COURSE NOTES
SIGGRAPH 2014, VANCOUVER, CANADA

CHANGES IN THE WORKFLOW

The transition to raytracing at Sony Pictures Imageworks

PRESENTED BY:

Jesse Andrewartha
Sony Pictures Imageworks

Contents

1 Introduction

In 2006 Sony Pictures Imageworks (SPI) released *Monster House*, the first film from the studio to utilize a raytracing-only strategy with the Arnold renderer. The success of the rendering in the film spurred development and our raytracer, *SPI Arnold*, co-developed with Solid Angle SL, quickly became the standard renderer in the facility. The focus of this presentation is the fundamental change from the *REYES* paradigm to a raytracing toolset with the broad implications of this shift on the production environment.

Key to understanding the shift SPI experienced in practices, pipeline, tools and artistic approach is to understand the difference in how our rendering pipeline acquired a pixel value before and after the transition. The change is a complete shift in rendering philosophy; we move the work of illumination from the artist to the machine. This core shift affects everything from hardware considerations and disk space to communication with clients.

2 The Transition

In 2014, the use of raytracers is widespread throughout the industry. But when SPI first investigated the use of pure raytracing in 2004, it was the wild west. In fact, the idea to switch to a pure raytracing environment wasn not a foregone conclusion; in seeking a "claymation" style look for Monster House, SPI was charged with the task of coming up with the best approach. At the time, our pipeline was completely *REYES*, but a new renderer was being tested; a raytracer called *ARNOLD*.

Figure 1: An early test of raytracing in *Monster House*.

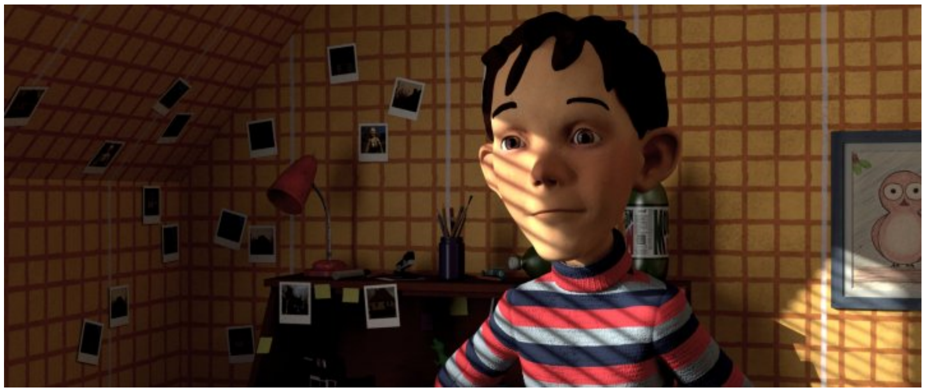


Image courtesy Columbia Pictures ©2006 Columbia Pictures Industries, Inc. All rights reserved.

One of our earliest tests, shown in Figure 1, left supervisors in no doubt about our direction. Lit with only three lights, it gave us a clear glimpse to the future. In 2004 Sony Pictures Imageworks licensed the source code to Arnold and entered into a partnership with *Solid Angle SL* founder, Marcos Fajardo, to co-develop it and adopt Arnold as Imageworks' main renderer. The new renderer was foundationally different: it was a path tracer.

The shift represented a whole new pipeline. Technically, the obstacles were daunting; we did not have any pipeline in place, there was no hair model implemented, the motion blur slowed performance so much as to be unusable in production. Even then, images had to be rendered 1K to get meaningful iterations in lighting. But even with such a limited palette, the imagery was compelling. The difference in how our established pipeline with the *REYES* renderer and Arnold evaluated a pixel value had implications for our entire workflow. (Figure 2) and (Figure 3) show the basic differences in the two approaches.

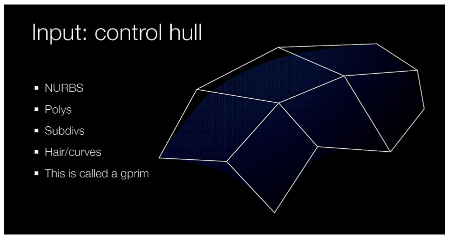
Moving forward meant implementing the new renderer and adjusting to a completely new set of changes:

Everything was brute force:
In the new raytracing architecture, everything was brute force, no tricks. This meant that for features such as motion blur, where we were used to 'smearing' objects, we were now sampling temporally between shutter open and shutter close. Technically it allowed for things that a *REYES* renderer could not do, like motion blur on shader functions that change the look of the surface over time or animating camera parameters. It was also more accurate. But it was also slower and had more artefacts which needed optimization.

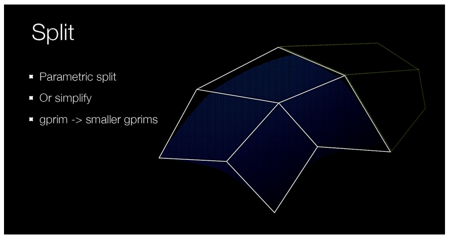
Shading sampling and antialiasing were one and the same:
Shading Rate was no longer applicable. In *REYES*, we were used to having separate control over the shading rate and the pixel samples. In Arnold, these were one and the same: `AA_samples`, which controlled both. This meant there was a lingering belief that Arnold was inherently 'softer' when rendering textures and shader detail. In fact, the change simply moved controls for textures to new parameters but there was a learning curve with this change that took years to overcome.

There was no longer any caching:
Shadowmaps, brickmaps, point clouds were no longer written out. It's important to recognize that we have at various times used these tools, but only until a pure raytracing method became available, at which time they were promptly and irrevocably deprecated. We did experiment with using shadowmaps on *Monster House* and *Cloudy With A Chance Of Meatballs*, but they were created and stored in memory, used and then discarded, so they were never persistent. While there were some initial performance improvements, the extra artist work to tweak and manipulate them was more expensive and their use was discontinued soon after. Other optimizations proved far more valuable than any saving realized via shadowmaps.

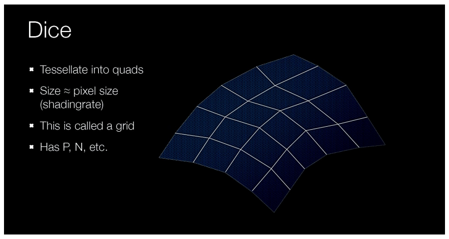
Figure 2: Computing the color of a pixel in a *REYES* renderer. *Images courtesy Larry Gritz.*



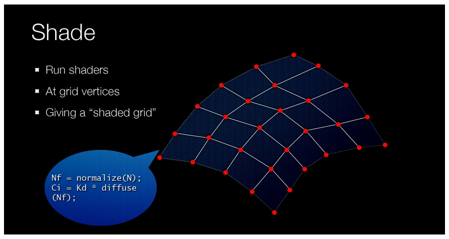
(a) Start with the input geometry



(b) The input geometry, the gprim, is simplified

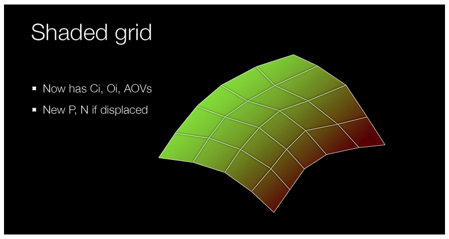


(c) The smaller gprim is subdivided to micropolygons (governed by shading rate)

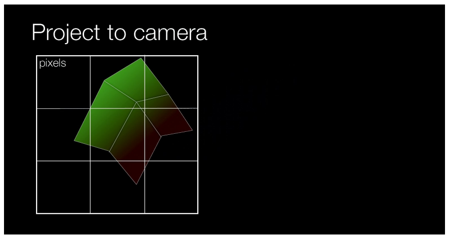


```
Nf = normalize(N);
Ci = Kd * diffuse
(Nf);
```

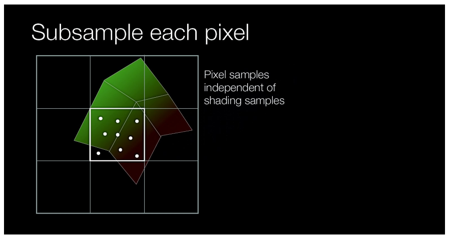
(d) The micropolygon surface is shaded



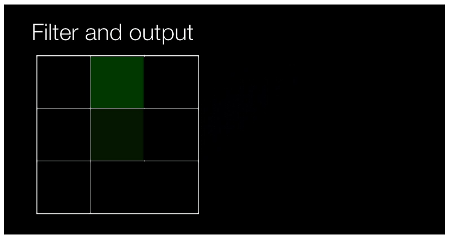
(e) Surface is displaced



(f) Camera position and view is calculated



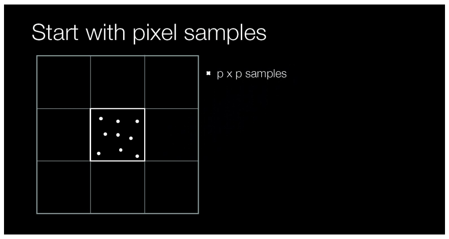
(g) Pixel sampling samples the shaded/displaced surface



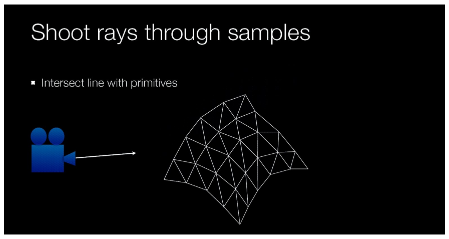
(h) The pixel samples are filtered and color value given to the pixel

No longer having to create and manage supporting map types was an enormous change. This had two effects:
(a) On one hand, we no longer had to reserve and manage these maps, which had a huge impact on I/O and resources. There was no longer any risk of losing files in a mistaken backup, there was no longer any problem with linking to the

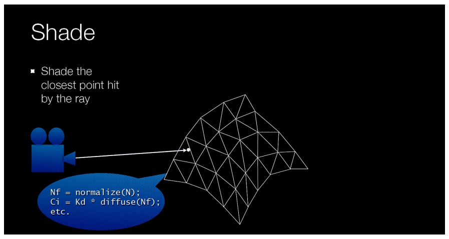
Figure 3: Computing the color of a pixel in a raytracer. *Images courtesy Larry Gritz.*



(a) Start with the camera & create sampling distribution

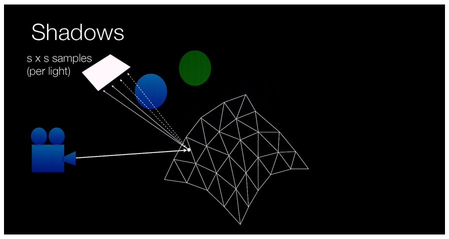


(b) Rays are fired along the pixel sample points

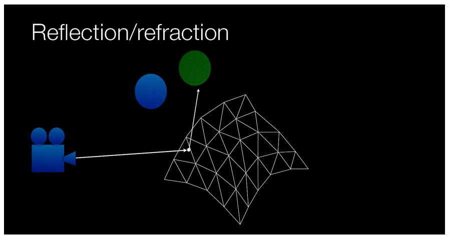


```
Nf = normalize(N);
Ci = Kd * diffuse(Nf);
etc.
```

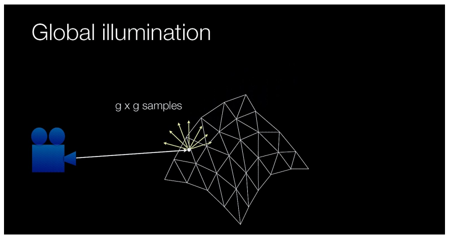
(c) Intersection with the closest mesh calls the surface shader



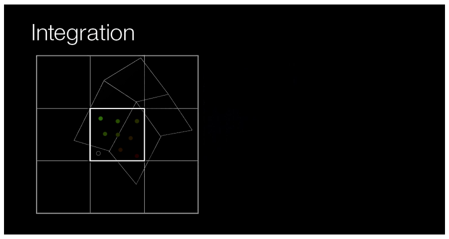
(d) Lights are sampled from the intersection point



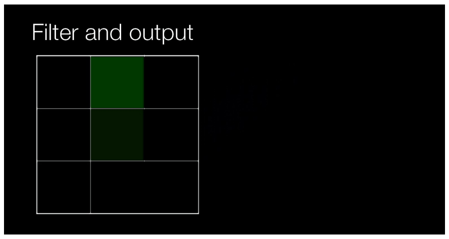
(e) Reflections and refractions are calculated with glossy/reflection rays



(f) Global Illumination is calculated with secondary rays



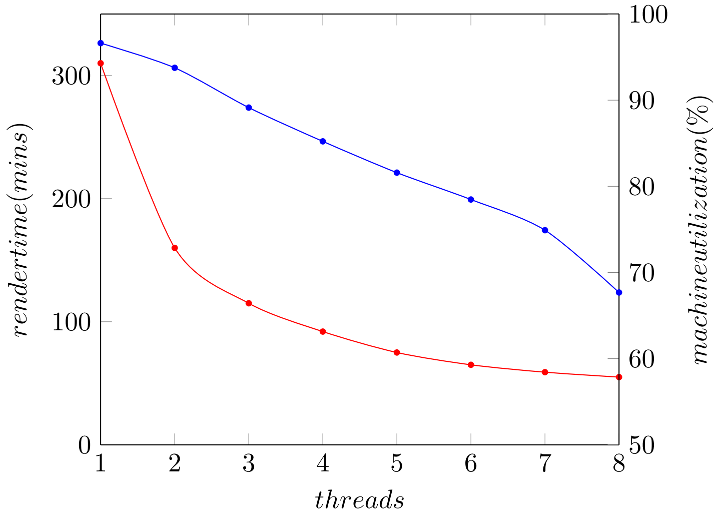
(g) All samples within a pixel integrate the color result of all surface, lighting and GI contributions



(h) The integrated samples are filtered and color value given to the pixel

wrong file version and no need to tweak and manage those files in the scene. (b) On the flipside, it meant that everytime a shot had to be rendered, we now faced a complete re-render. Ultimately, in terms of consistency and management, the benefits of removing these passes meant that the downside of having to re-render was seen as an acceptable loss.

Figure 4: The impact of multithreading on render time (red) and machine utilization (blue) Note that machine utilization decreases with more cores; during single threaded processes the balance of cores are idle. The more cores, the more core-time that is spend idle and the lower the utilization.



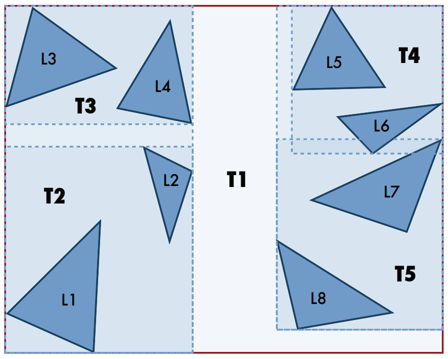
***Tiling* was gone:**
When faced with a complex scene in our *REYES* pipeline, it was possible to split it vertically or horizontally and render each "tile" on a different CPU. This worked because each CPU only had to load the geometry behind each tile. In a raytacing pipeline, this no longer worked because all geomtry had to be loaded into memory regardless of its location in the scene. Fortunately, Arnold's multithreaded performance is excellent between one to eight cores, scaling nearly 8x on 8 cores, meaning that we can get high performance from just one machine (Figure 4).

***Micropolygons* were no longer applicable:**
The largest impact of this was in displacement and subdivision. Because shading occurs AFTER hiding, the raytracer must displace first via subdivision; as a result, micropolygon displacement is not possible. Only subdivision iterations can govern the appearance of displacement. This means that we are sometimes forced to subdivide objects into tens of millions of triangles. Arnold has an impressive capability to handle massive amounts of geometry, but it is still a bottleneck. Interestingly, because Arnold was not micropolygonizing the mesh, moving the cv's one unit or a hundred units was basically the same cost, since you already subdivided as much as you were going to prior to displacing. This

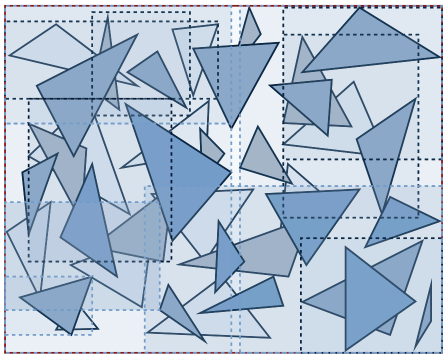
allows for “modeling with displacement”, something that was typically discouraged when working with a *REYES* renderer.

***Acceleration structures* were different:**
One side effect of raytracing was that partitioning the meshes changes significantly. In our earlier *REYES* architecture, we essentially had a Z-depth hider, in which the foremost geometry was retained but anything not visible was dumped from memory and furthermore, any back faces were discarded. We cannot do this in a raytracer.

Figure 5: Acceleration using BVH trees



(a) In a simple scene, the triangles are easy to partition.



(b) In a complex scene, with many overlapping triangles, the scene can become impossible to partition.

When a ray is fired into the scene, we must first determine where that ray intersects geometry. This task is expensive. When Whitted presented his landmark paper in 1980, most of the time in his images involved ray intersections; for complex scenes, the time spent on intersections between rays and objects approached 95%! [2] Clearly, we need a method to make these computations as efficient as possible. We use accelerations structures to optimize this process. The concept of acceleration structures is to quickly determine whether a ray can possibly hit a particular mesh in the scene in order to avoid expensive ray/primitive intersection tests.

Arnold uses BVH acceleration structures based on the paper by Kajiya and Kay.[1]. As illustrated in Figure 5, a scene is partitioned into a pyramid of smaller and smaller bounding boxes, each time checking whether the ray intersected any of the bounding boxes computed. In this way, whole sections of geometry can be quickly skipped without performing a linear check of all the primitives in a mesh or scene. Normally, this methodology can substantially cut down on the number ray/primitive intersections performed, but there are certain cases which can pose significant problems. Effects passes, such as particle systems or hair, can create conditions where all of a sudden, we have huge numbers of triangles overlapping. In worst case scenarios, the intersections can become so slow as to make the frame unrenderable, though these occurrances are rare.

Ray tracer from ground up:
In the *REYES* architecture, any raytracing required calls to a separate ray tracer and point clouds/support maps. This system is difficult to manage and requires maintenance of two separate renderering pipelines, complicating the task of maintenance for the software engineers. Furthermore, it retains the same system of support maps and caching. This increases complexity and introduced weaknesses. Adopting a ground-up raytracer, we are now maintaining only one renderer. This frees the renderer from dependencies and having to maintain the hooks into any parallel system. In this way, the system remains efficient and fast and permits much more sweeping optimization and development.

Physically accurate, one pass, no tricks:
No biasing methods were required to achieve sophisticated lighting and no support passes were necessary. This significantly reduced resource management and artist time in tweaking.

Complex lighting phenomena come standard:
In converting to the new pipeline, dozens of parameters were reduced to a few parameters that are interdependent with other parameters and affect them appropriately. This dramatically simplified the user experience. Initially, the response was negative; artists were used to control over every aspect of the lighting phenomena, but the benefit in consistency, complexity and sophistication in lighting lead to acceptance. The flipside is that this simplified system also requires more planning from the look developer. It goes beyond simply compositing a specular highlight and involves considerations of the properties of the surface itself.

A whole new language had to be learned:
Finally, along with the change came a whole new language and a new way of communication had to be developed around raytracing. Gone was *shading rate*, *shadowmaps*, *biasing*, *dso's*. We now had to contend with *Camera rays*, *Diffuse rays*, *Shadow rays*, *Glossy rays*, *Reflection rays*, *Refraction rays*, *Ray depth*. This new lexicon of raytracing had to be adopted and every artist had to become fluent in it. This lead to a new training curriculum.

3 Early Days

The success of rendering *Monster House* spurred development, but the obstacles were not simply technical, they were psychological. On one hand, there was the change in the actual renderer which had proven itself on a full CG feature.

To those that had participated, the advantages were obvious. However, to productions already on a tight schedule, it was risky. *REYES* renderers had the full backing of decades of user experience; not only were the pipelines mature, but also technicians and artists were readily available. Most CG supervisors had risen through the ranks knowing how to get things done with the *REYES* architecture and to move to an unkown system was a leap of faith few were willing to take.

Between 2006 and 2007, several productions used Arnold for selected elements; films such as *Beowulf* used Arnold as a utility renderer. *Don't Mess With The Zohan* used Arnold for the "cat hacky-sack" scene (and was SPI's first use of hair in Arnold), *G-Force* used Arnold for it's ability to handle the massive geometric complexity of the"Clusterstorm" robot and *Valkyrie* for some weaponry in battle sequences. But it was *Eagle Eye* was one of the first visual effects films at SPI to take up Arnold as its primary renderer.

Figure 6: A digital set extension rendered in Arnold for *Eagle Eye*.

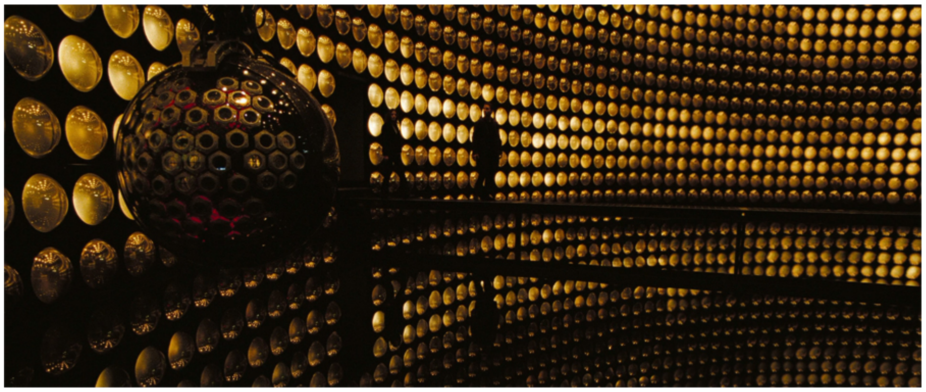


Image courtesy of DreamWorks Pictures ©2008 DreamWorks SKG. All rights reserved.

By that time, motion blur had been vastly improved, rendering speed had progressed such that 2K frames were possible within a regular dailies schedule and the shading system had improved to where truly photorealistic rendering was feasible. (Figure 6). By the time *Alice in Wonderland* and *2012* were evaluating rendering options, there was a broader discussion at facility level on SPI's philosophy and future. After exeriencing a pure raytracing pipeline with *Cloudy With A Chance Of Meatballs*, Rob Bredow, Imageworks' CTO, made clear the push for more photorealistic imagery out of the gate. The motivation was three-fold:

> **Consistency across shots:** When you are lighting with full raytracing, you are relying less on shot specific lighting to achieve the look. With careful look development, an environment can be lit and the template shared for all artists within a given sequence with minimal relighting on a per-shot basis.

Artist efficiency With no shadow maps, brick maps or point clouds artists are free to light the scene and concentrate on the look, rather than supporting passes. This effectively moves the work from the artist to the computer.

Interface simplification: The standard model for *REYES* shaders up until that time had been the 'uber shader'; these models relied on artist input for almost all aspect of the shading model. As a result, the UI to such models often had over 350 separate parameters that had to be managed and tweaked. By contrast, in our raytracer, renderer development moved toward handling all ray decisions. This meant that control no longer rested with the materials and simplified the interface; shaders in the current set of materials have less than 50 parameters.

Until this time, SPI and Solid Angle SL had been working together on Arnold. But with the development and implementation of Open Shading Language into SPI's version in 2010, the branches split. Because SPI's development of Arnold from that point forward occurred parallel but separately from *Solid Angle SL* Arnold from that juncture is described as *SPI's Arnold. Alice in Wonderland* marked the first show that instituted *SPI's Arnold* as the official facility renderer; every show subsequent to *Alice* has used *SPI's Arnold* without a requisite "renderer bake-off" during pre-production.

The last of the pre-physical lights and shader-directed sampling have been completely removed from the *SPI Arnold* source code; it is now all OSL with closures, correct units and completely physicaly-based BSDFs and lights.

The year 2012 saw a slew of productions, including *Men in Black 3*, *The Amazing Spider-Man* and *Hotel Transylvania*, all with *SPI's Arnold* as the primary renderer (Figure 7). With all the elements long since stable, 2013 and beyond have seen additional productions and releases of *Oz the Great and Powerful*, *The Smurfs 2*, *Cloudy With a Chance of Meatballs 2*, *The Edge of Tomorrow* and, of course, *The Amazing Spider-Man 2*.

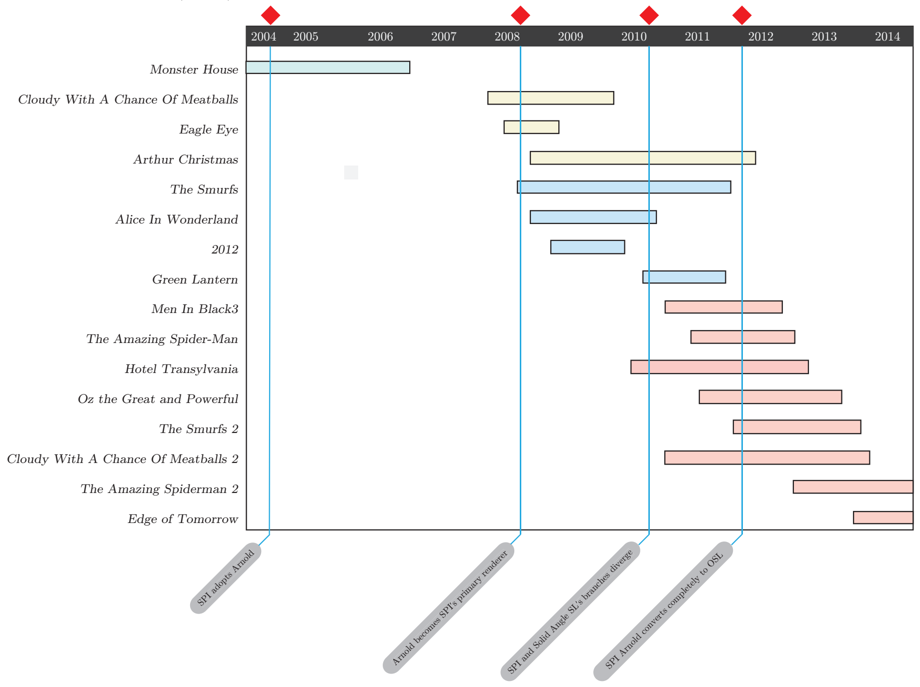
4 The State of the Art

4.1 Growing complexity

In the years since, the sophistication of the imagery has grown by orders of magnitude. When I wrote the first list of recommendations for rendering with *SPI's Arnold* in 2007, the maximum recommended number of rays per pixel was 750. In *The Amazing Spider-Man 2*, 2000-3000 rays per pixel was not uncommon. In terms of features, *The Amazing Spider-Man 2* has not only motion blur,but also geometry lights, skydome lights, fully participating hair and volumes, ray-traced BSSRDF, particles and more!

When the facility committed to Arnold, there was a strong emphasis on limited camera passes. It was considered ideal if the render was done in a single pass; this was due to several factors, not limited to consistency and the expense

Figure 7: The timeline of SPI productions using Arnold and *SPI Arnold* since 2004, with major renderer milestones. The three colors of the bars represent the shading systems in use: Maya-to-Arnold (Green), C-shaders (yellow), physically-based C-shaders/transition (blue) and finally, OSL (pink)



of multi-pass rendering at the time. Subsequent development provided AOVs and the single-pass preference has unavoidably given way to multipass AOVs which reduces the benefits to resource management. But we must also face the reality that some flexibility is required.

4.2 Talent and education

In these days of compressed schedules and tight deadlines, artist education becomes much more of a concern. Previously, there was a rotating pool of technicians and artists all fluent in *REYES* renderers. In shifting to a ray-traced paradigm, SPI was suddenly faced with educating artists on a totally new pipeline.

On one hand, the idea in transforming to the new pipeline was to shorten the training time. In line with the broader goals at SPI, the latest versions of our rendering and shading pipeline present a much simplified interface where the renderer only is in charge of sampling. Furthermore, the benefits in shifting the effort of complex lighting to the core ideally reduces the artist's job to lighting with as little between them and the renderer as possible. The thinking was that if a lookdev lead can set up a sequence and any render from that shot within that sequence is 95% complete on the first try, there there is less emphasis needed on training. In reality, however, not all sequence templates are perfectly tuned, not all lookdev is efficient and in the haste to get images before the VFX supervisor, corners are often cut.

This means that in fact education becomes paramount. But it is a double-edged sword. The more time an artist spends in training, the more expensive the hire becomes. Yet too little and the artist may not develop the strategies to become efficient users; the struggle becomes a question of how much education becomes necessary to become effective. At SPI, the training is ongoing throughout the production. Initial training is augmented with weekly "lighting tips" classes. Initial training is augmented throughout production with weekly lighting tips classes. Furthermore, any scene that's investigated and optimized is written up as an example which is distributed to all lighters every show.

