-Jack falls off from the duct and uncontrollably cries with his entire body. You feel his confusion and fear in his action and the volumetric simulation accentuates the feeling of angst. It is this mixture of keyframed comical timing on his body emphasized by volumetric simulation that enhances the moment (Figure6) (Wong et al. (2018)). Also, because the volumetric sim insures that his cheek to shoulder interaction and chubby arms to chest collide with correct volume distribution there is enough space between reacting skin surfaces to prevent unstable cloth simulation caused by snagging.

Robust Skin Simulation (Kautzman et al. (2018)): We use skin simulation on our characters to loosen up the movement of the bodies also making them feel more physically believable in action. However, you may not realize that we have been running skin simulation on our fur covered characters.

Figure 6: Volumetric simulation tetrahedral mesh and final pose ©Disney/Pixar 2018

Characters such as a horse, a bear or even a raccoon fighting with Jack-Jack all benefit from running skin simulation and is an important part of the cartoon physics. The simulation relaxes the skin making the motion feel coherent across the whole surface. We further tweak these settings based upon the character distance from camera. Loosing up the skin sim parameter when the character is further away allows the subtle motion to read at a distance and tightening it when the character is closer to camera keeps the simulation from moving the hair too dramatically which would distract our attention from the performance.

In the first *Incredibles* we did not simulate vegetation instead we applied noise fields to create the appearance of simulated plants and for large interactions between characters and vegetation we animated it by hand. On *The Good Dinosaur* (Soares et al. (2016)) we built a system to allow us to dress and simulate motion at a large scale by making a large series of cycled clips correspond to the wind levels. On the *Incredibles 2* we extended that work even further to allow the easy combination of simulation and noise. While the vegetation was generally calm, having the ability to simulate large regions of vegetation allowed us to create reactions to large forces such as shock waves and collateral damage. This really amped up the impact of these scenes.

Even apparently simple prop simulation can present challenges. For example, in one shot we had over 150 people holding wine glasses while moving on a boat. By removing all world space translation, and applying the local translation and rotation of the each glass to the fluids gravity field we were able to make the simulation process an order of magnitude faster.

Simulation on this film faced many new challenges but this is the studios management style. We focus on relying on the quality of our leadership and their ability to adjust and streamline the approval process to face these challenges even under a compressed schedule. Over years the studio has cultivate artistic eyes, cross department collaboration and transparency including the communication with our tools department to invite innovations while also increasing our productivity. Simulation in the first incredibles was in the infancy stage and on the *Incredibles 2* simulation department has reached the maturity.

Designing Effects

by Hiroaki Narita

Figure 1: *Incredibles 2* ©Disney/Pixar 2018

1 Seeking for Storytelling

The effects department at Pixar is getting more involved in the early stage of production. For *Incredibles 2*, we collaborated early with departments such as storyboard, layout and animation to boost our storytelling. To be more precise about the collaboration, we worked with layout and animation in the early stage to provide rough effects based on storyboards and artwork. Look development is also active collaboration among the effects, lighting, and the art department.

All artists in Pixar are highly encouraged to see the screenings of current story development. The screenings are usually edited with storyboards and rough layout. The production gathers feedback from everyone and refine the story for better storytelling. It also helps artist to understand the whole picture of the film and gives opportunities to explore how to support the story with their creativity.

The effects department usually holds an internal sequence review with the whole team before beginning a new sequence that involves heavy effects work. The artists on the FX team are encouraged to review the unassigned FX work on the film by looking at the current reels and asking for the shots they would like to be cast on.The leadership then can try and cast them on this work. This is a great part of Pixars FX culture that creates great moral and allows artists to have a say in the work they get to do.

2 General Effects Workflow

2.1 Generating effects data

The effects team at Pixar uses SideFXs Houdini as the main program to create effects work in the past several films. We basically use vanilla Houdini same as everyone else use outside the studio. There are some custom Houdini digital assets but they are mainly for input and output data in our pipeline. It helps artists to move from one project to the other smoothly. It is like having same canvas and brushes when you start painting. Since artists are also technical directors, we come up with new tools or new methods to improve our workflow but they are designed to be compatible with Houdinis fundamental workflow. Most of the shot work starts from importing USD assets into a Houdini session. The USD contains assets such as animated characters, sets, props and camera. We create effects based on the assets in Houdini and write out fx data as an effect USD file which is referenced into the main USD file for the shot. The output data types from effects are point clouds, mesh, F3D volume data or levelset data. The point clouds are used for lighting and shading signals in the render and the levelset data is used for ocean effects by our proprietary water tools we call Gin. We also overlay assets in the main USD file. For example we would use an overlay for fracturing building assets or street assets from the sets department.

2.2 Sharing knowledge in the team

Effects artists at Pixar are very collaborative with each other. We can see everyones current work in not only the reviews but also in our intranet video site or on a large TV looping our latest shot works in the effects department. Whenever we see someone else working on something that can benefit our current or future work, we can just ask the artist to share his or her setup. We have 3 ways, Copy & Paste, Template Container and Library, to share the setup with the team.

Method	Copy & Paste	Template Containers	Library
Purpose	Quick sharing	Show specific sharing Multi-shots population	Cross show sharing Archive

The Copy & Paste is literally a method that allows artists to copy and paste their specific part of setup among the team through the intranet. It also keeps record of the copy, so that artist have access to it later. The Template Containers are packaged templates mostly for show specific effects to generate the base setup in multiple shots. Artists worked on early development usually prepare the template setups and share with other artists when they start working on the same effects. The Library is where artists publish their effects setups or cached data for references or reuse purposes. It holds variety of example files that are designed for cross-show sharing. It keeps metadata such as the version, author, history and categories for searching.

2.3 Rendering and Comp

Once we have all effects data ready, we start setting up shaders and rendering in Foundrys Katana. Katana is very customised for our production pipeline. We share only one Katana master file for all shots and each department works in containers called live group in this master file. All effects related rendering setups are created in the live groups assigned to effects department. There are 3 kinds of Effects live groups, shot, sequence and show. Most of effects are rendered together with all assets but volumes are rendered as separate pass for efficiency and post flexibility. Pixar Renderman is our main renderer and it is well integrated in Katana so that we do all our setup in Katana. We sometimes touch up our effects with compositing in Foundrys Nuke. In *Incredibles 2*, we used the Nuke precomp system within a master Nuke file.

3 Artistic Approach and Technical Approach in *Incredible 2* Effects

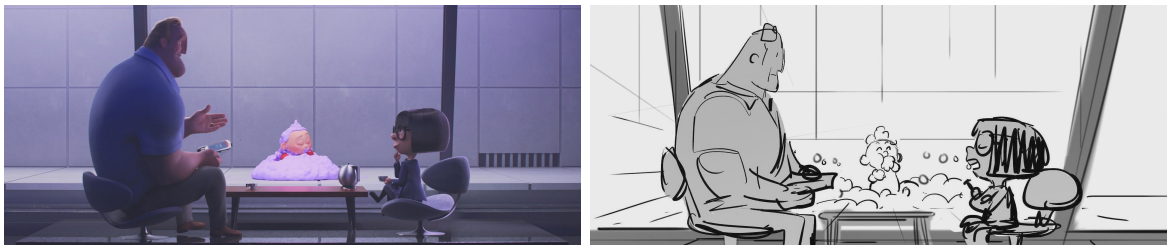


Figure 2: Final Image (left) and Storyboard (right)

3.1 Direction and Observation

Almost all effects are created by some artistic decisions. No matter how accurate and realistic the computer might be able to simulate an effect, it is very rare to see the pure result as final image on the screen. Therefore, effects need to be designed by a flexible and navigable workflow satisfying both realistic and stylized look. As an example of a technical solution to achieve a stylied look, I would like to describe the creative process of Jack-Jacks fire retardant foam effect in *Incredibles 2* (Figure2). Jack-Jack is a baby that has many superpowers in *Incredibles 2*. The fire retardant foam is one of safety functions which his supersuit can activate when he ignites fire on his body. The foam is emitted inside his suit and spills out from the suit. Art direction for the effects in this shot are listed below (Figure3).

Direction

- The foam is very fine viscous foam, not bubble foam.
- It comes out from the neck and sleeves opening on his supersuits and covers up his whole body at some point.
- Jack-Jack plays with it.

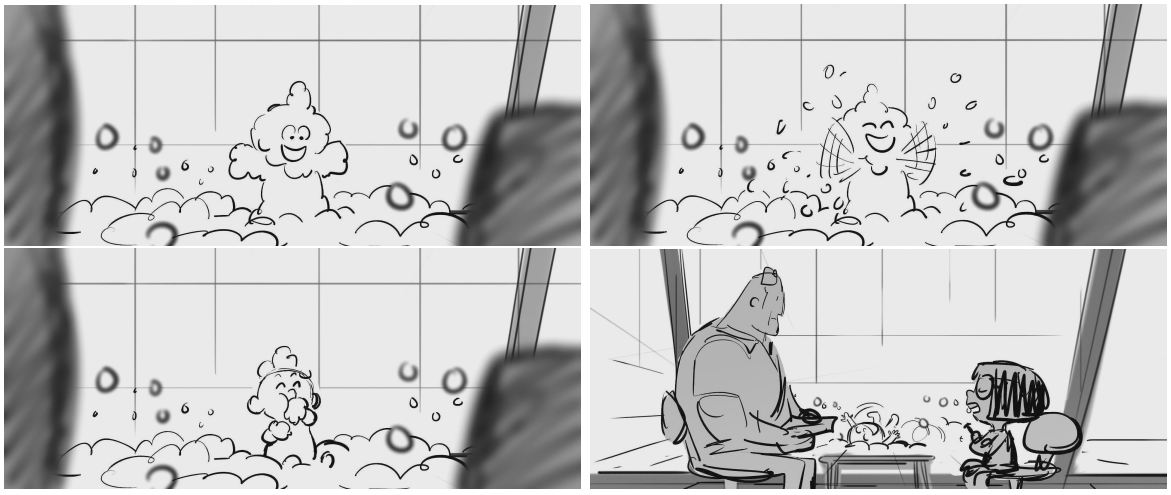


Figure 3: Storyboard for Jack-Jacks retardant foam

	Blend Shape	FLIP Simulation
Shape	**Easy** to get desired ending shape	**Not Easy** to cover the character body and get desired ending shape
Movement	**Not Easy** to avoid linear motion Not so believable motion	**Easy** to achieve variety of motion with interaction
Details	**Easy** to add and keep detail and feature of foam surface	**Not Easy** to keep detail and feature of foam surface

Table 1: Comparison chart for Blend Shape and FLIP Simulation

I found shaving cream as a reference of the very fine viscous foam and started observing its characteristics. Below is the list of features I observed.

Observation

- Light mass.
- Deforming very easily by interaction but hold the shape by itself.
- Sticking on the surface.

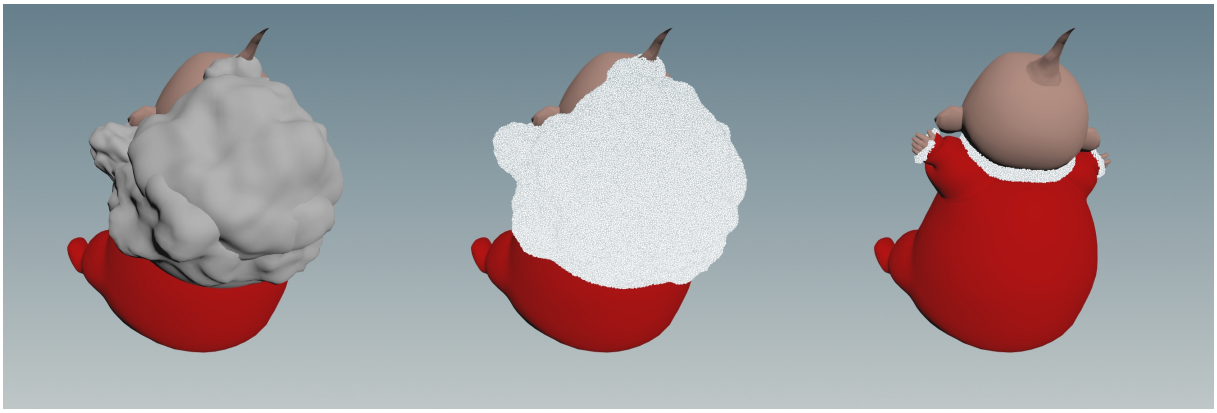
Another element I observed for the effect was the character animation. Brett Coderre, an animator of Jack-Jack, and I started a conversation in the early phase of work on this shot. We exchanged ideas how the character could play with the foam many times. It was very efficient to share the vision and understand the purpose of each motion before receiving final animation. I could design the effects setup without guessing cases character actions that might break the effect. It also helped me support the moments that the animator wanted to attract the audience with the effects.

3.2 Artistic Decision and Solution

Firstly, I needed to come up with a solution to make the foam expand along the characters surface and build the shape of foam pile in a controllable way. I tested two approaches, blend shapes and a FLIP simulation in Houdini. Both approaches had a advantages and disadvantages (Table1).

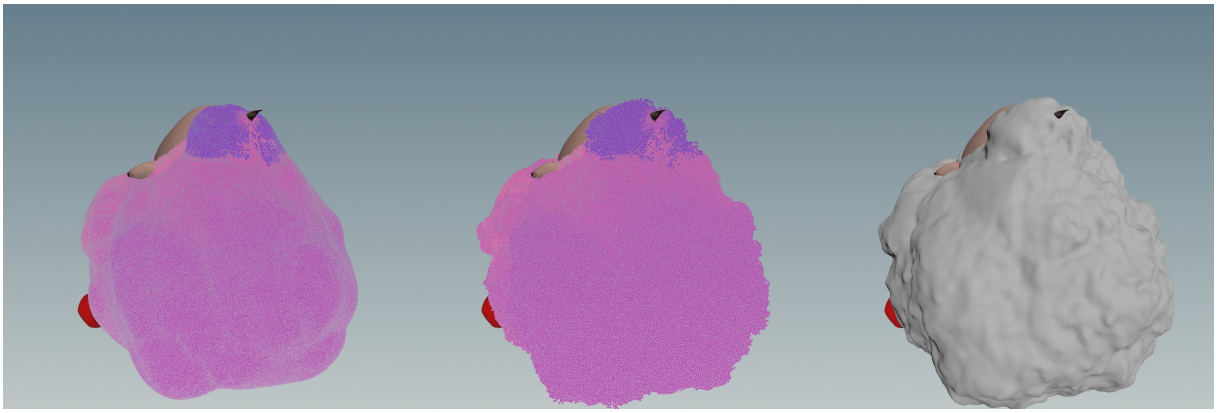
Based on these results, I tried to find a hybrid method to combine the advantages from the both method. My solution was replacing the mesh used for blending shape with point cloud and referencing the point cloud applied the blend shape animation as a goal for points in the FLIP simulation. Basically this allowed me to make result first and then use the simulation to fill the in-between animation from emitting position to goal position. Firstly, I prepared point cloud with the goal shape. Then I place them to the emitting position (starting position) and used it as source points for a FLIP simulation. I also prepared final shaped point cloud that was deformed with character animation for every frame. In the FLIP simulation, points feeded as source look for the current goal position and get small guide force towards there. To avoid having unnatural motion, I applied several conditions to stop the force and let points completely depend on the simulation (Figure4).

The solution worked as I expected except preserving surface details. FLIP simulation with viscosity tends to smooth out the surface as the simulation goes. It is a natural behavior of viscous fluid since velocity gets smoothed out with the velocity of fluid around it and I needed the behavior in the effect. Since I didnt want to change the behavior, I decided to apply post-simulation treatment by assigning a rest position attribute to the point cloud and carrying it through the simulation. The rest position attribute was used to add detail noise after the simulation (Figure5).



(a) Sculpt desired shape (b) Convert to point cloud (c) Move the points to emitting position

Figure 4: Preparation for guided FLIP simulation



(a) Simulation result (b) Apply noise based on rest position (c) Rasterize points to mesh

Figure 5: Detail restoration in post-simulation

Those two solutions enabled me to have a lot of artistic control with the simulation and to repose to new art-direction quickly.

3.3 Technical Decision and Solution

There was another challenge on the effect which was found when an animator and I were working on test animation in the early stage. The challenge was that the FLIP simulation had a hard time to sticking with a quick deforming character. Increasing substeps for the simulation helped but the foam still peeled off from the character surface time to time. It needed a more robust solution to change its sticking behavior without losing dynamic motion.

My solution was constraining FLIP simulation points to the character surface by looking at their states on the previous simulation frame. I kept the nearest surface information and the distance on each simulation point from the previous frame and used that information to get the nearest surface transformation to calculate a constraint velocity. Then blended the constraint velocity with current velocity by the distance between the points and surface (Figure6).

To control the constraint weight, I used two factors (Figure7).

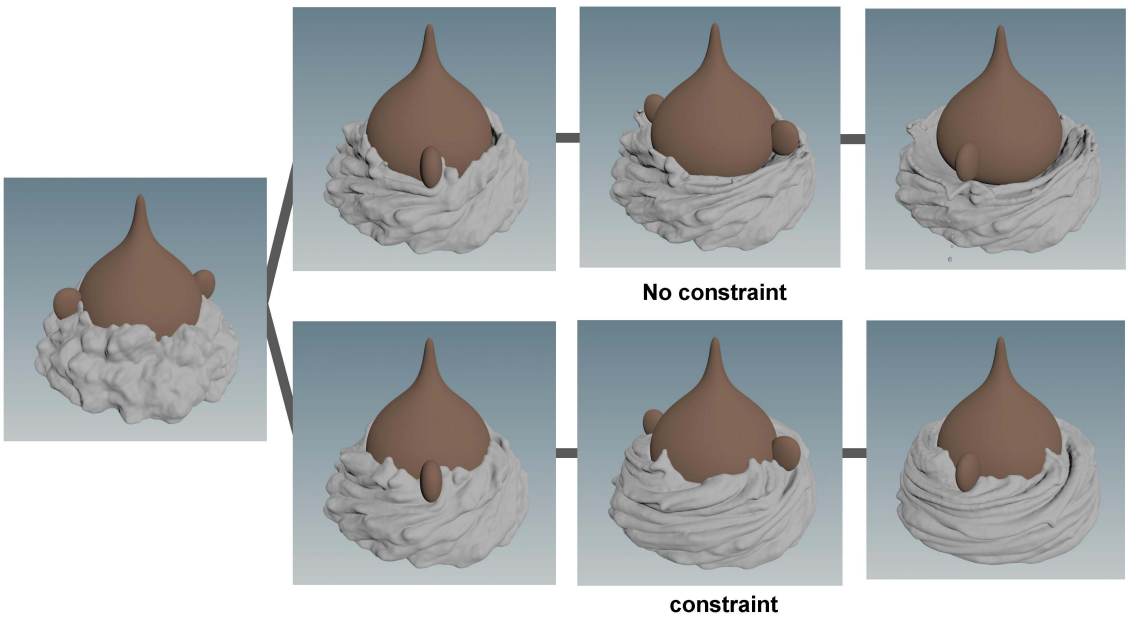
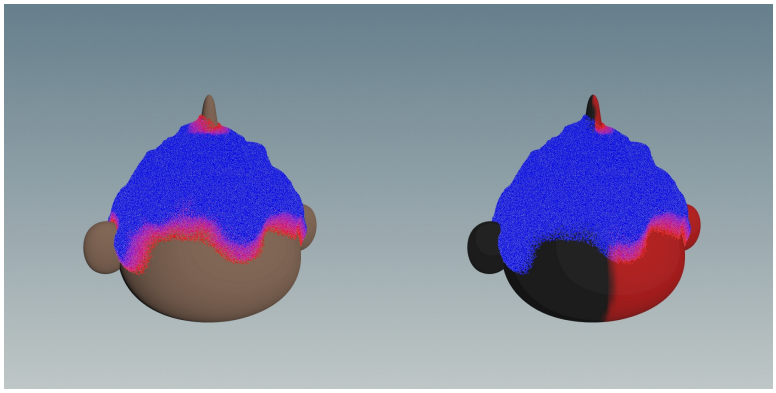


Figure 6: Constraint on the model with anticlockwise rotation and deformation

- Distance between the simulation points and the character surface
- Assign weight attribute on the character surface by painting directly



(a) SDF Constraint Weight (b) Painted Constraint Weight

Figure 7: Two factors to control Constraint Weight

4 Conclusion

The technology we used advanced significantly from the first *Incredibles* in 2004 to the *Incredibles 2* in 2018. We have more machine power, newer algorithms and better tools in effects now than ever. Thereunder, our work gets more complicated and requires fine balance between artistic quality and technical adaption. On the top of it, we are always seeking that fine line between stylized look and realistic look. The final quality of effects is still up to artists decision and it is the artists responsibility to design how art and technology contribute to storytelling from effects point of view.