5 The Workflow

SPI's Arnold represents a shift in the methods and approach taken with previous renders at SPI. Unlike the *REYES* renderers used until our conversion, *SPI's Arnold* demands more effort in pipeline preparation and setup at the outset, but once a pipeline is in place it represents a greater level of consistency and throughput. *SPI's Arnold* is less forgiving, spending time to solve problems is always more expensive than anticipating issues before they become problems. The key is to start off on the best footing possible and this extends from pre-production to the correct lighting and compositing set up.

5.1 Resources and I/O

With our *REYES* pipeline, with the required support maps there was an emphasis on sheer disk space and the need for efficient diskspace management. A decade ago the computing machinery itself was slower; fast renders with high numbers of iterations relied on a *REYES* pipeline with more work on the part of the artist and technical directors to manage the images and supporting map-types. In our shift to pure raytracing, this relationship has reversed. We are relying less on disk space and file management and more on the machines themselves. The change in rendering philosophy at the core raytracing, moving the work from the artist to the machine, meant a fundamental shift in the role of resources and I/O.

As computing hardware becomes cheaper and faster, more and more complex lighting and scenes become easier to render on schedules and budgets required in production pipelines. Complex lighting models and interactions in an efficient raytracing pipeline relies on fast machines and many of them. This shift has continued to evolve along with our raytracing pipeline. Everything was moving to memory and the CPU; the work of lighting, all geometry, everything. Even when we did experiment with caching with shadowmaps, they were created on the fly and only ever existed in memory. All these factors meant that we need as much RAM per machine as permissible. Intial renders in complex scenes have been known to climb as high as 50GB of RAM before optimization. It should be noted that while far from ideal, many of these renders did in fact succeed, completing (though crawling) through the render farm.

5.2 Modeling Considerations

With a *REYES* architecture, it was generally the case that you could model with a certain amount of impunity; not all geometry was held in memory and displacement was micropolygon-based. This is not the case in a raytracing paradigm: all geometry is held in memory until the conclusion of the render. While the growth in CPU power and RAM have made massive datasets more tenable, care has to be taken in the modeling phase to ensure efficient rendering.

Reduce geometric complexity wherever possible: Assets are often modeled with deep levels of hierarchy; this is often to simplify look development. This can have performance implications at render time and generally a "flatter" hierarchy is recommended. Furthermore, combining objects into single meshes wherever possible performs two tasks.

Level Of Detail models: Level of detail does not function in *SPI's Arnold* as it did in *REYES*. Even so, substitution of complex models in a scene for simpler, pre-modeled versions at mid and far distances can make a huge difference. Remember that *SPI's Arnold* performs subdivision and displacement based on object space, not by pixels on screenspace. Any excessive iterations or displacement on mid to far meshes can be massively inefficient. There is a potential downside: Level-Of-Detail (LOD) models can lead to extra memory usage. Instead of one big mesh, we now have a big mesh, a medium mesh, and a small mesh. To minimize any impact on memory, it is important to ensure that the medium and small mesh should be small enough to not come close in size to the large mesh. Also, the differing LOD's may present difficulties during instancing.

SPI's Arnold can subdivide based on the pixels in screen-space (ie- "adaptive" subdivision) where we can subdivide until each quad is roughly X pixels in size. This feature relies on a camera to provide the viewpoint and the size of the final subdivided quads is set by specifying the parameter "pixel_error". Care needs to be taken to limit camera movement, otherwise mesh may move in and out of a size boundary, resulting in "popping".

Hair: Hair is integral to the quality of our images and it is one of the more complex primitives to shade and raytrace efficiently. As everything is brute force/no tricks, we are often forced to try and sample what amounts to very thin specular geometry. Add to that we are creating millions of hairs with potentially hundreds of lights, low opacity, indirect diffuse and all of a sudden our performance drops. But we can take steps to limit the expense.

Can you reduce transparency of the hair and increase opacity?, Is the hair too thin? Can it be widended? Do you have the leanest lighting rig possible? Do you need to trace indirect diffuse between hairs?

Procedurals: Procedurals are another area where there was a change. In the *REYES* world, procedurals were generally considered beneficial. Storage was optimal and when rendering you could load them in lazily, that is, only load them when necessary in the scene and then discard them as they are rendered. But in a raytracer all procedurals need to be loaded and retained. Depending on the packages used, you can also incur additional overhead from the software that translates the procedurals to the raytracer. In our own experience with *SPI's Arnold* it was common for half of all memory in an fx heavy scene to be solely due to such overhead. It is therefore important to keep procedurals as lean as possible by limiting user and arbitrary data attached to procedurals.

5.3 Look Development and Lighting

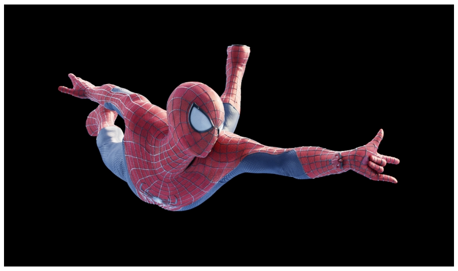
Look development and lighting had to alter quite considerably with the adoption of a raytracer. We eschewed the use of support maps and moved the work of lighting to the computer, but we need to remain vigilent about the implications of look development and lighting choices. Excessive geometry (refer to "Modeling Considerations" above), unnecessary use of opacity, subsurface scattering and excessive numbers of lights all can mean a massive decrease in performance and an increase in render time. We had to change the way we approached look development; the process is more powerful but less forgiving, requiring a clear idea of where an asset is being used, what the context of a given asset may be and how to make surface choices based on that information.

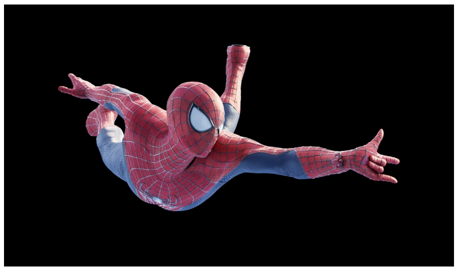
When reviewing assets at the beginning of a production, it is easy to underestimate the cost of individual meshes in a scene. Often, assets are created with flexibility in mind. With productions in flux well into the schedule, it's often difficult to have an exact conception of a given assets' final requirements. But in most cases the propagation is predictable and expensive features can be offset with cheaper and often close-matching alternatives.

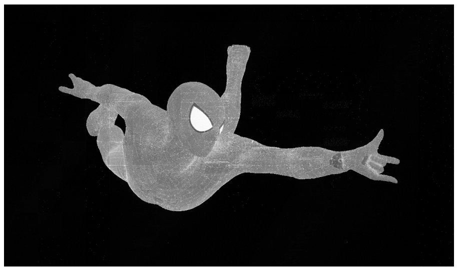
For example, in *The Amazing Spider-Man 2* we had finely modeled props for New York City, but we had screws that were modeled with 900,000 traingles each. In a *REYES* renderer we could deal with this level of geometry due to hiding, but in a raytracer all of these traingles are kept in memory. Another case is the power station in *The Amazing Spider-Man 2*; at look development, towers were created in isolation. Individually, they rendered in a reasonable time, despite having many geometry lights and high detail. However, the final Power Station had hundreds of these towers. This created over one thousand geometry lights and a low performance render. The shots involved required considerable optimization.

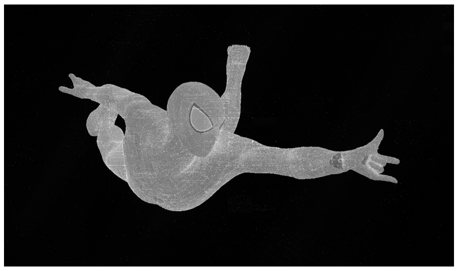
Anticipate and manage ray costs: It's important during the lookdev phase to pay close attention to ray counts. Checking the performance

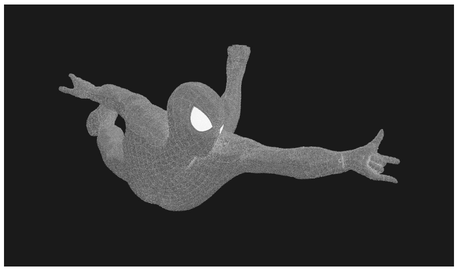
Figure 8: A comparison of the beauty pass, the cputime AOV and the raycount AOV. Note there's almost no difference between the two beauty renders (a) and (d), but in the cputime and the raycount the expense of the lenses is clear. Adjusting refraction samples from 3 to 1 significantly reduces render cost.

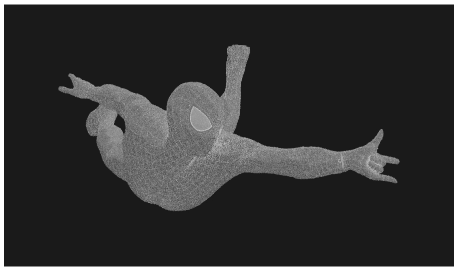

(a) Original beauty render (refraction samples=3)


(b) optimized (refraction samples=1)


(c) cputime AOV


(d) cputime AOV


(e) raycount AOV


(f) raycount AOV

using metrics such as the raycount AOVs allow the artist to visually check inefficiencies early on in the pipeline and rectify them. This can be a fine line as sometimes renders can be economical on the ray count but the shading is very compute intensive; the cputime AOV is another related metric that allows the user to see where the renderer is taking the time and together are very effective visual tools for evaluating efficiency (Figure 8).

Multi-resolution shader sets: Materials do and will become very complex; this complexity only grows as our pipeline evolves. Unfortunately, one by-product can be that huge amounts of compute cycles are consumed by calculating these surfaces, when all we need is a simpler representation to refine lighting, or for cases where objects are mid to far from camera and most of the complex lighting cannot be perceived at distance. Creating multi-resolution shader sets that use simpler, non-layered materials for testing that can then be substituted for full-shaders in the final renders can significantly lighten the render and permit faster feedback on lighting and faster, more optimized final renders in some cases.

5.4 Dailies and Rounds

Even daily schedules had to change. If we have moved the work of the render to the machine, we are now dealing with more expensive renders. In an environment where an artist is required to present dailies in the morning, prepare rounds by the afternoon and then set off new render for the next day, more expensive renders may mean these daily milestones are missed. With our *REYES* pipeline, the use of shadowmaps and other support maps often meant that s small change in light color or position was of little consequce; a new version could be rendered out without shadows and the same shadowmaps re-applied. In our conversion to a raytracer, each re-render was a complete re-evaluation of the scene and to manage the impact, we have had to develop new ways of working with the requirements of dailies and rounds:

Rendering at low resolutions: Frames in *SPI's Arnold* rendered at different resolutions/samples do not alter the color or properties of the image, they simply alter noise levels. This makes it possible to use lower resolution renders to evaluated lighting in the initial iterations. Rendering at higher resolutions with higher antialiasing settings can be performed once lighting has reached a satisfactory point. This does require confidence at the team level and an acknowledgement that noise will be resolved in final renders.

Single frame evaluation: Rendering single frames during the day for inital lookdev followed by turntables overnight/weekends can dramatically improve render turnaround. As a shot progresses and higher levels of optimization can be done, more and more frames can be rendered.

6 Conclusion

Originally tested to achieve a realistic "claymation" look, the advantages of raytracing at a production level were quickly realized. Adoption facility wide followed and began what is now a decade long process of converting our production environment from a well-established *REYES* pipeline to a completely different, raytracing paradigm. Understanding how each rendering method achieved the value of a pixel was key to adapting and optimizing our new workflow. This anecdotal account of the transition and documentation of the changes to our way of working conveys some of the successes and some of the trials experienced, the loss and the gain. At each step, our workflow has to be tweaked and the changes collate into an effective methodology of approaching a raytracer on ever shrinking production schedules.

As pure raytracing workflows become normalized throughout the visual effects industry, the more experienced artists and technicians will become experienced with its idiocyncracies, its strengths and its weaknesses. With computing machinery getting ever cheaper and the simulation of light transport ever more sophisticated, these techniques will expand. Already in 2014, we see the shift toward techniques such as bidirectional path tracing that will usher in yet another transition akin to the move from *REYES* to raytracing, bringing with it a new jump in techniques and workflow.

7 Acknowledgments

Thanks to Sony Pictures Imageworks and CTO Rob Bredow and for supporting the continued development of raytracing at SPI; the SPI Arnold Team both past and present, Larry Gritz, Christopher Kulla, Cliff Stein, Alejandro Conty, Rene Limberger. Additional thanks to Larry Gritz for the diagrams used in Figures 2 and 3, created for a discussion on *REYES* vs. raytracers at Sony Pictures Imageworks.

References

[1] T. L. KAY and J. T. KAJIYA. Ray tracing complex scenes. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, volume 20, pages 269–278, 1986.

[2] T. WHITTED. An improved illumination model for shaded display. In *Communications of the ACM*, volume 23, June 1980.

Raytracing in Production at Double Negative

Søren Ragsdale

Double Negative Visual Effects

sorenr@gmail.com

Siggraph 2014, Vancouver Canada

1 Summary

An overview of the last 3 years of production at Double Negative. This paper and course presents a series of attempts, decisions, problems, and solutions by Double Negative as we attempt to evolve a traditional REYES pipeline into a robust, flexible physically plausible solution for production cinematic rendering.

2 Introduction

"Strategy is a system of expedients; it is more than a mere scholarly discipline. It is the translation of knowledge to practical life, the improvement of the original leading thought in accordance with continually changing situations."
- General Helmuth von Moltke "On Strategy" (1871)

For many years rendering for visual effects was largely dominated by rasterizing renderers. Lighting artists used raytracing sparingly, often as a last resort for specific problems. By the early 2000s a combination of new research, robust raytracing solutions, and more capable compute resources created renewed interest in raytracing as a general approach. At the same time research was also progressing with new techniques to more accurately simulate the propagation of physical light and reflection properties of physical surfaces.

Beyond obvious visual improvements, there are clear practical advantages of a rendering solution of physically based BRDFs and Monte Carlo integration. A hand-tuned raster scene may run faster than an equivalent raytraced scene, but reductions in storage and pass management make "slower" raytracing a net economic win. As the price of computing resources continues to drop, physically plausible assets allow fewer artists to turn over more shots. Physically plausible assets also save effort by facilitating easy re-use between shows: a police car asset developed for a chase scene in one production can be dropped into a junkyard scene in another production. With proper shading and look development practices, it "just works".

In 2010, the Double Negative shading team began a major update to our shading library. Two complete rewrites our code base resulted in significant changes to our overall rendering strategy. This paper covers the first ~3 years of our ongoing effort to stay abreast of best practices for production efficiency and physical realism. Beyond research papers, maths, and algorithms, other concerns such as artist expertise and confusion, legacy tools, and financial constraints add unavoidable complications the already challenging task of keeping pace with evolving research.

3 New beginnings with “V4”

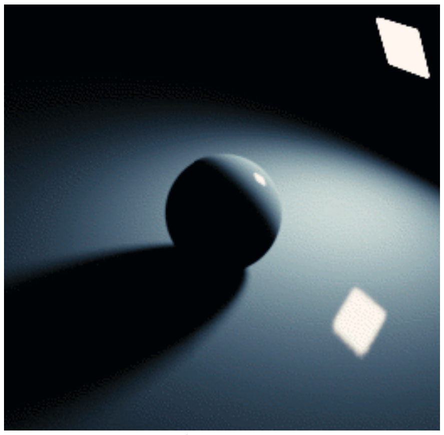
In 2010, the Double Negative rendering pipeline made heavy use of Pixar's RenderMan. We began in late 2010, targeting the upcoming release of RenderMan 16 for a new round of shows slated to begin in six months. The first public API for “16.0b1” in November 2010 made no mention of physically plausible shading, so we wrote an entire sampling and integration stack to handle everything from sample generation to final integration in RSL. This first shading library was completely outside the 16.0b3 API which would be announced in March 2011.

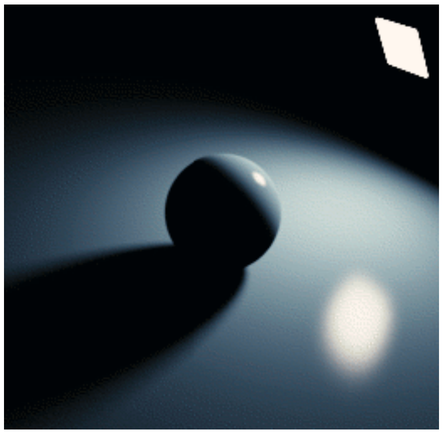
3.1 Design Decisions

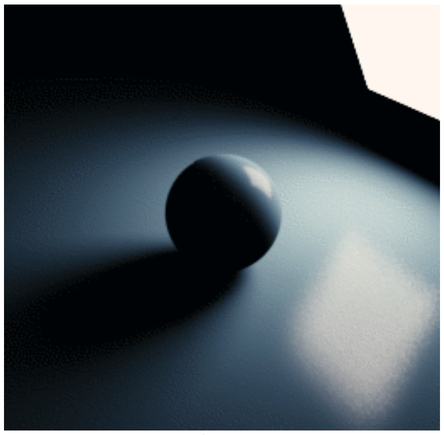
Early on, there was disagreement about the meaning of a light's “brightness” means. Some artists and lighting packages favour a brightness parameter dictating irradiance, or energy per square unit of area light. Double Negative artists found it more comfortable familiar to use a lighting parameter representing the overall power of the light. A light’s power is maintained by scaling its per-unit irradiance as its overall area increases, allowing the softness of the light’s shadow to be adjusted independently of its brightness.

V4 had been written with support for a modified Lambert, Microfacet, Beckmann, GGX, Phong, Distributions, and Ward BRDFs. In production most of these options were unused - look development artists generally used modified Lambert for diffuse and Ward for specular, or GGX in cases where rough specular surfaces should be realistic. The “long tail” of the GGX reflection curve extending out to glancing angles takes a large number of rays to sample without variance, but the results speak for themselves.

3.2 New Concepts for Artists

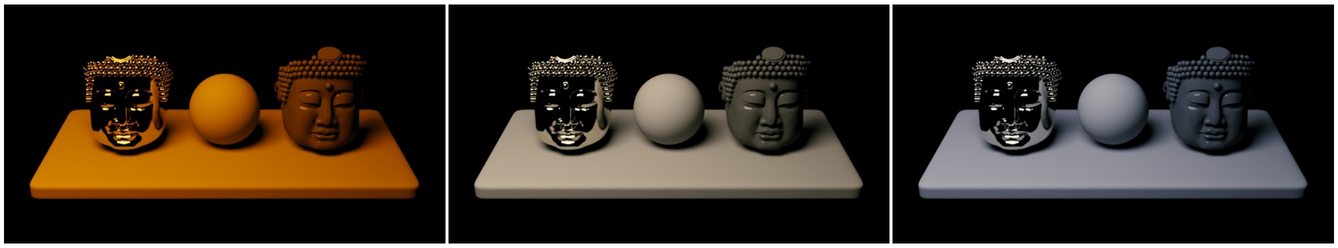

small light / smooth surface


small light / rough surface

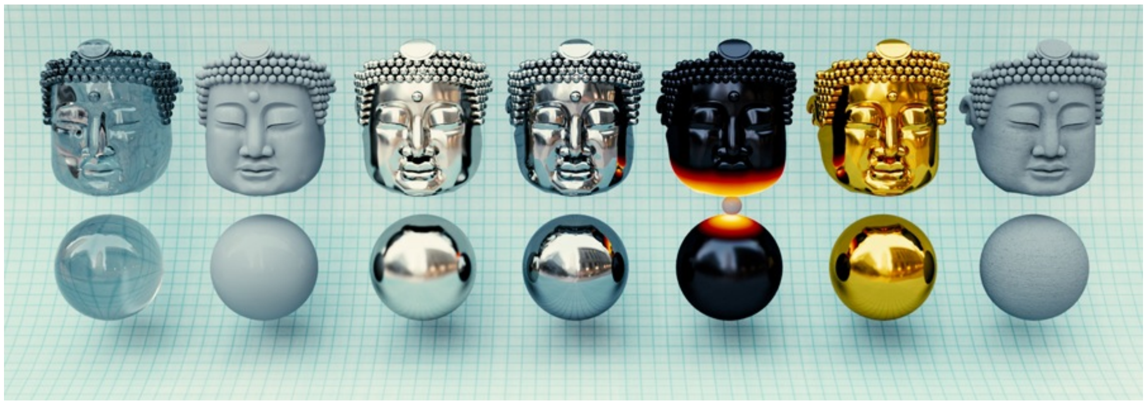

large light / rough surface

Artists transitioning to a physically plausible pipeline were required to cast off the artificial abstractions from old workflows and assumptions. There are no such thing as point light sources in the real world, or in V4. Spotlights are now objects with physical size and shape. Parameters such as “specular size” or “shadow blur” are driven implicitly by a light’s size and position or a surface’s roughness.

V4 light colours are configurable by familiar RGB values, and also with realistic black body colour temperatures. This decision was not only helpful for matching physical set lights with known values, but during artistic lighting to provide a realistic richness which can be difficult to achieve by guessing at values. These three scenes are illuminated by 1700K, 5000K, and 9300K lights.

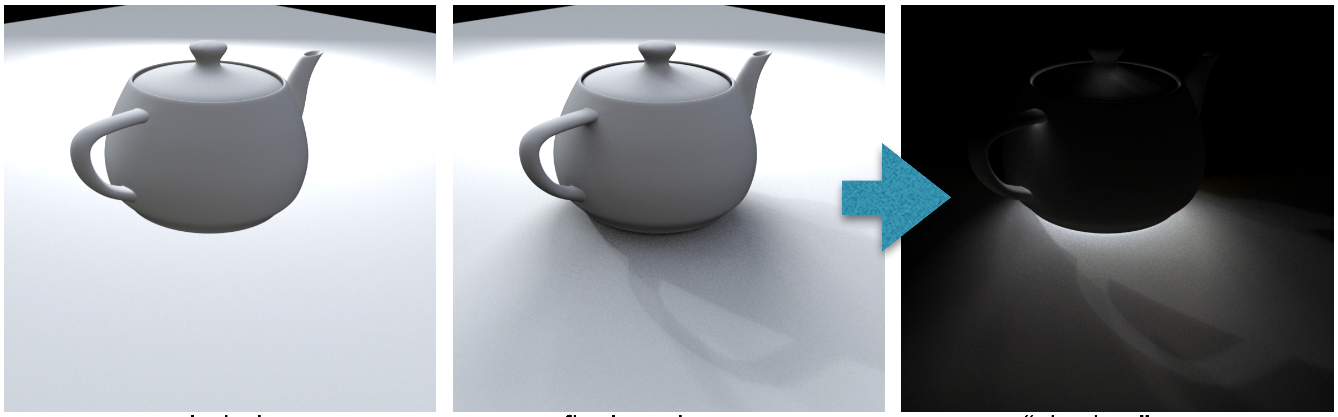


In a traditional pipeline, artists could take an ad hoc approach to look development by changing parameters like “specular size” and “reflection colour” until the render matched the reference. In a physically plausible pipeline, artists must actually understand the physical properties of materials - the difference between the reflection of a conductor and a dielectric, or the real physical structure of a layered material. Lighting has become simpler, but lighting artists must now understand how light actually behaves. Using physically based BRDFs and attribute values from physical tables can yield realistic results quickly.



Left to right: glass, ceramic, silver, chrome, blackbody emission, gold, and matte

Compositors were also required to adjust their thinking and workflows. Comp-friendly shadow passes were obsolete. In a scene where opaque surfaces can create shadows from multiple lights, and those shadows can be illuminated by indirect light or emissive surfaces, the notion of “shadow” is poorly defined. Light is additive but shadows are not. Where necessary, compositors could approximate shadow passes by subtracting the final image from an “unoccluded” pass:

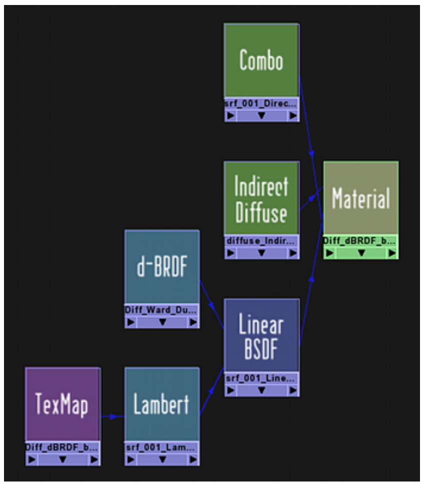

unoccluded pass final render pass “shadow” pass

3.3 Control Variates

Control variates[1] were applied to decrease variance in sampling, by augmenting ray-based integration methods with analytical methods. In this case, diffuse integration may require a large number of samples and still produce substantial variance. In the example of diffuse illumination, a convolution of the environment map provides a perfectly smooth representation, but without occlusion. Monte Carlo integration may also sample occlusion, but with high variance unless many expensive samples are used. Control variates are a way to blend between these methods using smooth convolution maps when occlusion is low, and ray-sampled diffuse when occlusion is high. Control variates can also work for arbitrary area lights if they have an analytic approximation.

3.4 Network Shading Structure

Artists were presented with a new shading structure. Surface shaders were configured with direct and indirect lighting integration nodes and an “aggregate material” node containing one or many BRDFs, which could reference procedural or texture pattern generators.



3.5 V4 In Production

The first production test for the V4 code base was "The Pirates! Band of Misfits" (2012). “Pirates” was a relatively easy first target,

[1] Mark Colbert, Simon Premoze, and Guillaume Francois “Importance Sampling for Production Rendering” § 2.3.3, Siggraph 2010

with simple clay characters, controlled lighting environments, and a stop-motion aesthetic without motion blur. Picking a smaller, less challenging production for deployment of new tools is important both for the success of the production and the perceived success of your tools.

“The Dark Knight Rises” was also a highly successful production. The director’s affinity for physical effects and practical environments limited rendering to the addition of characters or vehicles. Reference photographs and physical surveys on set of lights, environment maps, and set pieces allowed the creation of assets which performed consistently across different action sequences.

“Snow White and the Huntsman” presented a greater challenge due to a number of interior shots lit primarily by large windows. To efficiently render these scenes we implemented “light portals” to guide the direction of light samples. When rendering interior scenes, only a small percentage of occlusion rays will escape the scene and reach exterior lights. Light portals are rectangular shapes which designate these open areas. Portals do not change the appearance of a render, but they can vastly improve render time by restricting rays to productive directions. If your renderer parameterises the space as polar coordinates, portals may work better for windows at the horizon than skylights overhead. If your renderer supports bidirectional path tracing, this is a better general solution which can obviate the need for portals entirely.

3.6 Rendering Challenges

“Total Recall” presented larger challenges than previous shows due to the content of the work. Concept art and on-set reference photography showed scenes with hundreds of physical lights, and artists dutifully created scenes with 400-5000 area lights. The associated performance problems were solved with a dedicated light tracing DSO[2] presented at “Stupid RAT Tricks 2012”.

“Total Recall” concept art featured geometrically dense, procedurally generated virtual city environments. Scenes with dozens of densely detailed, procedurally generated buildings resulted in impractical render times and memory requirements. To control the complexity of distant buildings we

[2] http://www.renderman.org/RMR/Examples/srt2012/dnLightTracer.pdf

developed “lightable brick map” object proxies. Brick maps[3] are a type of file-based octree implemented in RenderMan which contain volumetric information such as surface colour or opacity. By “baking” a simple subset of geometry and surface parameters such as diffuse, reflection, and glossy albedo into file-based brick maps we could provide an efficient proxy representation of static geometry. At render time this baked representation is rasterised into a camera-facing discs, retrieved at a sufficient depth to provide detail at the scale dictated by camera distance. This strategy was visually superior to 2D cards, while remaining very efficient in speed and memory, and surprisingly compatible with Monte Carlo, although extremely large trace bias settings were often necessary to avoid self-shadowing artefacts. Without large trace bias values, a diffuse ray with large derivatives departing a brick map object might intersect with a lower-resolution level of the same object.

Shots from “Man of Steel” (2013) using V4 and relightable brick maps to render large numbers of intact and collapsed buildings.

“Man of Steel” followed with an even larger scope of work, as the nearly 2000 buildings in the city of Metropolis were destroyed during a climactic fistfight between Kal-El and General Zod. While lightable brick maps were a crucial component in this show’s rendering pipeline, brick maps could not be used in place of dense animated geometry such as collapsing buildings. The difficulty of our RenderMan/V4 approach to render animated collapsing buildings and the limitations of lightable brick maps necessitated a new approach to renderer diversity.

3.7 Renderer Diversity

A handful of shots from “Man of Steel” which could not fit within the constraints of RenderMan/V4 were rendered using Mantra, the first step into Double Negative’s current effort toward renderer diversity. The ability to render identical assets across multiple renderers has also been helpful in identifying bugs and performance issues in internal and external code. Investigation into the problems associated with rasterisation of large polygon data sets - the root causes of these performance issues and how to address them - resulted in significant improvements to RenderMan, formed the basis of a new shading library, and spurred an effort toward renderer diversity. Lightable

[3] http://renderman.pixar.com/resources/current/rps/brickmapgprim.html

brickmaps were crucial for finishing the majority of the film's static buildings, but could not be used in scenes with animated collapse.