Production Rendering

by Magnus Wrenninge and Ryusuke Villemin

Figure 1: *Incredibles 2* ©Disney/Pixar 2018

1 Introduction

Beginning with *Finding Dory*, the rendering pipeline at Pixar completely switched to raytracing replacing the traditional REYES architecture from Cook et al. (1987) that was used for more than 20 years. Although the move to raytracing enabled us to render realistic and convincing water and glass effects for that particular movie, the lighting tools in this new world were pretty barebone at the beginning, compared to the toolkit that was available to lighters during the REYES era. Thus for *Cars 3*, *Coco* and recently *Incredibles 2* (Figure2), we focused on improving artistic control in this new raytraced world. When introducing new technologies, we must first ensure the technology works, but also that it is intuitively controllable by our artists (as well as fitting inside the existing production workflow and pipeline).

Although pathtracing[1] became our main rendering algorithm, ad-hoc solutions were still used to achieve special effects like subsurface or hair shading, and specialized renders were still necessary to efficiently render volumetric effects. In this presentation, we will show how we improved unpon those approaches towards a pipeline where subsurface, hair, volumes are just another part of the general rendering process, not requiring any special attention.

[1]Quick word on pathtracing vs raytracing. We call path tracing trying to preserve one unique path from camera to light. This is not rigorously true in production (Fascione et al. (2018)), but close enough to enable progressive and interactive rendering.

Figure 2: *Cars 3* 2017, *Coco* 2017, *Incredibles 2* 2018 ©Disney/Pixar 2018

2 Pathtraced subsurface

For a long time, efficient rendering of subsurface relied on diffusion models (Jensen et al. (2001)). The model itself went through various improvements and extensions (Christensen (2015), Chiang et al. (2016)), and in parallel, sampling methods were also improved. In particular, while early days methods required preprocessing like creation of a point cloud, or cached data (Jensen and Buhler (2002), Hery (2005)), recent methods used raytracing to generate that same data on the fly, alleviating the need for any pre-rendering computation. That computation itself was improved multiple times, making the method more and more robust (King et al. (2013), Villemin et al. (2016)). But throughout all those modifications and improvements, the core idea stayed the same, and the computation ultimately stayed an approximation of the real phenomena.

In order to get to a more physically correct handling of subsurface effects, we are now executing subsurface rendering in a pathtraced way, effectively stochastically simulating light bouncing in the volume until it escapes it. This is made possible with the advent of faster and optimized raytracing engines (Christensen et al. (2018)) as well as available computational processing power.

Moving to pathtracing allows us to handle additional object properties that were problematic for the earlier approximations techniques:

- Correctly handle internal objects and surface geometry variation, through volume path tracing.
- Change phase functions (isotropic, anisotropic), and experiment with different free flight models (which we discuss in greater detail in section).

2.1 Sampling internal volumetric paths

Raytracing was already used in diffusion methods, but only to find exit points, around the entry point (the current shading point) (Figure3). Various methods have been used to dynamically generate those exit points, the most popular of which is multiple importance sampling (MIS) between planar and spherical distribution around the shading point, with recent improvements using resampled importance sampling (Villemin et al. (2016)). Although those methods make use of raytracing, they do not reflect

the actual path that the light is taking throughout the medium. This means that internal objects, as well as displacement of the surface itself is not taken into account correctly (Figure4).

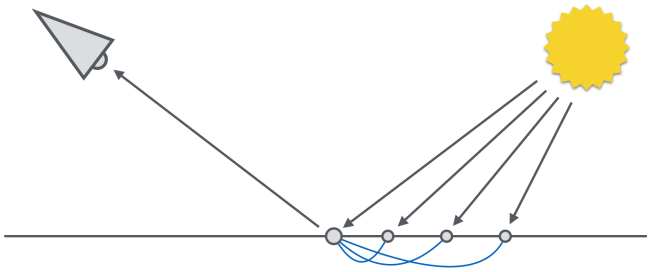


Figure 3: Diffusion SSS.

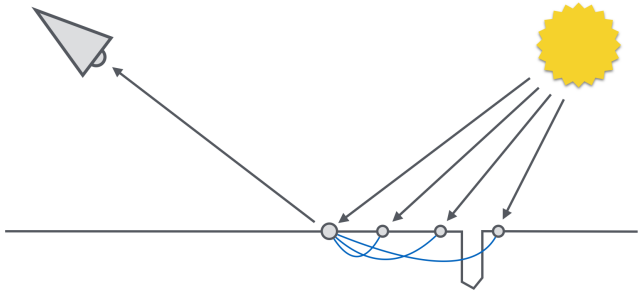


Figure 4: Diffusion SSS failing on concave geometry.

Moving to pathtraced subsurface scattering (PT SSS), we actually follow the path of a ray when entering an object (Figure5). If there is any internal object, we will hit it while marching through the medium, or if there is any surface variation, it will naturally be taken care of (Figure6). This is effectively equivalent to performing proper pathtraced volumetric rendering, and we are using the same techniques such as in Kutz et al. (2017), in order to reduce variance. In practice, although the math behind it is the same, the implementation is a little more lightweight than a real volumetric render, as we are handling all the internal bounces in one shader call, instead of going back to the main integration loop. For more implementation details, please refer to Wrenninge et al. (2017).

In practice, this means that any geometric surface variation will now be rendered correctly, without having to rely on any additional manual control to counteract the undesirable effects. Comparing the diffusion model with the pathtraced version in the closeup face render (Figure7), the bright concave area around the nose appear more natural, and while keeping the subsurface effect, we retain the displacement and micro bump details of the surface. These details are often difficult to retain with diffusion models, because they assume flat semi-infinite objects. In practice, artists had to counteract the loss of detail via excessive blur by exaggerating all the surface details,. This was a time consuming and very laborious process involving trial and error.

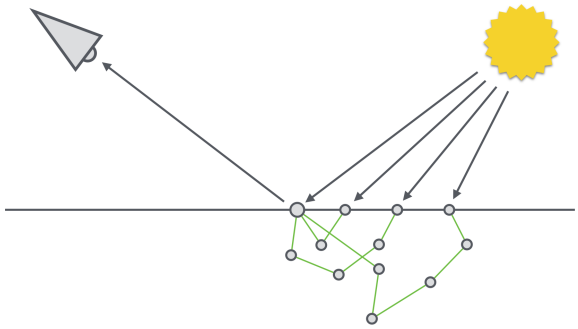


Figure 5: PT SSS.

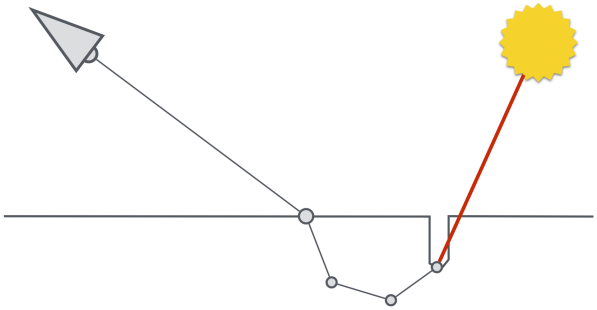


Figure 6: PT SSS correctly handling concave geometry.

2.2 Phase Functions, and free flight models

An additional benefit of running a full volumetric solution, is that we can control the phase function used at each internal bounce. Using a strong forward scattering phase function, that is closer to real skin behaviour, allows us to a even more realistic result (Figure8). We use the popular phase function from Henyey and Greenstein (1941), but any other phase function is trivial as an in place replacement.

One request we repeatedly received over the years, is a way to make the subsurface effect stronger without having to increase the mean free path (MFP) too much, which has an indesirable side effect of making everything blurry, and looking gummy.

Looking at the xxx litterature, we found an expression of non-exponential free flight in Davis and Xu (2014) that is suitable in rendering too. The details of the implementation can be found in Wrenninge et al. (2017), but the main idea is to have a second control on how particles are absorbed in the medium that is orthogonal to the usual mean free path. The Davis model was a good candidate as it has a parameterization suitable for artistic control in the form of an easily understandable tail exponent parameter (Figure9).

While using volumetric controls improved directability and quality, shading artists (especially surfacing artists) are used to painting albedo colors, which is very different than setting scattering and extinction coefficients on an object. In order for them to continue to work in that way, we introduced an automatic albedo inversion step, which takes the inputs (MFP tint, albedo colors) and converts them into volumetric components (scattering, extinction, absorption coefficients). Unfortunately, contrary to previous models like the better dipole (Hery (2005), Hery (2012)), there is no closed form inversion formulation. Additionally our model, now has more controls, like the phase function directionality,

Figure 7: Left: Diffusion SSS, Right: PT SSS

Figure 8: Phase Function, Left: Isotropic, Right: Anisotropic

our inversion process is a little more complex, but we use the same idea described in Chiang et al. (2016). We perform inverse rendering, by wedging albedo of an infinite plane under neutral lighting, varying the inputs and reading the output colors, then doing a fitting of all the observed points onto an analytic curve (Figure10).

3 Path traced hair rendering

Hair rendering has always been a difficult problem. While technically a surface rendering problem, in practice we found ourselves hafway between surfaces and volumes, because we never really render hair fibers individually, but almost always as an aggregate of them. Previous methods are indeed using volumetric techniques to render hair (Kajiya and Kay (1989)), or a mix of surface and volume techniques (Petrovic et al. (2005)).

Independently of what technique is used, in order to render hair correctly, we need to solve 3 main problems:

- Find a good hair Bidirectional Scattering Distribution Function (BSDF) model.

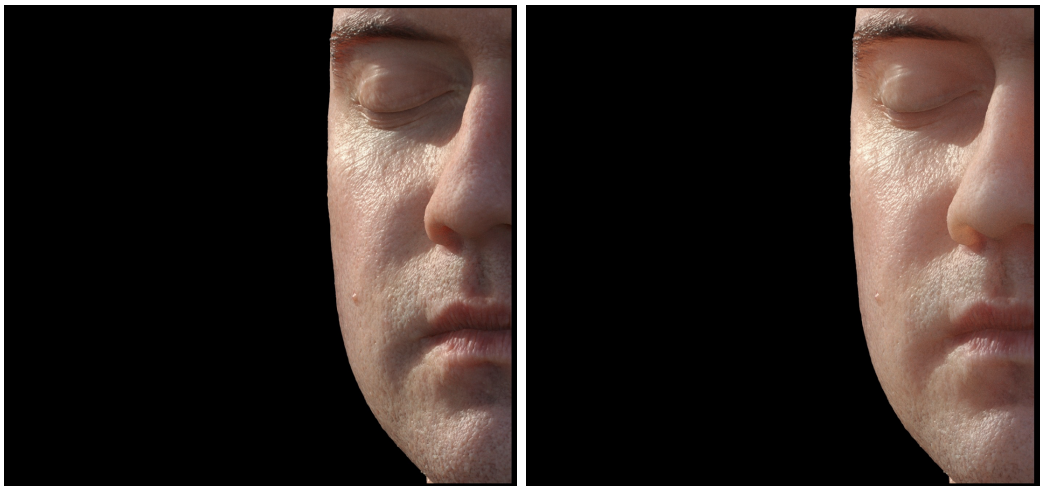


Figure 9: Non-exponential free flight, Left:tailExp=3, Right:tailExp=1

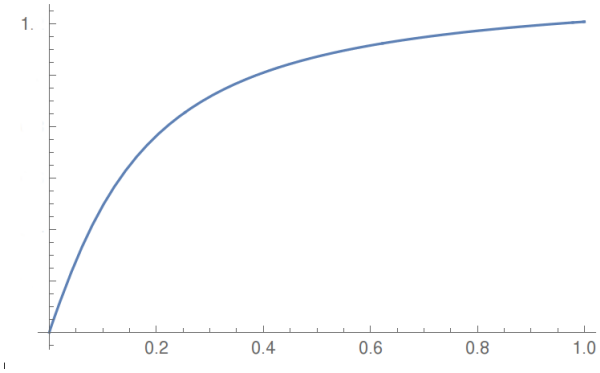


Figure 10: PT SSS inversion curve for (albedo=0.2, g=0.5)

- Efficiently compute hair shadowing and indirect lighting.
- Have a predictible and easy to control hair shading model.

Before going fully raytraced, the final look of the hair would depend on the combination of the 3 points, which would make it very difficult to control.

3.1 Hair BSDF Model

The most common and simple model is the one from Kajiya and Kay (1989). While being very easy to sample and evaluate, it has a very simplistic formulation. It consists of 2 components, one diffuse and one specular, and both assume a perfect cylindrical curve section. Later Marschner et al. (2003) improved the model by actually studying the geometry of a hair fiber, analyzing its response and decomposing it into 3 components based on the number of light interactions with the hair fiber: one reflection R, 2 transmission TT, 2 transmissions and 1 reflection TRT. R is a monochromatic reflection, TRT tinted reflection from light traveling inside the fiber, TT colored transmission (formerly approximated by tinted shadows in earlier models). A special mode of TRT interactions necessitates a special computation, and is taken care of by a separate GLINTS lobe, which is really just a sub part of the TRT lobe. Further interactions with more than 1 reflection are ignored.

While being physically more correct than the Kajiya model, it was found to be a little less artist friendly. as a result, a few years later a new model from Disney Animation, the artist friendly model

appeared, that focused on artist usability (Sadeghi et al. (2010)). Roughly it is based roughly on a Marschner model but with no real normalization, and fully decoupled lobes for ease of use. Later d'Eon et al. (2011) introduced a model that went further than Marschner's by accounting for TRRRT. Unfortunately this model was not easily importance sampled. At Pixar, we use an improved Marschner with an adhoc deep bounce model TRR..T (which we call residual) and is fully importance sampled (Pekelis et al. (2015)) including glints and eccentricity. This allow us to hit realistic hairs, but also completely artist driven looks while maintaining energy conservation and other properties that makes it suitable for a pathtracer (Figure11, Figure12).

3.2 Hair Shadowing and Lighting

Shadowing used to be addressed using shadow maps, or some volumetric representation. Both are susceptible to bias in the result if the resolution is too low, or not representing the actual hair geometry close enough. One important limitation is that they were also specialized for shadows, so indirect lighting would need to work with completely a different representation.

But with any recent renderer, it is now reasonable to ray trace visibility rays through millions of curves. And by using a physically based hair BSDF model, we don't need to fake tinted transparent shadows anymore, which brings down the cost of calculating shadows.

Indirect lighting is essentially a problem for light hair. For dark or black hair, indirect bounces between hair does not account for much lighting. However for lighter hair like blonde hair, most of the color is actually coming from all the indirect bounces. It is common for light to bounce more than 20 times in hair and still contribute a non negligable amount to the final look (Fascione et al. (2018)). That is why using an optimized russian roulette algorithm is preferred to having a hard limit on the number of bounces, to avoid any bias and loss of energy in the final render. At pixar we used to use point based subsurface to mimic light diffusion through out the hair volume, other techniques like dual-scattering simplification were used in some cases (Zinke et al. (2008)), photon spherical harmonics Moon et al. (2008), but all those models were pretty crude approximations, and didn't produce realistic blonde hair. The existance of all those methods was to avoid pathtracing in the first place, a limitation that is less relevant nowadays. We now trace all the indirect bounces with no limit on the path depth. In order to still improve performance, we use a few tricks described in Fascione et al. (2018), but for the most part we just let the renderer converge to the right color.

Figure 11: Blonde Hair with all components

Figure 12: TopLeft: R, TopRight: TT, BottomLeft: TRT, BottomRight: GLINTS

3.3 Hair Shading

Our last topic is somewhat similar to volume rendering with single scatter albedo. Specifying the color of a single hair fiber does not predict what the full hair color will be. This is because the final color we see is the result of the accumulation of all possible light bounces in the hair and not a single interaction with one fiber (Figure13).

Unfortunately, there is no easy way to predict the final color for every possible scenarios, so we use empirical models. Disney started using inversion tables (Chiang et al. (2015)), rendering a cube of hair and looking at the resulting colors. We do the same except we use generic groomed models since they gave results closer to their artist expectations.

The albedo needs to be boosted to match the target color (Figure14). For the actual inversion calculation process, we tried different hair grooms but ended up with something similar to the hair balls shown in Figure15.

Figure16 represents the result of only direct lighting only with no shadow, in a white furnace. What we see is directly the sum of all the components building our hair model (R, TT, TRT, GLINTS, Residual). Figure17 shows the same models, but this time with normal render settings, with shadows and unlimited indirect bounces. It is easy to see that the end result, after shadowing and indirect, is darker than the artist intended color. Comparing to the simple spheres on the row below, which retain the original color, hair is getting darker as the color we see is the aggregate color hair and not the fiber color. Figure18 shows the result after applying our correction curve to the inputs of the hair shader.

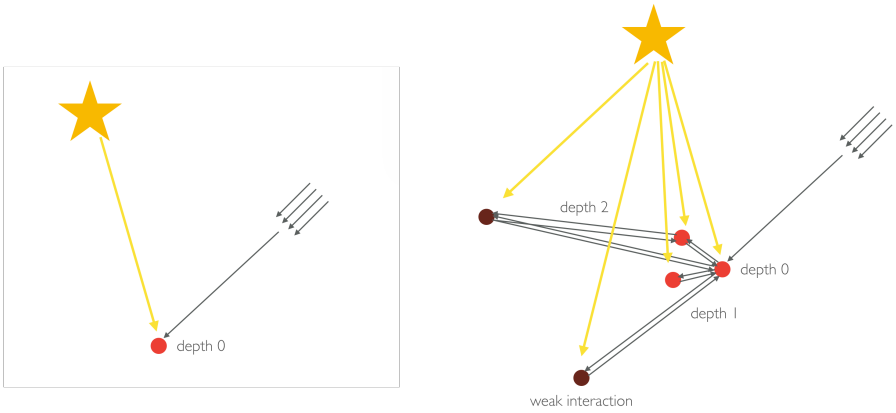


Figure 13: Hair/Light Interactions.

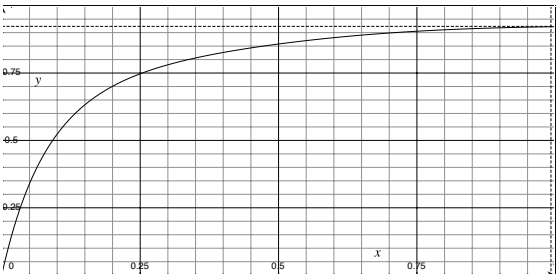


Figure 14: Hair Albedo Inversion Curve

The result is much closer to the simple spheres, which let our artists shade hair as any other surface.

The second row shows the same renders applied to a production asset, the raccoon from *Incredibles 2*.

Rendering fully raytraced hairs, with a physically based hair response coupled with advanced albedo inversion techniques, enabled us to hit both realistic hair, but also strong art directability, to render believable but very artistic hair.

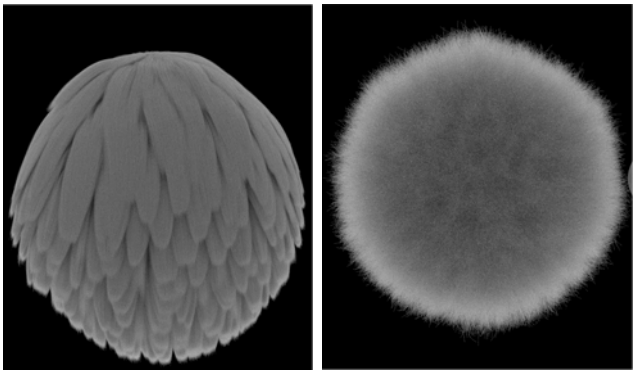


Figure 15: Hair examples used for inversion.

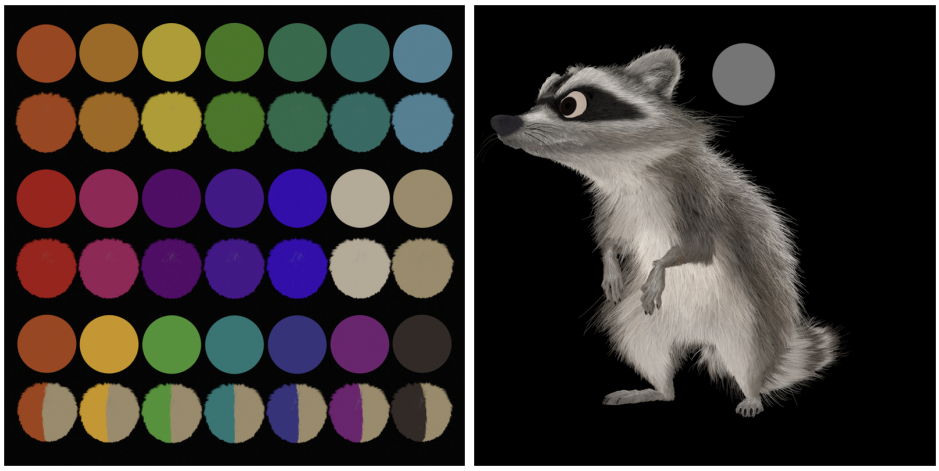


Figure 16: Input Colors.

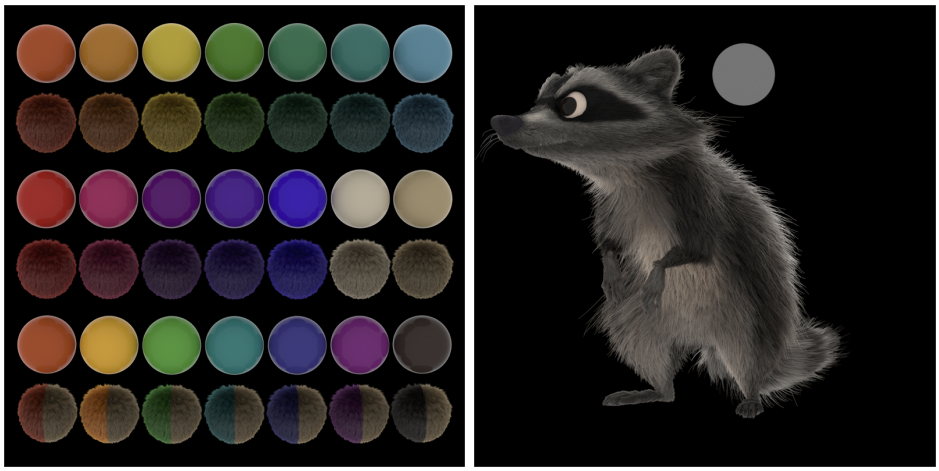


Figure 17: Raw Albedo Renders.

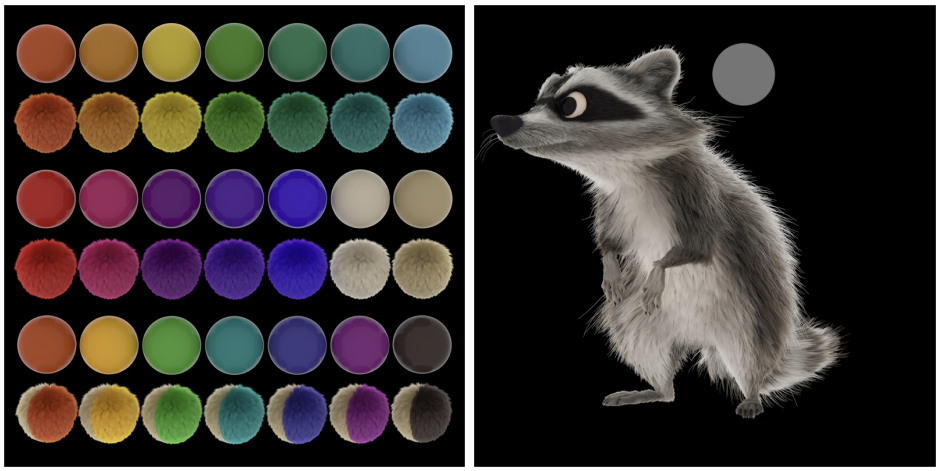


Figure 18: Corrected Albedo Renders.

Figure 19: Effects animation and 3D clouds from Incredibles 2. © Disney/Pixar 2018

4 Volume Authoring and Rendering at Pixar

Like many other animation and visual effects studios, Pixars volume authoring and rendering pipeline has grown and evolved over a period of time. A particularly large shift occurred around the time of Finding Dory, as RenderMan shifted from rasterization-based volume rendering to a path tracing-based framework (RIS). This shift required a new approach not only to rendering, but also to the supporting pipeline and the authoring tools used by artists and technical directors in several departments.

This part of the course gives an overview of the current authoring pipeline as well as a view into how the tools are used by some of the most volumetrically active departments.

4.1 Foundation

The current incarnation of Pixar's pipeline for creating volumetric effects was designed with two primary goals in mind: First, it had to be flexible so that artists could utilize a wide range of tools for authoring. And secondly, it had to ensure that the data created would be optimally structured for robust and fast rendering.

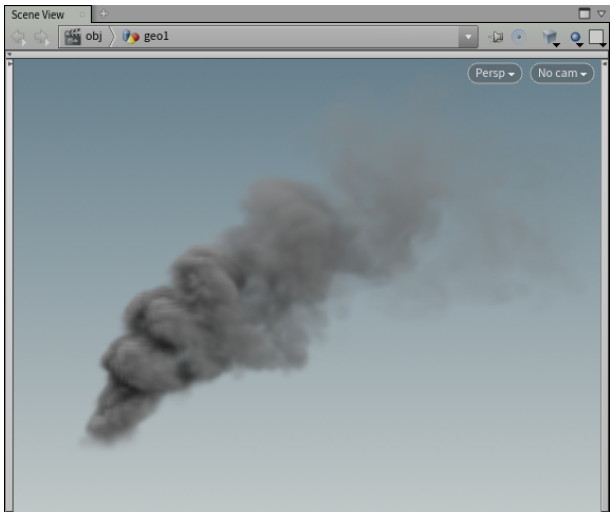


Figure 20: Field3D primitive in Houdini. The volume references a high-resolution voxel buffer on disk but is visualized with a light-weight representation by leveraging a pre-computed MIP.

At Pixar, several departments create volumetric elements. The Effects Animation department has the longest history, but starting with The Good Dinosaur, the Matte Painting department has created 3D clouds for several films, and more recently volumetric workflows have made their way into the Sets and Character departments as well. Next, we cover the common pipeline that supports all of this work, and also look at the workflows within some of the different departments.

4.1.1 Houdini

SideFx's Houdini is the foundation that nearly all of Pixar's volume authoring tools are built on. Houdini itself includes a wide range of tools for this purpose, from fluid simulations in DOPs to the OpenVDB toolset and many other ancillary tools. Artists regularly utilize these built-in tools but also build many custom tools in order to create their volumetric datasets.

4.1.2 Field3D

Although Pixar's authoring pipeline supports all of the formats and data structures used in Houdini as inputs, the only file formats used downstream of Houdini is USD and Field3D. Field3D stores the volumetric data itself, and USD is used to represent the rendering structure and placement of this volumetric data in the scene (see below.)

A set of custom primitive types allows Field3D volumes to be represented natively in Houdini. These primitive types allows both live, in-memory storage (`GEO_PrimF3D`), but also referenced volumes on disk (`GEO_PrimPackedF3D`). Live (i.e. mutable) Field3D volumes are created by a set of custom Field3D plugins that extend Houdini's built-in capabilities, and include both SOP, ROP and DOP nodes. The packed reference primitive (immutable volumes) is used extensively for visualization of high-resolution data, and in dressing- and instancing-based workflows such as the clouds pipeline.

4.1.3 USD

While both Field3D and OpenVDB provide industry-standard ways of storing volumetric data on disk, they do not provide the necessary features that clearly describe a volumetric element to the renderer. Modern path tracers need to know which sub-volumes need to be bounded (density/extinction), which ones define , and although it is possible to store these as generic metadata in both OpenVDB and

Files	Volumes per file	Primvar bindings	Use case
Single	Single	Single	Basic density field.
Single	Multiple	Single	Density composed of multiple volumes.
Single	Multiple	Multiple	Pyro or smoke simulation with motion vectors.
Multiple	Single	Single	Clustered smoke simulation generated in parallel.
Multiple	Multiple	Multiple	Clustered pyro simulation generated in parallel.

Table 2: Expressing the composition of a render element.

Field3D, it is desirable to have a more strict definition of these important settings. Additionally, shaders, shader assignments and shader primer bindings can be quite complex and can involve volumes composed of data from multiple files.

Table 2 shows some of the combinations that are commonly encountered in a production pipeline:

To solve this problem, Pixar uses two custom USD schema to encapsulate the description of a volume as it is seen by the renderer. `PxVolume` represents the volume as seen by the renderer, and each `PxVolume` can have N number of `PxField3D` children, which define the fields to load from one or more Field3D files:

- `PxVolume`
  - `densityMult` (float)
  - `extinctionAttributes` (string array)
  - `motionBlurLength` (float)
  - `velocityAttributes` (string array)
- `PxField3D`
  - `filename` (string)
  - `name` (string)
  - `attribute` (string)
  - `index` (int)

4.1.4 MIP Volumes

As authoring tools have improved and hardware specs have grown, volumetric data has become increasingly complex and high-resolution, sometimes ranging into multiple thousands of voxels, cubed. While this has been beneficial to image quality, it can make authoring and rendering more complex. Each time the resolution of a volume doubles, the memory footprint increases eight times, and if a volumetric element is authored at higher resolution than needed for a given shot, memory is wasted. At the same time, having additional resolution available means that e.g. the camera angle can be adjusted late during production without having to worry that an element will wind up having insufficient detail.

Rather than task artists with having to decide exactly what resolution to make each individual volumetric elements, Pixar's pipeline uses MIP representations for all volumes. A MIP volume works similarly to a MIP texture: a single file stores the high resolution data (e.g. a 2000^3 volume), as well as multiple lower resolution representations (1000^3, 500^3, 250^3, etc.). By making multiple representations available, the lighting package or renderer can automatically choose an appropriate resolution based on factors such as the distance to the element, the field of view of the lens, etc. This frees artists up to put as much detail as is needed into a given element, without having to worry about subsequent render-time memory use.

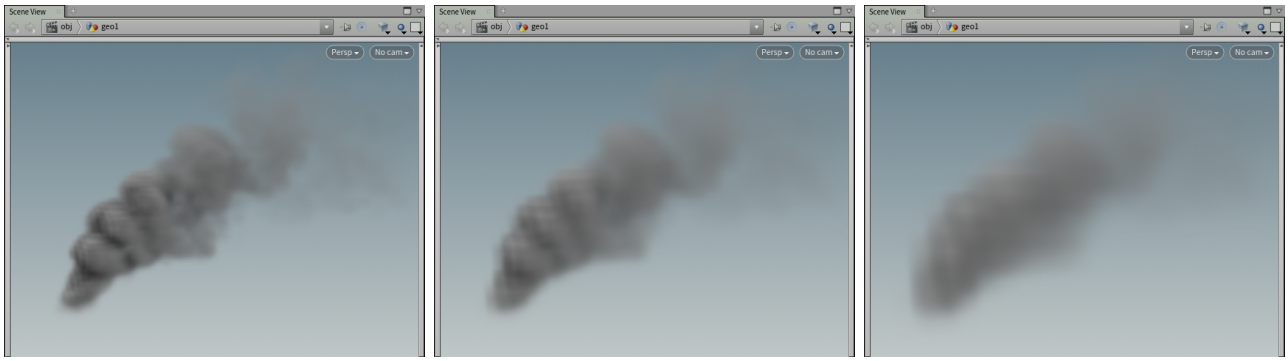


Figure 21: Multiple MIP levels from the same underlying high-resolution volume.

Field3D implements a generic MIP volume class that allows MIP versions of dense, sparse and temporal volumes to be stored natively. All of the different data structures support deferred loading, so that only MIP levels that are used by the renderer get loaded from disk.

In order to ensure that MIP volumes are available throughout the volume pipeline, they are automatically generated whenever a volume asset is created.

4.1.5 Min/Max Volumes

As mentioned previously, one of the key design goals for the pipeline is to ensure that the volumes can be rendered *efficiently*. In the days when raymarching was the primary rendering method, communicating the sparsity of the volume was key, for example using the inherent block structure of Field3D or OpenVDB volumes. However, modern, path tracing-based renderers need additional information about the volume in order to render as efficiently as possible.

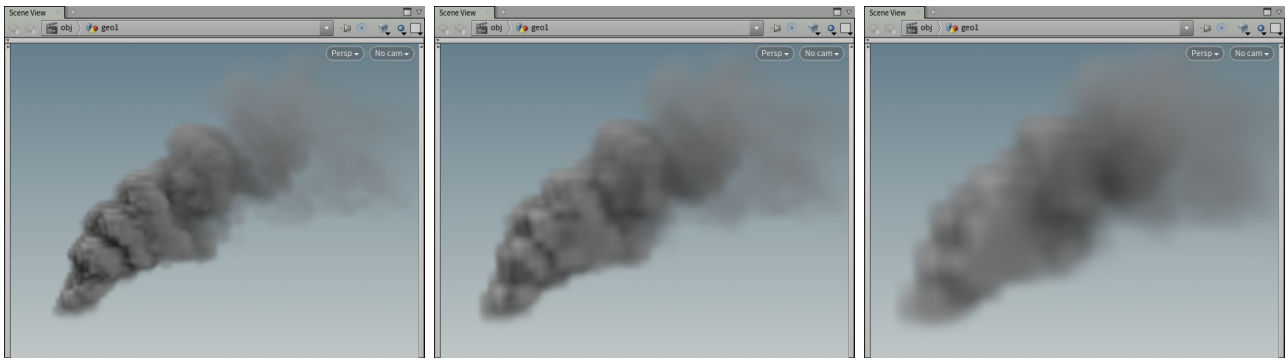


Figure 22: Visualizing the majorant density.

The most important information that needs to be communicated is *where* and *how much* of the volume is present. That is, not only the sparsity (i.e. placement), but also the magnitude of the volumetric properties.

Most modern path tracers use some form of *tracking* to render volumetric elements. These generally need to know the maximum density (a.k.a the *majorant*) present in the scene. The renderer then uses this information to optimize the placement of samples during integration. In order to best optimize sample placement, the majorant must be queryable for smaller sub-regions, so that the integrators sampling can be adapted to thin and thick regions as needed. Volume integration is a complex topic, and Novk et al. (2018) provides a very good overview of the current state-of-the-art.

In order to provide the majorant (and minoring) information, Field3D provides built-in support for

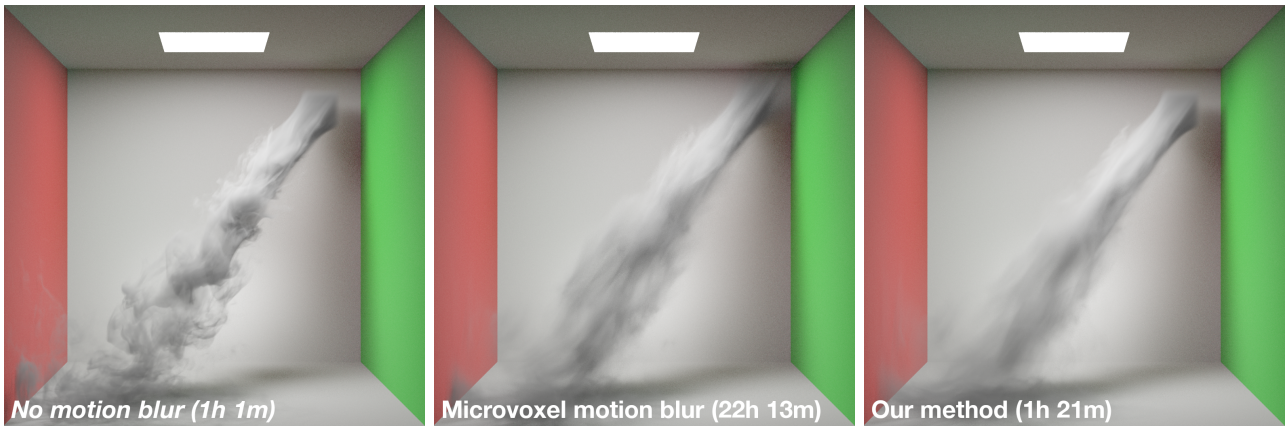


Figure 23: Motion blur using microvoxels versus temporal volumes.

generating what's called *min/max* representations of any volume element. The min and max properties are created by replacing the standard reconstruction filter in the MIP creation process with a min() or max() operation. The min/max MIP pyramid thus becomes a conservative bound on any region, and makes it fast to query min/max even for large regions, as they can simply look at the lower-resolution MIP levels. Figure 22 shows a base density volume and its majorant version at multiple resolutions.

The process of generating this meta-information is entirely automated and runs at the same time as the MIP creation during volume export time.

4.1.6 Motion blur and temporal volumes

Another challenge in production volume rendering is the handling of motion blur. In theory, motion blur is not much of artistic choice but rather something that either is or isn't present. In practice however, artists need to be aware of how motion blur will be generated for a given element, and Pixar's pipeline provides multiple ways of handling the process.

Volumetric motion blur is computationally complex to achieve in the path tracing regime, as velocity vectors effectively act as a *displacement shader*, pushing densities outside the carefully constructed min/max bounds discussed in the previous section.

In order to address these inefficiencies, *temporal volumes* were introduced by Wrenninge Wrenninge (2016), which encode the motion of the volume as a 4D signal, removing the need to account for displacement during rendering. Temporal volumes have since been integrated into the core of RenderMan, and provides a general and flexible solution to the motion blur problem. Figure 23 shows a image and render time comparison between temporal volumes and microvoxel rendering.

For most elements, any motion blur effects are baked into temporal volumes at the authoring stage, but in some cases, it is preferable to generate the motion blur effect using other times and other methods. The following is a description of all the options available in the Pixar pipeline:

1. **Simulated temporal motion blur**. For volumes that are the result of a Houdini DOPs simulation, one of Pixar's custom DOP plugins can optionally create temporal volumes as the simulation progresses, which accurately captures sub-frame motion and produces the highest quality motion blur.

2. **Post-process temporal motion blur**. For procedurally generated (i.e. non-simulated) volumes, a custom SOP plugin will generate temporal volumes given a volume to blur (density) as well as a velocity field. Temporal motion blur is computed for the full duration from the start of frame to the following frame, which allows arbitrary shutter length and shutter curves

to be used without having to re-compute any data. This method still provides the advantage of pre-computing motion blur, enabling quick time-to-first-bucket during rendering.

3. **Render-time temporal motion blur**. Optionally, in cases where it is impractical to pre-compute motion blur (for example if the velocity field is generated at a later stage than the density field), RenderMan can perform the same static-to-temporal conversion at render-time. This has the downside of increasing the startup time for any render that uses the volume, as it requires re-computation in each render.

4. **Advection or Eulerian motion blur**. With the introduction of *volume aggregates* (see below), it is practically feasible (although not as efficient or accurate as temporal volumes) to compute motion blur on-the-fly during rendering. In this case, the `impl_f3d` plugin (described below) will compute the necessary offset padding when bounding the volume, and will subsequently offset each sample point by the velocity vector field, yielding motion blur in the process.

5. **No motion blur**. Some volumes are either static or slow-moving to the point where motion blur would be unnoticeable. In these cases motion blur is simply ignored.

Additional details on rendering of volumetric motion blur can be found in the SIGGRAPH course by Fong et al. Fong et al. (2017).

4.2 Effects Animation

Much of the current volumetric pipeline at Pixar has its roots in the effects animation department, and going back as far as Wall-E, Field3D has been the primary volume interchange and rendering file format. Integration with the scene construction pipeline was handled through TIDScene (a predecessor to USD) and through RenderMan's native RIB files.

When USD was introduced at the studio, a new set of Houdini plugins were written to handle import and export, and the entire pipeline was made USD-native. In this incarnation, Field3D is still used to carry the individual volumes (voxel buffers) and their associated metadata, but the integration into the scene graph is handled entirely using USD.

4.2.1 USD specifics

USD is used extensively in the effects animation department and the Houdini USD plugins provide full round-trip import/export capabilities that allow everything from additions to deletions and modifications to the existing scene, at any level of the scene hierarchy. This functionality is used for tasks ranging from applying deformations, to authoring procedural surface shading signals to rigid body simulations and fracturing/destruction.

On the volumetrics side, the USD Volume Asset node is used to create new volumetric data and inserting it either at the show, sequence or shot-level, as well as for creating various types of library elements that can later be re-used to easily dress elements into specific shots. By leveraging the light-weight Field3D reference primitives, all of Houdinis procedural tools can be used to build up complex effects from these pre-canned elements, saving valuable artist time and shortening iteration intervals.