4 REYES and Rays

Published in 1984, the REYES[4] algorithm was designed to render within the constraints of early computing. Its performance innovations included:

- Geometry is kept in memory as high-order surfaces, and only diced into micropolygons when necessary for shading to minimise memory footprint.
- All micropolygons from a patch are shaded together using a data-parallel SIMD[5] approach to maximise CPU and texture cache coherency.
- Shading products from adjacent micropolygons are used to compute shading derivatives via finite differencing methods.

This strategy was tremendously effective during their time, but today these performance features can create problems unless their implications if they are not fully appreciated.

4.1 Dicing More Polys into Fewer Micropolys

For smaller assets, a relatively small number of higher-order shapes can be efficiently diced into a large number of micropolygons to create smooth shading. As geometric complexity increases, more shapes must be decimated to yield fewer micropolygons. It is not uncommon for current show assets to have a dozen polygons per pixel. A recent asset in "Jupiter Ascending" contained an average of 110 polygons per final rendered pixel. With increasing geometric density, the setup cost for dicing more patches into fewer shaded polygons becomes significant.

Supervisors working with high-complexity assets should be careful to avoid hidden or unnecessary detail. Simple diagnostic shaders can highlight assets with wasteful detail. The trend toward higher resolution asset is one factor currently driving production renders toward raytracing. Regardless how many objects are inside one render bucket, camera rays obviate dicing while limiting shading to the primitives that the rays hit.

---

[4] http://dl.acm.org/citation.cfm?id=37414

[5] Single Instructions executed on Multiple Data

4.2 Interpreter Overhead and Indirect Shading

The data-parallel execution of shader code across large patches of points comes with significant coherency benefits, and is very easy to vectorise if target hardware supports it. Unfortunately this approach comes with equally significant expenses such as variable binding and per-shadeop dispatch costs. These costs dominate in the above case of large collections of small patches, or during evaluation of indirect illumination. In this illustration, the interpreter costs for evaluating direct lighting in the primary patch (green) is offset by the large number of micropolygons simultaneously evaluated. Indirect samples (red) incur the same interpreter costs to evaluate proportionately fewer samples.

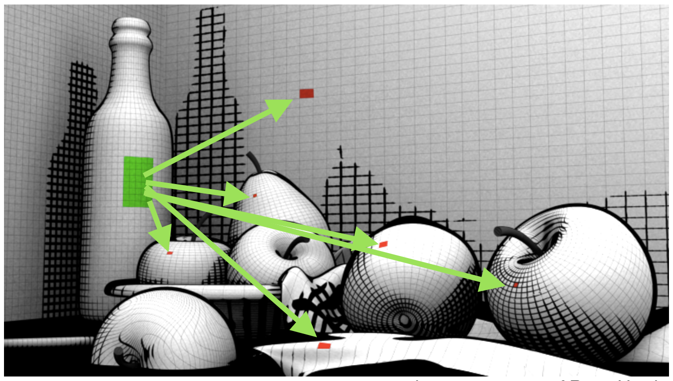


image courtesy of Ryan Hurd

4.3 Extraneous Points for Derivatives

Finite differencing is a method for calculating variable derivatives from adjacent samples. In the above illustration, evaluation of direct lighting on the bottle requires the evaluation of a relatively small number of extraneous points along the edge of the bottle's patch. For secondary illumination, the evaluation of a single sample on the wall may require two extraneous evaluations in 'u' and 'v', effectively tripling the cost of shading. In the above illustration, primary integration on the green portion of the bottle results in the evaluation of 40 primary samples and 13 extraneous samples. The secondary lighting from the wall - one of potentially hundreds or thousands - results in one primary sample and two extraneous samples.

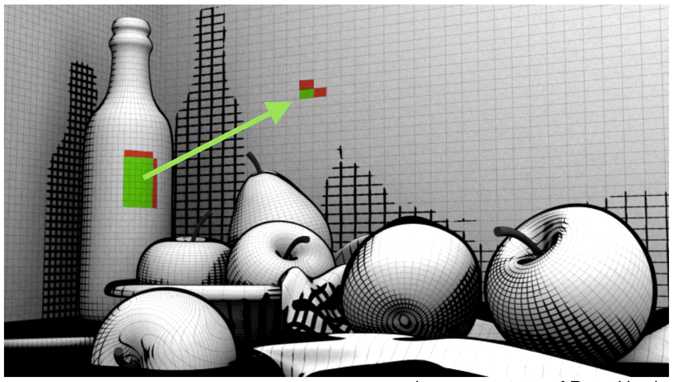


image courtesy of Ryan Hurd

To avoid unnecessary overhead in shaders which evaluate extraneous points for finite differencing, test whether your shader evaluation is occurring on primary shading. Run cheap evaluations (displacement only, if possible) if the point is extraneous.

4.4 What about OSL and RIS?

RenderMan RSL is not the only interpreted shading language, so why don't the languages OSL[6] and VEX, , which also compile shading their shaders into a bytecode representation, suffer from the same penalties?

OSL complies to LLVM bytecode which is eventually converted to native machine code, so there's no interpreter overhead. OSL is executed vertically, so there's no vectorisation penalty on small sets of points. OSL does not use finite differences for derivatives, so it does not calculate extraneous points: it uses automatic differentiation, computing the partial derivatives of the variables that lead to derivatives.

4.5 Irradiance Caching

The performance considerations mentioned above are most adverse when evaluating diffuse illumination, which often requires a very large number of widely distributed samples. Diffuse irradiance caching is employed to save the results of view-independent components for further use. Unfortunately this caching strategy also introduces complications. Low-importance secondary rays must be computed at high quality because their values may be cached. Burdened by this obligation, a renderer can fill the cache with unnecessarily high quality lighting information than it will never use.

4.6 RenderMan 19

The above performance issues are the major reason driving the significant changes announced in RenderMan 19.0 this year. Path tracing from the camera reduces the REYES overhead from dicing large numbers of patches into small numbers of polygons. With the RIX code path in C, the overhead from the RSL interpreter has been eliminated - shaders can be efficiently evaluated on single samples, which also makes irradiance caching unnecessary. Bidirectional integrators improve sampling in enclosed surfaces without the need for portals. The new "RIS Mode" API allows shaders written in C to shade multiple points at a time, in a data-parallel if the shader writer chooses, without the overhead of the RSL interpreter.

[6] http://github.com/imageworks/OpenShadingLanguage

5 V5 and Renderer Diversity

In late 2012, based on ~18 months experience with V4 in production and the performance issues mentioned in the previous section, we began a complete rewrite of our V4 code base which we imaginatively called "V5".

5.1 Moving from RSL to C

To minimise RSL interpreter costs, as much code as possible would be moved inside C DSOs. Layered BRDF combining, light and material sample generation, and BRDF evaluation were handled within DSOs. Testing for extraneous shading made sure that full surface calculations would not be wasted on points which would be visible.

5.2 Simplified Shading: Less Is More

In theory, node-based shader networks offer a flexible way for artists to configure shaders. In practice these networks were unwieldy to set up, invited wasteful errors such as variables duplicated between nodes, and contained configuration variables scattered between the integrator, master material, and aggregate material nodes. V5 surface shaders were rewritten to be monolithic, to create a more regular and simplified interface. Many optimisation settings were also eliminated - either removed when they were rarely helpful, or hardcoded when they were always helpful. Reflection function models were pared down to Burley diffuse[7], Ward[8], Generalized-Trowbridge-Reitz[9] (GTR), and GGX[10].

5.3 Texturing with SeExpressions

For almost every shading parameter there is some usage case where an artist would want that parameter driven by a texture map. Increasing sets of texture maps, and the necessity of parameters

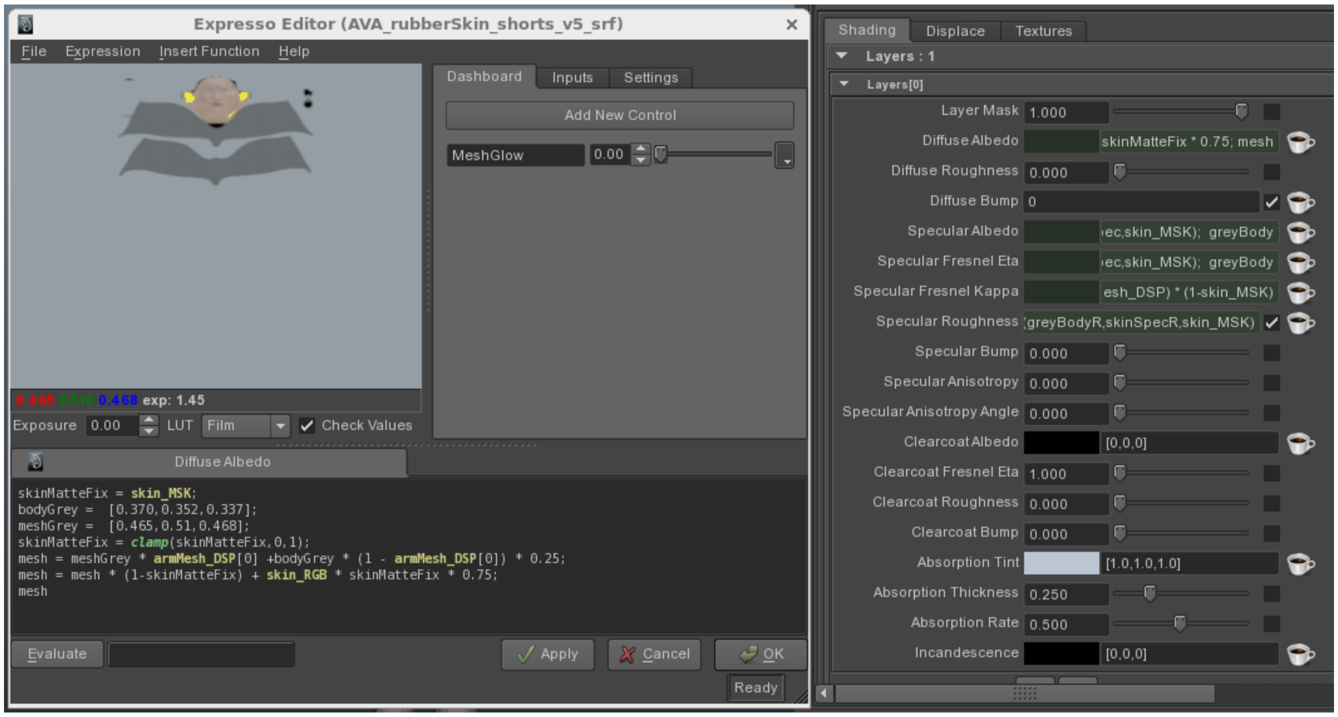
[7] http://blog.selfshadow.com/publications/s2012-shading-course/burley/s2012_pbs_disney_brdf_notes_v2.pdf

[8] Bruce Walter. Notes on the Ward BRDF. Technical Report PCG-05-06, Cornell Program of Computer Graphics, 2005.

[9] http://blog.selfshadow.com/publications/s2012-shading-course/burley/s2012_pbs_disney_brdf_notes_v2.pdf

[10] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In Proceedings of the Eurographics Symposium on Rendering, 2007.

to describe constant values, map names, and adjustment parameters like blend or contrast lead to a parameter explosion which are particularly detrimental to indirect shading for the reasons discussed



MeshGlow 0.00

0.465 0.510 0.468 exp: 1.45

Exposure 0.00 LUT Film ✓ Check Values

Diffuse Albedo

```
skinMatteFix = skin_MSK;
bodyGrey =  [0.370,0.352,0.337];
meshGrey =  [0.465,0.51,0.468];
skinMatteFix = clamp(skinMatteFix,0,1);
mesh = meshGrey * armMesh_DSP[0] +bodyGrey * (1 - armMesh_DSP[0]) * 0.25;
mesh = mesh * (1-skinMatteFix) + skin_RGB * skinMatteFix * 0.75;
mesh
```

Layers : 1

Layers[0]

Layer Mask	1.000
Diffuse Albedo	skinMatteFix * 0.75; mesh
Diffuse Roughness	0.000
Diffuse Bump	0 ✓
Specular Albedo	ec,skin_MSK); greyBody
Specular Fresnel Eta	ec,skin_MSK); greyBody
Specular Fresnel Kappa	esh_DSP) * (1-skin_MSK)
Specular Roughness	(greyBodyR,skinSpecR,skin_MSK) ✓
Specular Bump	0.000
Specular Anisotropy	0.000
Specular Anisotropy Angle	0.000
Clearcoat Albedo	[0,0,0]
Clearcoat Fresnel Eta	1.000
Clearcoat Roughness	0.000
Clearcoat Bump	0.000
Absorption Tint	[1.0,1.0,1.0]
Absorption Thickness	0.250
Absorption Rate	0.500
Incandescence	[0,0,0]

in §4.2 and §4.3.

To provide flexibility without increasing texture counts or shader parameters, V5 shaders have values represented as expressions evaluated by Disney's SeExpression[11] library. Nearly all variables in V5 are SeExpressions, from simple scalar values to multi-line expressions which combine input from several textures. This approach can facilitate rapid and flexible look development, although care should be taken to simplify expressions before production rendering begins. SeExpressions also support procedural pattern generation, but since SeExpressions do not natively support texture derivatives care must be taken to avoid aliasing.

5.4 *Cross-Platform Shading*

Support for renderer diversity which started during "Man of Steel" was a major design goal of V5. With the bulk of the shading library implemented in C, general-purpose shading functions can be called from any renderer which with support for a C API. Isotropix' "Clarisse" is a relatively new offline raytracer and interactive rendering environment. In early 2013 we created a Clarisse shading library using the core V5 shading library. This approach required a minimum of new code, with identically configured assets producing consistent output between RenderMan and Clarisse. Side Effects'

[11] http://www.disneyanimation.com/technology/seexpr.html

“Mantra” has also taken on a significant role in production, whose shaders also use calls into Core Shading.

6 V5 In Production

The “V5” shading core and its various manifestations have been the mainstay of production for the last year at Double Negative. “Thor: The Dark World” was predominantly a RenderMan show following in the footsteps of “Man of Steel”, with character and interiors traditionally rendered and large exterior flythroughs making heavy use of relightable brick maps. “Jupiter Ascending” made heavy use of the Mantra pipeline. “Godzilla” was the first show to make limited use of Clarisse, which is playing an increasing role in current productions.

7 Best Practices and Future Work

- Deploy major releases on small shows before use on large shows
- Take care with procedural geometry: emit instances where possible
- Renderer diversity
    - lets studios leverage different strengths on different shots
    - exposes visual and performance bugs
    - preserves negotiation flexibility, avoids expensive “lock-in”

8 Thanks

Team DNeg: Marc Bannister, Emmanuel Turquin, Simon Premoze

Team Pixar: Philippe LePrince, Dana Batali, and Wayne Wooten

Team Miscellaneous: Jesse Andrewartha, Larry Gritz, Adam Martinez

COURSE NOTES
SIGGRAPH 2014, VANCOUVER, CANADA

TURNING IT TO ELEVEN

Establishing best practices in raytracing at Sony Pictures Imageworks

PRESENTED BY:

Jesse Andrewartha
Sony Pictures Imageworks

Contents

1 Introduction

The establishment of “best practices” at Sony Pictures Imageworks (SPI) has been the culmination of over seven years of productions working solely with *SPI’s Arnold* renderer, transitioning from REYES to monte-carlo raytracing and optimizing performance issues in all subsequent productions. Since the adoption and co-development of *SPI’s Arnold* by SPI with Solid Angle SL almost a decade ago, raytracing has provided a tool that permits far greater consistency, more sophisticated lighting effects and removes the burden of maintaining libraries of supporting maptypes required at render time. In exchange, the system is less tolerant of non-photorealistic approaches, provides fewer controls and can easily be tweaked so that performance comes to a standstill. For VFX artists, the change was not without challenges; maintaining rendering efficiency in production has required a myriad changes in workflow philosophy and implementation. The creation and adoption of OSL further unified rendering but also further shifted the landscape from beneath artists and technical directors. The result has been a massive re-education of how to manage the rendering environment.

In our day-to-day experience, we see not only the benefits but also the shortcomings of our pipeline. In this document, we are going to reveal the culmination of over seven years experience working day-to-day with a raytracer-centric pipeline, our “best practices”. In this discussion, we will be using examples and discussing scenarios we encounter with our renderer, *SPI Arnold*, but many of the concepts should be general enough to apply to most modern production raytracers.

2 The nomenclature of raytracing

In examining optimization, there are a number of terms that will be used in the course of this discussion which have specific meaning in context to our renderer. These will be changed for both reader accessibility and proprietary reasons. Because SPI’s development of Arnold has occurred parallel but separately from *Solid Angle SL* (our branches of the renderer separating soon after version 3.0) Arnold will be called *SPI’s Arnold*.

In examining production examples, we will be using specific nomenclature in relation to raytracing; let us start out with a quick refresher on how a raytracer like *SPI’s Arnold* samples the scene.

> *Camera Rays* are fired into the scene from a camera; these are the initial rays that kick off the raytraced render. The ray travels into the scene until it hits an object, at which point the surface is evaluated. If required, further rays are fired and the final object color is returned. The number of camera rays per pixel fired into the scene are determined by *AA_Samples* (which stands for *Antialiasing_Samples*). All sampling in *SPI’s Arnold* is calculated “n x n”, so that if you have `AA_samples` = 8, then in reality you have 64 camera rays per pixel.

Shadow Rays originate from a ray intersection point on a surface . The purpose of the shadow ray is to determine if the point is in shadow. Shadow samples are set in the light and the default is 1, meaning one shadow ray is fired for each incoming ray.

Reflection and Glossy Rays are fired from the same intersection point if the shader has a specular value greater than 1.0. Whether or not the ray is glossy or reflection depends on the specular model: specular roughness greater than zero (ie- mirror) will result in a reflection ray, whereas any roughness will result in glossy rays.

Diffuse Rays are used to gather the indirect diffuse contribution to the the point being shaded. Diffuse rays are fired randomly in a 180 degree hemisphere tangent to the point. These rays are fired if the diffuse ray depth is greater than zero. The number of diffuse rays per pixel are determined by *GI_diffuse_samples*.

Refraction Rays are used for any surface with opacity less than 1.0 and the number of refraction rays is determined by *GI_refraction_samples*. The refraction depth is determined by *GI_refraction_depth*.

You will note from this that any increase in the *AA_samples/Camera Rays* will result in the exponential increase in any subsequent raytype. This is important in the following sections.

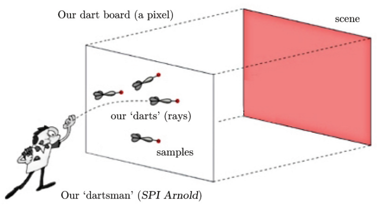
3 Approaching Optimization

So imagine yourselves lighting a shot; you have started to establish the mood and are fairly settled on the light configuration. There is only a week budgeted for lighting the shot (if that) and you need to render a final quality test. Your supervisor is anxious to get something in front of the visual effects supervisor and informs you that you need something on the cue by the afternoon. Many people faced with this situation react by 'turning it to 11': overcranking the antialiasing samples with the idea that if there are simply enough camera rays, their render will be smooth. It will take time, but the ethos is to move the work of lighting from the artist to the machine, right? While technically true, simply dialing the antialising often results in heartache, with unrenderable images and missed deadlines. We need to intercept this process, both from the point of view of the artist and the supervisor. It takes time, but optimization must be seen by both parties as an investment; the determination of the best settings for the shot through simple, small tests, only submitting the full version to the cue when optimized. The result is that for most shots, we are able to achieve considerably faster frames with just several hours testing. It is possible to see those hours as time lost, but consider that those core hours would have otherwise been taken cooking these frames that ultimately will not have finished or worse still, shown problems and required a re-render. Instead, the farm was free to render other artist's jobs and when we finally submit to the cue, the frames are optimized and run much faster, completing in less time than the original estimate even considering the time investment in optimization. So how do we approach optimization?

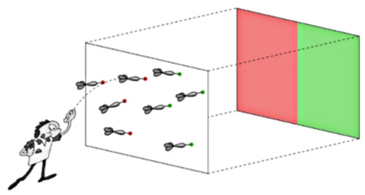
The short answer is that we must become detectives. To resolve image artifacts and improve render performance we must first understand how these problems are created, then divide and conquer. So what problems are there and how are they created?

Take a look at Figure 1 and imagine that you have a dartboard with a photo on it (on our screen, the pixel is the "dartboard" and the scene is the "photo") and you want to know the average colour of the photo. It is easy to see what the colour is at any one point, so you throw a bunch of darts and then average the results.

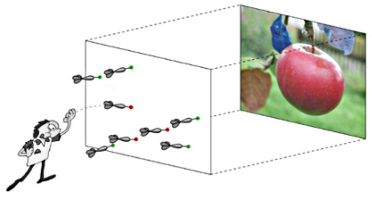
Figure 1: An illustration of how we sample in a raytracer such as *SPI's Arnold*



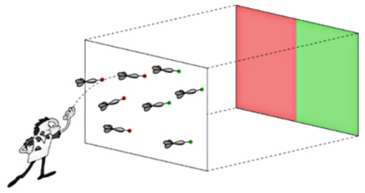
(a) If the dartboard is all one color, it takes just a few darts to get the correct average.



(b) If the dartboard is half one color and half another, a whole lot of darts will still give you a good average.



(c) But if you have a complex pattern (say, this image of an apple) and use just a few darts, say 8 darts...



(d) ...then sometimes half will be red and half green, sometimes 6 will be one color or the other, and sometimes just by coincidence all will be on the red apple, despite only a fraction of the board being the red apple.

So how do we fix this? There are three ways:

Option 1: Improve your dart throwing so they do not bunch up as much (that is what the *SPI Arnold* team does).

Option 2: Increase the number of darts (sampling).

Option 3: Try to make the pattern on the dartboard simpler, so fewer darts are necessary to estimate the average (such as the use of MIP-mapping, limiting noise octaves and decreasing the frequency content of patterns).

For the artist, let us evaluate these options: *Option 1* is literally beyond artist control; engineers consistently work toward better resolution of noise within an image, but we are limited by current research and investigation. *Option 2* resolves noise at the expense of performance. *Option 3* typically involves the optimization of scene elements but oversimplification can impact image integrity. Optimizing rendering performance and reducing noise usually is a combination of *Option 2* and *Option 3*. If optimization with *Option 2* still leaves renders that exhibit noise, then it is often a matter of *Option 3*, which may be an optimization that is required in any scene element from the textures to the meshes.

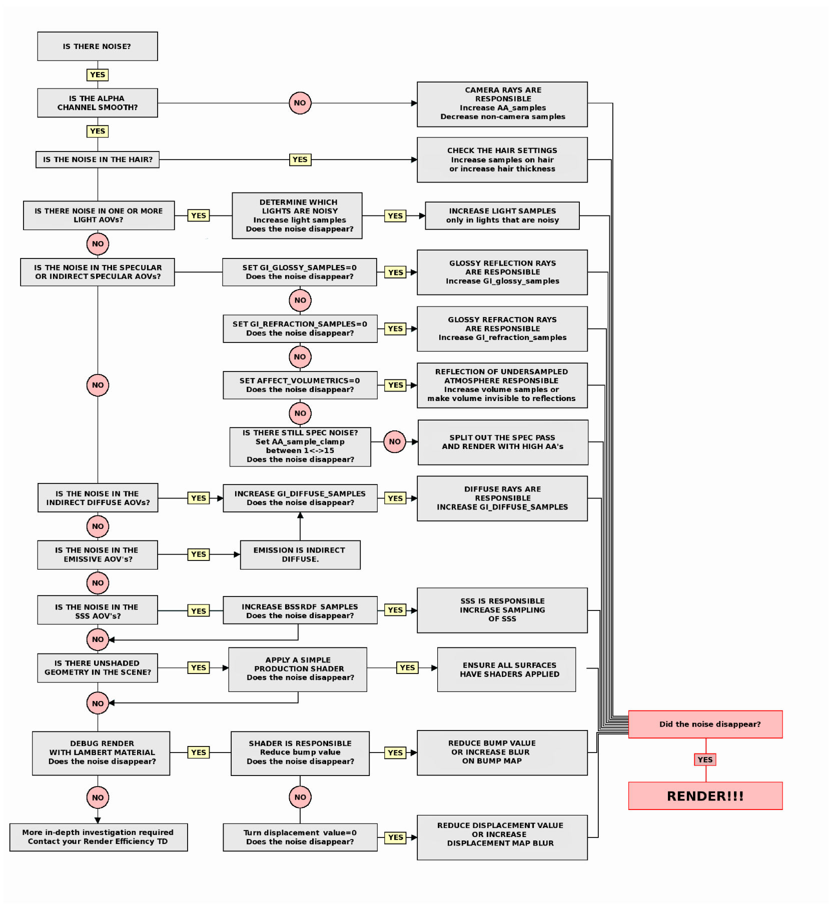

IS THERE NOISE?
YES
IS THE ALPHA CHANNEL SMOOTH?
NO
CAMERA RAYS ARE RESPONSIBLE Increase AA_samples Decrease non-camera samples
IS THE NOISE IN THE HAIR?
CHECK THE HAIR SETTINGS Increase samples on hair or increase hair thickness
IS THERE NOISE IN ONE OR MORE LIGHT AOVs?
DETERMINE WHICH LIGHTS ARE NOISY Increase light samples Does the noise disappear?
INCREASE LIGHT SAMPLES only in lights that are noisy
IS THE NOISE IN THE SPECULAR OR INDIRECT SPECULAR AOVs?
SET GI_GLOSSY_SAMPLES=0 Does the noise disappear?
GLOSSY REFLECTION RAYS ARE RESPONSIBLE Increase GI_glossy_samples
SET GI_REFRACTION_SAMPLES=0 Does the noise disappear?
GLOSSY REFRACTION RAYS ARE RESPONSIBLE Increase GI_refraction_samples
SET AFFECT_VOLUMETRICS=0 Does the noise disappear?
REFLECTION OF UNDERSAMPLED ATMOSPHERE RESPONSIBLE Increase volume samples or make volume invisible to reflections
IS THERE STILL SPEC NOISE? Set AA_sample_clamp between 1<->15 Does the noise disappear?
SPLIT OUT THE SPEC PASS AND RENDER WITH HIGH AA's
IS THE NOISE IN THE INDIRECT DIFFUSE AOVs?
INCREASE GI_DIFFUSE_SAMPLES Does the noise disappear?
DIFFUSE RAYS ARE RESPONSIBLE INCREASE GI_DIFFUSE_SAMPLES
IS THE NOISE IN THE EMISSIVE AOV's?
EMISSION IS INDIRECT DIFFUSE.
IS THE NOISE IN THE SSS AOV's?
INCREASE BSSRDF SAMPLES Does the noise disappear?
SSS IS RESPONSIBLE INCREASE SAMPLING OF SSS
IS THERE UNSHADED GEOMETRY IN THE SCENE?
APPLY A SIMPLE PRODUCTION SHADER Does the noise disappear?
ENSURE ALL SURFACES HAVE SHADERS APPLIED
DEBUG RENDER WITH LAMBERT MATERIAL Does the noise disappear?
SHADER IS RESPONSIBLE Reduce bump value Does the noise disappear?
REDUCE BUMP VALUE OR INCREASE BLUR ON BUMP MAP
More in-depth investigation required Contact your Render Efficiency TD
Turn displacement value=0 Does the noise disappear?
REDUCE DISPLACEMENT VALUE OR INCREASE DISPLACEMENT MAP BLUR
Did the noise disappear?
RENDER!!!

Figure 2: A generalized flowchart to help diagnose noise in a raytraced render.

Raytracers are powerful but can be fickle and easy to abuse; simply turning up sampling without careful consideration will result in untenable renders that cannot be iterated in a reasonable time, which was our original problem. It is essential to consider where the noise is coming from. Early on in the process of developing a methodology for optimizing shots, I created a flowchart to assist in locating the source of noise (refer to Figure 2). This can be useful as a checklist to work through noise issues and most cases can be narrowed down or solved using such a chart.

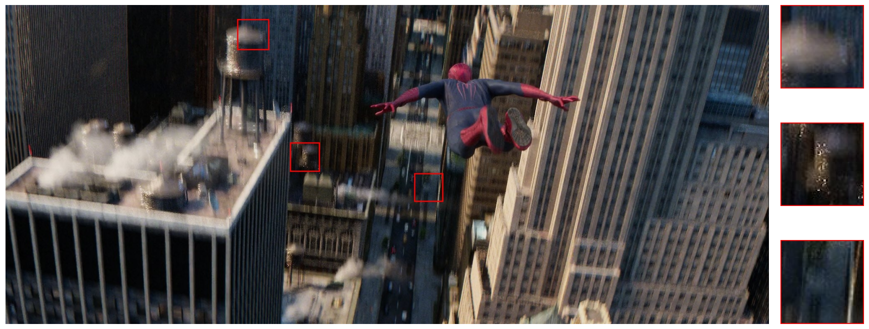
4 Examples and Best Practices

In this section, we will be examining the diagnosis and causes of rendering and performance issues. It is most useful to use actual examples from productions to put these items into practice, so we will be investigating two shots from *"The Amazing Spider-Man 2"*: ***fr1214***, A flythrough of NYC, and ***ee4418***, A shot with Electro transforming into electricity. Please note all times discussed in these examples are single-core render times.