4.2.2 Clustering

One particularly common workflow in effects animation is *clustering*, which refers to the process of parallelizing simulation and geometry generation into independent pieces. An example would be the generation of foot interaction dust on a sandy surface: rather than simulate the entire motion of a character in a single space, each foot hit can be simulated independently and in parallel, which is

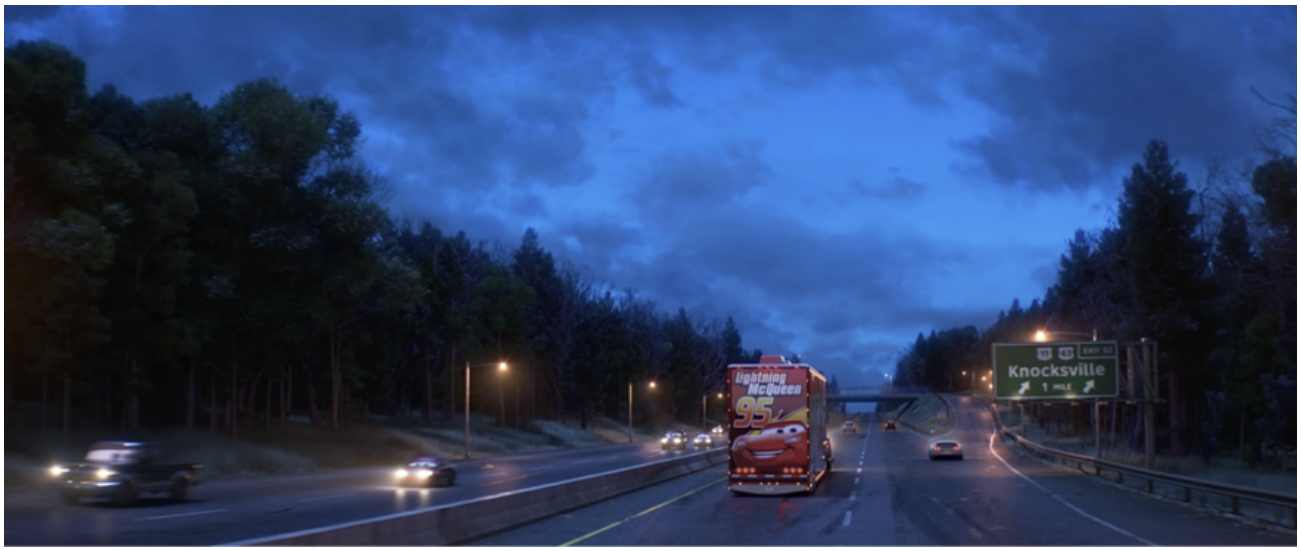


Figure 24: Clouds from Cars 3. © Disney/Pixar 2018

both faster and more flexible. The pipeline tools give artists several choices for how the resulting volumes are exposed to RenderMan: as a single, large volume, with the individual pieces hidden under a single scene graph location, or individually, such that e.g. lighting artists can modify the density of individual elements at render time. The pipeline takes care of optimizing each case, letting artists focus on the look and the control thats needed, rather then on how a particular volumes structure should be communicated to the renderer.

4.2.3 Temporal tools & Reves

With temporal volumes serving an important role in efficient rendering, it was important to provide the artists with tools that made the creation process flexible. During DOPs simulation, a custom DOP node can be added to the simulation chain, which acts as a temporal memory during execution. Using this node, the output of the simulation is a native temporal volume, and the full motion of the simulation, down to individual subsets, is captured. For volumes that are created procedurally, a Create Temporal Volume SOP takes ordinary volumes (e.g. density) as well as a velocity vector field, and outputs a temporal volume ready for rendering.

4.3 3D Clouds and Hybrid Matte Paintings

Skies and clouds have been present in Pixar films since the original Toy Story, but they were traditionally done using classic matte painting techniques, sometimes with individual clouds on 3D cards for added parallax effects. On The Good Dinosaur, Director Pete Sohn and Director of Photography Sharon Calahan wanted the environment to be *alive*, from grass and branches close to our protagonist, all the way to the distant clouds. With the existing Houdini-based volumetric pipeline in place for effects animation, the clouds team saw an opportunity to leverage these tools to create fully rendered 3D cloud scapes Webb et al. (2016). The outcome was successful and the workflow was later evolved and used on Finding Dory, with rendering handled by the (at the time) new RIS version of RenderMan.

While the fully 3D approach meant that lighting artists had the greatest degree of control possible, render times were significant, and the clouds team started looking for alternative approaches that could provide both the benefits of fully rendered clouds with the detailed control and fast turn-around that is possible with a painting-based approach. These constraints, along with the particular production

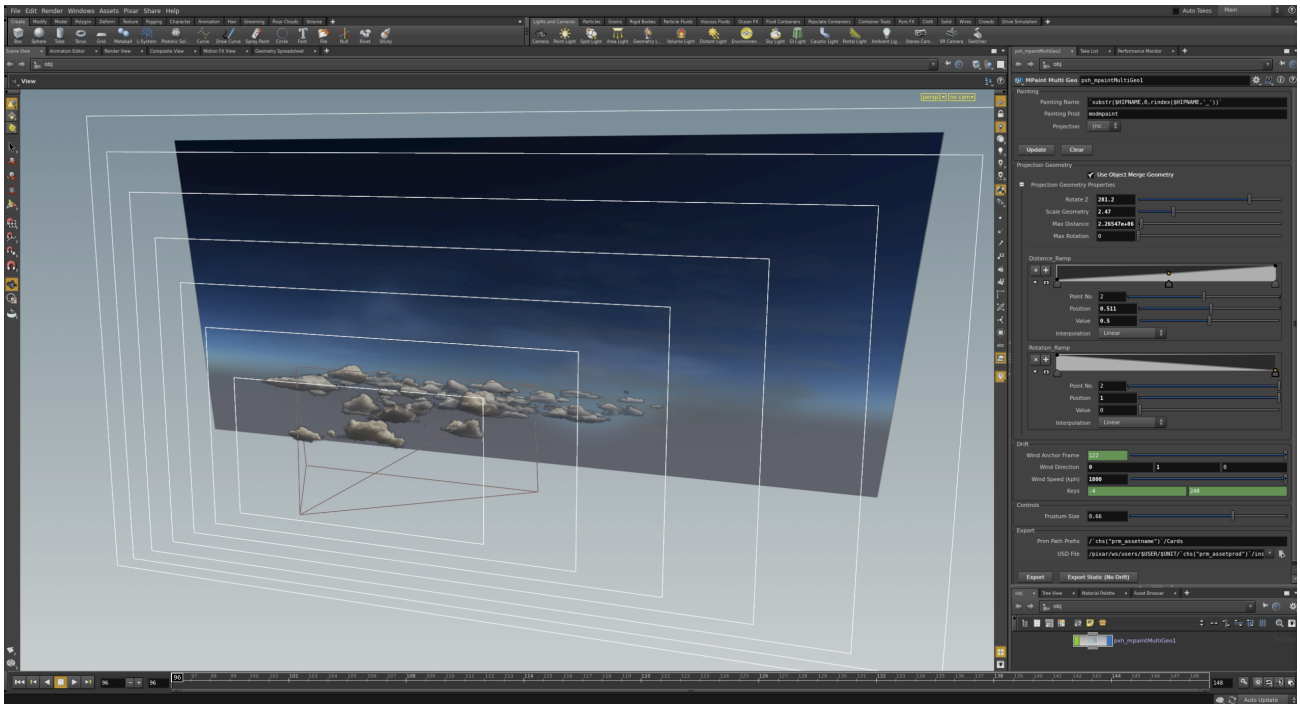


Figure 25: 3D rendered clouds on 2D cards from Cars 3. © Disney/Pixar 2018

design requirements in Cars 3, led to the development of a hybrid 2D/3D workflow that started with 3D renders of clouds, but also incorporated paint work on top of what had been rendered. Finally, on Incredibles 2, this hybrid method incorporated a Nuke-based compositing workflow that made for a flexible and efficient approach to cloud creation Murphy et al. (2018).

5 Acknowledgments

Thanks to the whole team, Christophe Hery, Jeremy Newlin and Junyi Ling. We would also like to thank Beth Albright, Kiki Poh and Markus Kranzler for their work and feedback on hair shading; Alex Marino, Chuck Waite, Robin Huntingdale and Ben Porter for providing assets and groom models; Reid Sandros and Tom Nettleship for rendering all those curves, as well as all the shading, lighting and rendering TDs especially on *Incredibles 2*. We also want to thank the full RenderMan team, who we worked with more closely than ever to make this raytracing world possible.

6 Conclusion

Overall moving to raytracing improved the out of the box renders, which translated into more creative exploration and refinement. At the same time, the uniformization of the rendering process means that we have a simplified pipeline process and less diversity in the techniques we have to maintain and support, which in turn makes it easier to concentrate our work and optimization efforts. 3 years after *Finding Dory*, our first movie using the new RIS raytracing architecture, we can now say that we are fully raytraced!

Compositing workflows on *Incredibles 2*

by Chia-Chi Hu

Figure 1: *Incredibles 2* ©Disney/Pixar 2018

1 Introduction

During *Incredibles 2*, Pixars compositing pipeline was re-worked and extended in order to better support the shows technical requirements, a new lighting workflow, and a very tight schedule. While initially driven by the the movies large amount of effects integration, in the end the compositing workflow became an integral part of the films production methodology, making it possible for artists to individually handle both lighting and compositing on a large number of shots concurrently.

On *Incredibles 2*, most of the compositing shot work was done by the sequence lighting artist, but the show also had a team of three dedicated compositing artists who were responsible for creating show setups and templates, and also for tackling particularly complex compositing shots.

There are some unique challenges in the way that compositing is used for feature animation. Every element is computer-generated, so there are no live-action plates to consider, however images are often broken down into individual elements, and elements are sometimes generated in different renderers. The shot count is often as large as any visual effects-driven feature film, and the lighting and compositing teams are often small and need to work in parallel with other departments. All of this made it clear that a new, more efficient compositing workflow was needed. In the end, the shows shot count reached over 2000, which was completed with only 36 lighting artists at its peak.

The lighting workflow on *Incredibles 2* was different than that of most previous shows. A single lighter would generally be assigned an entire sequence of shots, and would carry those shots from initial rough lighting through to final shot lighting polish. This change meant that the compositing workflow had to change as well, making it possible to manage a large set of shots in parallel.

Compositing was often leveraged to provide fast feedback and iteration time during production. Many tasks that could be addressed equally well in both the lighting and compositing stages would generally be handled in compositing, but the changes were subsequently pushed back into the lighting scripts. In particular, adjustments to light intensities could quickly be dialed in compositing, but in order to minimize unexpected changes in noise levels, these changes would usually be pushed back into

the lighting setup, so that the renderer could optimize sample distribution accordingly.

Director of Photography (DP) Erik Smitt and compositing lead Esdras Varagnolo along with Shawn Neely, Greg Finch and Joachin De Deken from the Software & Tools Department developed a set of new tools to accommodate this workflow.

2 Sequence-level Nuke Scripts

Pixar has long had a lighting workflow that allows top-down control of lighting across many shots at once. For example, when a sequence is started, a Master Lighter will develop a lighting configuration that works for the majority of shots, establishing the broad strokes of the sequences look. Shot Lighters will then refine this setup for individual shots, finalizing the look. However, by leveraging custom tools, the choices made during master lighting are still live, and if a DP wants to apply a lighting change consistently across multiple shots at a later stage, this can be done directly, without every shot lighter having to make corresponding adjustments on shots that they are already in progress on. This is quite different from compositing workflows in live action, where a compositing supervisor may establish a template for a sequence, but once the sequence is actively being worked on, there is only one script in play, and sequence-wide adjustments require careful coordination between artists. Figure2 shows an example of these templates.

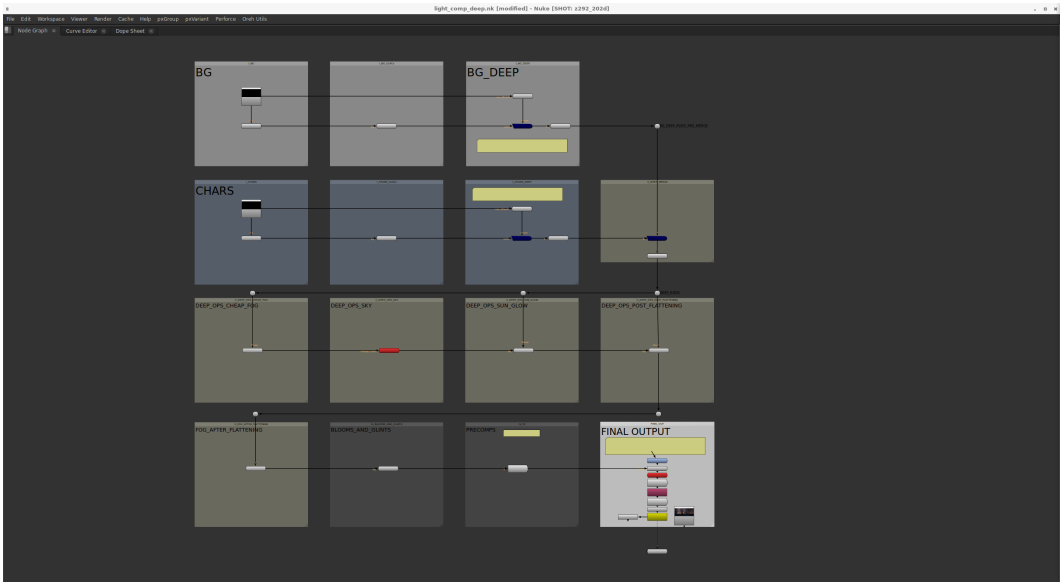


Figure 2

In order to support a compositing workflow where a single artist could work on all the shots in a sequence in parallel, it was clear that a different approach was needed.

3 Shot Board

When working with an entire sequence at once, it is important to provide the artist with a clear overview, and to make context switching quick and effortless. To this end, a custom Nuke panel called the Shot Board was developed. It gave the artist a thumbnail overview of all the shots in the sequence, and by clicking one of these thumbnails, the Nuke script would switch the active shot. This allowed artists to work actively on manipulating the comp script within the context of one shot, but

to immediately see the effect the change would have on a second shot. Figure3 shows an active shot template with shot board on the right.

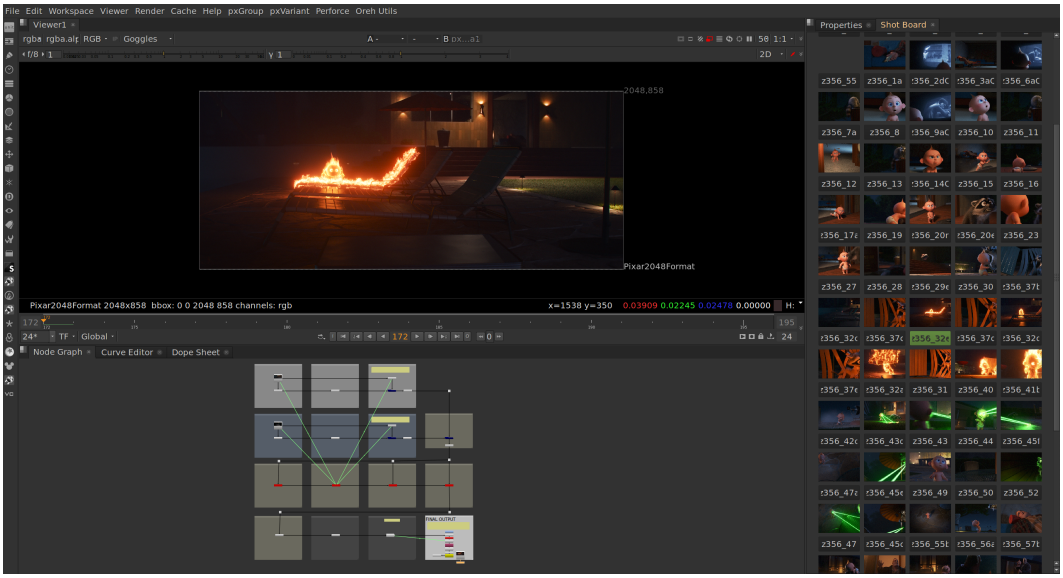


Figure 3

4 Switch Shot & Shot Enable Group

Being able to execute the same node graph for multiple shots in a sequence would not be particularly useful if every shot had to follow the exact same node graph. Instead, two types of nodes were developed that gave artists the option to enable or disable parts of the graph based on which shot (or set of shots) was currently active. Depending on the particular template, a Switch node would enable a portion of a graph in the script, or a Shot Enable Group (SEG) would activate an entire sub-graph. Figure4 shows how wildcard patterns and exclusions could be used to apply changes to all shots except one.

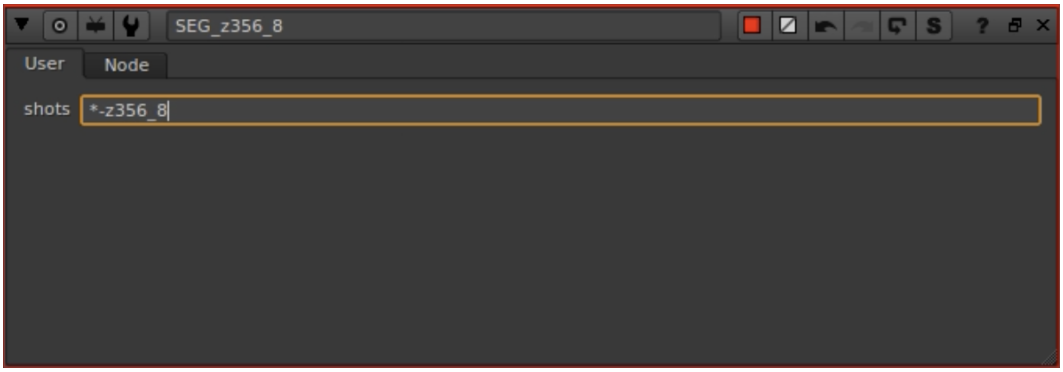


Figure 4

Between these two workflow nodes, artists were able to quickly make exceptions from the template graph, without having to leave the single-graph workflow. They served as the foundation for making packet-based (group of shots) or individual shot tweaks on the entire film.

5 Precomps

Not only did lighting artists work on entire sequences of shots in parallel, they also had to do so while multiple other departments were actively making changes to each of the individual shots in particular, the effects department leveraged compositing extensively to assemble final looks from individual elements. In these cases, the other departments needed to make adjustments to the compositing script without disrupting the lighting artists workflow. Attempting to lock and version the script file would be impractical, as multiple departments often needed to keep the files active and perform edits concurrently.

As a solution, a pre-comp workflow was put in place where other departments had their own dedicated compositing scripts. When rendering these scripts, the output would automatically be routed through the main compositing script, which gave artists from multiple departments the necessary control, while still providing necessary context to show what their work would look like in its final stage.

6 Nuke plugins

Deep compositing was leveraged heavily on the show in order to make spatially dependent adjustments to the image. Many elements were rendered as separate deep EXR layers and combined using Nukes deep compositing toolkit. Pixar also has a wide range of in-house deep compositing tools, from the SkyGen tool that was used to generate in-composite atmosphere and background skies, to two particularly prevalent tools called Deep Rod and Deep ID.

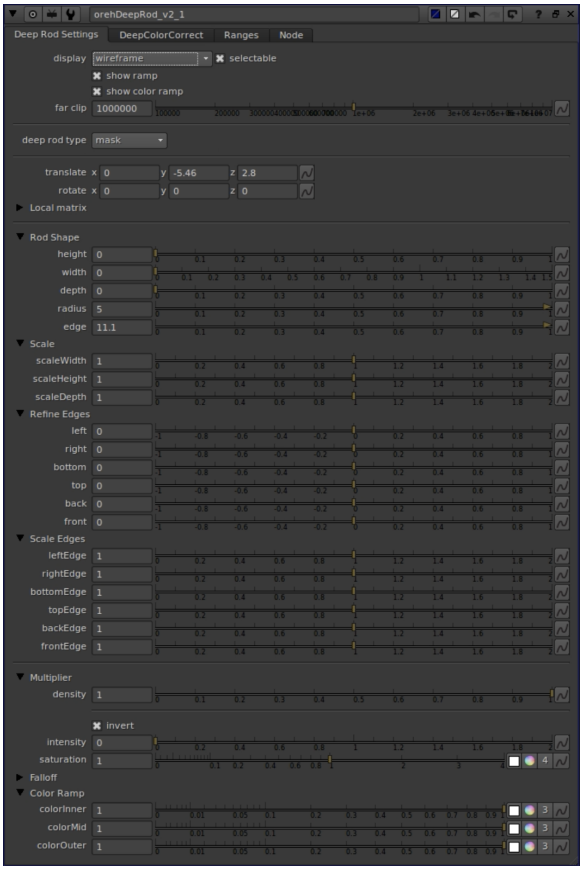


Figure 5

Rods are a common tool at Pixar for shaping lights, with multiple years of development and refinement. The same Rod concept was brought into the compositing workflow and exposed to the user as a set of parameters and a set of 3D interactive handles. Using the Deep Rod node, artists could quickly isolate areas in an image and make adjustments with detailed control over inner/outer regions and falloffs. Additionally, a USD importer made it quick and easy for artists to track these regions to moving geometry, such as a characters limb or a moving car. Figure5 shows the Deep Rod UI.