4.1 Optimizing fr1214: The 'Spider-Man Flythrough'

I chose this shot because the problems present in this shot are common in renders that require optimization and offer a good test bed for the concepts outlined above.

Figure 3: The initial render: Max 48GB RAM and 155 hours per frame. Note the noise present throughout the image



4.2 Sampling

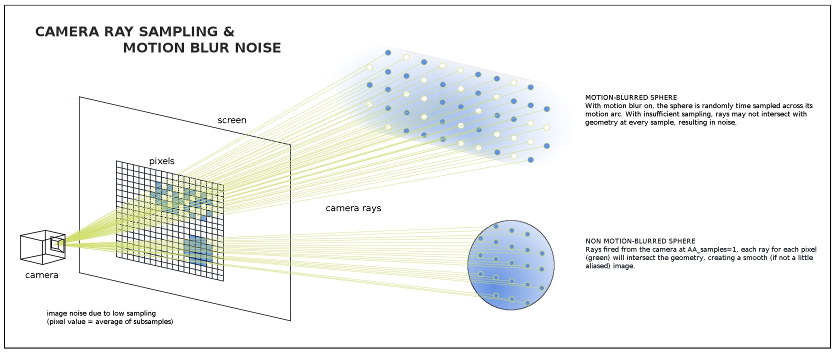
Figure 3 shows clearly the artefacts that need to be addressed: noise on the ceiling props, as well as general noise. This was exacerbated as Spider-Man flew through the chasms between buildings, combining high speed pans from light to shadow with highly specular materials and massive amounts of geo visible to the camera at any given moment. The artist was attempting to render the images at 4K to overcome problems in noise and sampling and were running past memory and render performance limits. With some optimizations, we were able to reduce the memory footprint down from 48GB to 19GB, bring the resolution back down to 2K and reduce render time from 155 single-core hours to 35 single-core hours while maintaining image integrity.

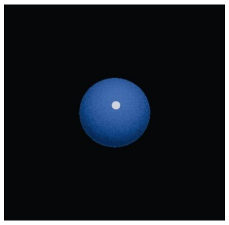
The first aspect we are going to tackle is sampling, our *Option 2*. But we cannot simply just increase sampling, that would result in even worse performance; our aim is to improve performance and reduce artefacts at the same time. The best place to start is to examine the logs. Looking at the log, we can easily find the settings for each of the main ray types:

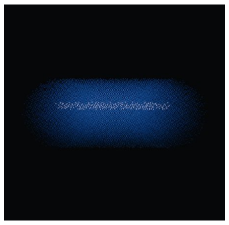
```
AA_samples: 8
AA_sample_clamp: inf.
GI_diffuse_samples: 3
GI_total_depth: 20
GI_refraction_depth: 16
```

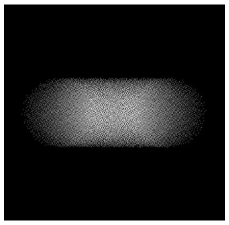
These are high settings, but we still have noise. It is a fast paced shot and any undersampling of the blur is going to show. You can confirm whether it is motion blur noise by checking the alpha; if there is insufficient rays to sample the geometry, then the alpha of the geometry will appear noisy. Note the two different sampling contexts that are occuring in Figure 4:

Figure 4: An illustration of motion blur in *SPI's Arnold*


CAMERA RAY SAMPLING &
MOTION BLUR NOISE
screen
pixels
camera rays
camera
image noise due to low sampling
(pixel value = average of subsamples)
MOTION-BLURRED SPHERE
With motion blur on, the sphere is randomly time sampled across its motion arc. With insufficient sampling, rays may not intersect with geometry at every sample, resulting in noise.
NON MOTION-BLURRED SPHERE
Rays fired from the camera at AA_samples=1, each ray for each pixel (green) will intersect the geometry, creating a smooth (if not a little aliased) image.


(b) motion blur=off 1 camera ray/pixel


(c) motion blur=on 1 camera ray/pixel

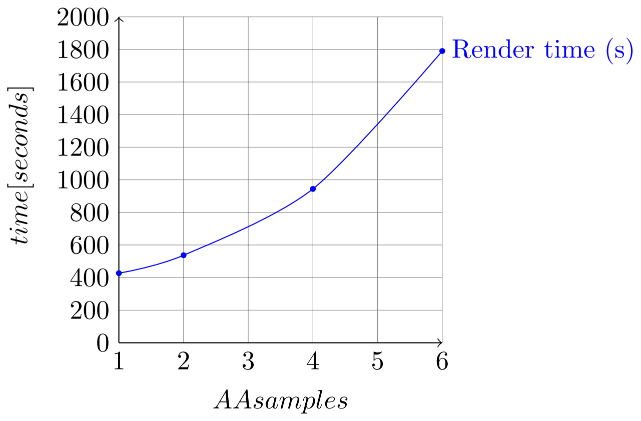

(d) motion blur=on, alpha

Motion-blurred sphere: With motion blur on, the sphere is randomly time sampled across its motion arc. With insufficient sampling, rays may not intesect with geometry at every sample. This results in noise.

Non-motion-blurred sphere: Rays fired from the camera will intersect the sphere even at `AA_samples` = 1, creating a smooth, if not a little aliased, image.

In a heavily motion blurred shot like this where we see motion blur noise, we must increase the camera rays, or `AA_samples` to decrease motion blur noise. As previously mentioned, if we simply just increase `AA_samples`, then our performance is going to decrease. To combat this, we can pull down other subsequent raytypes to compensate for the increase in camera rays. Let me explain: Remember how I mentioned earlier that all sampling in *SPI's Arnold* is "n x n"? What this means is that if you increase the camera rays, then all subsequent raytypes will increase exponentially; you will notice this occuring in Figure 5. But assuming other raytypes were rendering without noise, then any additional sampling is simply wasteful.

Figure 5: The impact of increasing *AA_samples* on render time.



Let us take our example shot; we have the original render which used `AA_samples` = 8, but there is noise. The lighter compensated for the noise by increasing `diffuse_samples` to 3 and each of the 10 lights to 2. This dealt with noise, but the motion blur was still noisy. So the `AA_samples` needs to be increased, but diffuse and light samples must then be decreased to avoid oversampling.

Note in table 1 that with a simple increase in the `AA_samples` to 12, we get a massive increase in total rays as seen in column 3. By column 4, we have reduced `GI_diffuse_samples` to 1 and each light to `samples` = 1, resulting in a massive reduction in total raycounts while maintaining high camera rays. Even though this render would have 1/3 the original number of rays, motion blur and geometry will be better sampled. It is worth testing renders with the aggressive optimizations; many cases will provide quality images.

Table 1: Ray numbers resulting from modifying *AA_samples*, *GI_diffuse_samples* and light *samples*

Ray type	original sampling	AA_samples increase only	aggressive optimization	measured optimization
AA_samples	8	12	12	12
GI_diffuse_samples	3	3	1	2
light *samples*	2	2	1	1
camera rays	64	144	144	144
diffuse rays	576	1296	144	576
shadow rays	2560	5760	1440	1440
total rays	**3200**	**7200**	**1728**	**2160**

Remember, it is not necessary to eliminate all noise; in comp, there is often some amount of grain that is added back in. We are after images that can match the plate, not perfection. Artists often end up increasing sampling in a vain effort to eliminate noise completely, when you only really need to approximate the grain that exists in the final comp.

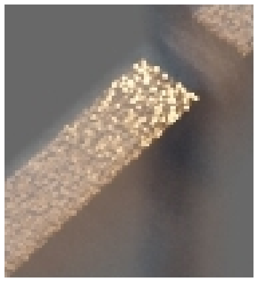
This scenario is a vast simplification of a complex issue involving many participating media and raytypes. But it is clear with even this limited example how we can work towards applying sampling only where we need it while keeping all other raytypes as close to the original sampling level possible. We can go even further: Do we need all 10 lights? Check the output of each; if we consolidate we can increase sampling on the remaining lights. This leads us to another issue: specular noise.

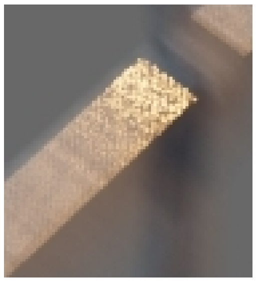
You will notice on some of the metal and other thin, specular geometry there is noise. This is a common phenomenon often mistaken for glossy noise, when in fact this is caused mostly by direct specular highlights from bright lights, not glossy reflections. This is not an *SPI Arnold* only problem, but common to all renderers.

Also, basically any high intensity, low relative-radius light source has the potential to create noise on a low-roughness specular surface. When the geometry is thin and motion blurred, this effect is exacerbated (Figure 6). There may be a myriad reasons why this condition exists in a scene. Sometimes it is an inherited rig where there is a light that has been set too high. Sometimes it is a specific sample setting that is appropriate in one shot but not others and the artist does not even consider there may be an issue until images return with untenable noise. Because this is not due to glossy rays, increasing the `GI_glossy_rays` will have no impact.

Increasing light samples in the lights can improve the direct specular in some cases, but it is important to isolate and test the lights responsible for the specific

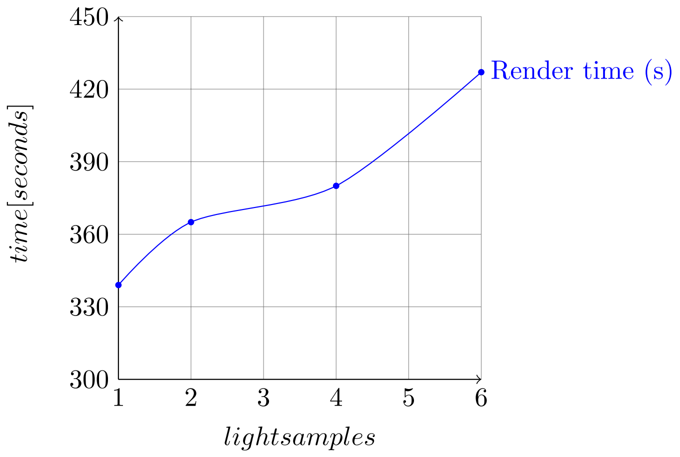
Figure 6: A close-up of a highly specular, motion-blurred object


(a) At $AA\_samples = 8$, artifacts are clearly visible


(b) At $AA\_samples = 20$, it still shows noise.

specular in question. Each light adds more samples to the scene, which adds render time (Refer to Figure 7).

Figure 7: The impact of increasing light samples for 3 lights on render time.



More often than not, it is simply a matter of increasing `AA_samples`. But here again we face a situation where all other sampling might be sufficient, even the motion blur on non-specular surfaces, but it is just these glints that are problematic. It is not always immediately obvious, but any specular that requires more than 12-14 samples in either the `AA_samples` or `GI_glossy_samples` should be considered for a workaround. The cause of the hot pixels can be varied. It is possible that the ray sampled a super-hot reflection, or the surface of an area light or the bright sunspot of a skydome. There are some simple tests that can be used to isolate and resolve each of these scenarios. Here are the steps to resolve specular noise:

1. ***In-camera solutions:***

   (a) *AA_sample_clamp* to limit the maximum permissible value of any given sample.
   (b) *light_decay_clamp* to limit the intensity at the light surface
   (c) Checking the skydome for any ultra-hot sources, such as sunspots.
   (d) Splitting out the responsible specular: While ugly, it permits targeted optimization without incurring the expense of any other scene elements. Isolate the geo and turn off all GI and material components, then render at ultra-high `AA_sample` rates like 25-30.
   (e) vector blur: Ugly, problematic but may deliver a smooth rendered frame when all else fails.
   (f) Motion blur a projection: render a single frame of the isolated specular, then reproject and render normally.

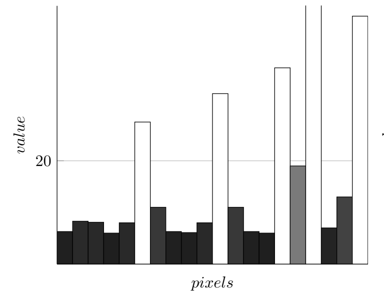
2. ***Comp solutions:***

   (a) Filtering in comp: Use of a median filter or equivalent to smooth the offending specular.

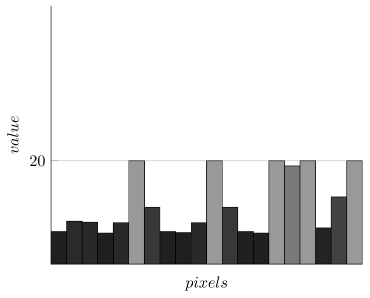
Clamping is a biased but effective way in which we can have absolutely no impact on render time or performance, but a substantial impact on the appearance of noise. By default, there is practically no limit to the value any sample can return. This is particularly true in the case of physically-based lights where the light intensity at the light surface can literally be measured in the millions. What this means is that you can get ultra-hot pixel noise in your scene. At the global level, we can adjust the `AA_sample_clamp` can help control these highlights. Let us look at a theoretical line of pixels in a noisy area of an image.

In the row of twenty pixels in Graph (a) of Figure 8; the white pixels result from subsamples returning very high values.

Figure 8: Representation of *AA_sample_clamp*.



(a) *AA_sample_clamp* = inf. (default)

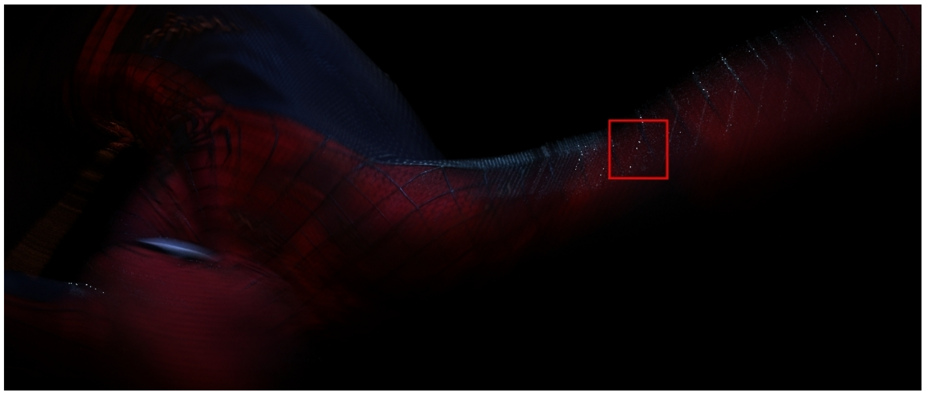


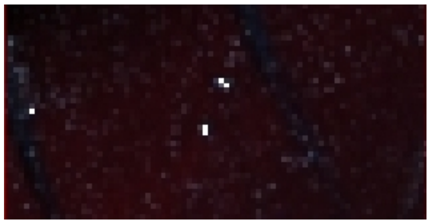
(b) *AA_sample_clamp* = 20

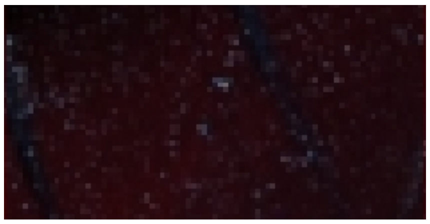
This is in contrast to the surrounding pixels which have a much lower value; this difference shows as noise. In Graph (b) we have clamped the number to

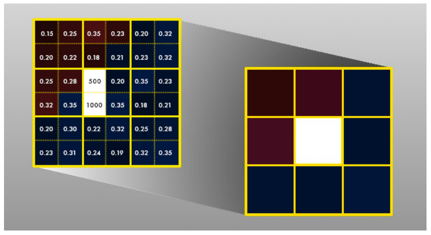
20. This means any subsample values above 20 are clamped and the new pixel value will be much closer to the values of the pixels around it. Let us take this information and apply it with a closer look at Spider-Man in our shot:

Figure 9: *AA_sample_clamp* in use to control white hot pixels on Spider-Man's arm.

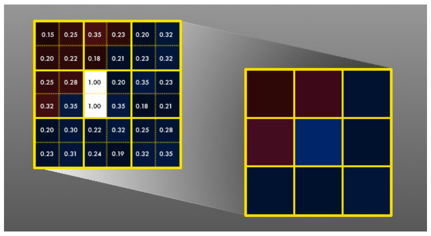

(a) the original image with the analyzed section isolated


(b) A close-up of the problem pixels


(c) A close-up of the same pixels with the *AA_sample_clamp*=12



(d) A diagram of the pixel samples for*AA_samples=2*

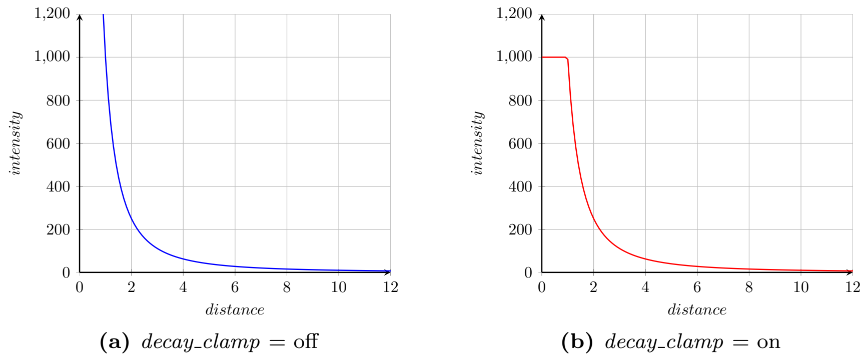


(e) A diagram of the same pixels with the *AA_sample_clamp*=1

Note the white-hot pixels on Spider-Man's arm. If we tried to increase samples to resolve this noise, we would have to raise the number extremely high, which would not be renderable. Or we might consider splitting out the spec pass to try and oversample that alone. If we reduce the *AA_sample_clamp* to a low number, we can avoid any oversampling. You can see in Figure 9 that the ultra-hot pixels have somewhat dissipated. This is because the clamp has reduced the average value across the subsamples for each pixel, resulting in a more homogeneous array of pixel values.

Another method that can be used to control these spec glints is by limiting the intensity of a light at its surface. When implementing a physically plausible light, the intensity at the surface can reach millions of units. While this method still can result in high sample values, the restraining of surface intensity often makes a visible difference in the appearance of noise and in combination with other methods can be a very effective tool to control noise without any impact on performance (Refer to Figure 10).

Figure 10: With *decay_clamp* on, brightness never gets above the intensity at the decay radius, which prevents any ultra-hot pixels



(a) *decay_clamp* = off

(b) *decay_clamp* = on

The use of floating-point images for skydomes is a common method in ray-tracing to recreate skylight. Even though we fully utilize Importance Sampling [2] on the skydomes, if that image contains a sunspot, we can occasionally create conditions ideal for noise: a small, ultra-bright light source that can be hard to sample, particularly when the object reflecting the skydome has any kind of bump mapping, displacement or motion.

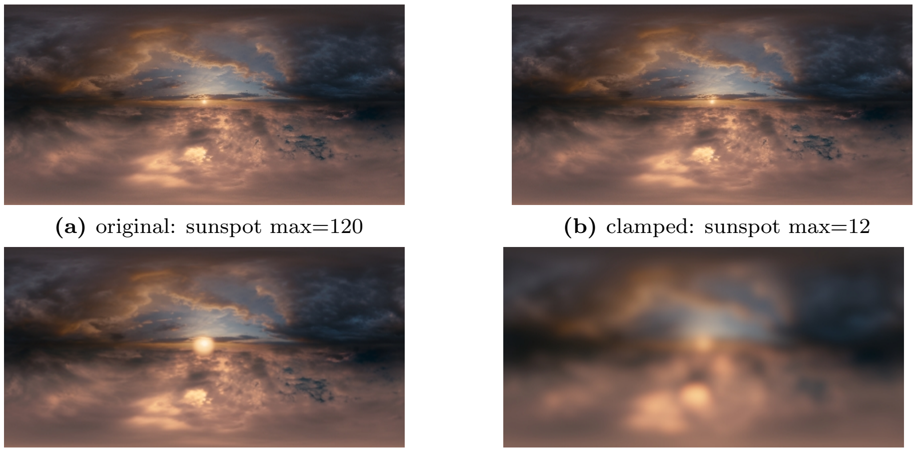
In these cases, substitution of the skydome texture for a homogenous color can determine if this is the root cause: the noise will simply disappear. If it is determined that the sydome is responsible for noise, there are several options available:

1. **Clamping the image:** postprocessing the image to clip any high values can create a map that is still an excellent skydome but eliminates any ultra-high values that can cause noise
2. **Enlarge the sunspot:** Increasing the size of the sunspot makes it easier to sample.
3. **Blur the skydome:** Similar to item 2, this makes the sunspot easier to sample, reducing hard edges between bright/dark boundaries and increasing the size of the sunspot.

Figure 11 shows the resulting map from each of these methods.

Modifying the skydome in this way is an effective way to use our *Option 3*; there is no performance penalty and it requires no change in sampling to reduce

Figure 11: Modifying the skydome to help noise

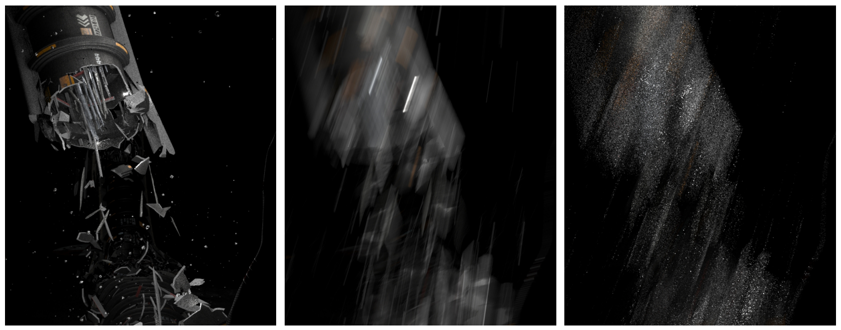

(a) original: sunspot max=120 **(b)** clamped: sunspot max=12

(c) resized: max=120, but larger **(d)** Blurred, max=120, softer

noise. If we absolutely need to retain the value and sharpness of the skydome, then we can use clamping in other ways to reduce artefacts.

It is recommended that no brightness be applied to an image in comp. If you need to brighten an image, use the comp to guide how much to dial the lights in the shot and re-render. If you increase the brightness of a render, this will exaggerate any subtle noise in the image and will create noise where none was noticeable before and may result in unnecessary efforts to resolve noise. Note in Figure 12 that I have increased the gamma 3.5 stops to show the noise clearly:

Figure 12: *motion vectors vs. motion blur* The highly specular lighting and materials combines to make this case one where motion vectors, although a biased solution, more preferable under a tight deadline.

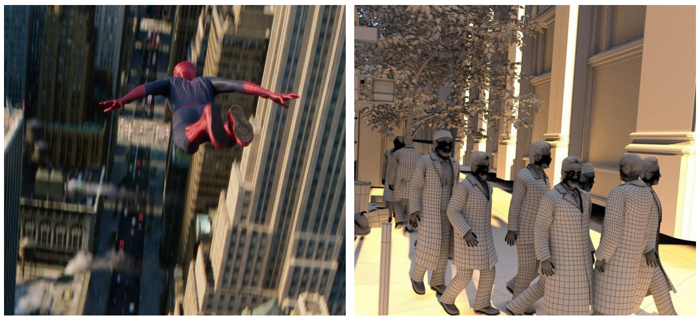

(a) No motion blur **(b)** *with motion vectors.* **(c)** *True motion blur.*

However, in the case of the flythrough, the increased `AA_samples` was sufficient to control the specular. With sampling under better control, we can look again at our Global Settings. It is time to see what we can do about *Option 3*.

4.3 Subdivision and meshes

Elimination or reduction of subdivisions on cars, crowds and other small geometry was essential. The vehicles and crowd geometry were developed at 2 subd iterations, even though there are hundreds of crowd avatars and a similar number of cars, yet none above a few pixels high. We may not see it, but the raytracer will still subdivide them all. In one instance, when we actually moved the camera to street level and rendered a wireframe of the view, we found the following:

Figure 13: Subdivided crowds with *subd_iterations*=2. Note the camera is 1000ft away.



(a) the camera position **(b)** ...and the subdivisions at street level

This complexity was widespread, even though it was not visible to camera. Without optimization, the cars and crowds will easily consume gigabytes of RAM and in our shot was responsible for the majority of the 48GB consumed (Figure 13). As stated previously, it is important in sequence layout the context in which particular assets will be shown. Set the `subd_iterations` to 0 to prevent any unnecessary subdivision. This cuts the memory consumption significantly. As the edge profile is the main concern with most subdivision, a simple check with a wireframe render with original `subd_iterations` and at zero allows the user to adjust the iterations so that there is no discernible difference between the two renders.

Many wide shots did not need any subdivision at all and most objects could be converted to polymesh. Why would we go to the effort to convert to polymesh when we could simply turn the iterations to zero? Even with iterations=0, *SPI's Arnold* will still attempt to move vertices to the limit surface. The process to move a vertex takes a small amount of time, but if you have a massive scene with incredibly dense meshes, this can make a meaningful difference in scene build time. Ok, then, why not simply turn the subdmesh subdivision off using "ignore subdivision" in the globals? Does not that do the same thing? The short answer is no, it is quite different: when an object is a polymesh, it will still displace. When an object simply has "ignore subdivision", the displacements no longer affect the mesh.

So it is recommended that one of the first renders should be a wireframe test at the original settings and as a polymesh. This will give you an idea what needs to be subdivided, what can be left at iterations 0 and what can be converted to polymeshes. In the case of this shot, most of the objects could be converted to polymesh with no visual impact.

4.4 Wrapping it up

The resulting combination of sampling changes and scene modification actually cut render time approximately to 30% of the original. As you can see in Figure 14, the presence of noise is also markedly reduced. To recap, let us cover what we have altered.

Figure 14: The optimized render: 19GB RAM and 35 core hours (down from 155 hours), with significantly reduced noise.

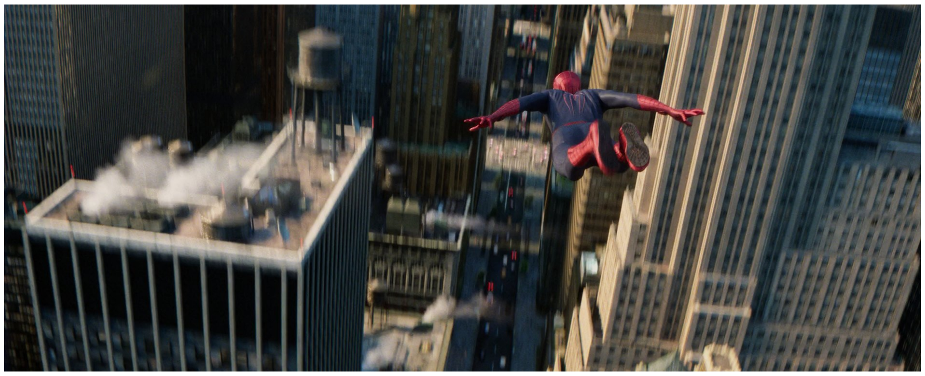


Image courtesy Columbia Pictures ©2014Columbia Pictures Industries, Inc. All rights reserved.

1. ***For geometry:***
   (a) Limited unnecessary *subdivision iterations* on meshes.
2. ***For sampling:***
   (a) We *increased* the `AA_samples` from 8 to 12. This eliminates motion blur noise.
   (b) We reduced the `AA_sample_clamp` from infinity to 20, which limits the max of any given sample.
   (c) We reduced the `diffuse_samples` from 3 to 1. This compensates for the increase in camera rays
   (d) We reduced the `total_depth` from 20 to 12. This limits the ray depth and the possible number of rays.
   (e) We reduced the `refraction_depth` from 16 to 8. As for the `total_depth`, this limits the number of rays in the scene.

3. ***For volumes:***
    (a) We increased the `Volume_step_size_mult` from 1 to 5. At this distance and this speed, this reduction in the detail of the volume is not noticeable but speeds up volume rendering.
    (b) We increased the `Volume_shadow_step_size_mult` from 1 to 10. The volume is thin and shadows do not have a significant impact on the scene; reducing the detail of the volume shadows is not noticeable but further speeds up volume rendering.

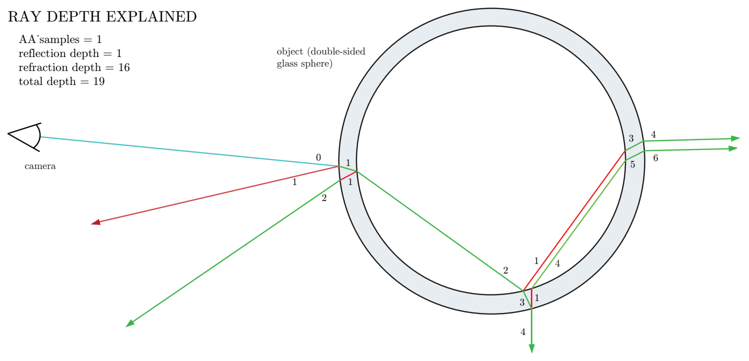
4.5 Optimizing ee4418: Electro

Following on from the investigation of the Spider-Man Flythrough, I would like to delve a bit deeper into the topic of optimizing opacity and transparency. Meshes and media with opacity less than 1.0 present their own challenges and with optimization we can decrease render times by orders of magnitude.