Figure 6: *Incredibles 2* ©Disney/Pixar 2018

The Deep ID node was originally developed at Industrial Light & Magic, and allows users to click-select geometry in an image and have an accurate, anti-aliased mask created instantly. Pixar augmented the node by tying the ID information in the image to the USD scene graph, making it possible for artists to perform pick walking, which lets a user select e.g. a characters eyeball, then go up-hierarchy to the head, again to the neck, to the entire body, and eventually up to the entire character scope. Regular expressions were also supported and could be matched to any part of the USD scope, for example `*Helen*`, `*crowd*` or `/world/geo/sets/*`. This made is straightforward to create mattes of even complex geometry, independently of how the individual gprims were organized at render-time. Figure6 shows the creation of a rod with falloff being used to isolate Dashs face.

7 Conclusion

In the end, the new compositing workflow and tools helped advance the way lighting was executed on the show. As an example, eye highlight refinement was traditionally a tedious process that required constant re-rendering to visualize the effect of the adjustments. In the new workflow, this was sped up considerably by utilizing material lobe AOVs and deep ID mattes, making refinement a fast and interactive process. Figure7 shows the Deep ID UI along with its in-viewer visualization.

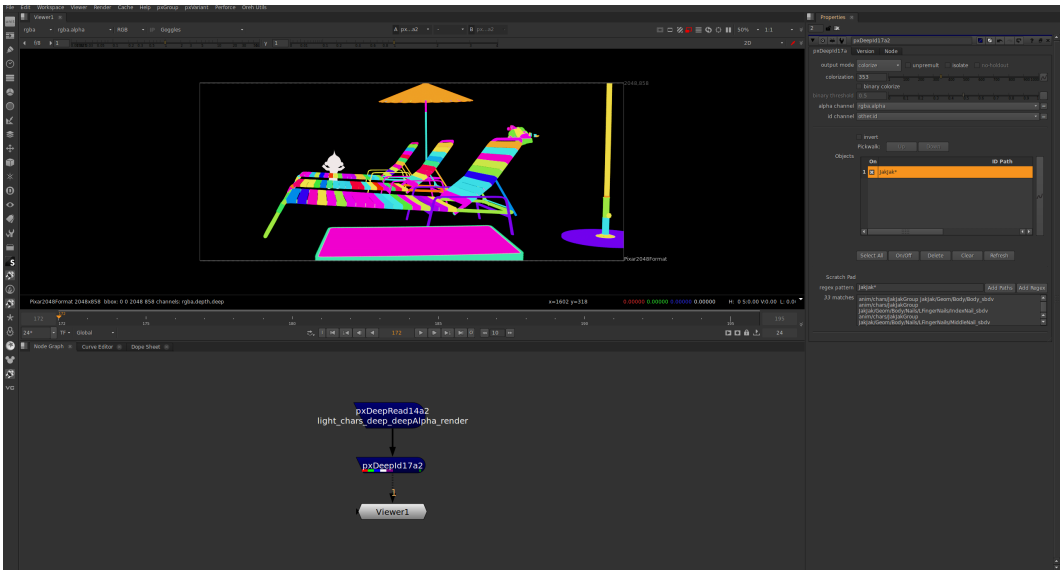


Figure 7

Real-time Tools for Film Production

by David G Yu

1 Introduction

The process of making an animated feature film involves massive amounts of data in a production pipeline being manipulated by many artists making creative decisions. In this section we will describe some of the tools that Pixar has developed to help organize and work with this production data. A particular focus is on expressive and efficient specification of the data needed to describe scenes and models in order to support robust interchange of assets across applications in the pipeline, and to support real-time feedback helping artist to work in context.

Specifically, we will present:

- Universal Scene Description (USD)
- Hydra Render Engine
- OpenSubdiv

2 Universal Scene Description (USD)

The data used to describe a 3D scene is refered to as scene description. Universal Scene Description (USD) has been designed with consideration of the complex requirements of film production and other digital content creation based on years of experience at Pixar and our other partners. USD is the foundation of the development of real-time tools at Pixar since it helps ensure that all of our 3D scene description can be represented fully and efficiently and robustly throughout our pipeline.

2.1 Composed Scene Description

One of the most challenging aspects of working with 3D scene description in a large production is determining how to support individuals and departments iterating to refine distinct parts of the scene while still preserving a coherent view of the whole. USD is a composed scene desceription format, meaning that the representation supports the expression of intent to compose different opinions in a way that can be resolved consistently in the context of the task at hand whether that task is working on an individual asset in isolation or working with many assets in a larger context.

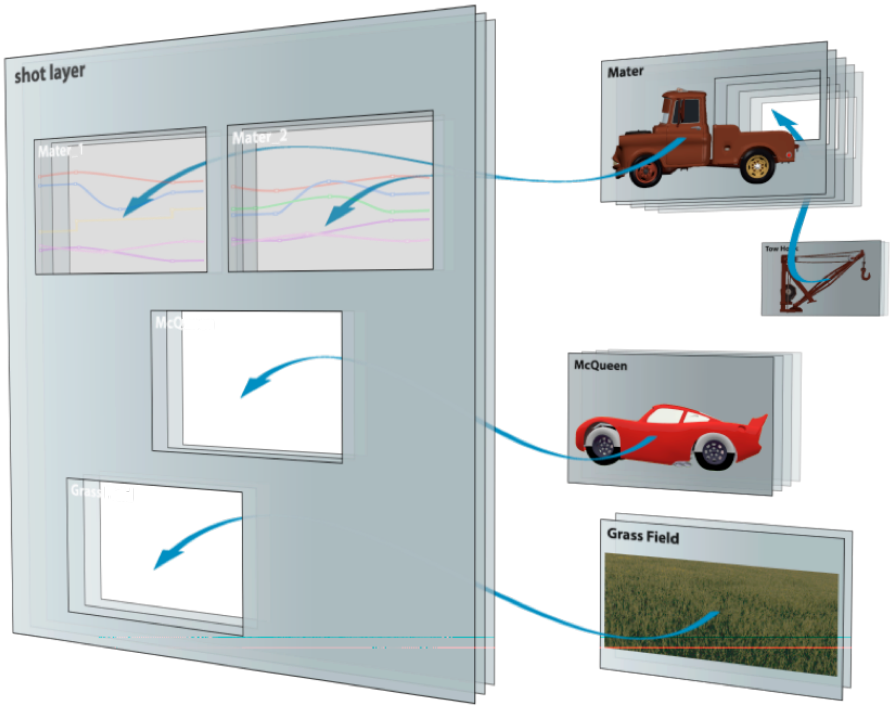


Figure 1: Composition of Scene Layers and References

2.2 Namespace, Prims, and Properties

USD organizes data into hierarchical namespaces of Prims (short for "primitive"). In addition to child prims, each prim can contain Attributes and Relationships, collectively known as Properties. Attributes have typed values that can vary over time; Relationships are multi-target "pointers" to other objects in a hierarchy, and USD takes care of remapping the targets automatically when referencing causes namespaces to change. Both prims and properties can also have (non-time-varying) metadata. Prims and their contents are organized into a file abstraction known as a Layer.

Built on top of this low-level, generic scene description, USD provides a set of schemas that establish a standard encoding and client API for common 3D CG concepts like:

- **Geometry**. The UsdGeom schemas define (OpenSubdiv-compliant) meshes, transforms, curves, points, nurbs patches, and several intrinsic solids. It also defines: the concept of arbitrary primvars as attributes that can interpolate across a geometric surface; geometric extents and aggregate, computed bounding boxes; pruning visibility; and an attribute called purpose that expresses a (non-animatable) conditional visibility useful for deploying level-of-detail proxies and guides.

- **Shading**. The UsdShade schemas define primitive shader nodes that can be connected into networks and packaged into materials, on which one can create a public interface of attributes that will drive parameters in the contained shader networks. Although the UsdShade schemas are used in the USD plugins for transmitting renderman shading from Maya to Katana, please be aware that these schemas are in flux until the 1.0 USD release.

- **Model** and **Asset** . USD's composition operators allow you to construct arbitrarily large, complex scenes. As an aid to processing, analyzing, and decomposing such scenes, USD formalizes the concepts of model and asset. The "model" prim classification allows scenegraphs to be

partitioned into logical, manageable chunks for traversal, working-set management, and data coalescing/caching. The concept of "asset" shows up in USD at two levels: as a core datatype for referring unambiguously to an external file, which identifies which data needs to participate in asset/path resolution; and in the AssetInfo schema for depositing a record of what assets have been referenced into a scene, which survives even if the scene is flattened.

2.3 References

For more references, please visit http://graphics.pixar.com/usd/

3 Hydra Render Engine

Hydra is Pixar's real-time rendering architecture. Hydra is designed to be able to complement the scalability of USD while also being flexible and extensible to support USD within the Digital Content Creation applications that are an important part of a production pipeline. Hydra is the renderer inside of the "usdview" viewport but also supports high performance rendering within other applications and rendering engines.

3.1 Hydra Architecture

The main role of a renderer is to render an image of a scene.

In order to accomplish this, the renderer must process the input scene description data and prepare it into the form that is needed by the rendering algorithm.

In an interactive context, this processing must be done for each new frame whenever the scene is updated. In order for this update to be done efficiently, it is useful to track changes to the scene data so that additional renderer preparation can be limited to only data which has changed.

The fundamental organization of the Hydra renderer architecture is to distinguish these separate responsibilites. The SceneDelegate is responsible for discovering scene objects affecting the rendered image and observing any changes to those objects, the RenderDelegate is responsible for fetching and preparing data for those scene objects for rendering, and the RenderIndex is the data structure which maintains the correspondence between scene object and the resulting rendering resource objects as well has providing an interface to record and track changes.

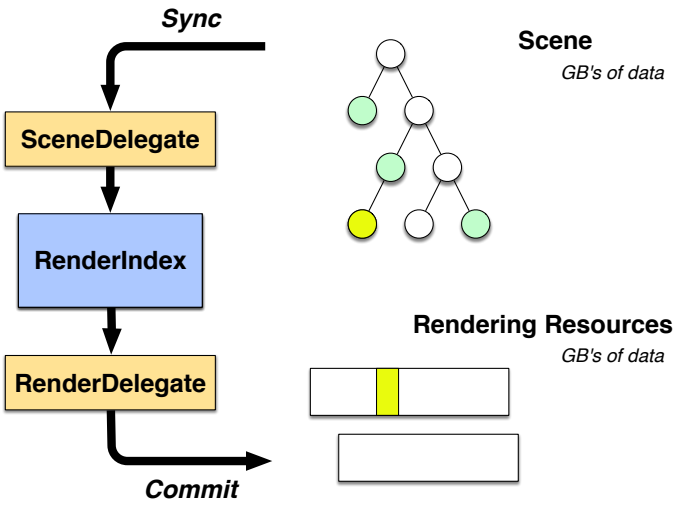


Figure 2: Hydra Architecture

In code the SceneDelegate and RenderDelegate are represented by abstract interfaces which can be implemented for specific scene description and rendering resource representations respectively. For instance, in order to use OpenGL to rasterize an image of a UsdStage, Hydra can be configured with a UsdImaging SceneDelegate and an OpenGL stream RenderDelegate.

One benefit of this organization is that the RenderIndex representation can serve as a common bridge between different SceneDelegate "front-end" implementations and different RenderDelegate "back-end" implementations.

If we think of the different SceneDelegate instances as "heads" and the different RenderDelegate instances as "tails" then we can see how our Hydra render engine (like the Hydra of Mythology) can have multiple heads and multiple tails.

This is not just an abstract concept and it is something that we use every day in our applications, e.g. our Presto Animation System can be configured to populate a RenderIndex from both native USD scene description and custom Presto rigging data and render to either a rasterized OpenGL viewport or a path traced RenderMan viewport.

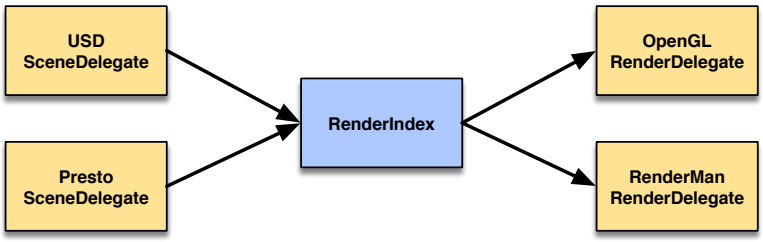


Figure 3: Hydra Heads and Tails

Pixar and others have implemented a growing set of SceneDelegate and RenderDelegate implementations, e.g. SceneDelegates which can read directly from Katana and Maya scene representations and RenderDelegates which can render using Metal, Vulkan, Embree, NVIDIA's Optix, and other path tracers.

An important emergent aspect of this is that it allows a way to organize hybrid rendering situations, e.g. to allow a full scene to be rendered in a single render pass even if the RenderIndex for that scene was populated by multiple different kinds of SceneDelegates.

3.2 Flattening and Instancing and Identity

As described in the section presenting USD, the input scene data is typically organized in a hierarchical name space in order to match the conceptual and practical requirements for organizing of scene data in a production context.

This is typically not the ideal representation for this data within a renderer, e.g. a GPU rasterizer may benefit from aggregating similar data together in order to vectorize execution of GPU shaders and draw calls while a path tracer may want to organize data spacialy in order to accellerate ray traversal.

We generalize this abstraction by saying that the RenderIndex is a "flattened" representation of the scene, e.g. it is possible to directly iterate through all of the geometry needed to render a frame without going back and walking through the whole scene graph name space (and reference hierarchy, etc).

It is important to note that this "flattened" representation can and should take advantage of efficiencies afforded by instancing and other data deduplication.

Each item populated into the RenderIndex has an identity, and Hydra uses an SdfPath to represent this identity. There is a correspondence between these paths an the actual scene paths in the input

scene description, and Hydra has optimized hierarchical namespace gather operations so that it is possible to quickly identify and operate on RenderIndex objects corresponding to subtrees in the input scene namespace. Hydra also augments this path namespace in order to identify special objects in the RenderIndex. But most operations treat these SdfPaths as simple identifies to support the benefits of "flattening".

An important quality of the RenderIndex is that it is not required to retain a copy of scene description data. If it is necessary to copy any bulky data from the input scene description to rendering resource data buffers then that should occur only when absolutely necessary as directed by the RenderDelegate implementation.

3.2.1 Rprim, Sprim, Bprim, Task

There are currently four kinds of objects that are populated into the RenderIndex. These base classes are futher classified into specific subclasses. The following class hierarchy is not exhaustive, and the system is designed to be extended.

- HdRprim – render prim corresponding to rendered geometry
  - HdBasisCurves – linear and parametric curves
  - HdMesh – polygonal and subdivision surface meshes
  - HdPoints – points
  - HdVolume – volumetric data
  - ...
- HdBprim – buffer prim corresponding to things like texture memory
  - HdTexture – UV, UDIM, Ptex image textures
  - HdRenderBuffer – rendered AOV attachment
  - HdField – volumetric data
  - ...
- HdSprim – state prim corresponding to state like lights and materials which affects rendering
  - HdCamera – camera
  - HdComputation – computation applied to data, e.g. skinning
  - HdLight – light
  - HdMaterial – GLSL or OSL material shading networks
  - HdField – volume data
  - ...
- HdTask – tasks which organized the rendering of a frame
  - HdxRenderTask – rendering passes
  - ...

3.3 Tasks and RenderPasses and Collections

The simplest case for a renderer is to render an image in a single pass consisting of all of the renderable items in the scene. Typically it is more complicated. For example, consider the case of supporting shadow maps in a forward rasterizer. In order to render the image, first a shadow map must be created by rendering only objects which should cast shadows into shadow buffer, then the a second pass can render only objects which should be visible and receive shadows. The situation can be even more complicated in interative applications considering additional passes that might be needed to support specific behaviors for rendering visual affordances like manipulators, guides, selection highlighting, etc.

Hydra supports both simple and more complex cases via a system of Tasks, RenderPasses, and Collections. Rendering execution proceeds by traversing a set of tasks. A task which produces rendered outputs can execute a RenderPass which operates on a portion of the items in the RenderIndex identified by a Collection.

4 OpenSubdiv

Subdivision surfaces are a fundamental geometric primitive at Pixar. Their expressive and economical representation is well suited to the characters and other models that we work with in film production.

With OpenSubdiv, Pixar provides a high performance and full-featured implementation of subdivision surface refinement, evaluation, and drawing.

As an open source project, OpenSubdiv provides a foundation for consistent representation of subdivision surface geometry across the industry including the real-time tools we use at Pixar.

4.1 Overview

As the name suggests, subdivision surfaces are fundamentally *surfaces*.

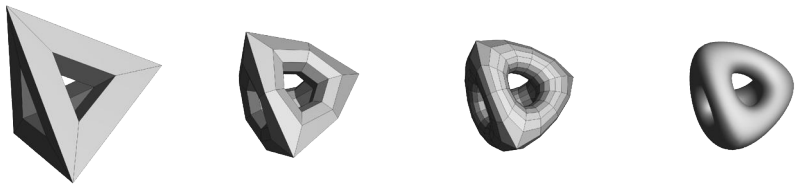


Figure 4: Subdivision Surface

More specifically, subdivision surfaces are *piecewise parametric surfaces* defined over meshes of *arbitrary topology* – both concepts that will be described in the sections that follow.

Subdivision is both an operation that can be applied to a polygonal mesh to refine it, and a mathematical tool that defines the underlying smooth surface to which repeated subdivision of the mesh converges. Explicit subdivision is simple to apply some number of times to provide a smoother mesh, and that simplicity has historically lead to many tools representing the shape this way. In contrast, deriving the smooth surface that ultimately defines the shape – its "limit surface" – is considerably more complex but provides greater accuracy and flexibility. These differences have led to confusion in how some tools expose subdivision surfaces.

The ultimate goal is to have all tools use subdivision surfaces as true surface primitives. The focus here is therefore less on subdivision and more on the nature of the surface that results from it. In addition to providing a consistent implementation of subdivision – one that includes a number of

widely used feature extensions – a significant value of OpenSubdiv is that it makes the limit surface more accessible.

Since its introduction, OpenSubdiv has received interest from users and developers with a wide variety of skills, interests and backgrounds. This document is intended to present subdivision surfaces from a perspective helpful in making use of OpenSubdiv. One purpose it serves is to provide a high level overview for those with less experience with the algorithms or mathematics of subdivision. The other is to provide an overview of the feature set available with OpenSubdiv, and to introduce those capabilities with the terminology used by OpenSubdiv (as much of it is overloaded).

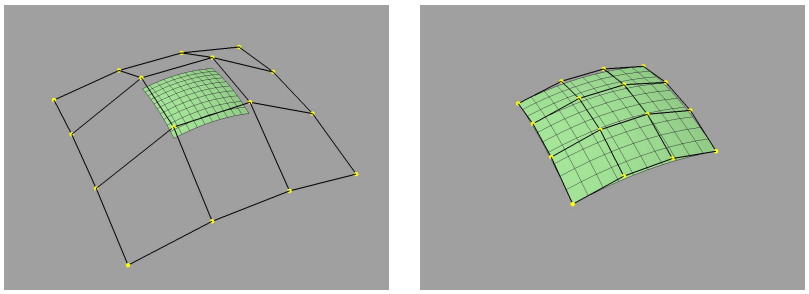
4.2 Piecewise Parametric Surfaces

Piecewise parametric surfaces are arguably the most widely used geometric representation in industrial design, entertainment and many other areas. Many of the objects we deal with everyday – cars, mobile phones, laptops – were all designed and visualized first as piecewise parametric surfaces before those designs were approved and pursued.

Piecewise parametric surfaces are ultimately just collections of simpler modeling primitives referred to as patches. Patches constitute the "pieces" of the larger surface in much the same way as a face or polygon constitutes a piece of a polygonal mesh.

4.2.1 Parametric Patches

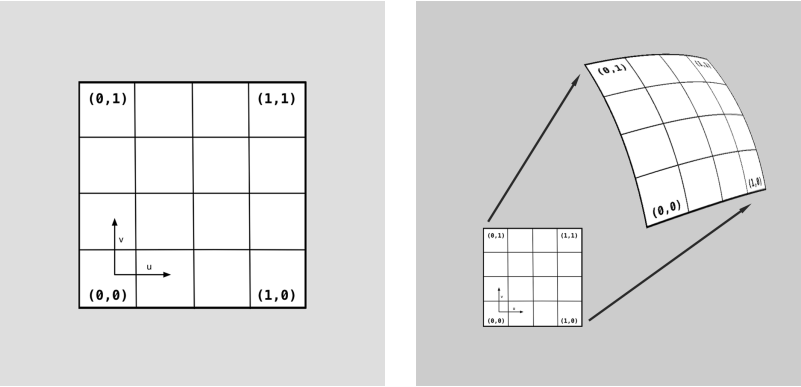
Patches are the building blocks of piecewise smooth surfaces, and many different kinds of patches have evolved to meet the needs of geometric modeling. Two of the more effective and common patches are illustrated below:



Single bicubic B-Spline patch Single bicubic Bezier patch

Figure 5: Parametric Patches

Patches consist of a set of points or vertices that affect a rectangular piece of smooth surface (triangular patches also exist). That rectangle is "parameterized" in its two directions, transforming a simple 2D rectangle into the 3D surface:



2D (u,v) domain of a patch Mapping from (u,v) to (x,y,z)

Figure 6: Mapping Parametric Patches

The points that control the shape of the surface are usually referred to as control points or control vertices, and the collection of the entire set defining a patch as the control mesh, the control hull, the control cage or simply the hull, the cage, etc. For the sake of brevity we will frequently use the term "cage", which serves us more generally later.

So a patch essentially consist of two entities: its control points and the surface affected by them.

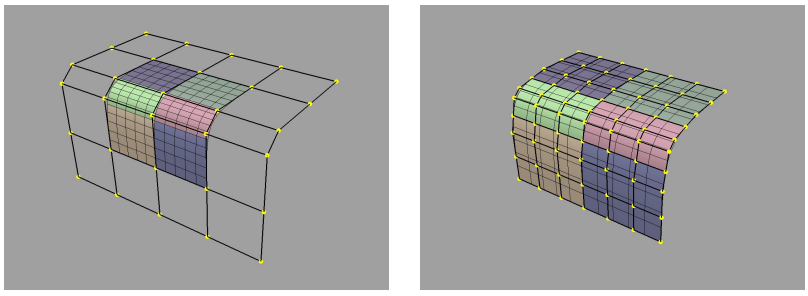
The way the control points affect the surface is what makes the different types of patches unique. Even patches defined by the same number of points can have different behavior. Note that all 16 points of the B-Spline patch above are relatively far from the surface they define compared to the similar Bezier patch. The two patches in that example actually represent exactly the same piece of surface – each with a set of control points having different effects on it. In mathematical terms, each control point has a "basis function" associated with it that affects the surface in a particular way when only that point is moved. It is these basis functions that often give rise to the names of the different patches.

There are pros and cons to these different properties of the control points of patches, which become more apparent as we assemble patches into piecewise surfaces.

4.2.2 Piecewise Surfaces

Piecewise parametric surfaces are collections of patches.

For rectangular patches, one of the simplest ways to construct a collection is to define a set of patches using a rectangular grid of control points:

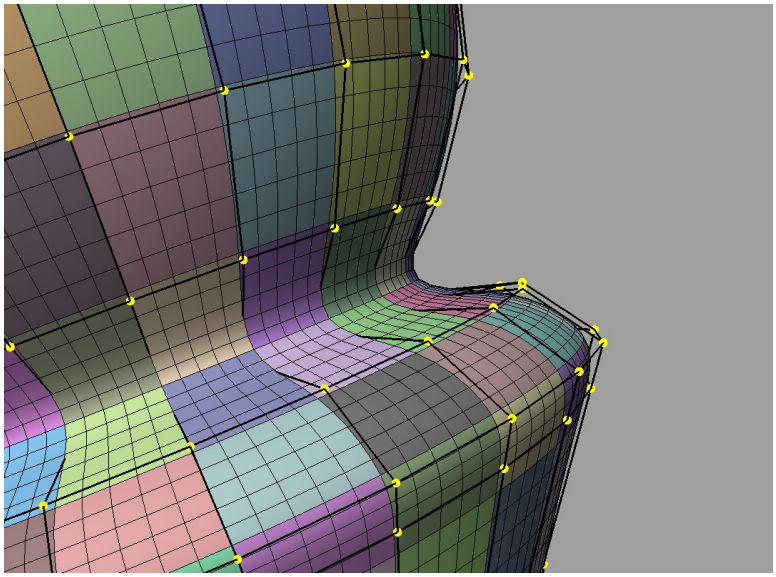


Piecewise B-Spline surface Piecewise Bezier surface

Note that we can overlap the points of adjacent B-spline patches. This overlapping means that moving one control point affects multiple patches – but it also ensures that those patches always meet

smoothly (this was a design intention and not true for other patch types). Adjacent Bezier patches only share points at their boundaries and coordinating the points across those boundaries to keep the surface smooth is possible, but awkward. This makes B-splines a more favorable surface representation for interactive modeling, but Bezier patches serve many other useful purposes.

A more complicated B-spline surface:



Part of a more complicated B-Spline surface

Just as a patch consisted of a cage and a surface, the same is now true of the collection. The control cage is manipulated by a designer and the surface of each of the patches involved is displayed so they can assess its effect.

4.3 Arbitrary Topology

Piecewise surfaces discussed thus far have been restricted to collections of patches over regular grids of control points. There is a certain simplicity with rectangular parametric surfaces that is appealing, but a surface representation that supports arbitrary topology has many other advantages.

Rectangular parametric surfaces gained widespread adoption despite their topological limitations, and their popularity continues today in some areas. Complex objects often need many such surfaces to represent them and a variety of techniques have evolved to assemble them effectively, including "stitching" multiple surfaces together or cutting holes into them ("trimming"). These are complicated techniques, and while effective in some contexts (e.g. industrial design) they become cumbersome in others (e.g. animation and visual effects).

A single polygonal mesh can represent shapes with far more complexity than a single rectangular piecewise surface, but its faceted nature eventually becomes a problem.

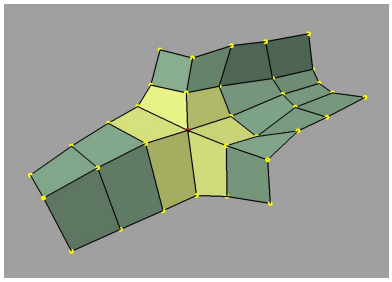
Subdivision surfaces combine the topological flexibility of polygonal meshes with the underlying smoothness of piecewise parametric surfaces. Just as rectangular piecewise parametric surfaces have a collection of control points (its cage stored as a grid) and an underlying surface, subdivision surfaces also have a collection of control points (its cage stored as a mesh) and an underlying surface (often referred as its "limit surface").

4.3.1 Regular versus Irregular Features

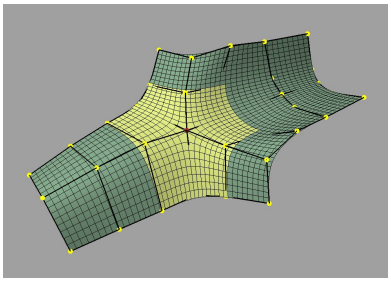
A mesh contains the vertices and faces that form the cage for the underlying surface, and the topology of that mesh can be arbitrarily complex.

In areas where the faces and vertices of the mesh are connected to form rectangular grids, the limit surface becomes one of the rectangular piecewise parametric surfaces previously mentioned. These regions of the mesh are said to be "regular": they provide behavior familiar from the use of similar rectangular surfaces and their limit surface is relatively simple to deal with. All other areas are considered "irregular": they provide the desired topological flexibility and so are less familiar (and less predictable in some cases) and their limit surface can be much more complicated.

Irregular features come in a number of forms. The most widely referred to is an extra-ordinary vertex, i.e. a vertex which, in the case of a quad subdivision scheme like Catmull-Clark, does not have four incident faces.



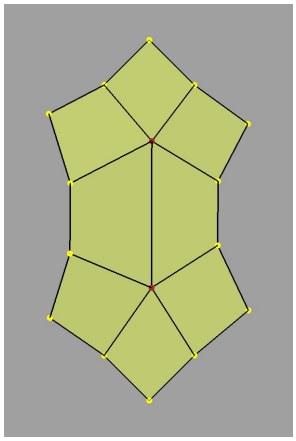
Irregular vertex and incident faces



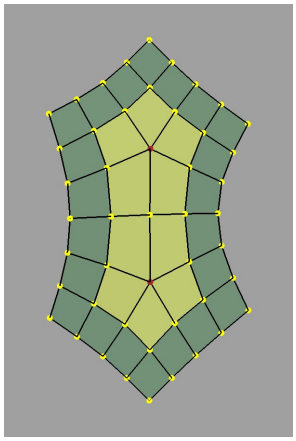
Regular and irregular regions of the surface

The presence of these irregular features makes the limit surface around them similarly irregular, i.e. it cannot be represented as simply as it can for regular regions.

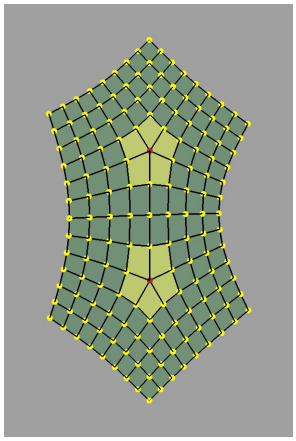
It's worth noting that irregular regions shrink in size and become more "isolated" as subdivision is applied. A face with a lot of extra-ordinary vertices around it makes for a very complicated surface, and isolating these features is a way to help deal with that complexity:



Two valence-5 vertices nearby



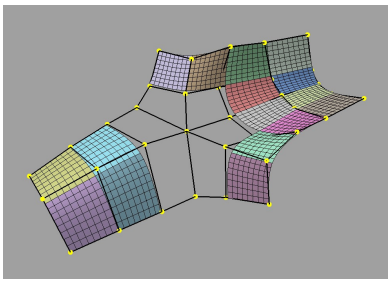
Isolation subdivided once



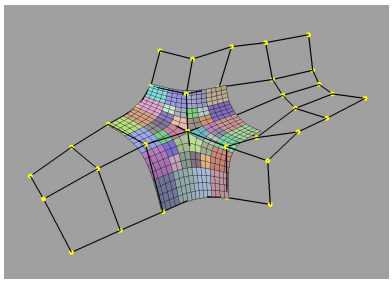
Isolation subdivided twice

It's generally necessary to perform some kind of local subdivision in these areas to break these pieces of surface into smaller, more manageable pieces, and the term "feature adaptive subdivision" has become popular in recent years to describe this process. Whether this is done explicitly or implicitly, globally or locally, what matters most is that there is an underlying piece of limit surface for each face

– albeit a potentially complicated one at an irregular feature – that can be evaluated in much the same way as rectangular piecewise surfaces.



Patches of the regular regions



Patches of the irregular region

While supporting a smooth surface in these irregular areas is the main advantage of subdivision surfaces, both the complexity of the resulting surfaces and their quality are reasons to use them with care. When the topology is largely irregular, there is a higher cost associated with its surface, so minimizing irregularities is advantageous. And in some cases the surface quality, i.e. the perceived smoothness, of the irregular surfaces can lead to undesirable artefacts.

An arbitrary polygonal mesh will often not make a good subdivision cage, regardless of how good that polygonal mesh appears.

As with rectangular piecewise parametric surfaces, the cage should be shaped to affect the underlying surface it is intended to represent. See ¡Modeling Tips¿ for related recommendations.

4.3.2 Non-manifold Topology

Since the cage of a subdivision surface is stored in a mesh, and often manipulated in the same context as polygonal meshes, the topic of manifold versus non-manifold topology warrants some attention.

There are many definitions or descriptions of what distinguishes a manifold mesh from one that is not. These range from concise but abstract mathematical definitions to sets of examples showing manifold and non-manifold meshes – all have their value and an appropriate audience. The following is not a strict definition but serves well to illustrate most local topological configurations that cause a mesh to be non-manifold.

As mentioned earlier, many tools do not support non-manifold meshes, and in some contexts, e.g. 3D printing, they should be strictly avoided. Sometimes a manifold mesh may be desired and enforced as an end result, but the mesh may temporarily become non-manifold due to a particular sequence of modeling operations.

Rather than supporting or advocating the use of non-manifold meshes, OpenSubdiv strives to be robust in the presence of non-manifold features to simplify the usage of its clients – sparing them the need for topological analysis to determine when OpenSubdiv can or cannot be used. Although subdivision rules are not as well standardized in areas where the mesh is not manifold, OpenSubdiv provides simple rules and a reasonable limit surface in most cases.

As with the case of regular versus irregular features, since every face has a corresponding piece of surface associated with it – whether locally manifold or not – the term "arbitrary topology" can be made to include non-manifold topology.

4.4 Subdivision versus Tessellation

The preceding sections illustrate subdivision surfaces as piecewise parametric surfaces of arbitrary topology. As piecewise parametric surfaces, they consist of a cage and the underlying surface defined

by that cage.

Two techniques used to display subdivision surfaces are subdivision and tessellation. Both have their legitimate uses, but there is an important distinction between them:

- *subdivision* operates on a *cage* and produces a refined *cage*
- *tessellation* operates on a *surface* and produces a discretization of that *surface*

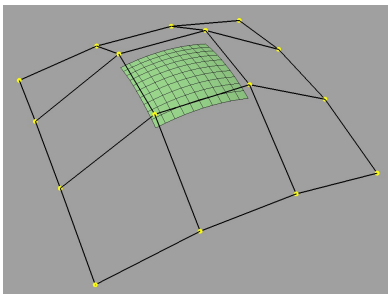
The existence and relative simplicity of the subdivision algorithm makes it easy to apply repeatedly to approximate the shape of the surface, but with the result being a refined cage, that approximation is not always very accurate. When compared to a cage refined to a different level, or a tessellation that uses points evaluated directly on the limit surface, the discrepancies can be confusing.

4.4.1 Subdivision

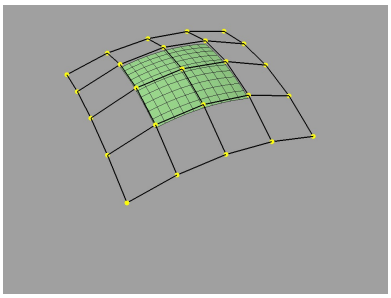
Subdivision is the process that gives "subdivision surfaces" their name, but it is not unique to them. Being piecewise parametric surfaces, let's first look at subdivision in the context of the simpler parametric patches that comprise them.

Subdivision is a special case of *refinement*, which is key to the success of some of the most widely used types of parametric patches and their aggregate surfaces. A surface can be "refined" when an algorithm exists such that more control points can be introduced *while keeping the shape of the surface exactly the same*. For interactive and design purposes, this allows a designer to introduce more resolution for finer control without introducing undesired side effects in the shape. For more analytical purposes, it allows the surface to be broken into pieces, often adaptively, while being faithful to the original shape.

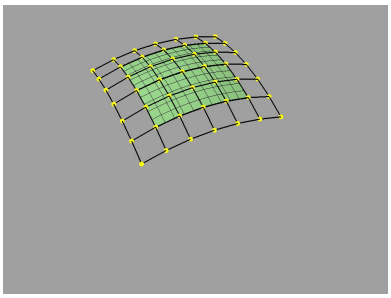
One reason why both B-spline and Bezier patches are so widely used is that both of them can be refined. Uniform subdivision – the process of splitting each of the patches in one or both of its directions – is a special case of refinement that both of these patch types support:



B-Spline surface and its cage



Cage subdivided 1x



Cage subdivided 2x

In the cases illustrated above for B-Splines, the uniformly refined cages produce the same limit surface as the original (granted in more pieces). So it is fair to say that both uniform B-splines and Bezier surfaces are subdivision surfaces.

The limit surface remains the same with the many more control points (roughly 4x with each iteration of subdivision), and those points are closer to (but not on) the surface. It may be tempting to use these new control points to represent the surface, but using the same number of points evaluated at corresponding uniformly spaced parametric locations on the surface is usually simpler and more effective.

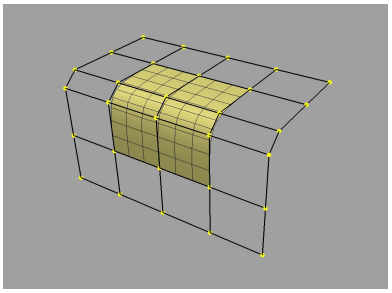
Note also that points of the cage typically do not have any normal vectors associated with them, though we can evaluate normals explicitly for arbitrary locations on the surface just as we do for

position. So if displaying a cage as a shaded surface, normal vectors at each of the control points must be contrived. Both the positions and normals of the points on the finer cage are therefore both approximations.

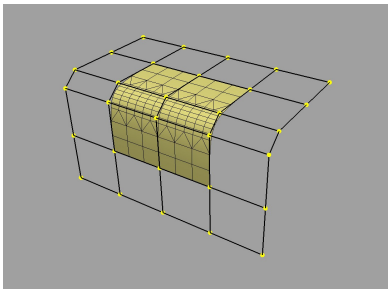
For more general subdivision surfaces, the same is true. Subdivision will refine a mesh of arbitrary topology, but the resulting points will not lie on the limit surface and any normal vectors contrived from and associated with these points will only be approximations to those of the limit surface.

4.4.2 Tessellation

There is little need to use subdivision to approximate a parametric surface when it can be computed directly, i.e. it can be tessellated. We can evaluate at arbitrary locations on the surface and connect the resulting points to form a tessellation – a discretization of the limit surface – that is far more flexible than the results achieved from uniform subdivision:



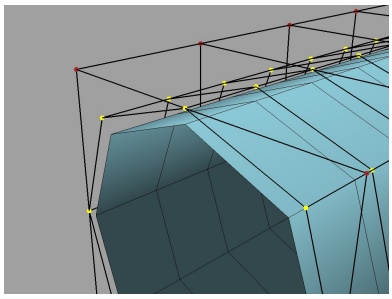
Uniform tessellation of B-Spline surface



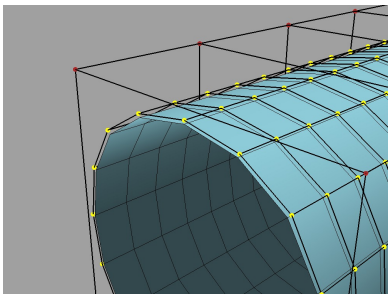
Curvature-adaptive tessellation of B-Spline surface

For a simple parametric surface, the direct evaluation of the limit surface is also simple, but for more complicated subdivision surfaces of arbitrary topology, this is less the case. The lack of a clear understanding of the relationship between the limit surface and the cage has historically lead to many applications avoiding tessellation.

It's worth mentioning that subdivision can be used to generate a tessellation even when the limit surface is not available for direct evaluation. The recursive nature of subdivision does give rise to formulae that allow a point on the limit surface to be computed that corresponds to each point of the cage. This process is often referred to as "snapping" or "pushing" the points of the cage onto the limit surface.



Subdivided 1x and snapped to limit surface



Subdivided 2x and snapped to limit surface

Since the end result is a connected set of points on the limit surface, this forms a tessellation of the limit surface, and we consider it a separate process to subdivision (though it does make use of it). The fact that such a tessellation might have been achieved using subdivision is indistinguishable from

the final result – the same tessellation might just as easily have been generated by evaluating limit patches of the cage uniformly 2x, 4x, 8x, etc. along each edge.

4.4.3 Which to Use?

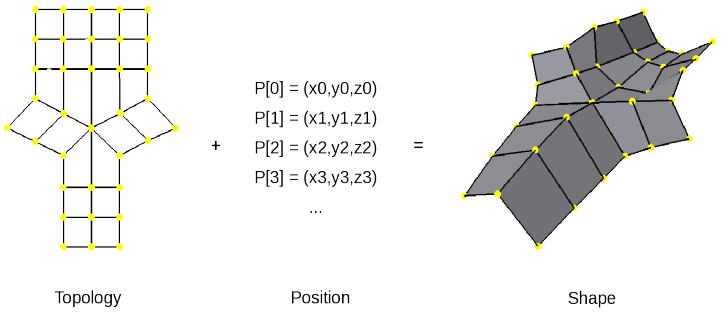
Subdivision is undeniably useful in creating finer cages to manipulate the surface, but tessellation is preferred for displaying the surface when the patches are available for direct evaluation. There was a time when global refinement was pursued in limited circles as a way of rapidly evaluating parametric surfaces along isoparametric lines, but patch evaluation, i.e. tessellation, generally prevails.

Considerable confusion has arisen due the way the two techniques have been employed and presented when displaying the shape in end-user applications. One can argue that if an application displays a representation of the surface that is satisfactory for its purposes, then it is not necessary to burden the user with additional terminology and choices. But when two representations of the same surface differ considerably between two applications, the lack of any explanation or control leads to confusion.