4.6 Refractions and rays

The challenges boil down to the fact that if we can see through an object, we now have to perform all the same sampling again for any new intersection as the ray traverses the scene. Depending on ray depth, this can repeat multiple times for every ray tree fired. This can be expensive, and if we are not careful with our shading pipeline, extremely wasteful. Let us start by taking a look at the way a ray propagates through the scene.

Figure 15: Ray depth in a raytracer. Each ray type is numbered: camera rays(blue), reflection rays (red) and refraction rays(green). If you count through each ray path, no ray depth limit is violated, yet you can see how complex even relatively small limits can become.



This setup is pretty simple, but you can see in Figure 15 how even a single ray can exponentially increase in complexity. Remember that the moment the ray goes through that first glass surface, it is now considered a refraction ray. This situation is often encountered in refractions through the outer layer of eyeballs or when the camera is shooting through a glass pane. In that case, only `AA_samples` and `GI_refraction_samples` are our only controls for resolving noise and we want to avoid increasing `AA_samples` when possible. Things become even more complex when we consider how we want to handle shadows.

So let us take a look at an example with Electro. The shot, *ee4418*, shows Electro transforming into electrical energy (Refer to Figure 16). When these frames came to our attention, the particle renders had been the slowest and had been running for a week. A lot of the frames were getting stuck at 95%. In the logs for successful contiguous render frames, the time between most buckets up until 95% was approximately 3 mins, with the remaining 5% taking over 70 single-core hours.

Figure 16: Electro transforming in ee4418. Before optimization, the particle renders were taking over 70 hours.

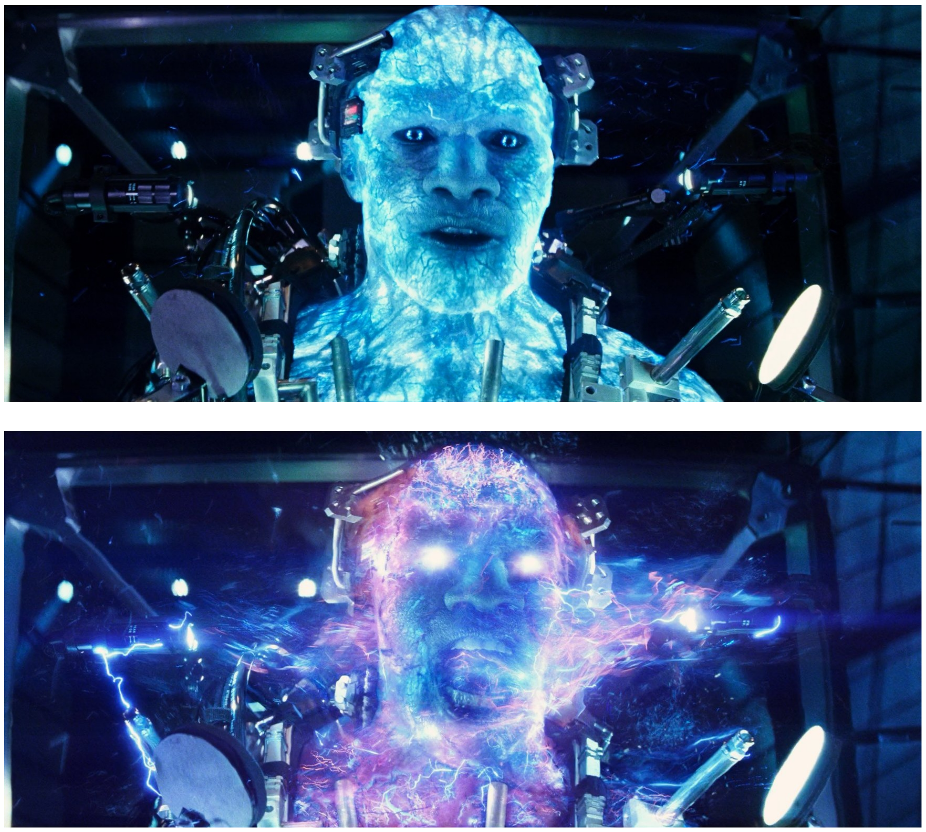
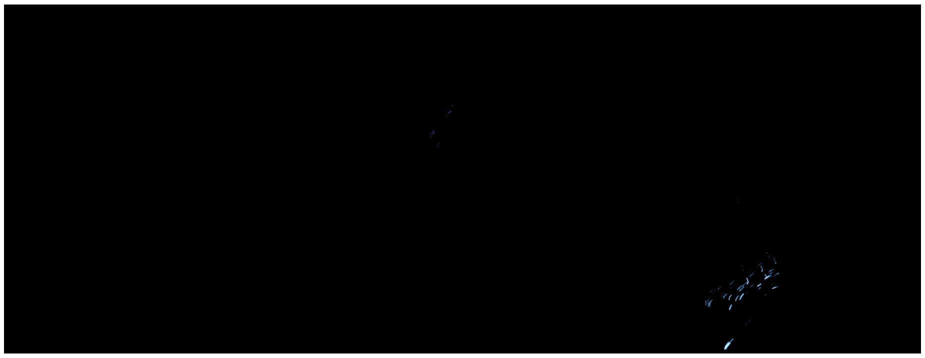


Image courtesy Columbia Pictures ©2014 Columbia Pictures Industries, Inc. All rights reserved.

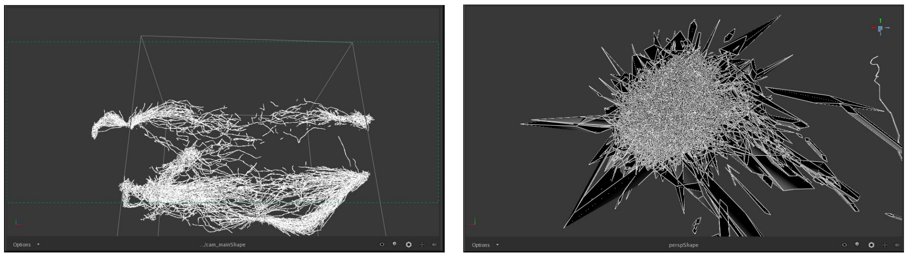
As can be seen in Figure 17, there is not too much that is visible, so upon first glance, it is hard to determine what the problem is. Note that barely anything is visible in the frame; that is alot of expense for little payoff. Checking the geometry in the viewer, the problem becomes immediately visible.

Figure 17: An example frame from a sequence that sat at 95% completion for *70+ hours!*



The bucket that represented the last 5% lay on the loci for spark generation. Zooming in, they were tiny bundles of millions of triangles, taking up less than a pixel in screenspace (Refer to Figure 18). Literally, a single pixel was taking orders of magnitude longer than the rest of the frame! Basically, the problem is the transparency combined with the sheer number of overlapping triangles. This would be a problem for any raytracer: it relates to how the renderer determines where any given ray intersects geometry [1].

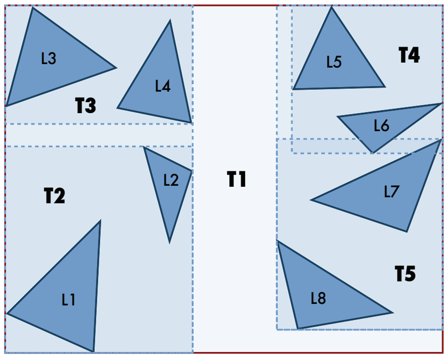
Figure 18: The fx passes in the *Katana* viewer. Note the jumble of polys shown exists in a space smaller than 1 pixel in screenspace



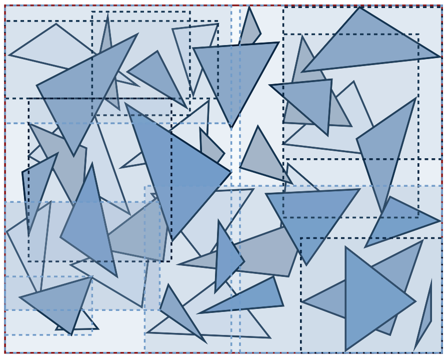
(a) The geo in the Katana viewer **(b)** and close-up!

Normally, the camera ray would traverse through the triangles until opacity hits a defined threshold, where it will stop calculating and exit. In this case, the opacity is zero, so there is no opacity accumulated to cause a ray exit and so Arnold has to try and determine the intersection with hundreds of thousands of triangles for each ray. With the enormous amount of overlapping meshes, the triangles cannot easily be isolated, so the raytracer effectively must test each one in a linear search (Refer to Figure 19).

Figure 19: Acceleration using BVH trees



(a) In a simple scene, the triangles are easy to partition.



(b) In a complex scene, with many overlapping triangles, the scene can become impossible to partition.

Camera rays are used to calculate transparency so this would still be true even if *all* non-camera rays are turned off. The combination of low opacity plus the massive numbers of overlapping triangles are what takes 70+ hours.

So any solution will rely on either reducing geometry or increasing opacity to force the ray to stop calculating faster. You can double check the veractiy of this theory by eliminating the opacity map and rendering with a lambert shader; the render time is reduced to seconds! Obviously a lambertian render is not deliverable, so here are some possible solutions:

1. *Increase opacity in and around those bundles* so that the ray stops faster or implement a transparency threshold that could be increased to cull rays even faster.

2. *Re-simulate with less geometry in those bundles and create a ramp on the width* This is so that the width within the bundles is small, which would lead to fewer intersecting traingles.

3. *Eliminate the OpacMap altogether* by using color. While not as sophisticated, if these effects are to be transparent and heavily processed, the difference may not be particularly noticeable.

5 Conclusion

Raytracing presents an opportunity to implement a robust, consistent and efficient rendering pipeline in a VFX facility. Since adopting a pure raytracing environment at SPI, the level of imagery and performance has been propelled to a new echelon of photorealism and sophistication. But the system relies on the thoughtful optimization of sequences to prevent untenable render times and substandard performance; it would be easy to write off an entire rendering pipeline for lack of knowledge on rendering optimization! In this discussion, we have presented a number of strategies technical directors can use to pre-empt and address inefficiencies and wastefulness.

The key is to understand the conditions that lead to problematic renders and address them before they are incorporated into a shot. Develop a process and a checklist to help progress through the optimization process: *Are all the textures tiled?*, *Has light linking been used effectively?*, *Have I checked the Subdivisions using a wireframe render?*, *Am I using a lot of direct light sources?*, *Can I reduce the number of lights?*, *Do I have lights with excessively large radii or intersecting geometry?*, *Is there a lot of noise on bumpy objects?*, *Do all of my direct light sources need to contibute to the GI solution?* and *Have I optimized the ray visibility of objects in your shot?* These questions and others you develop can help to overcome our *Option 2* and *Option 3* and gain artefact-free renders, fast.

However, the solutions here are the tip of the iceberg and this work extends far deeper. In this presentation we have not even discussed subsurface scattering, hair or explored optimizing volumes. At SPI, it has been a process over years to collate, organize and formalize strategies for render optimization and in hand with that, education of artists and supervisors about best practices for producing the best images in the shortest amount of time.

6 Acknowledgments

Thanks to Sony Pictures Imageworks and CTO Rob Bredow and for supporting the continued development of raytracing at SPI; the SPI Arnold Team both past and present, Larry Gritz, Christopher Kulla, Cliff Stein, Alejandro Contya and Rene Limberger for your inspiration and guidance.

References

[1] T. L. KAY and J. T. KAJIYA. Ray tracing complex scenes. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, volume 20, pages 269–278, 1986.

[2] E. VEACH and LEONIDAS J. G. Optimally combining sampling techniques for monte carlo rendering. In *SIGGRAPH 95 Proceedings*, pages 419–428. Addison-Wesley, 1995.

FRAMESTORE

HOW THE USE OF RAYTRACING & PHYSICAL PLAUSIBLE SHADERS CHANGED COMMUNICATION BETWEEN VFX ARTISTS AND FILM ARTISTS IN THE MAKING OF GRAVITY

Presented by **Paul Beilby**, Head of Lighting and Look Development at Framestore
Siggraph
August 2014

Gravity : Start Early and Change all the Rules of Post Production

Everyone together

- Early involvement
- Director and Producer in the building though out the project
- Pre and post-production happened at Framestore
- Framestore heavily involved with shooting process
- Close relationship between Framestore, Alfonso Cuaron, Producers, Editorial, Art Dept and other departments

PRE-VISUALISATION & LIGHTING PLAN DEVELOPMENT

IN HOUSE PREVIS FIRST STEP IN COLLABORATION

The 'first draft' of the film

- Storyboards used for first pass
- Alfonso Cuaron worked directly with framestore animators during previs
- Long shot durations
- Vcam - Mocap virtual camera
- Framestore collaboration established with The Third Floor
- Framestore and 3rd floor are able to share assets
- Allows head start in planning for all post production departments

PREVIS
Final Previs

- Shot Layout Is Finalised before shooting begins this gives huge advantages in planning what will be necessary in post production
- This knowledge allows us to evaluate current and future rendering techniques

Framestore
20/01/11
012_AS_0500_v038
monoPlayblast_lrussell_as012_0500[1001-3077]jpg
21mm
Shoot Layout Animation
1062

PRE-LIGHT
Pre-shoot Cinematography

- Working closely with Director of Photography: Emmanuel "Chivo" Lubezki
- Planning
  - Light Directions
  - Shadows
  - Hard light
- Physically Based Lighting
- Uses approximate low res assets
- Exterior Lights
  - Sun
  - Earth
- Common language

FrameStore
28/04/11
001_AH_2000_v036
ah001_2000_reviewcomp_v015[1001-1709]dpx
Lighting
1209

TECHNICAL REQUIREMENTS OF PRELIGHT

- Accurate physical plausible light interaction
- Real time feedback (nearly)
- Complete flexibility for lighting changes
- Common language with Director of Photography
- All “virtual” lighting must be able to be recreated on set
- Very Long Takes

FRAMESTORE LIGHTING AND RENDERING PIPELINE PRE-GRAVITY

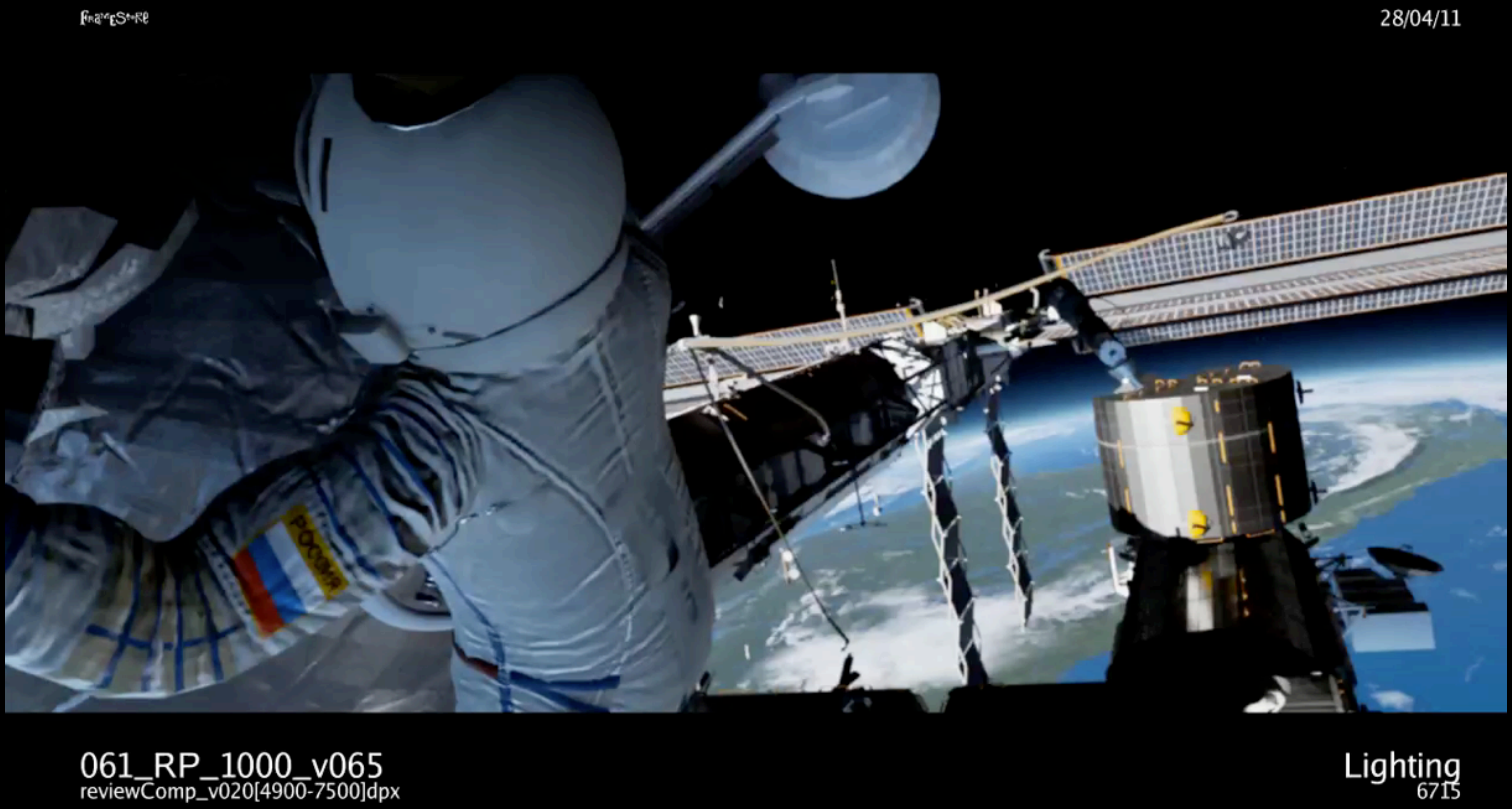
Reyes & Baking

- Completely Reyes based pipeline
- All secondary Light bounces are produced approximately by baking
- Slow set up time
- Requires Strong Technical Knowledge
- Not Real Time!
- Not flexible to quick turn around lighting changes
- Not always physical plausible
- Many virtual lighting techniques do not have comparable on set solution

GRAVITY PRELIGHT PIPELINE

Moving to path tracer

- IPR gives instant feedback to lighting changes
- All secondary Light bounces are accurately represented
- Fast set up time
- Common language with Non VFX artists
- Always physical plausible
- All lighting techniques have comparable on set lighting technique
- No heavy bake assets so can be done without connection to main studio servers



Advantages of Prelight Stage

- Many common post production challenges are solved before shoot begins
- Builds much stronger relationship with on set crew
- Allows both on set and vfx artists to plan lighting much further ahead without onset prices!
- Gives actors and actresses excellent visual feedback while shooting
- Gives forewarning of the lighting and look development challenges ahead

What We Learned During the Pre light Stage

we are going to need a bigger boat!

- We cannot assume the sun will be a common position!
- All of the assets will be viewed from wide to extreme close up from all angles in extreme lighting conditions
- Much of the lighting will be indirect
- A huge amount of detail will be required on all assets
- The shots are extremely long so many vfx short cuts will not be able to be used i.e.all assets are hero assets and require hero detail
- The shot length means that all lighting transitions will happen during the shot
- Most shots will be 90% CGI for most of the movie
- Only Photo real lighting and assets will be accepted by the Director.
- CONCLUSION : Our current reyes rendering pipeline is not going to work for our Production Quality Assets.

CONVERTION FROM VIRTUAL PATH TRACER LIGHTING TO ONSET

PLANNING

How to how do we covert from virtual to on set lighting in zero-g

- What did we need to consider?
  - Long shots in Zero G
  - Exterior shots: only shoot faces
  - Light rigs
  - Break long shots into “shoot setups” with “beats”
  - Add bounce cards
  - Lighting transitions
  - Rotating environment instead of actor

- Opening Shot
  - 12.5 minutes
  - 17 setups joined together

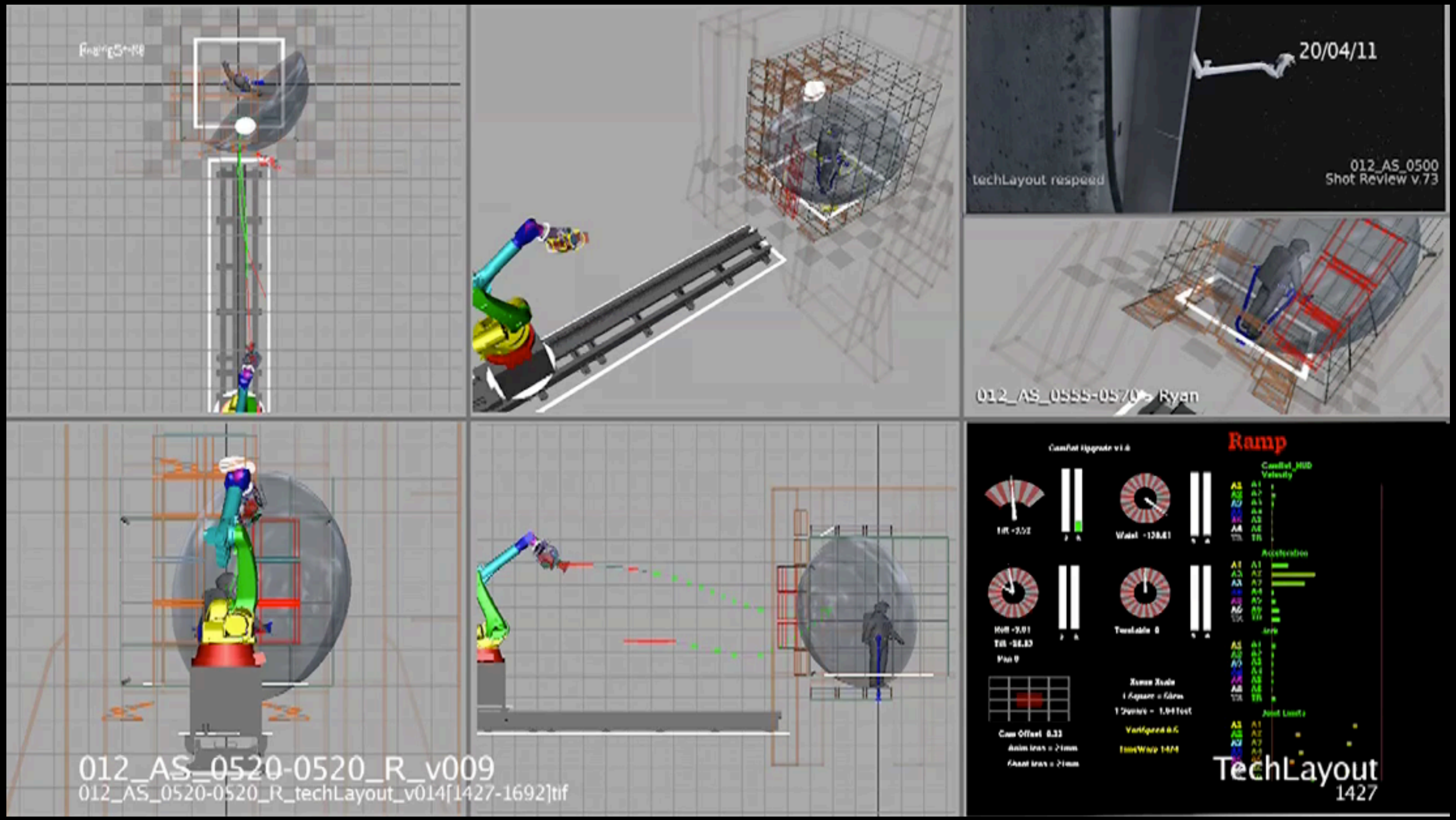
- Initial Tech Vis
- How can each setup be shot?
- How do camera and light move?

TECH-LAYOUT

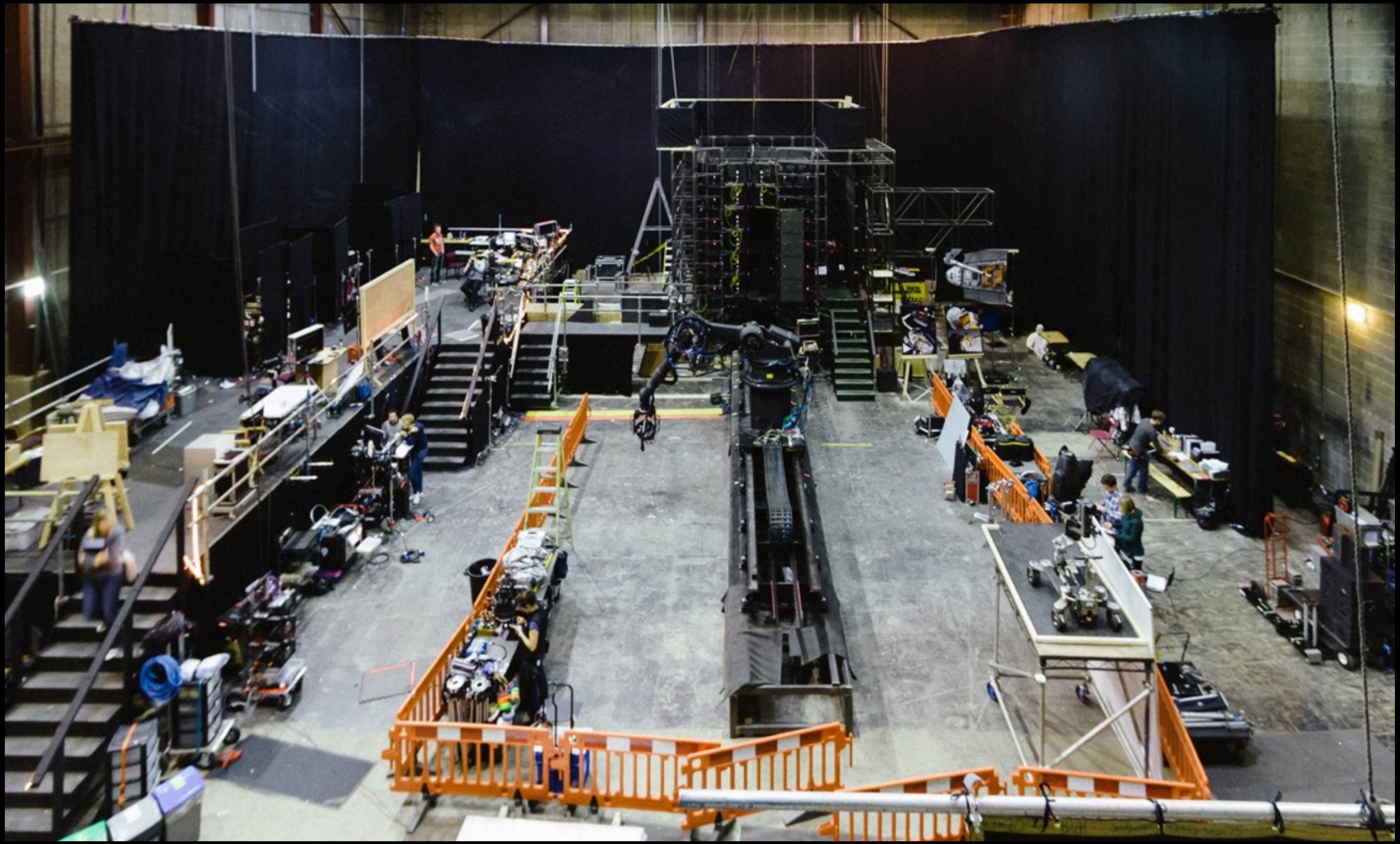
Converting vitual cameras and lighting for on set shooting

- Generate Virtual Cameras
  - From point of view of actor
  - Apply "reversed" animation moves to give the impression extra movement and rotation of the actor
  - Check safety of actor
  - Robot/Rig tolerances

- From Prelight generate environment maps from point of view of actor to use as :
  - Eye line Markers
  - Character lighting in LightBox