As long as applications make different choices on how to display the surface, we seek a balance between simplicity and control. Since subdivided points do not lie on the limit surface, it is important to make it clear to users when subdivision is being used instead of tessellation. This is particularly true in applications where the cage and the surface are displayed in the same style as there is no visual cue for users to make that distinction.

4.5 Mesh Data and Topology

The ability of subdivision surfaces to support arbitrary topology leads to the use of meshes to store both the topology of the cage and the data values associated with its control points, i.e. its vertices. The shape of a mesh, or the subdivision surface that results from it, is a combination of the topology of the mesh and the position data associated with its vertices.



When dealing with meshes there are advantages to separating the topology from the data, and this is even more important when dealing with subdivision surfaces. The "shape" referred to above is not just the shape of the mesh (the cage in this case) but could be the shape of a refined cage or the limit surface. By observing the roles that both the data and topology play in operations such as subdivision and evaluation, significant advantages can be gained by managing data, topology and the associated computations accordingly.

While the main purpose of subdivision surfaces is to use position data associated with the vertices to define a smooth, continuous limit surface, there are many cases where non-positional data is associated with a mesh. That data may often be interpolated smoothly like position, but often it is preferred to

interpolate it linearly or even make it discontinuous along edges of the mesh. Texture coordinates and color are common examples here.

Other than position, which is assigned to and associated with vertices, there are no constraints on how arbitrary data can or should be associated or interpolated. Texture coordinates, for example, can be assigned to create a completely smooth limit surface like the position, linearly interpolated across faces, or even made discontinuous between them. There are, however, consequences to consider – both in terms of data management and performance – which are described below as the terminology and techniques used to achieve each are defined.

4.5.1 Separating Data from Topology

While the topology of meshes used to store subdivision surfaces is arbitrarily complex and variable, the topology of the parametric patches that make up its limit surface are simple and fixed. Bicubic B-Spline and Bezier patches are both defined by a simple 4x4 grid of control points and a set of basis functions for each point that collectively form the resulting surface.

For such a patch, the position at a given parametric location is the result of the combination of position data associated with its control points and the weights of the corresponding basis functions (*weights* being the values of basis functions evaluated at a parametric location). The topology and the basis functions remain the same, so we can make use of the weights independent of the data. If the positions of the control points change, we can simply recombine the new position data with the weights that we just used and apply the same combination.

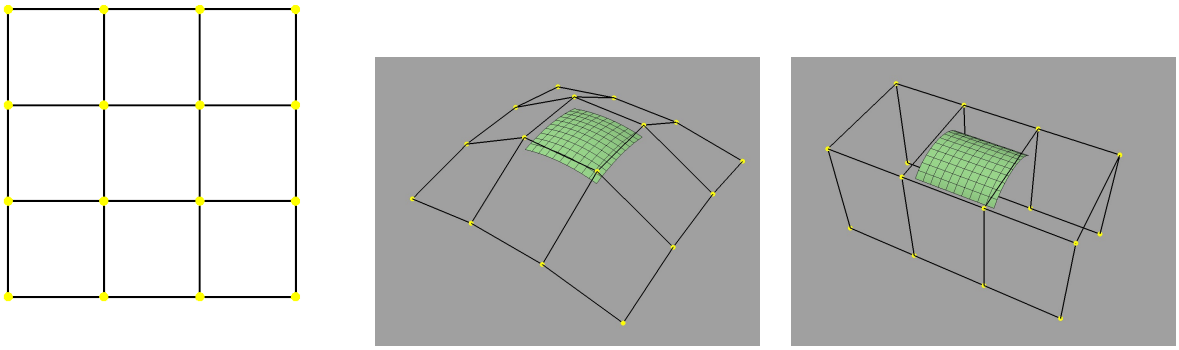


Figure 7: The fixed topology of a parametric patch and two shapes resulting from two sets of positions.

Similarly, for a piecewise surface, the position at a given parametric location is the result of the single patch containing that parametric location evaluated at the given position. The control points involved are the subset of control points associated with that particular patch. If the topology of the surface is fixed, so too is the topology of the collection of patches that comprise that surface. If the positions of those control points change, we can recombine the new position data with the same weights for the subset of points associated with the patch.

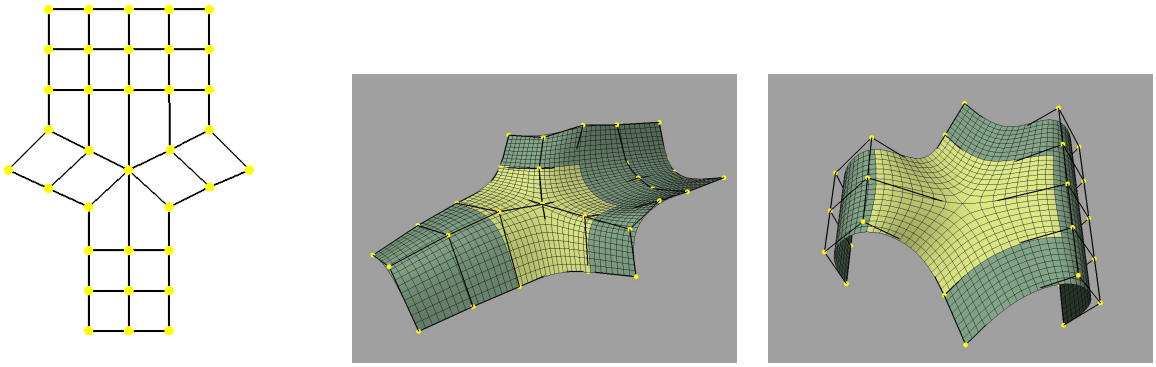


Figure 8: More complex but fixed topology of a surface and two shapes resulting from two sets of positions.

This holds for a piecewise surface of arbitrary topology. Regardless of how complex the topology, as long as it remains fixed (i.e. relationships between vertices, edges and faces does not change (or anything other settings affecting subdivision rules)), the same techniques apply.

This is just one example of the value of separating computations involving topology from those involving the data. Both subdivision and evaluation can be factored into steps involving topology (computing the weights) and combining the data separately.

When the topology is fixed, enormous savings are possible by pre-computing information associated with the topology and organizing the data associated with the control points in a way that can be efficiently combined with it. This is key to understanding some of the techniques used to process subdivision surfaces.

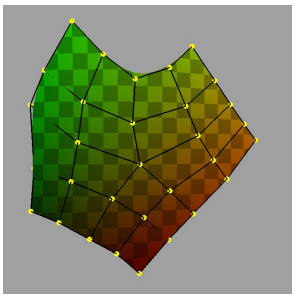
For a mesh of arbitrary topology, the control points of the underlying surface are the vertices, and position data associated with them is most familiar. But there is nothing that requires that the control points of a patch have to represent position – the same techniques apply regardless of the type of data involved.

4.5.2 Vertex and Varying Data

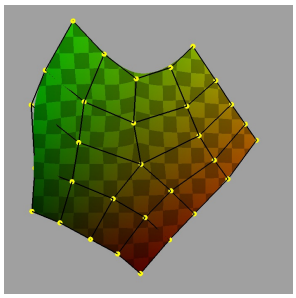
The most typical and fundamental operation is to evaluate a position on the surface, i.e. evaluate the underlying patches of the limit surface using the (x,y,z) positions at the vertices of the mesh. Given a parametric (u,v) location on one such patch, the data-independent evaluation method first computes the weights and then combines the (x,y,z) vertex positions resulting in an (x,y,z) position at that location. But the weights and their combination can be applied to any data at the vertices, e.g. color, texture coordinates or anything else.

Data associated with the vertices that is interpolated this way, including position, is said to be "vertex" data or to have "vertex" interpolation. Specifying other data as "vertex" data will result in it being smoothly interpolated in exactly the same way (using exactly the same weights) as the position. So to capture a simple 2D projection of the surface for texture coordinates, 2D values matching the (x,y) of the positions would be used.

If linear interpolation of data associated with vertices is desired instead, the data is said to be "varying" data or to have "varying" interpolation. Here the non-linear evaluation of the patches defining the smooth limit surface is ignored and weights for simple linear interpolation are used. This is a common choice for texture coordinates as evaluation of texture without the need of bicubic patches is computationally cheaper. The linear interpolation will not capture the smoothness required of a true projection between the vertices, but both vertex and varying interpolation have their uses.



Projected texture smoothly interpolatedfrom vertex data



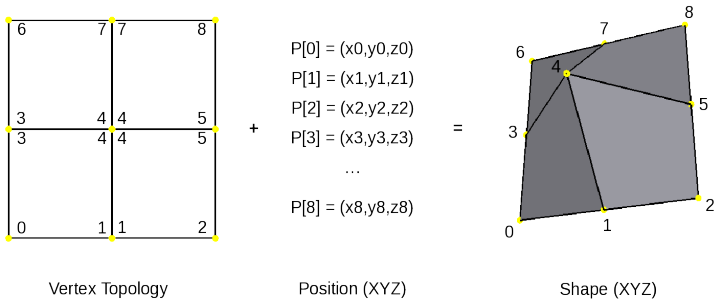
Projected texture linearly interpolated from varying data

Since both vertex and varying data is associated with vertices (a unique value assigned to each), the resulting surface will be continuous – piecewise smooth in the case of vertex data and piecewise linear in the case of varying.

4.5.3 Face-Varying Data and Topology

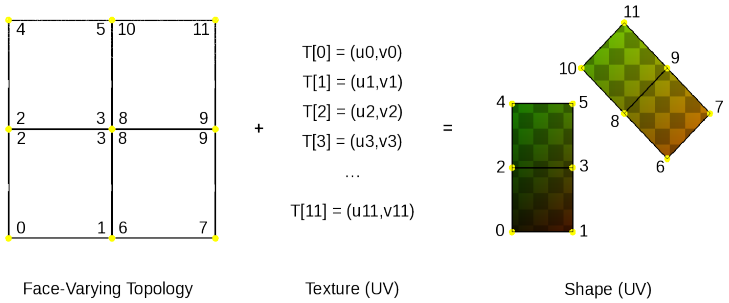
In order to support discontinuities in data on the surface, unlike vertex and varying data, there must be multiple values associated with vertices, edges and/or faces, in order for a discontinuity to exist.

Discontinuities are made possible by assigning values to the corners of faces, similar to the way in which vertices are assigned to the corners of faces when defining the topology of the mesh. Recalling the assignment of vertices to faces:

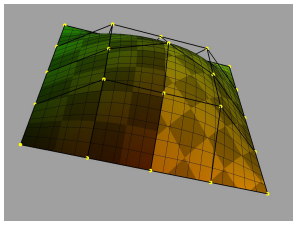


Vertex indices are assigned to all corners of each face as part of mesh construction and are often referred to as the face-vertices of an individual face or the mesh. All face-vertices that share the same vertex index will be connected by that vertex and share the same vertex data associated with it.

By assigning a different set of indices to the face-vertices – indices not referring to the vertices but some set of data to be associated with the corners of each face – corners that share the same vertex no longer need to share the same data value and the data can be made discontinuous between faces:



This method of associating data values with the face-vertices of the mesh is said to be assigning "face-varying" data for "face-varying" interpolation. An interpolated value will vary continuously within a face (i.e. the patch of the limit surface associated with the face) but not necessarily across the edges or vertices shared with adjacent faces.



Disjoint face-varying UV regions applied to the limit surface

The combination of associating data values not with the vertices (the control points) but the face corners, and the resulting data-dependent discontinuities that result, make this a considerably more complicated approach than vertex or varying. The added complexity of the data alone is reason to only use it when necessary, i.e. when discontinuities are desired and present.

Part of the complexity of dealing with face-varying data and interpolation is the way in which the interpolation behavior can be defined. Where the data is continuous, the interpolation can be specified to be as smooth as the underlying limit surface of vertex data or simply linear as achieved with varying data. Where the data is discontinuous – across interior edges and around vertices – the discontinuities create boundaries for the data, and partition the underlying surface into disjoint regions. The interpolation along these boundaries can also be specified as smooth or linear in a number of ways (many of which have a historical basis).

A more complete description of the different linear interpolation options with face-varying data and interpolation is given later. These options make it possible to treat the data as either vertex or varying, but with the added presence of discontinuities.

An essential point to remember with face-varying interpolation is that each set of data is free to have its own discontinuities – this leads to each data set having both unique topology and size.

The topology specified for a collection of face-varying data is referred to as a *channel* and is unique to face-varying interpolation. Unlike vertex and varying interpolation, which both associate a data value with a vertex, the number of values in a face-varying channel is not fixed by the number of vertices or faces. The number of indices assigned to the face-corners will be the same for all channels, but the number of unique values referred to by these indices may not. We can take advantage of the common mesh topology in areas where the data is continuous, but we lose some of those advantages around the

discontinuities. This results in the higher complexity and cost of a face-varying channel compared to vertex or varying data. If the topology for a channel is fixed, though, similar techniques can be applied to factor computation related to the topology so that changes to the data can be processed efficiently.

4.6 Schemes and Options

While previous sections have described subdivision surfaces in more general terms, this section describes a number of common variations (often referred to as *extensions* to the subdivision algorithms) and the ways that they are represented in OpenSubdiv.

The number and nature of the extensions here significantly complicate what are otherwise fairly simple subdivision algorithms. Historically applications have supported either a subset or have had varying implementations of the same feature. OpenSubdiv strives to provide a consistent and efficient implementation of this feature set.

Given the varying presentations of some of these features elsewhere, the naming chosen by OpenSubdiv is emphasized here.

4.6.1 Subdivision Schemes

OpenSubdiv provides two well known subdivision surface types – Catmull-Clark (often referred to more tersely as "Catmark") and Loop subdivision. Catmull-Clark is more widely used and suited to quad-dominant meshes, while Loop is preferred for purely triangulated meshes.

The many examples from previous sections have illustrated the more popular Catmull-Clark scheme. For an example of Loop:

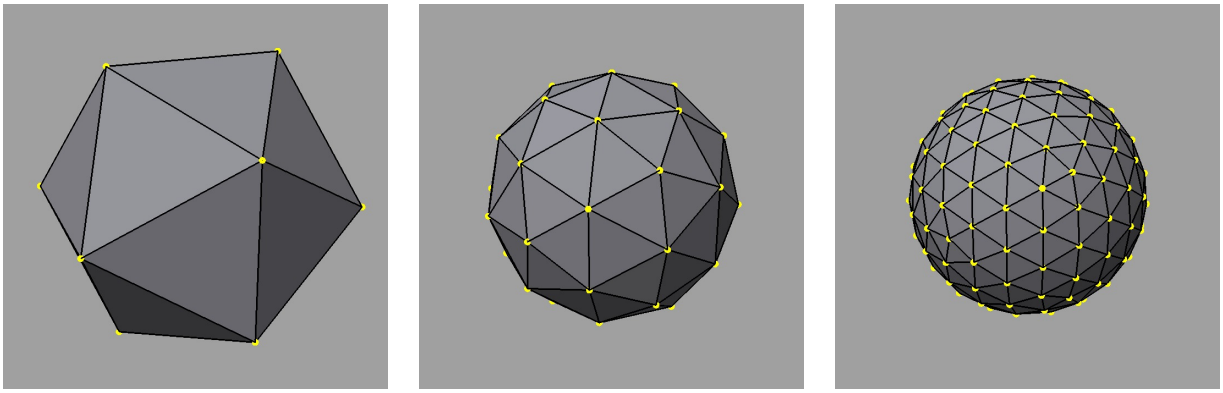


Figure 9: Loop scheme subdivision on a triangle mesh

Note that while Loop subdivision has long been available, support for the limit surface of Loop subdivision (i.e. arbitrary evaluation of the surface via patches) is not supported prior to version 3.4.

4.6.2 Boundary Interpolation

Boundary interpolation rules control how subdivision and the limit surface behave for faces adjacent to boundary edges and vertices.

The following choices are available:

Mode	Behavior
VTX_BOUNDARY_NONE	No boundary edge interpolation should occur; instead boundary faces are implicitly tagged as holes so that the boundary vertices continue to support the adjacent interior faces, but no surface corresponding to the boundary faces is generated
VTX_BOUNDARY_EDGE_ONLY	A sequence of boundary vertices defines a smooth curve to which the limit surface along boundary faces extends
VTX_BOUNDARY_EDGE_AND_CORNER	Similar to edge-only but the smooth curve resulting on the boundary is made to interpolate corner vertices (vertices with exactly one incident face)

On a grid example:

In practice, it is rare to use no boundary interpolation at all – this feature has its uses in allowing separate meshes to be seamlessly joined together by replicating the vertices along boundaries, but these uses are limited. Given the global nature of the setting, it is usually preferable to explicitly make the boundary faces holes in the areas where surfaces from separate meshes are joined.

The remaining "edge only" and "edge and corner" choices are then solely distinguished by whether or not the surface at corner vertices is smooth or sharp.

4.6.3 Face-varying Interpolation

Face-varying interpolation rules control how face-varying data is interpolated both in the interior of face-varying regions (smooth or linear) and at the boundaries where it is discontinuous (constrained to be linear or "pinned" in a number of ways). Where the topology is continuous and the interpolation chosen to be smooth, the behavior of face-varying interpolation will match that of the vertex interpolation.

Choices for face-varying interpolation are most commonly available in the context of UVs for texture coordinates and a number of names for such choices have evolved in different applications over the years. The choices offered by OpenSubdiv cover a wide range of popular applications. The feature is named face-varying *linear* interpolation – rather than *boundary* interpolation commonly used – to emphasize that it can be applied to the entire surface (not just boundaries) and that the effects are to make the surface behave more linearly in various ways.

The following choices are available for the `Sdc::Options::FVarLinearInterpolation` enum – the ordering here applying progressively more linear constraints:

Mode	Behavior
FVAR_LINEAR_NONE	smooth everywhere the mesh is smooth
FVAR_LINEAR_CORNERS_ONLY	linearly interpolate (sharpen or pin) corners only
FVAR_LINEAR_CORNERS_PLUS1	CORNERS_ONLY + sharpening of junctions of 3 or more regions
FVAR_LINEAR_CORNERS_PLUS2	CORNERS_PLUS1 + sharpening of darts and concave corners
FVAR_LINEAR_BOUNDARIES	linear interpolation along all boundary edges and corners
FVAR_LINEAR_ALL	linear interpolation everywhere (boundaries and interior)

These rules cannot make the interpolation of the face-varying data smoother than that of the vertices. The presence of sharp features of the mesh created by sharpness values, boundary interpolation rules, or the subdivision scheme itself (e.g. Bilinear) take precedence.

All face-varying interpolation modes illustrated in UV space using a simple 4x4 grid of quads segmented into three UV regions (their control point locations implied by interpolation):

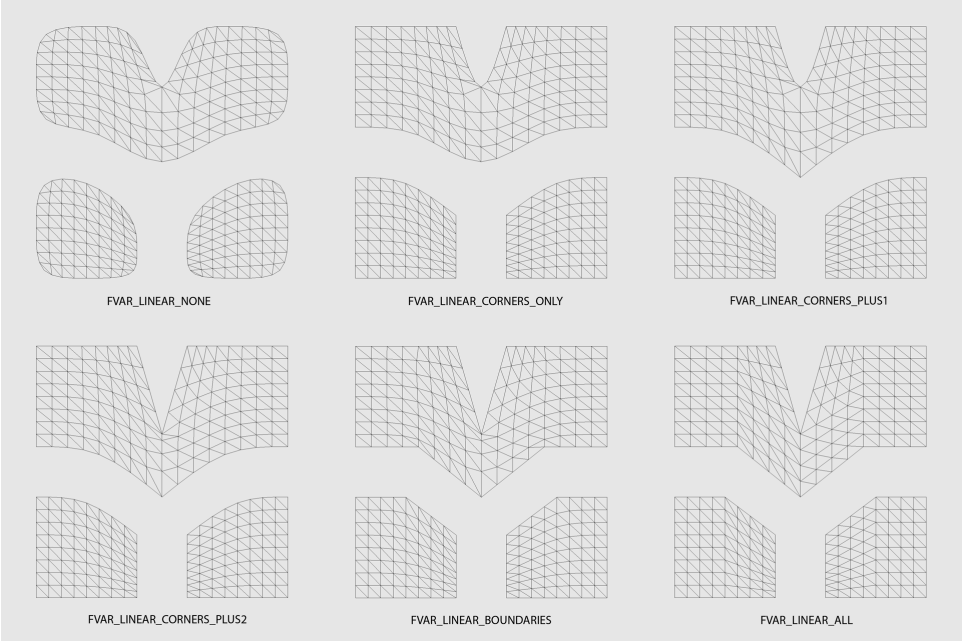


Figure 10: Face-varying interpolation modes

(For those familiar, this shape and its assigned UV sets are available for inspection in the "catmark_fvar_bound1" shape of OpenSubdiv's example and regression shapes.)