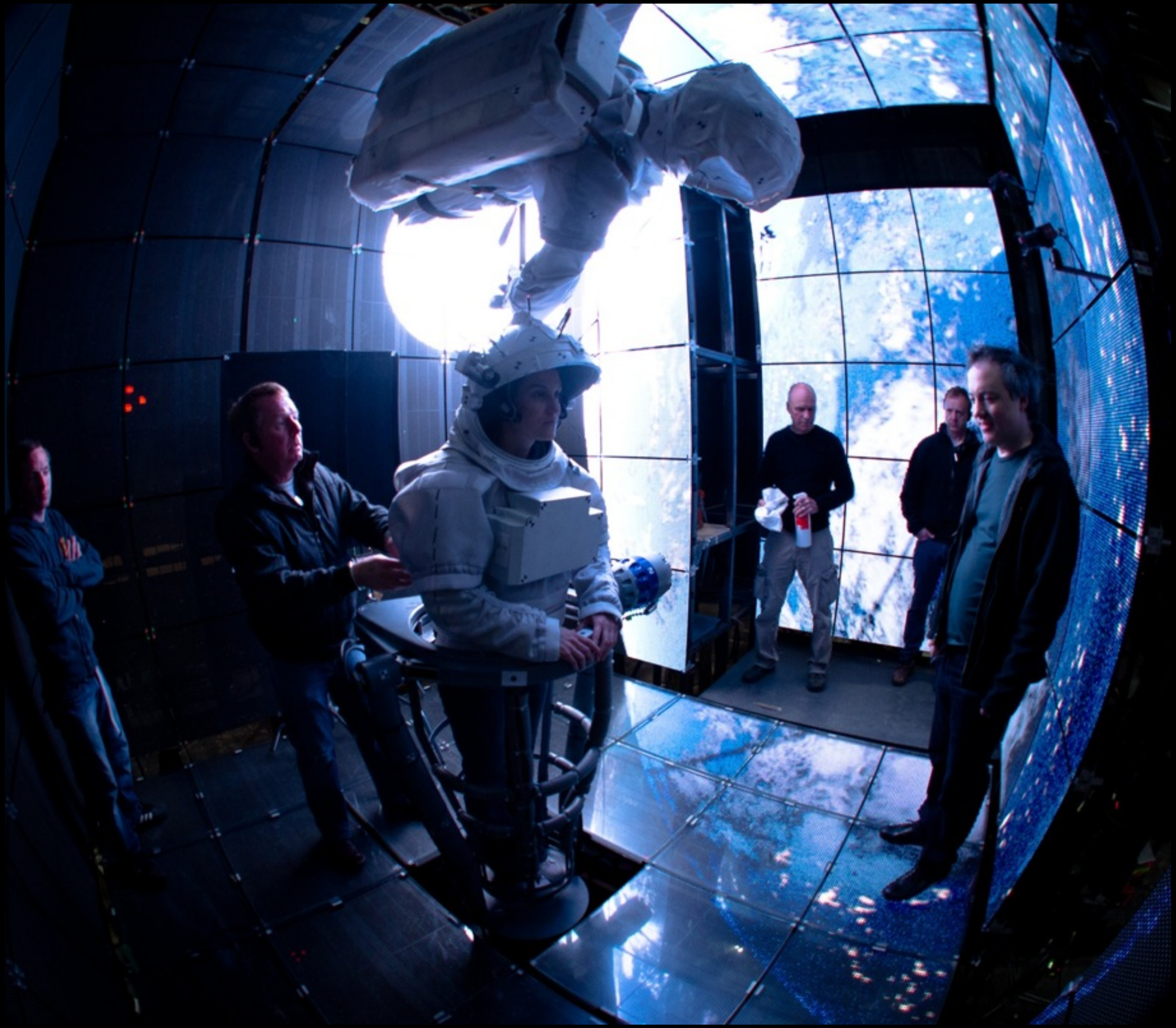
SHOOTING GRAVITY
R-Stage



LED PANELS

Interactive Lighting Environment
Path tracer generating environments
Projected around the actor

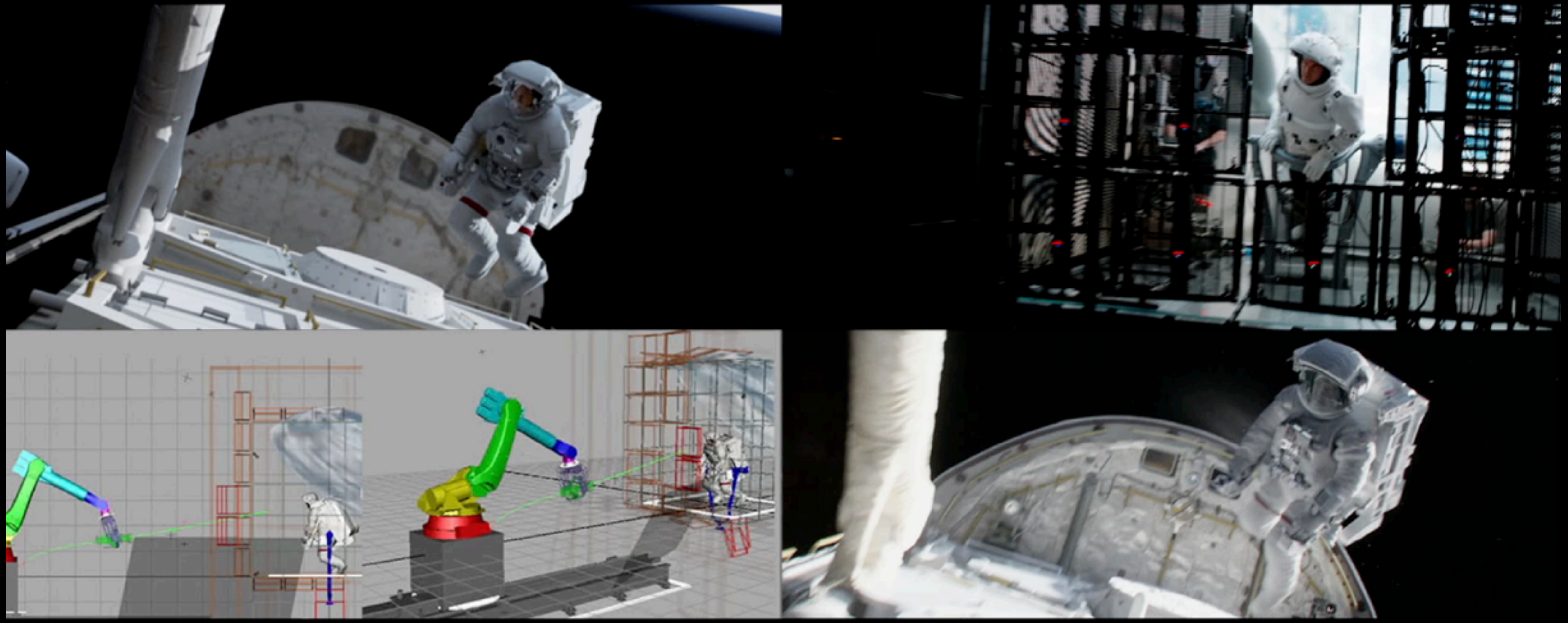
- The “light box”



SHOOTING IN THE LIGHTBOX

Previs Animation to SFX Robot Rig

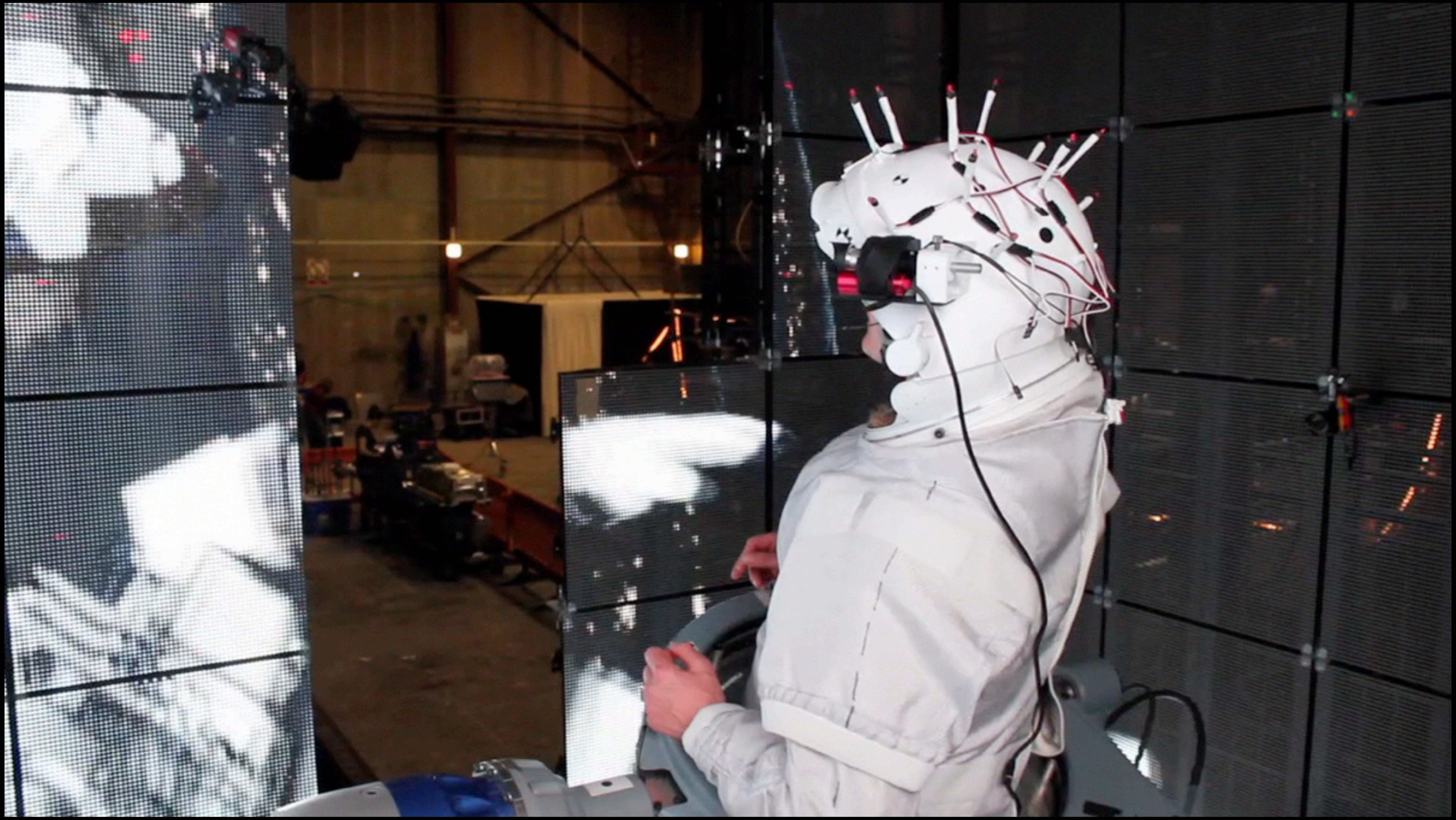
- A typical Shot
  - Lightbox
  - Robot
  - Tilt Rig



VFX Action Lighting and Feedback for Actors

Inside the lightbox

- All robot moves tested
  - Gives visual feedback of virtual action
  - Safe
  - Eyes reflect action for integration
  - Lighting comes directly from VFX assets
- Small adjustments
  - On the stage
  - Shared Tools
  - Framestore/Bot & Dolly
- Bigger adjustments
  - Back office



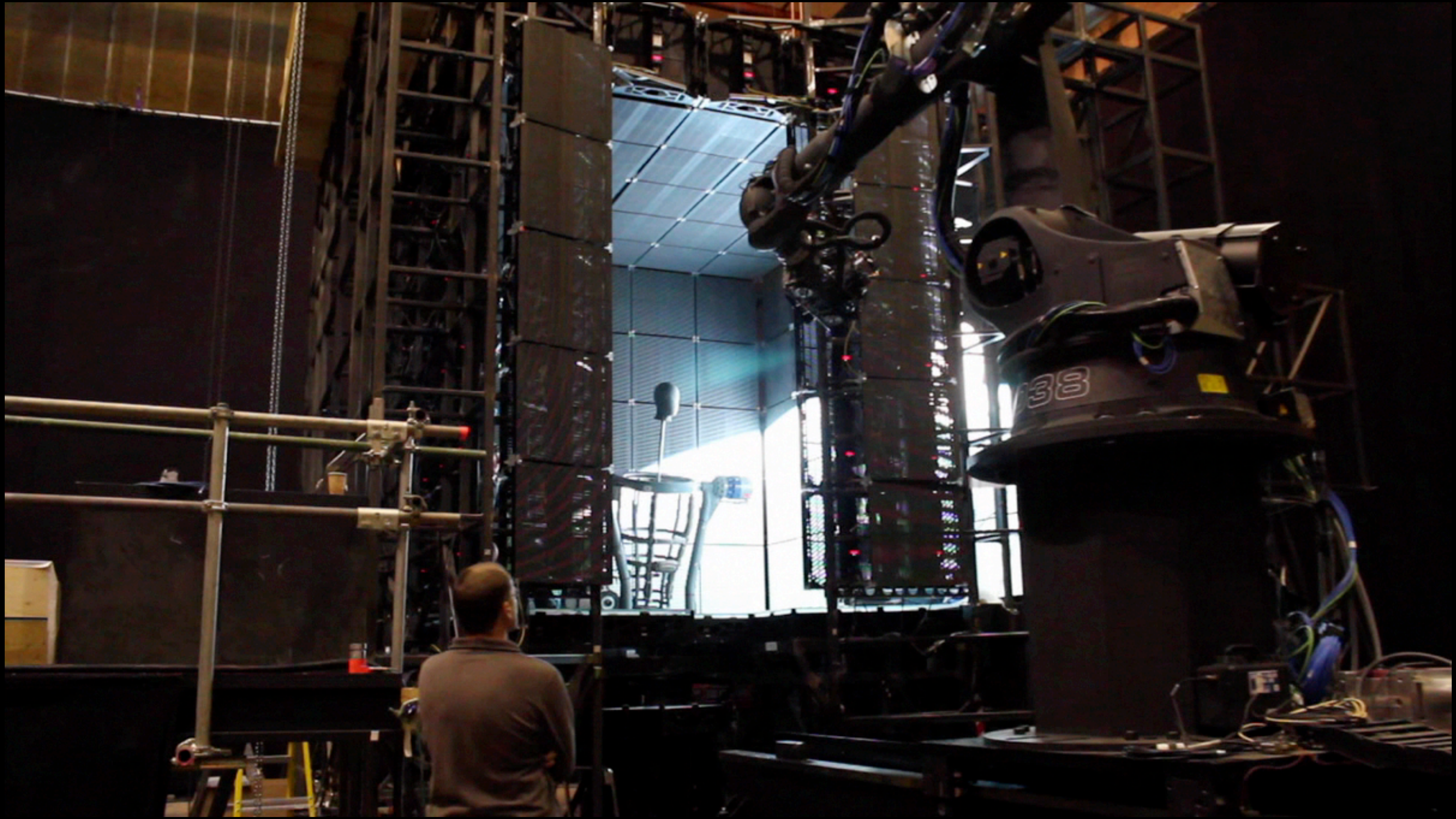
SHOOTING GRAVITY
An unusual way to make a movie

- Many required on set lighting direction adjustments are applied in realtime by Framestore “on set” Lighting Team and new lighting data is supplied back to “Virtual“ Lighting Team back at the studio

LIGHTING ENVIRONMENT SHOOT TESTING

Framestore Shepperton

- All robot moves tested
  - Within specifications
  - Clearance
  - Safety
  - Cabling
  - Lighting
- Small adjustments
  - On the stage
  - Shared Tools
  - Framestore/Bot & Dolly
- Bigger adjustments
  - Back office



THE LIGHT BOX
Physically accurate Lighting Integration

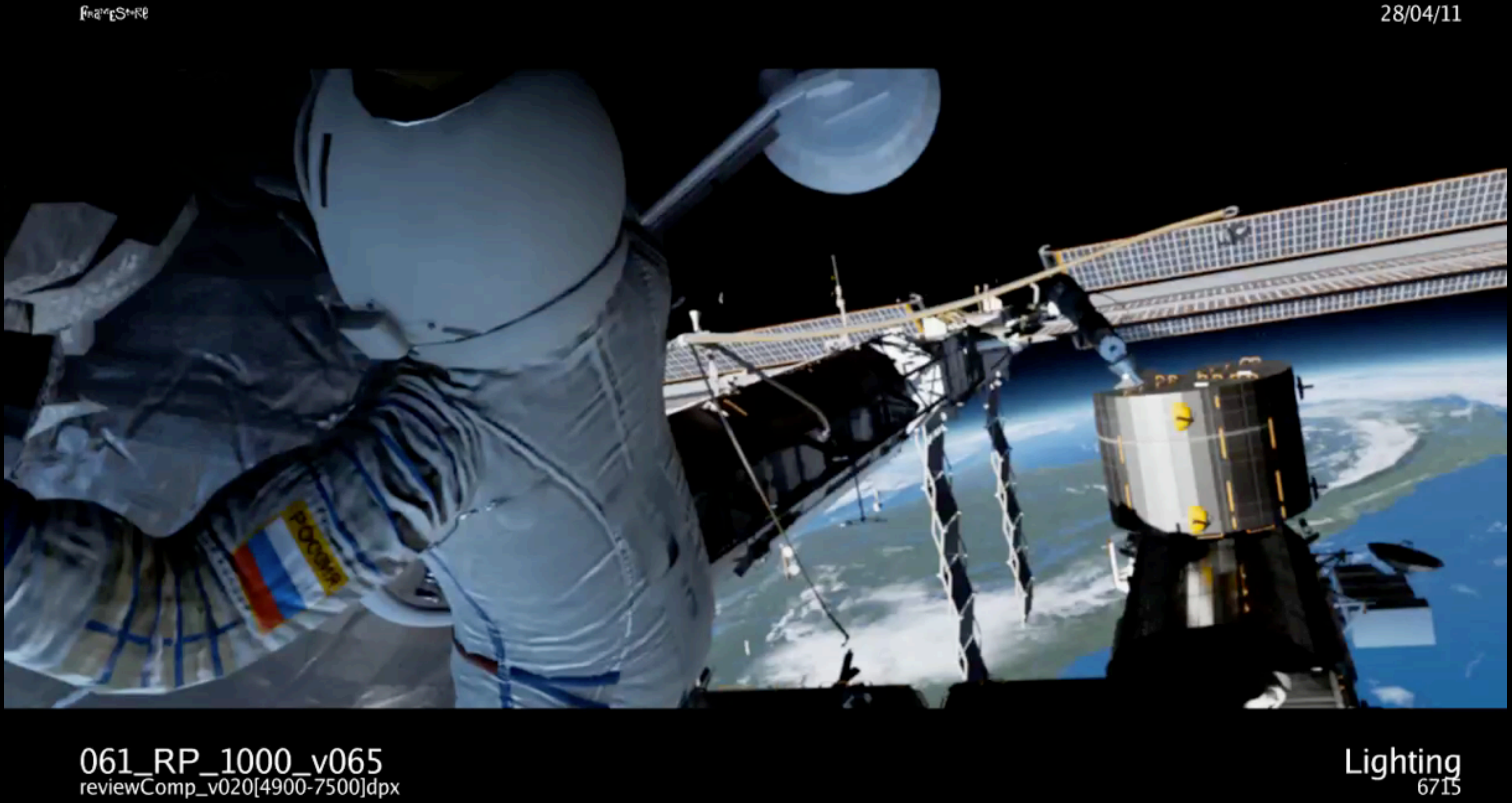
- Animated realtime lighting transitions
- Actors eyes reflect virtual action for increased integration
- Accurate bounce light from inside of helmet

THE LIGHT BOX
Adding to the box

- Combined with traditional lights to match virtual directional lights with hard shadows
- Robot or Manually Controlled

SHOOTING GRAVITY
Lightbox & Special Cases

- Complex Lighting Interaction with Soyuz
  - Lightbox
  - Robot Controlled Lighting
  - Turntable
  - Set Piece Bounce Cards
  - Wires
  - Animated action assets environment maps generated in mental ray

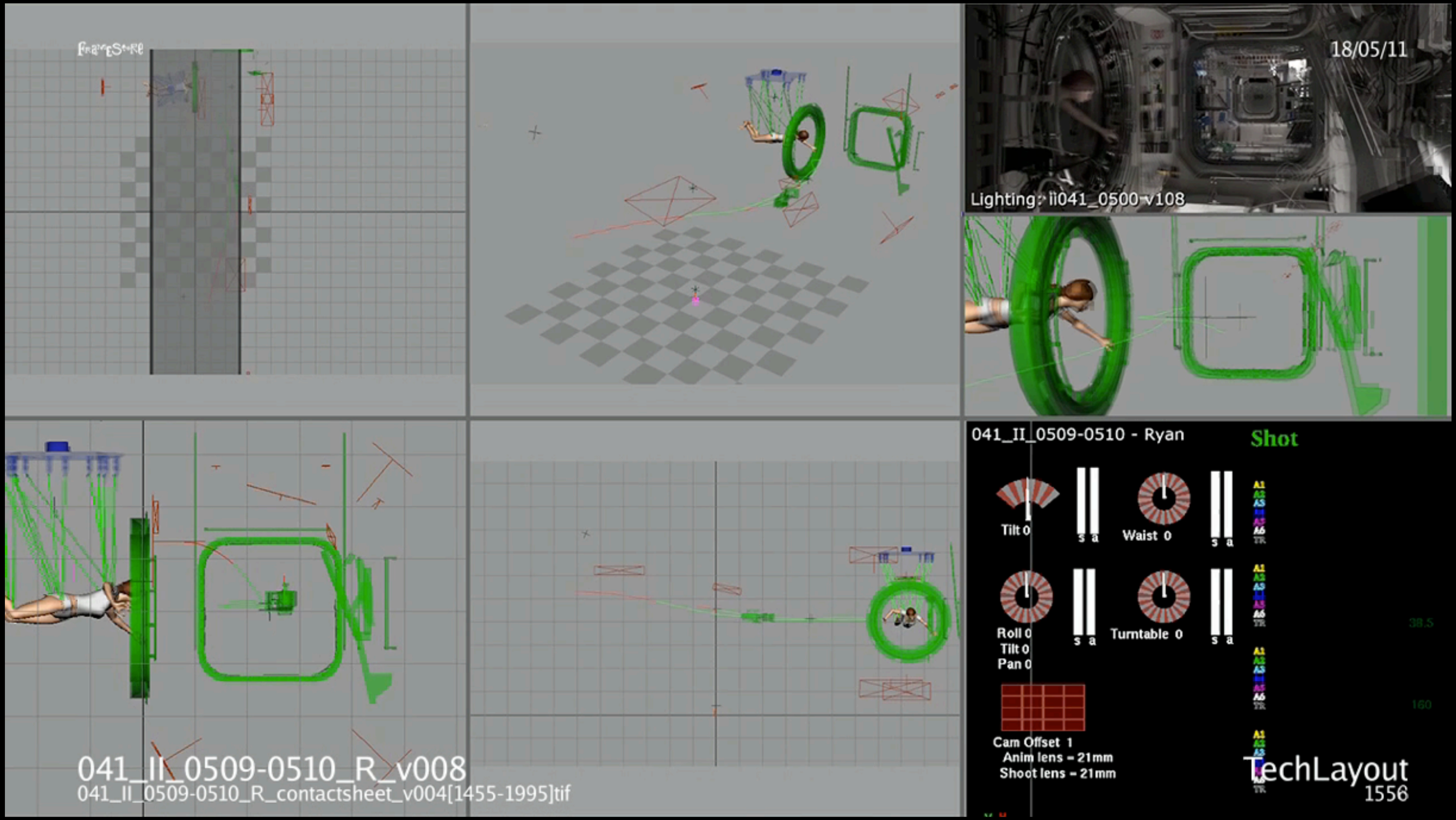


THE LIGHT BOX
Other uses: Walls of bounced sunlight

- Set pieces and on set lighting built to matches virtual environments for interaction
- On Set light direction matches virtual sun generated by lightbox panel and conventional light combination

INTERIOR SHOT
Using all techniques at once to match pre light action

- ‘Flying Shot’
  - 12 Wire Rig
  - Programmed Move
  - Puppeteers
- Proxy set added for lighting interaction
- Join seamlessly to Lightbox shot



LIGHTING
& LOOK DEVELOPMENT

What We Learned During the Pre light Stage

We must completely change our pipeline to allow for huge hero assets under extreme lighting conditions
Framestore moves to arnold

Advantages of path tracer for Gravity

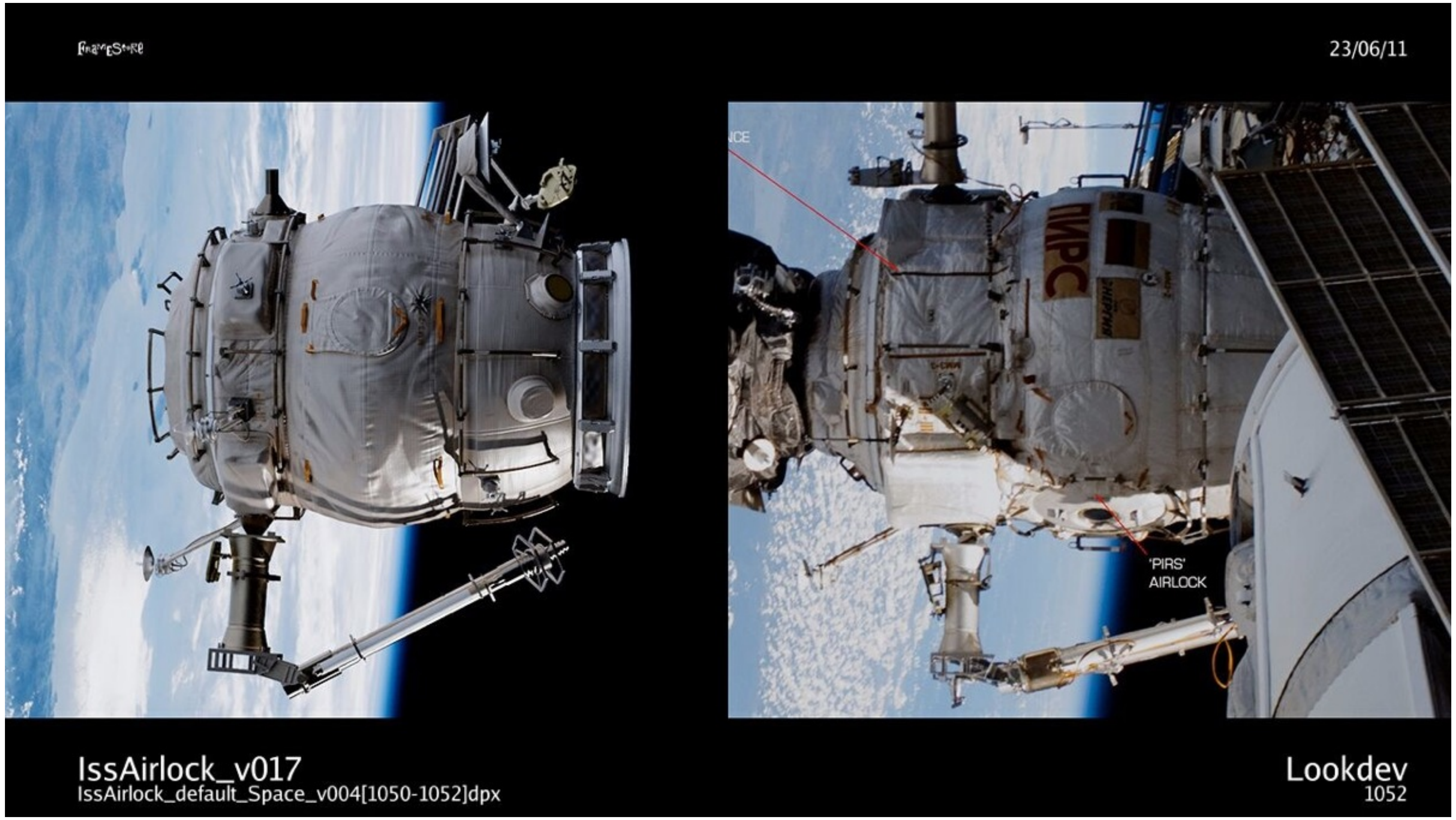
- Moving to a pathtracer allows us to alter lighting transitions without rebaking
- Arnold very memory efficient allows us to use hero assets though out shot as we have no opportunity to swap
- No bake data : would be unmanageable on shots of this length
- Physically accurate lighting model no approximation
- Easy, flexible lighting set up on all shots

- Disadvantages
  - Extreme lighting environment "worse case" for path tracer specular noise
  - Framestore crew must be trained in path tracing approaches and reyes assumptions must be unlearned!
  - Rendertime in stereo

LOOKDEV STAGE 1 VEHICLES

Fast image prototyping using in-house co-shaders

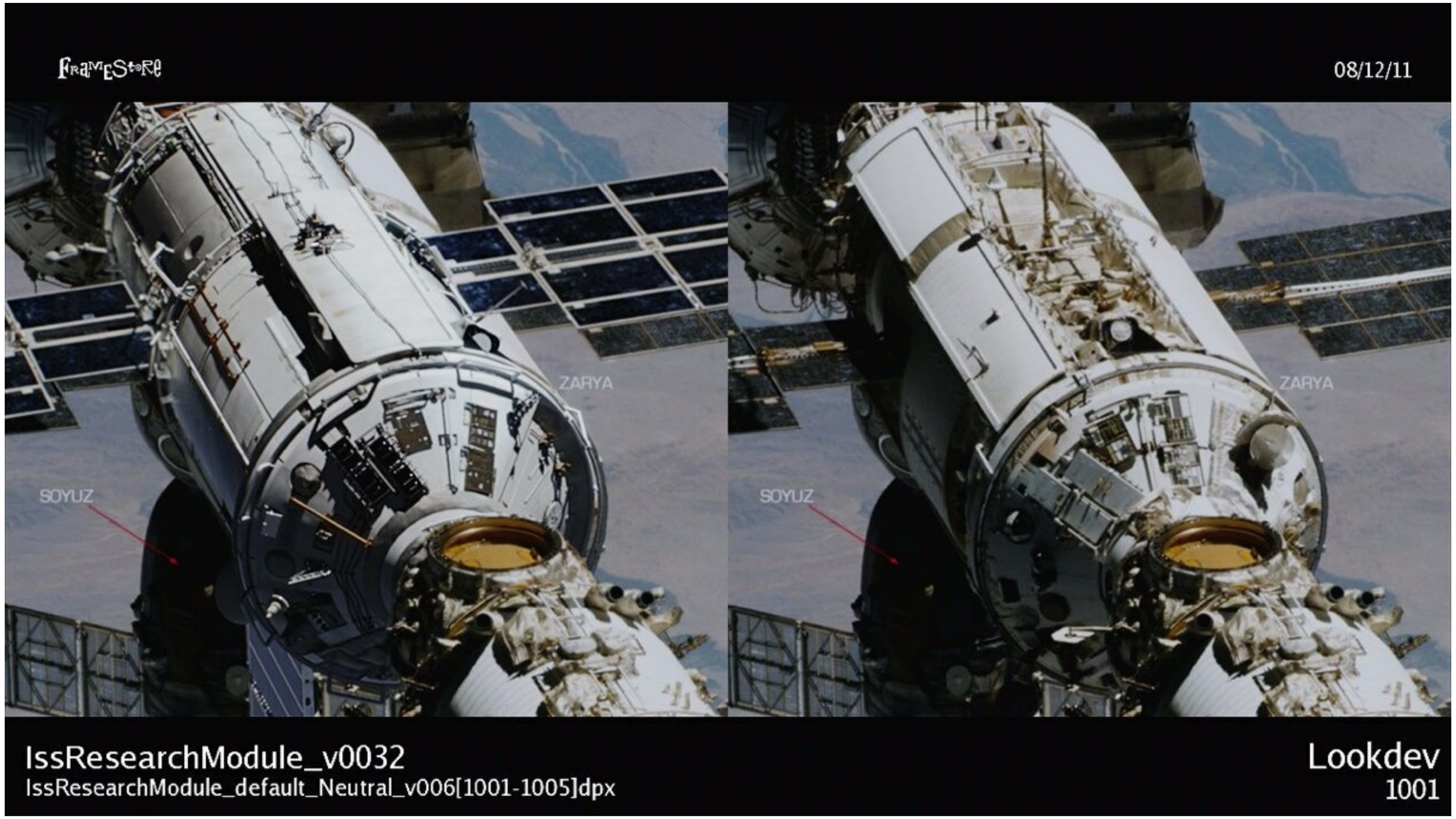
- NASA reference material
  - no access to real assets for texture reference
  - Variable unknown lighting environment
- Use Physically based rendering to match lighting environment
  - 'Arnold' software
  - Path tracing
  - Use coshaders for fast prototyping of materials and texture feedback



LIGHTING and LOOK DEVELOPMENT
Use of reference material

- Advantages
  - Great for client approval!