4.6.4 Semi-Sharp Creases

Just as some types of parametric surfaces support additional shaping controls to affect creasing along the boundaries between surface elements, OpenSubdiv provides additional sharpness values or "weights" associated with edges and vertices to achieve similar results over arbitrary topology.

Setting sharpness values to a maximum value (10 in this case – a number chosen for historical reasons) effectively modifies the subdivision rules so that the boundaries between the piecewise smooth surfaces are infinitely sharp or discontinuous.

But since real world surfaces never really have infinitely sharp edges, especially when viewed sufficiently close, it is often preferable to set the sharpness lower than this value, making the crease "semi-sharp". A constant weight value assigned to a sequence of edges connected edges therefore enables the creation of features akin to fillets and blends without adding extra rows of vertices (though that technique still has its merits):

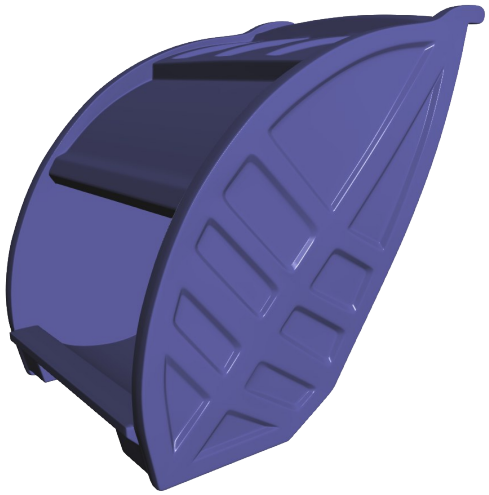


Figure 11: A shape modeled with semi-sharp creases

Sharpness values range from 0-10, with a value of 0 (or less) having no effect on the surface and a value of 10 (or more) making the feature completely sharp.

It should be noted that infinitely sharp creases are really tangent discontinuities in the surface, implying that the geometric normals are also discontinuous there. Therefore, displacing along the normal will likely tear apart the surface along the crease. If you really want to displace a surface at a crease, it may be better to make the crease semi-sharp.

4.6.5 Other Options

While the preceding options represent features available in a wide-variety of tools and modeling formats, a few others exist whose recognition and adoption is more limited. In some cases, they offer improvements to undesirable behavior of the subdivision algorithms, but their effects are less than ideal. Given both their limited effectiveness and lack of recognition, these options should be used with caution.

4.6.6 Chaikin Rule

The "Chaikin Rule" is a variation of the semi-sharp creasing method that attempts to improve the appearance of creases along a sequence of connected edges when the sharpness values differ. This choice modifies the subdivision of sharpness values using Chaikin's curve subdivision algorithm

to consider all sharpness values of edges around a common vertex when determining the sharpness of child edges. The creasing method can be set using the values defined in the enumeration `Sdc::Options::CreasingMethod`:

Table 3: Chaikin rule

Mode	Behavior
CREASE_UNIFORM	Apply regular semi-sharp crease rules
CREASE_CHAIKIN	Apply "Chaikin" semi-sharp crease rules

Example of contiguous semi-sharp creases interpolation:

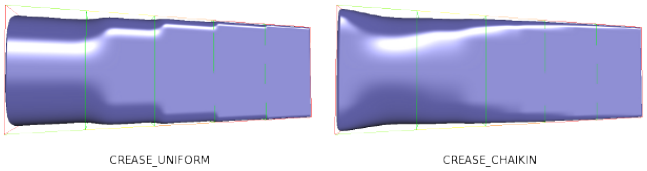


Figure 12: Chaikin rule example

4.6.7 Triangle Subdivision" Rule

The triangle subdivision rule is a rule added to the Catmull-Clark scheme that modifies the behavior at triangular faces to improve the undesirable surface artefacts that often result in such areas.

Table 4: "Triangle Subdivision" Rule

Mode	Behavior
TRI_SUB_CATMARK	Default Catmark scheme weights
TRI_SUB_SMOOTH	"Smooth triangle" weights

Cylinder example :

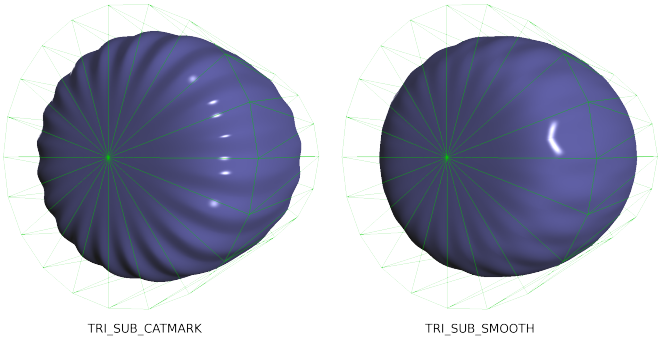


Figure 13: Triangle subdivision rule

This rule was empirically determined to make triangles subdivide more smoothly. However, this rule breaks the nice property that two separate meshes can be joined seamlessly by overlapping their boundaries; i.e. when there are triangles at either boundary, it is impossible to join the meshes seamlessly

4.7 API Overview

API Layers

OpenSubdiv is structured as a set of layered libraries. This structure facilitates operation on a variety of computing resources, and allows developers to only opt-in to the layers and feature sets that they require. From a top-down point of view, OpenSubdiv is comprised of several layers, some public, and some private.

Layers list:

Table 5: API Layer list

Sdc (Subdivision Core)	The lowest level layer, implements the core subdivision details to facilitate the generation of consistent results. Most cases will only require the use of simple public types and constants from Sdc.
Vtr (Vectorized Topological Representation)	A suite of classes to provide an intermediate representation of topology that supports efficient refinement. Vtr is intended for internal use only.
Far (Feature Adaptive Representation)	The central interface that processes client-supplied geometry and turns it into a serialized data representation ready for parallel processing in Osd. Far also provides a fully-featured single-threaded implementation of subdivision interpolation algorithms.
Osd (OpenSubdiv cross platform)	A suite of classes to provide parallel subdivision kernels and drawing utilities on a variety of platforms such as TBB, CUDA, OpenCL, GLSL and DirectX.

Client mesh data enters the API through the Far layer. Typically, results will be collected from the Osd layer. However, it is possible to use functionality from Far without introducing any dependency on Osd.

Although there are several entry-points to provide topology and primitive variable data to OpenSubdiv, eventually everything must pass through the private Vtr and Sdc representations for topological analysis.

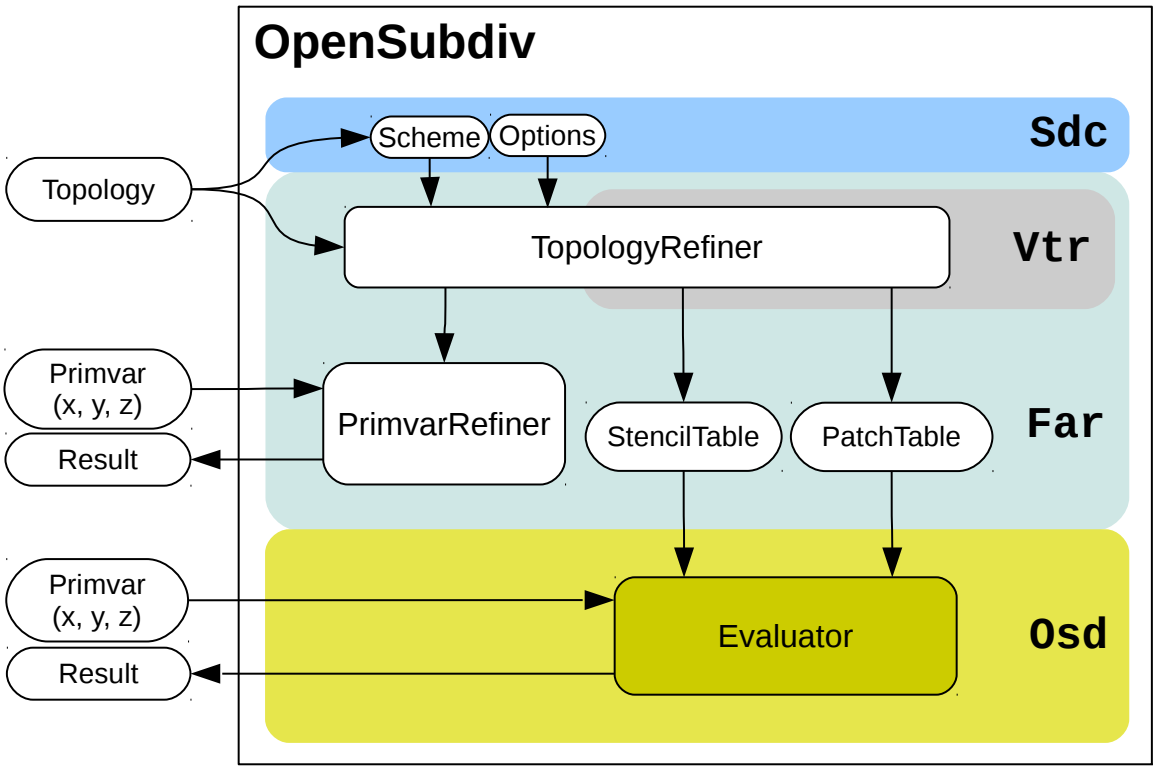


Figure 14: API layers

4.8 References

For more references, please visit http://graphics.pixar.com/opensubdiv/

References

Andrew Butts, Ben Porter, Dirk Van Gelder, Mark Hessler, Venkateswaran Krishna, and Gary Monheit. 2018. Engineering Full-fidelity Hair for Incredibles 2. In *ACM SIGGRAPH 2018 Talks (SIGGRAPH '18)*. ACM, New York, NY, USA, Article 14, 2 pages. https://doi.org/10.1145/3214745.3214798

Gordon Cameron, Robert Russ, and Adam Woodbury. 2007. Acting with Contact in Ratatouille: Cartoon Collision and Response. In *ACM SIGGRAPH 2007 Sketches (SIGGRAPH '07)*. ACM, New York, NY, USA, Article 64. https://doi.org/10.1145/1278780.1278858

Matt Jen-Yuan Chiang, Benedikt Bitterli, Chuck Tappan, and Brent Burley. 2015. A Practical and Controllable Hair and Fur Model for Production Path Tracing. In *ACM SIGGRAPH 2015 Talks (SIGGRAPH '15)*. ACM, New York, NY, USA, Article 23, 1 pages. https://doi.org/10.1145/2775280.2792559

Matt Jen-Yuan Chiang, Peter Kutz, and Brent Burley. 2016. Practical and Controllable Subsurface Scattering for Production Path Tracing. In *ACM SIGGRAPH 2016 Talks (SIGGRAPH '16)*. ACM, New York, NY, USA, Article 49, 2 pages. https://doi.org/10.1145/2897839.2927433

Per Christensen, Julian Fong, Jonathan Shade, Wayne Wooten, Brenden Schubert, Andrew Kensler, Stephen Friedman, Charlie Kilpatrick, Cliff Ramshaw, Marc Bannister, Brenton Rayner, Jonathan Brouillat, and Max Liani. 2018. RenderMan: An Advanced Path-Tracing Architecture for Movie Rendering. *ACM Trans. Graph.* 37, 3, Article 30 (Aug. 2018), 21 pages. https://doi.org/10.1145/3182162

Per H. Christensen. 2015. An Approximate Reflectance Profile for Efficient Subsurface Scattering. In *ACM SIGGRAPH 2015 Talks (SIGGRAPH '15)*. ACM, New York, NY, USA, Article 25, 1 pages. https://doi.org/10.1145/2775280.2792555

Robert L. Cook, Loren Carpenter, and Edwin Catmull. 1987. The Reyes Image Rendering Architecture. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. ACM, New York, NY, USA, 95–102. https://doi.org/10.1145/37401.37414

Anthony B. Davis and Feng Xu. 2014. A Generalized Linear Transport Model for Spatially-Correlated Stochastic Media. arXiv:physics.optics/1410.8200

Fernando De Goes and Doug L. James. 2018. Dynamic Kelvinlets: Secondary Motions Based on Fundamental Solutions of Elastodynamics. *ACM Trans. Graph.* 37, 4, Article 81 (July 2018), 10 pages. https://doi.org/10.1145/3197517.3201280

Fernando de Goes, William Sheffler, Michael Comet, Alonso Martinez, and Aimei Kutt. 2018. Patch-based Surface Relaxation. In *ACM SIGGRAPH 2018 Talks (SIGGRAPH '18)*. ACM, New York, NY, USA, Article 43, 2 pages. https://doi.org/10.1145/3214745.3214768

Eugene d'Eon, Guillaume Francois, Martin Hill, Joe Letteri, and Jean-Marie Aubry. 2011. An Energy-conserving Hair Reflectance Model. In *Proceedings of the Twenty-second Eurographics Conference on Rendering (EGSR '11)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1181–1187. https://doi.org/10.1111/j.1467-8659.2011.01976.x

Luca Fascione, Johannes Hanika, Rob Pieké, Ryusuke Villemin, Christophe Hery, Manuel Gamito, Luke Emrose, and André Mazzone. 2018. Path Tracing in Production. In *ACM SIGGRAPH 2018 Courses (SIGGRAPH '18)*. ACM, New York, NY, USA, Article 15, 79 pages. https://doi.org/10.1145/3214834.3214864

Julian Fong, Magnus Wrenninge, Christopher Kulla, and Ralf Habel. 2017. Production Volume Rendering: SIGGRAPH 2017 Course. In *ACM SIGGRAPH 2017 Courses (SIGGRAPH '17)*. ACM, New York, NY, USA, Article 2, 79 pages. https://doi.org/10.1145/3084873.3084907

L. G. Henyey and J. L. Greenstein. 1941. Diffuse radiation in the Galaxy. *Astrophysical Journal* 93 (1941), 70–83. https://doi.org/10.1086/144246

Christophe Hery. 2005. Implementing a Skin BSSRDF: (or Several...). In *ACM SIGGRAPH 2005 Courses (SIGGRAPH '05)*. ACM, New York, NY, USA, Article 4. https://doi.org/10.1145/1198555.1198584

Christophe Hery. 2012. Texture mapping for the Better Dipole model. *Pixar Technical Memo* (2012). http://graphics.pixar.com/library/TexturingBetterDipole/

Geoffrey Irving, Ryan Kautzman, Gordon Cameron, and Jiayi Chong. 2008. Simulating the Devolved: Finite Elements on WALL&Middot;E. In *ACM SIGGRAPH 2008 Talks (SIGGRAPH '08)*. ACM, New York, NY, USA, Article 54, 1 pages. https://doi.org/10.1145/1401032.1401102

Henrik Wann Jensen and Juan Buhler. 2002. A Rapid Hierarchical Rendering Technique for Translucent Materials. *ACM Trans. Graph.* 21, 3 (July 2002), 576–581. https://doi.org/10.1145/566654.566619

Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. 2001. A Practical Model for Subsurface Light Transport. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 511–518. https://doi.org/10.1145/383259.383319

J. T. Kajiya and T. L. Kay. 1989. Rendering Fur with Three Dimensional Textures. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '89)*. ACM, New York, NY, USA, 271–280. https://doi.org/10.1145/74333.74361

Ryan Kautzman, Gordon Cameron, and Theodore Kim. 2018. Robust Skin Simulation in Incredibles 2. In *ACM SIGGRAPH 2018 Talks (SIGGRAPH '18)*. ACM, New York, NY, USA, Article 50, 2 pages. https://doi.org/10.1145/3214745.3214793

Ryan Kautzman, Bill Wise, Meng Yu, Per Karlsson, Mark Hessler, and Audrey Wong. 2016. Finding Hank: Or How to Sim an Octopus. In *ACM SIGGRAPH 2016 Talks (SIGGRAPH '16)*. ACM, New York, NY, USA, Article 61, 2 pages. https://doi.org/10.1145/2897839.2927458

Alan King, Christopher Kulla, Alejandro Conty, and Marcos Fajardo. 2013. BSSRDF Importance Sampling. In *ACM SIGGRAPH 2013 Talks (SIGGRAPH '13)*. ACM, New York, NY, USA, Article 48, 1 pages. https://doi.org/10.1145/2504459.2504520

Aimei Kutt, Fran Kalal, Beth Albright, and Trent Crow. 2018. Collaborative Costume Design and Construction on Incredibles 2. In *ACM SIGGRAPH 2018 Talks (SIGGRAPH '18)*. ACM, New York, NY, USA, Article 5, 2 pages. https://doi.org/10.1145/3214745.3214790

Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. 2017. Spectral and Decomposition Tracking for Rendering Heterogeneous Volumes. *ACM Trans. Graph.* 36, 4, Article 111 (July 2017), 16 pages. https://doi.org/10.1145/3072959.3073665

Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. 2003. Light Scattering from Human Hair Fibers. In *ACM SIGGRAPH 2003 Papers (SIGGRAPH '03)*. ACM, New York, NY, USA, 780–791. https://doi.org/10.1145/1201775.882345

Jonathan T. Moon, Bruce Walter, and Steve Marschner. 2008. Efficient Multiple Scattering in Hair Using Spherical Harmonics. In *ACM SIGGRAPH 2008 Papers (SIGGRAPH '08)*. ACM, New York, NY, USA, Article 31, 7 pages. https://doi.org/10.1145/1399504.1360630

Laura Murphy, Martin Sebastian Senn, and Matthew Webb. 2018. Efficient hybrid volume and texture based clouds. In *ACM SIGGRAPH 2018 Talks*. ACM, 39.

Jan Novk, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. 2018. Monte Carlo Methods for Volumetric Light Transport Simulation. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)* 37, 2 (may 2018).

Leonid Pekelis, Christophe Hery, Ryusuke Villemin, and Junyi Ling. 2015. A Data-Driven Light Scattering Model for Hair. *Pixar Technical Memo* (2015). http://graphics.pixar.com/library/DataDrivenHairScattering/

Lena Petrovic, Mark Henne, and John Anderson. 2005. Volumetric Methods for Simulation and Rendering of Hair. *Pixar Technical Memo* (2005). http://graphics.pixar.com/library/Hair/

Iman Sadeghi, Heather Pritchett, Henrik Wann Jensen, and Rasmus Tamstorf. 2010. An Artist Friendly Hair Shading System. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. ACM, New York, NY, USA, Article 56, 10 pages. https://doi.org/10.1145/1833349.1778793

Kevin Singleton, Trent Crow, and Edgar Rodriguez. 2018. Making Mrs. Incredible More Flexible. In *ACM SIGGRAPH 2018 Talks (SIGGRAPH '18)*. ACM, New York, NY, USA, Article 49, 2 pages. https://doi.org/10.1145/3214745.3214785

Olivier Soares, Thomas Moser, and Frank Aalbers. 2016. Vegetation Choreography in the Good Dinosaur. In *ACM SIGGRAPH 2016 Talks (SIGGRAPH '16)*. ACM, New York, NY, USA, Article 19, 2 pages. https://doi.org/10.1145/2897839.2927435

Ryusuke Villemin, Christophe Hery, and Per Christensen. 2016. Importance Resampling for BSSRDF. *Pixar Technical Memo* (2016). http://graphics.pixar.com/library/Resampling/

Matthew Webb, Magnus Wrenninge, Jordan Rempel, and Cody Harrington. 2016. Making a dinosaur seem small: cloudscapes in The Good Dinosaur. In *ACM SIGGRAPH 2016 Talks*. ACM, 64.

Audrey Wong, David Eberle, and Theodore Kim. 2018. Clean Cloth Inputs: Removing Character Self-intersections with Volume Simulation. In *ACM SIGGRAPH 2018 Talks (SIGGRAPH '18)*. ACM, New York, NY, USA, Article 42, 2 pages. https://doi.org/10.1145/3214745.3214786

Magnus Wrenninge. 2016. Efficient Rendering of Volumetric Motion Blur Using Temporally Unstructured Volumes. *Journal of Computer Graphics Techniques (JCGT)* 5, 1 (31 January 2016), 1–34. http://jcgt.org/published/0005/01/01/

Magnus Wrenninge, Ryusuke Villemin, and Christophe Hery. 2017. Path Traced Subsurface Scattering using Anisotropic Phase Functions and Non-Exponential Free Flights. *Pixar Technical Memo* (2017). http://graphics.pixar.com/library/PathTracedSubsurface/

Arno Zinke, Cem Yuksel, Andreas Weber, and John Keyser. 2008. Dual Scattering Approximation for Fast Multiple Scattering in Hair. In *ACM SIGGRAPH 2008 Papers (SIGGRAPH '08)*. ACM, New York, NY, USA, Article 32, 10 pages. https://doi.org/10.1145/1399504.1360631