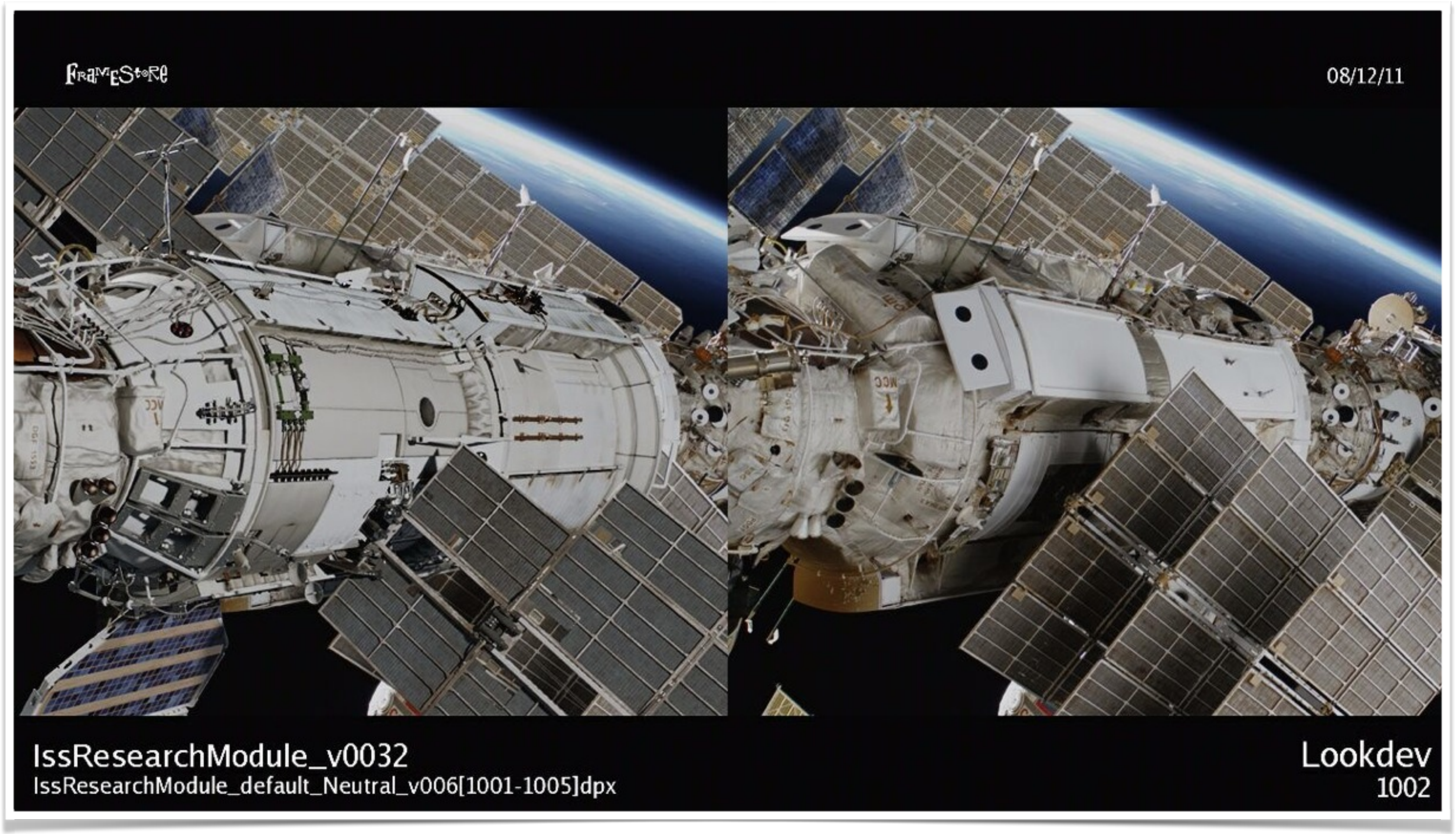
- Disadvantages
  - Material quality can be difficult to judge in unknown environment
  - No guarantee that all assets will work together when combined
  - No bench mark for render time



LIGHTING and LOOK DEVELOPMENT
Use of reference material

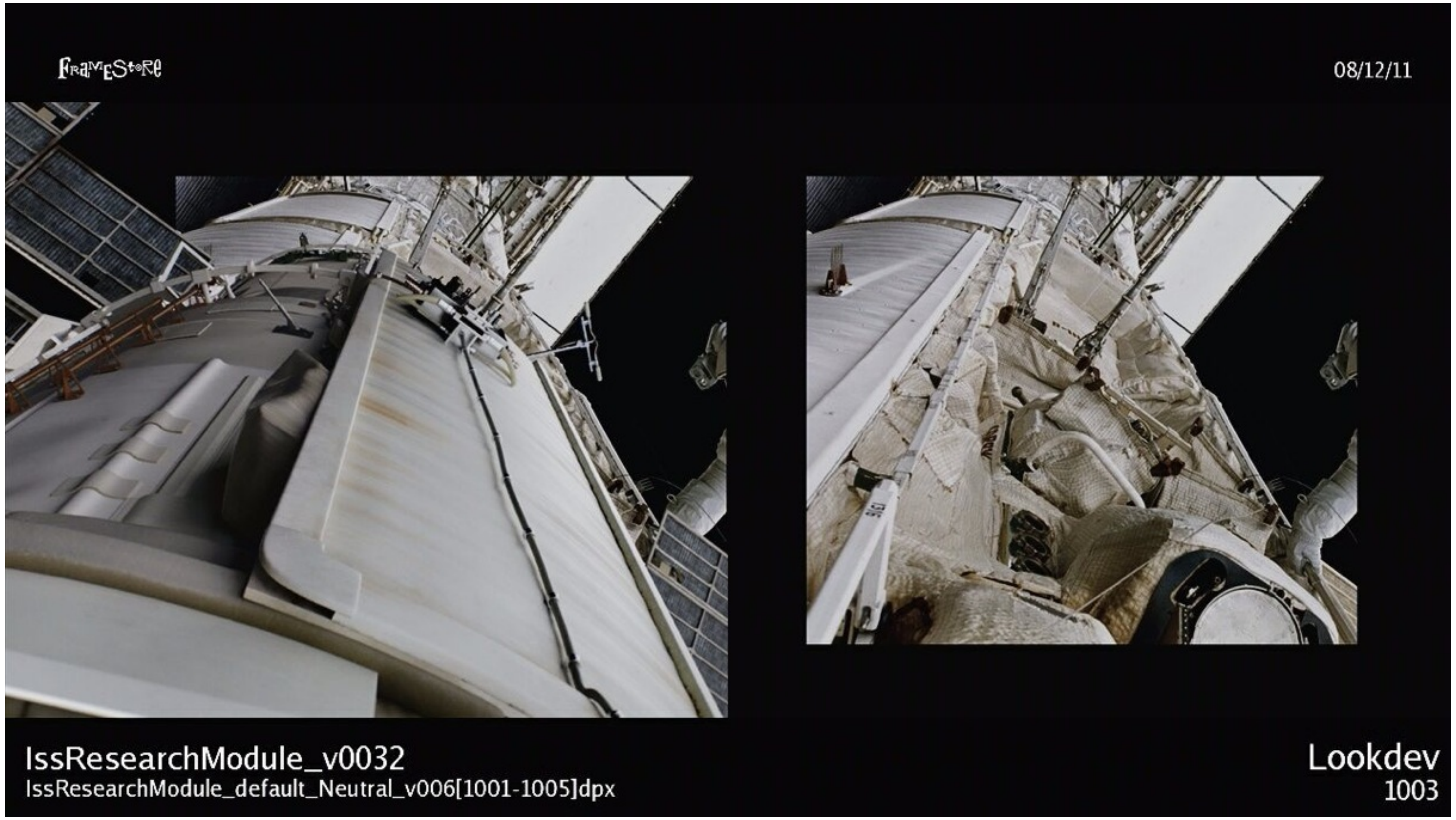
- Advantages
  - Great for client approval!

- Disadvantages
  - Material quality can be difficult to judge in unknown environment
  - No guarantee that all assets will work together when combined
  - No bench mark for render time



LIGHTING and LOOK DEVELOPMENT
From wide to close up

- All assets needed to work from all angles from wide to extreme close ups
- Complex modelled detail
- High resolution texture maps
- Long Rendertimes

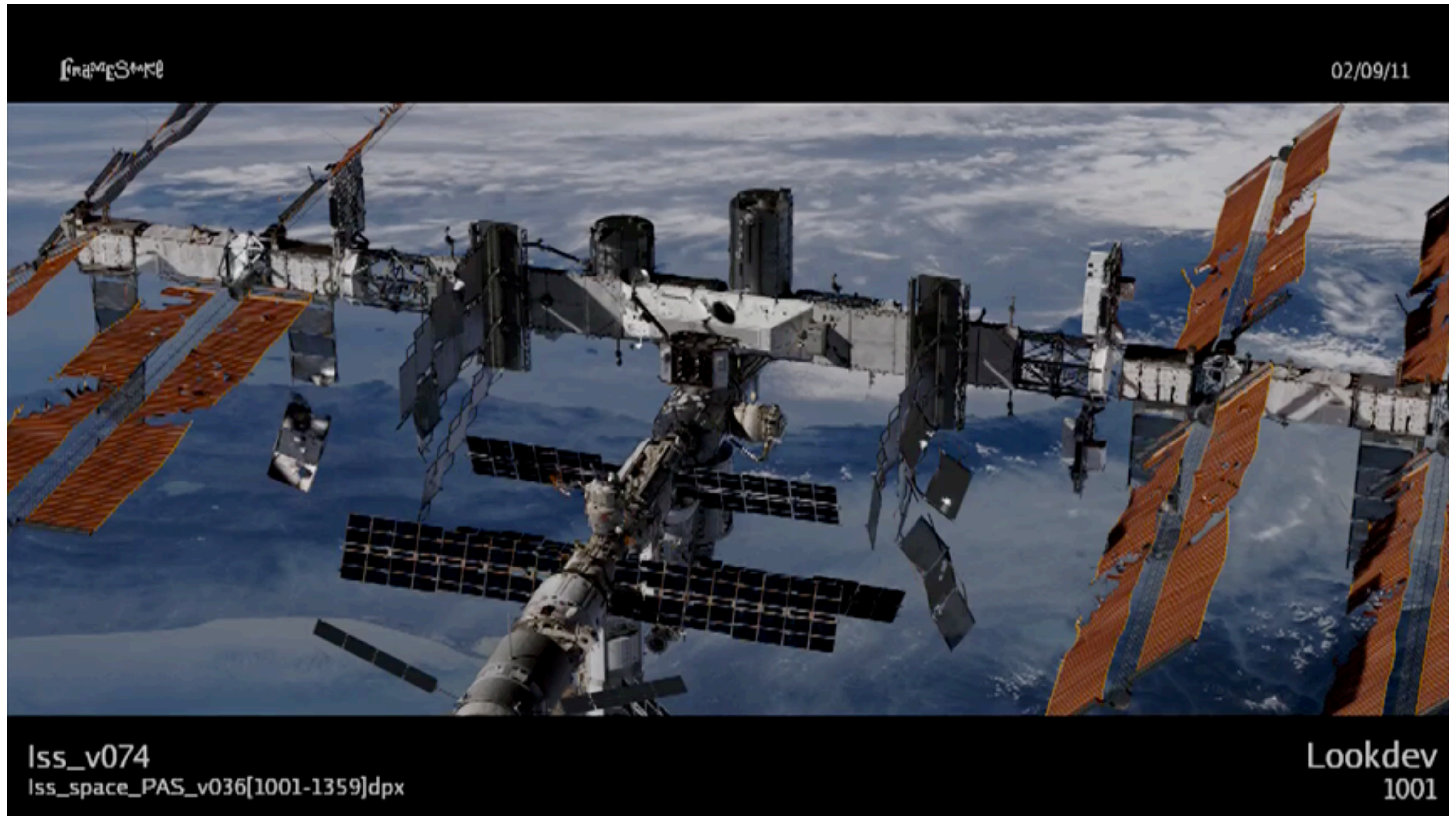


COMBINING ASSETS

Space station

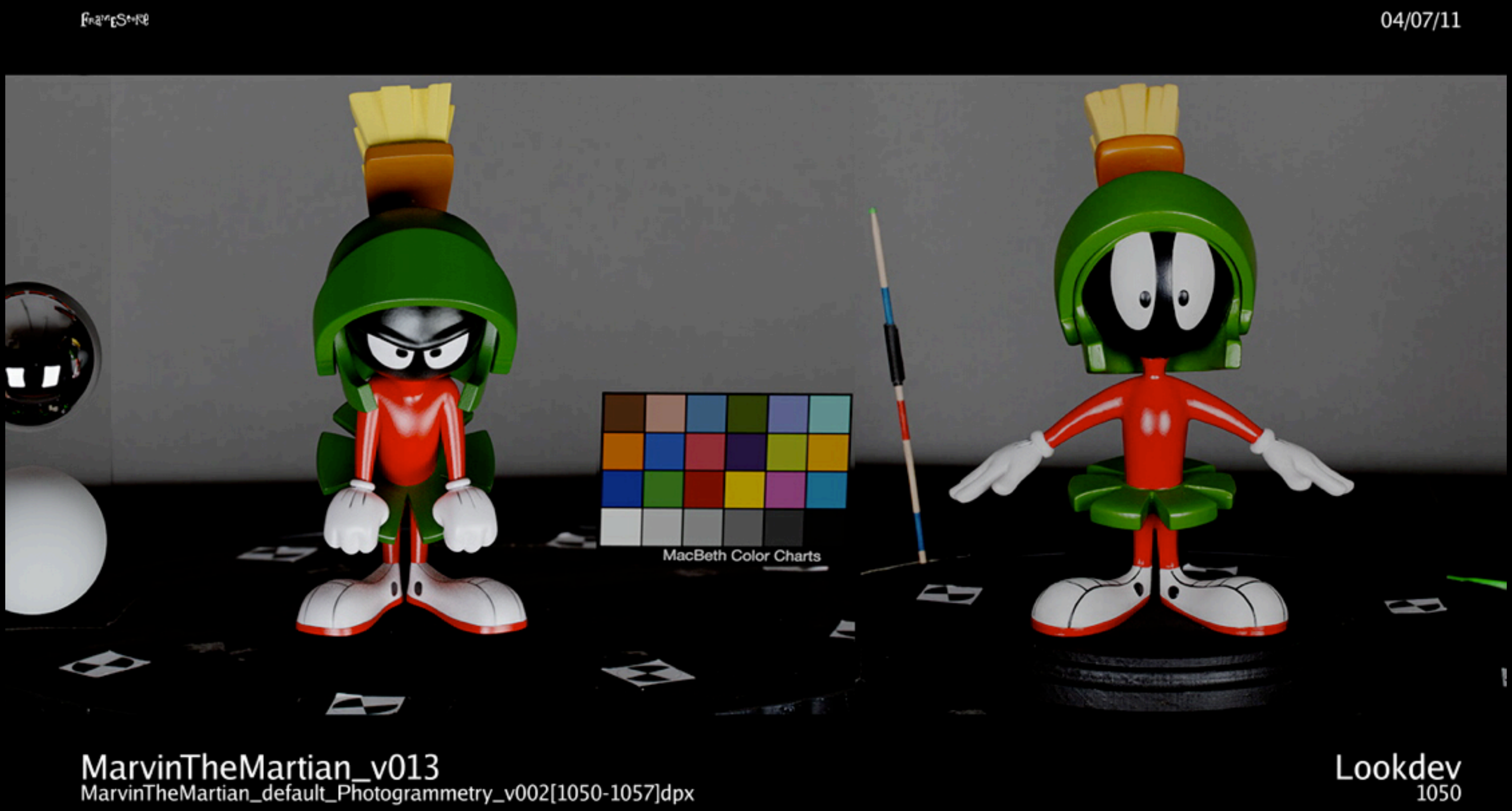
- All Assets could not be loaded into memory proxy geo used in lighting scenes and then replaced at render time

FrameStore
01/09/11
Tiangong_v052
Tiangong_default_Space_v029[1001-1360]dpx
Lookdev
1001



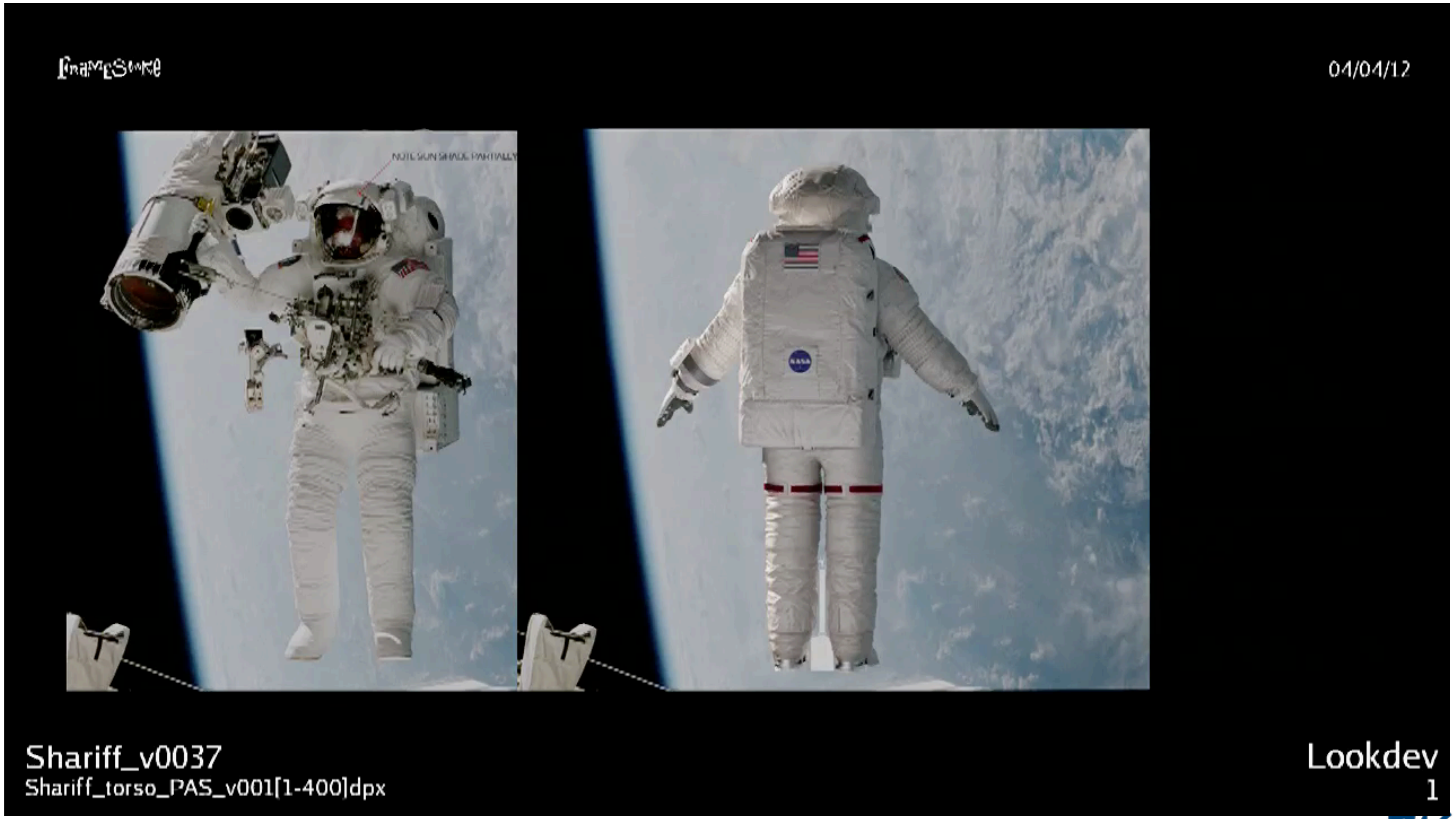
PROP LOOK DEVELOPMENT
Controlled ReferenceEnvironment

- Controlled lighting environment
- Development Lighting HDRI matches photo reference
- Polarised lighting used to match diffuse component



SUIT LOOK DEVELOPMENT
Suits: NASA suit

- Suits were tailored for cfx
- Cloth emulated with ptm for thread detail
- Uv in cloth direction
- Fur system used for extra detail
- Displacement shaders replacement by high resolution mesh



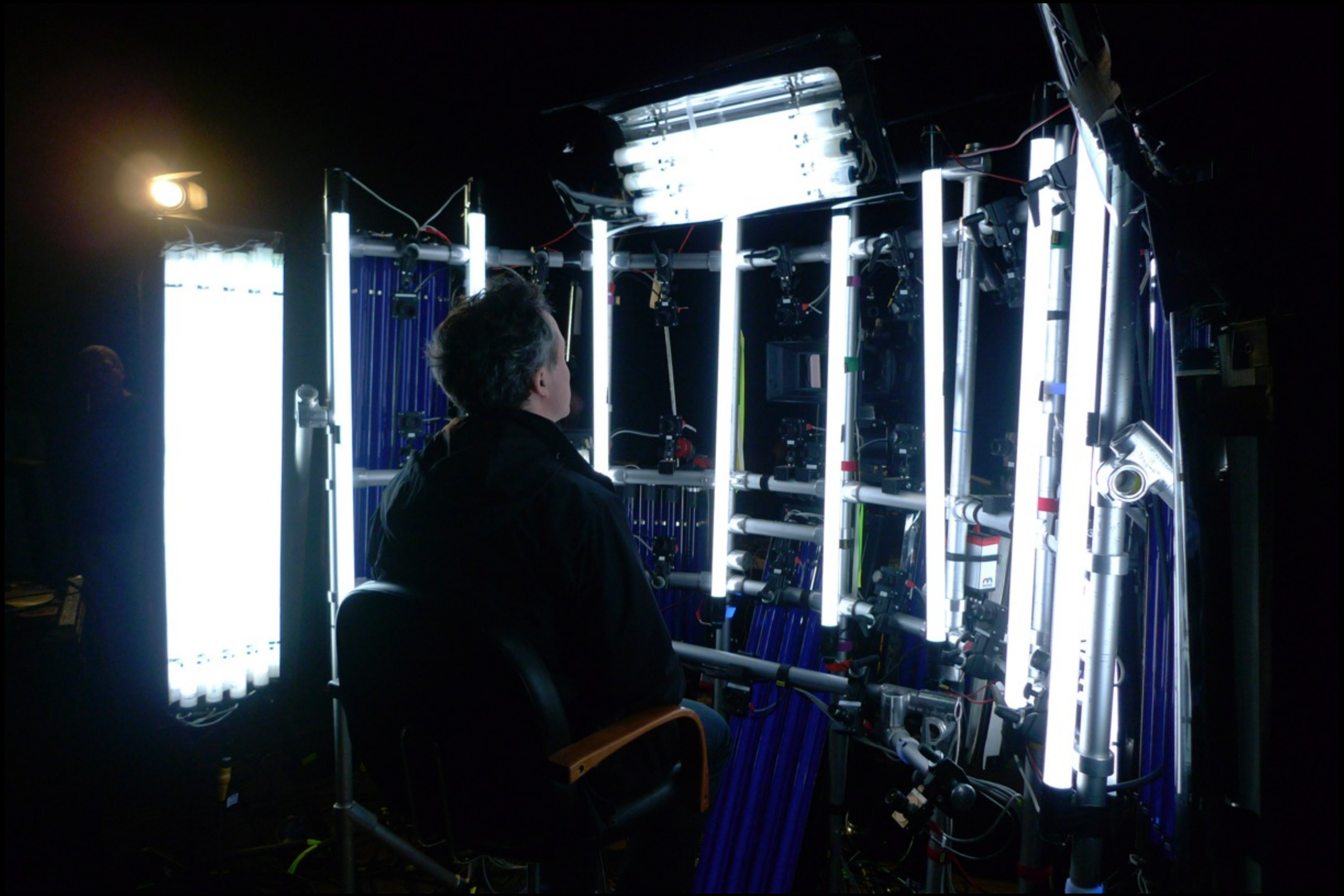
LOOK DEVELOPMENT

Iterations

- repeat until
  done !

DIGITAL HEADS

- Data captured under Neutral lighting
- Captured using ‘Mova’
- 5 cameras



DIGITAL HEADS

- If smaller then 1/8 screen height
- Animated Data captured for each subsurface layer
- Extremely detailed animated texture maps created at high resolution for Displacement and Specular Response

DIGITAL HEADS

- Digital heads used when action is too dangerous to be shot
- Visual reference for the actor provided by a wall of LED panels of Prelight

DIGITAL HEADS

- Arnold allowed low memory profile for large amount of texture detail and data.Captured using ‘Mova’
- Raytracing subsurface allowed accurate light modelling from distant light sources and indirect bounce
- Easy set up
- No technical knowledge of input data to light in shots

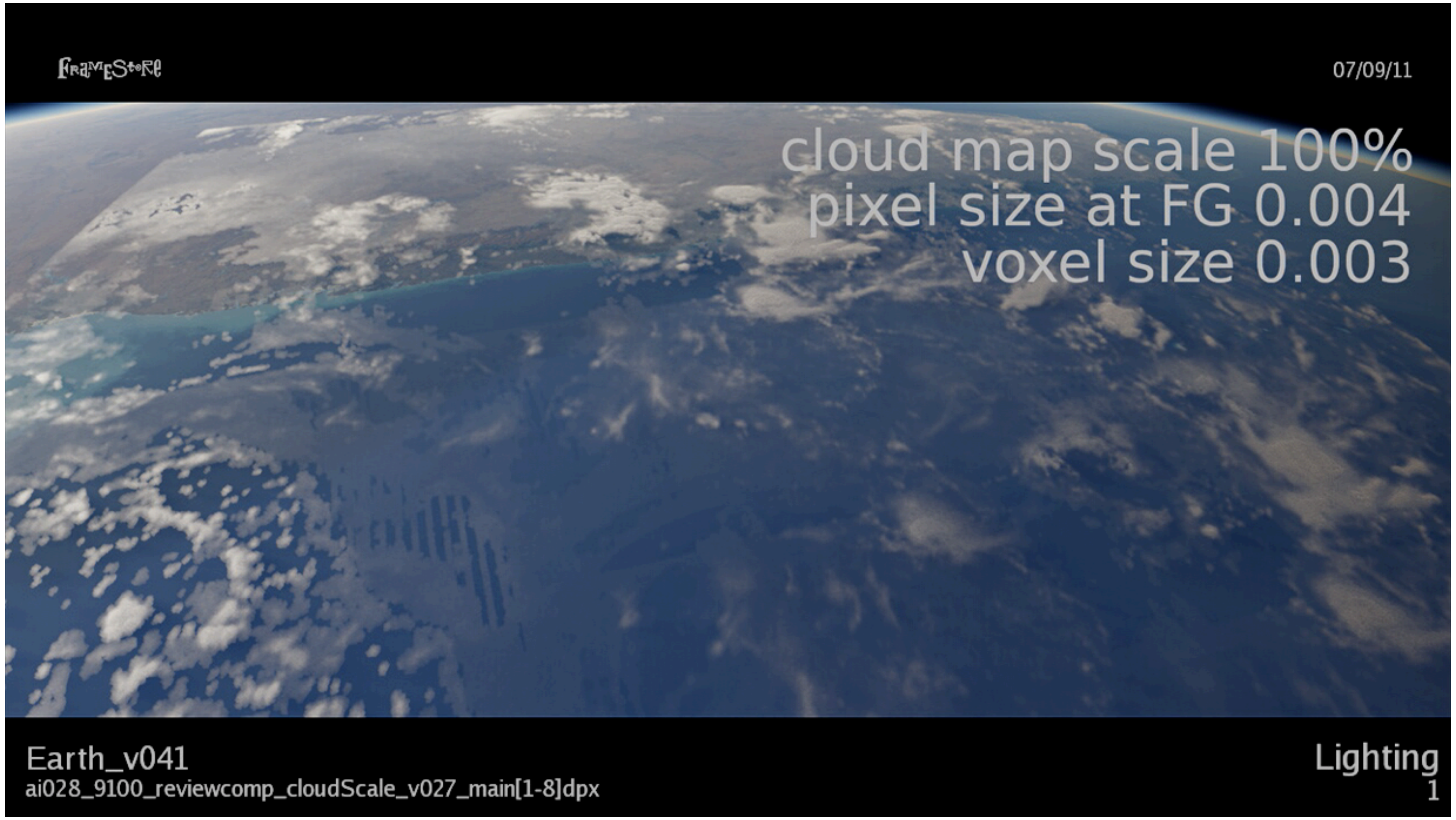
EARTH
Shading & Rendering

- Could not achieve required detail with shadow maps
- Shadow maps calculation was to slow for quick turnaround of client weather requests
- Moved to ray marching technique using Arnold
- Also allowed increased physical accuracy in lighting scatter model

FrameStore
02/04/11
Earth_v015
Earth_loodev_v002[1-250]dpx
Lookdev
0001

EARTH
Clouds and Weather

- All clouds were hand painted
- Paintings were converted into 3D volumes
- Ability to paint on multiple altitudes, and to hand dress in varying noise patterns
- Final clouds scattered and reflected into the overall lighting simulation
- Reyes system could not hold all data in memory
- Arnold Volume renders gave accurate scatter and lighting interaction on large extremely detail cloud data from Nasa.



Two Stage LookDevelopment

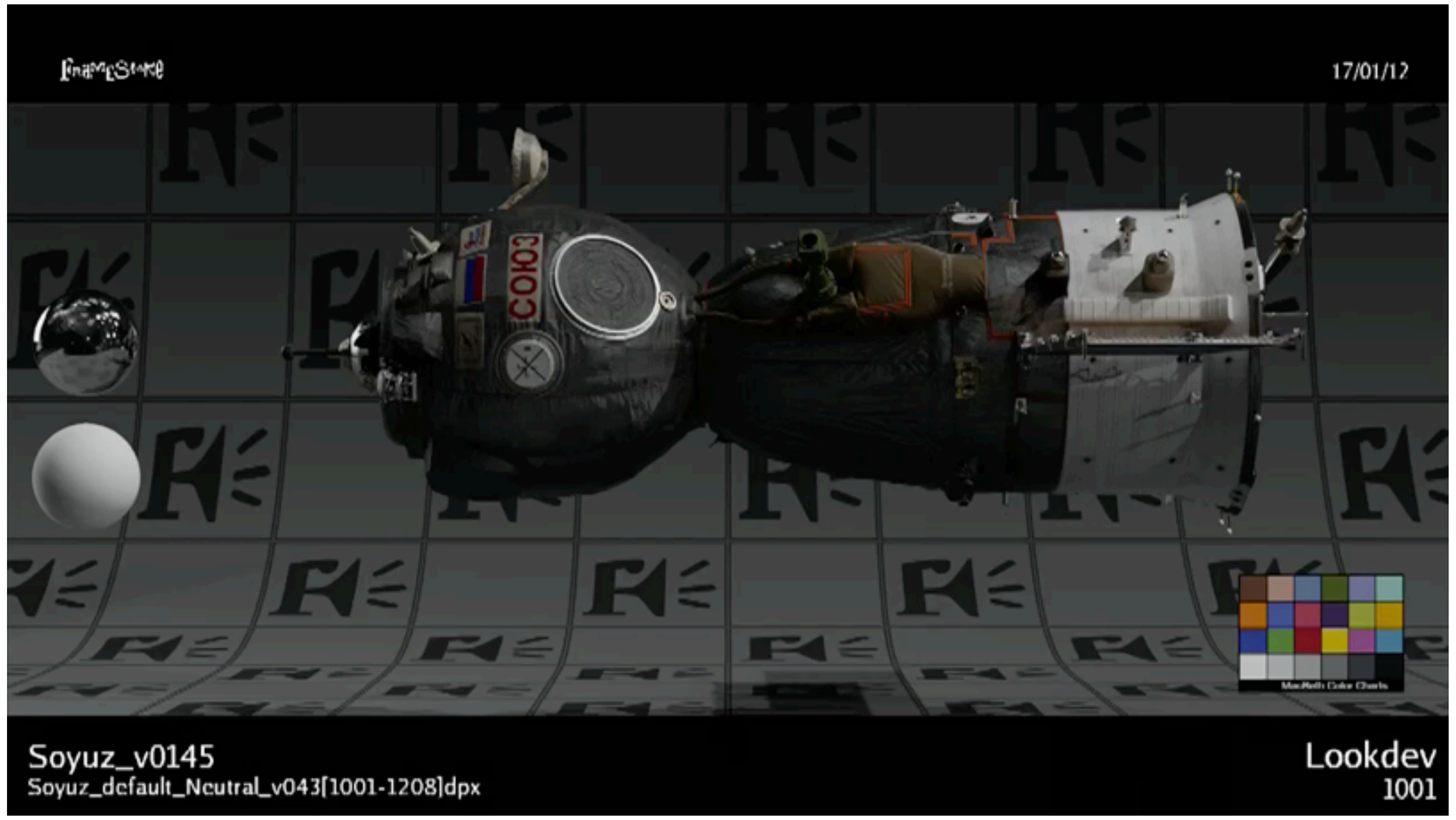
Once the Look development is physically plausible and cleared by client it is time to optimise

- Analyse all materials for assets and identify most efficient
- Where possible replace expensive co-shader networks with optimised Monolithic Shaders
  - These “monoliths” include automatic rayswitching to simpler shading models where possible
  - Combine multilayer textures and shader based colour corrections back into the colour map
  - Replace multilayer brdfs with IOR Maps and Roughness maps where possible
  - Cache as much data as possible between left and right eye to reduce stereo costs

- Disadvantages
  - Increases the number of texture map iterations
  - Optimisation of shaders requires higher technical knowledge of path tracing

Physical Plausibility and Optimisation Stage 2
Calibrated Neutral Lighting Environment

- Lighting environment calibrated to 18% grey ball and macbeth chart
- Diffuse macbeth chart gives value of 0.038 on black and .91 in white under this lighting environment
- High contrast environment used to check for specular response issues
- Same environment used for all assets
- Checks brdf balance and bounce response
- Checks Albedo response
- Checks render time
- Not good for client presentations



PROPS Testing and Optimisation

Props

- All props tested in shoot and calibrated lighting environment

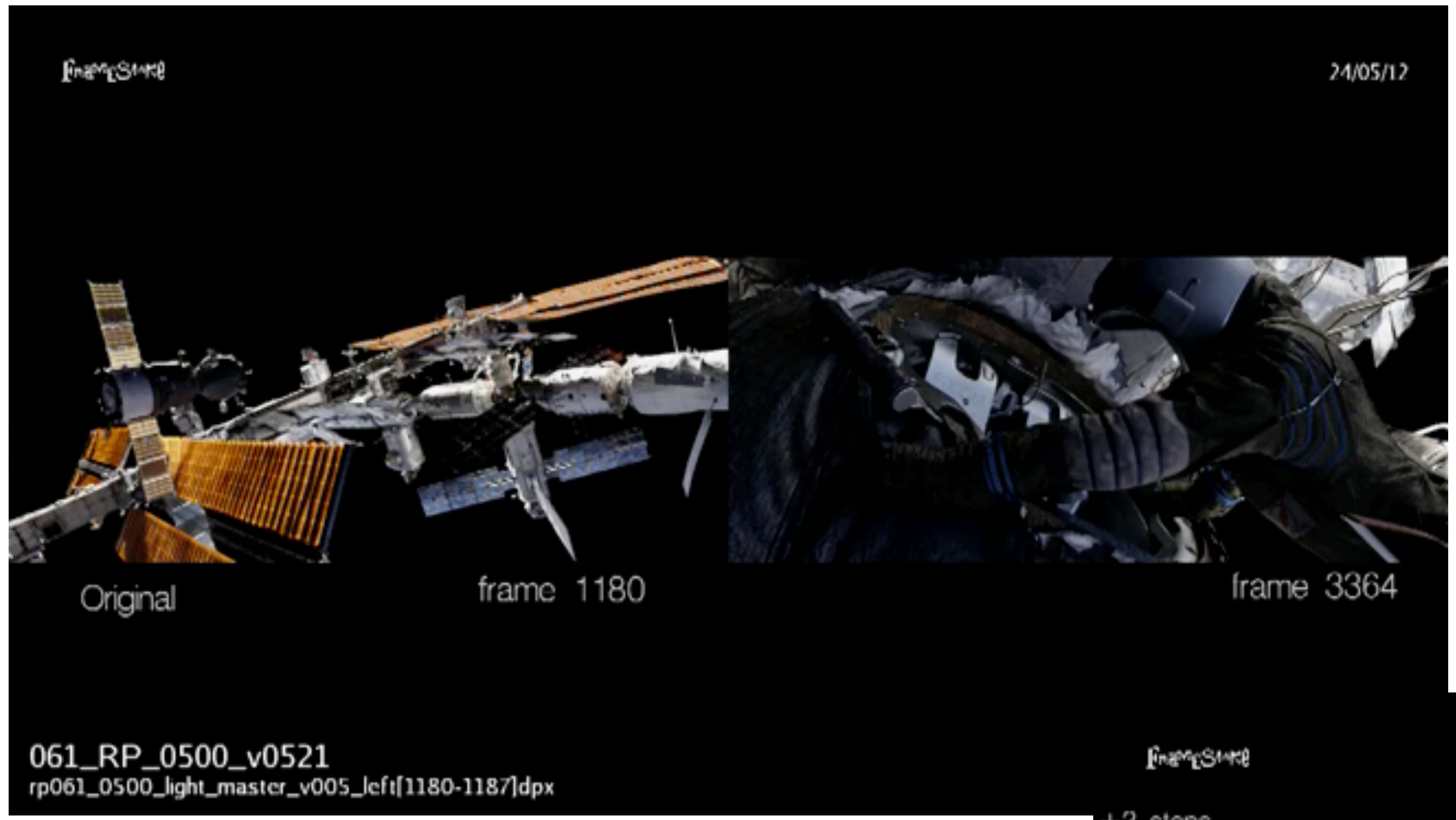
12/07/12
IssClearColaPouch_v0010
IssClearColaPouch_neutral_v004[1001-1050]dpx
Lookdev
1001

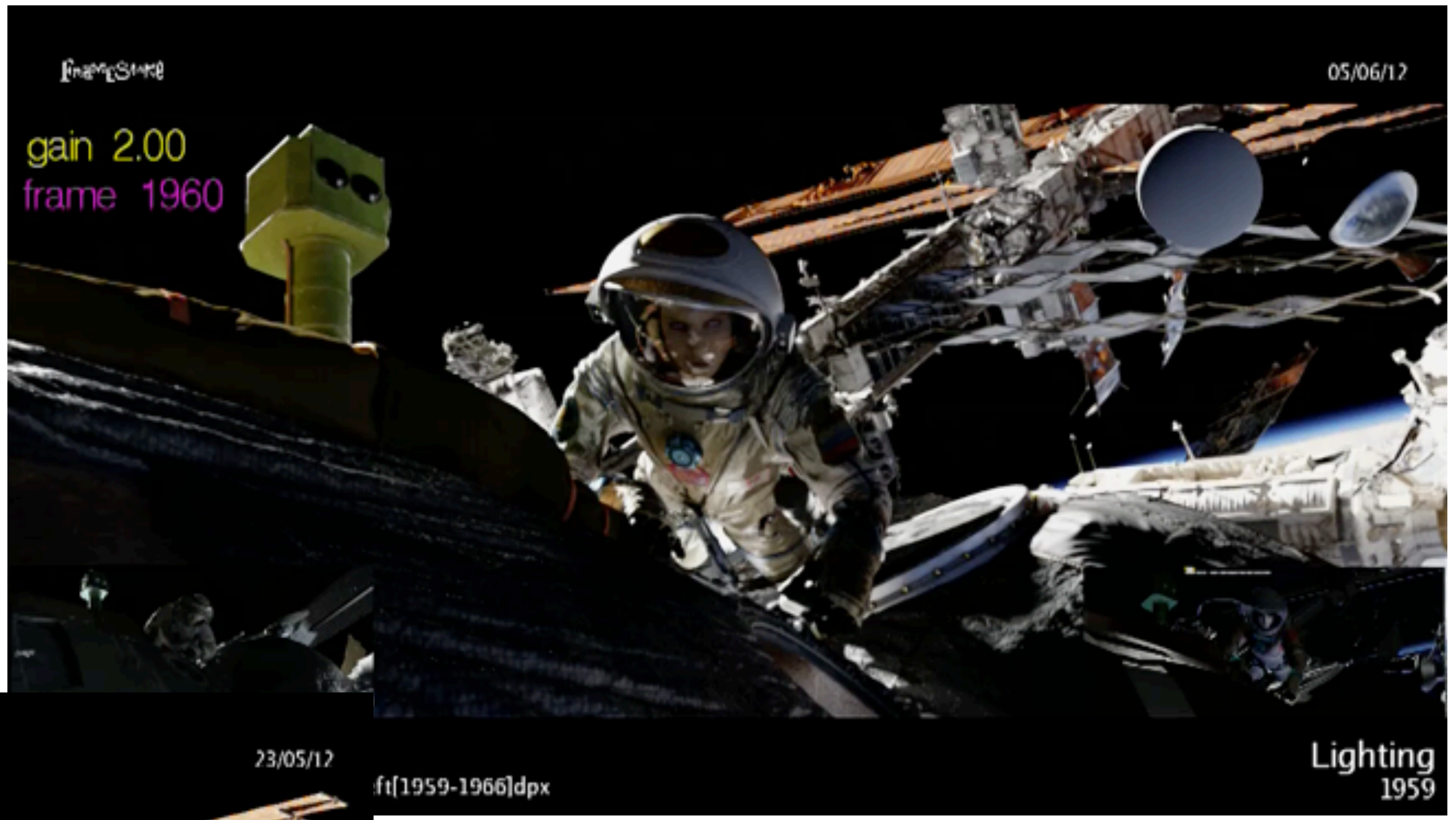
20/07/12
Houston Astros
IssBaseballHat_v0005
IssBaseballHat_default_Photogrammetry_v005[1001-1050]dpx
Lookdev
1001

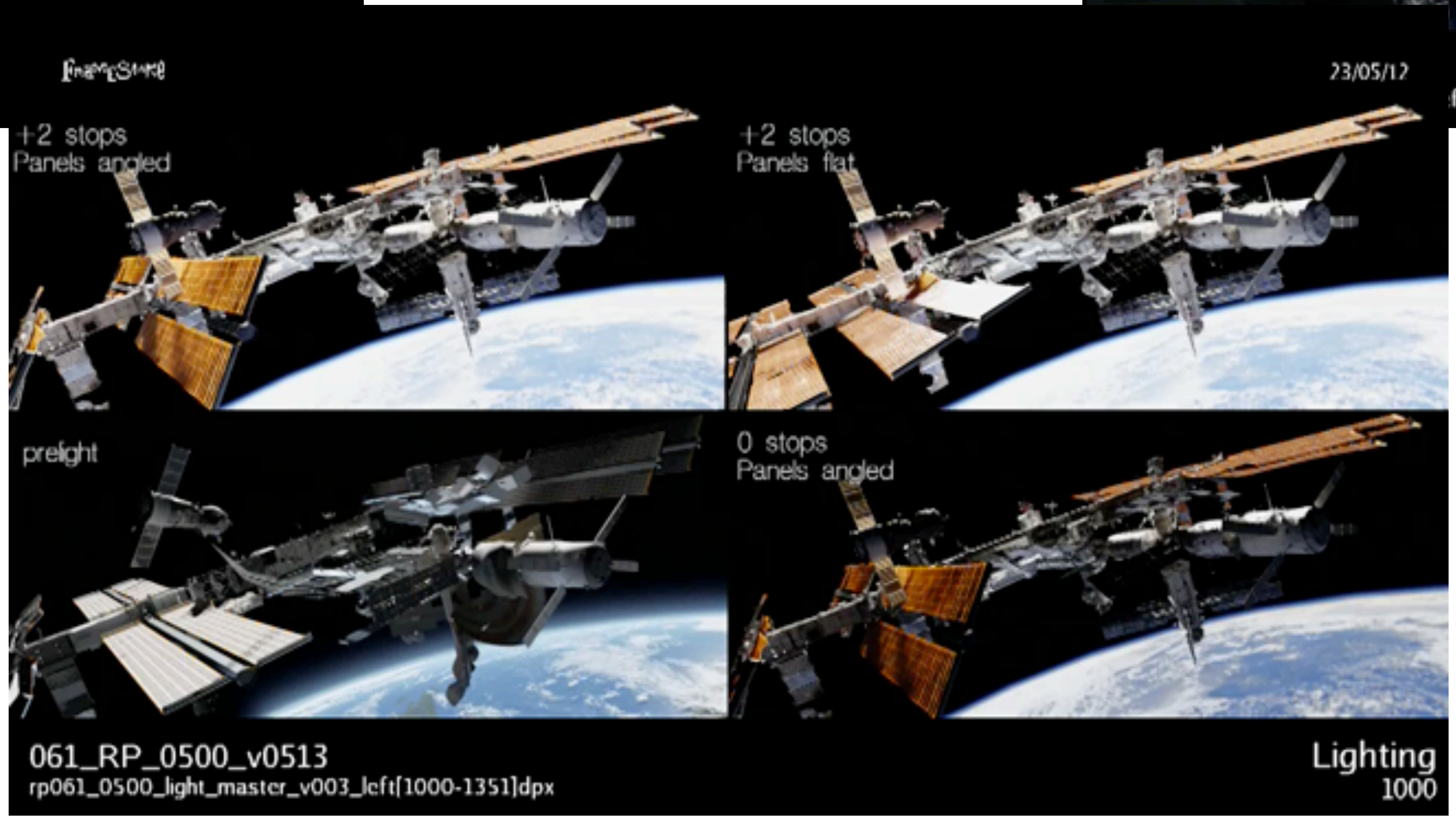
29/05/12
default
beige variant
green variant
blue variant
IssCargoBagB_v0007
IssCargoBagB_default_neutral_v002[1001-1050]dpx
Lookdev
1001

15/06/12
CssLuckyCat_v0007
CssLuckyCat_default_Neutral_v001[1001-1050]dpx
Lookdev
1001

LOOK DEVELOPMENT
Lighting & renders







Challenges in Lighting

Extreme lighting conditions, huge assets and shots of 12.5 minutes

- Reducing Render time
  - With a path tracing solution visually successful images are much easier to achieve however optimisation is far more important than with a reyes system. Every shader network should be analysed for speed as well as look achieved.
  - Don’t render till it’s ready!
    - We employed a system of very cheap Qc renders to check that all assets were working before even starting a lighting render
    - Because all shader are physically accurate you do not need to render all frames to test lighting, all lighting setups on Gravity where tested every 10th frame for clearance
    - All shots on Gravity were finalled with less than 1.5 2k full production iterations (and the 0.5 is because the Director changed his mind!)
- Lighting in a extreme environment
  - Noise in direct specular : use motion vectors to sample specular “through time”
  - Increase roughness in secondary bounces

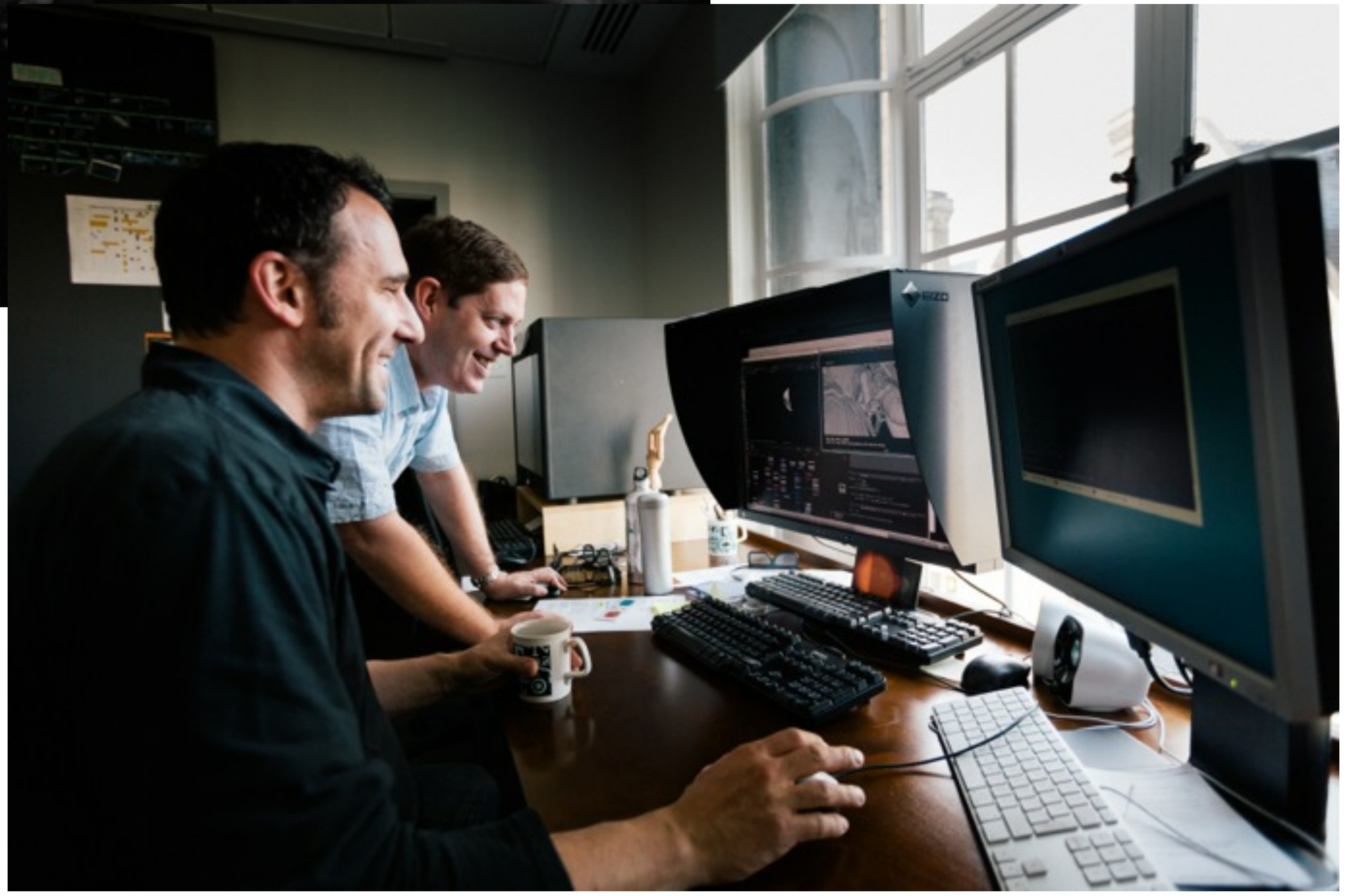
LIGHTING
The finished product

- Entire Framestore pipeline now path tracer based

FINISHING UP

STATS & FACTS

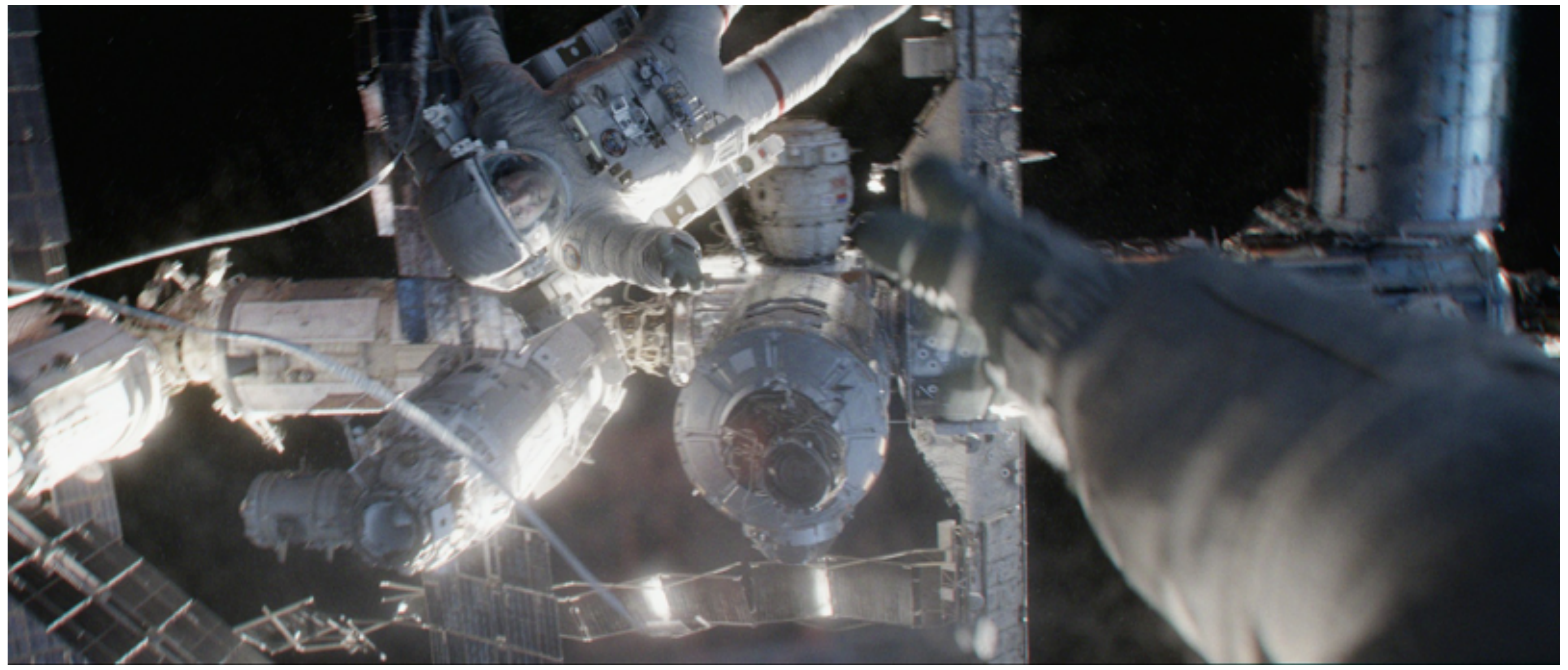
- Peak crew: 280
- Total crew: 440
- Project duration: 3 years
- 500Tb live data at any one time
- 15,000 dedicated processors



MILESTONES

- 7,000 years of computation
- production notes: 2.1 millions words (or 4 copies of War and Peace)
- greediest process: reading over 1.5 million files
- longest single-frame render: 380+ hours (before optimisation !)

Unique collaboration between Film-Makers and Visual Effects



CONCLUSION
Our most critical audience…

"I was so extravagantly impressed by the portrayal of the reality of zero gravity…"
Buzz Aldrin

Expedition 27
NASA
GRAVITY
NASA Astronaut Ronald J. Garan Jr.
Air Force Colonel Catherine Coleman