Into the Voyd: Teleportation of Light Transport in *Incredibles 2*

Patrick Coleman
Pixar Animation Studios
pcoleman@pixar.com

Darwyn Peachey
Pixar Animation Studios
peachey@pixar.com

Tom Nettleship
Pixar Animation Studios
tomn@pixar.com

Ryusuke Villemin
Pixar Animation Studios
rvillemin@pixar.com

Tobin Jones
Pixar Animation Studios
tobin@pixar.com

Figure 1: Recursive light transport through a portal (left) and a character passing through a portal from one set to another (right). (c) Disney/Pixar.

ABSTRACT

In *Incredibles 2*, a character named Voyd has the ability to create portals that connect two locations in space. A particular challenge for this film is the presence of portals in a number of fast-paced action sequences with multiple characters and objects passing through them, causing multiple views of the scene to be visible in a single shot. To enable the production of this effect while allowing production artists to focus on creative work, we've developed a system that allows for the rendering of portals while solving for light transport inside a path tracer, as well as a suite of interactive tools for creating shots and animating characters and objects as they interact with and pass through portals. In addition, we've designed an effects animation pipeline that allows for the art–directible creation of boundary elements that allow artists to clearly show the presence of visually distinctive portals in a number of fast-paced action sequences.

CCS CONCEPTS

• **Computing methodologies** → **Animation**; **Ray tracing**; **Graphics systems and interfaces**;

KEYWORDS

Ray tracing, Animation, Procedural Animation, Constraints, Visibility

DigiPro '18, August 11, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-5895-8/18/08.
https://doi.org/10.1145/3233085.3233092

ACM Reference Format:
Patrick Coleman, Darwyn Peachey, Tom Nettleship, Ryusuke Villemin, and Tobin Jones. 2018. Into the Voyd: Teleportation of Light Transport in *Incredibles 2*. In *DigiPro '18: The Digital Production Symposium, August 11, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3233085.3233092

1 INTRODUCTION

The character Voyd in *Incredibles 2* introduces a new superpower to the world of *The Incredibles*–she can open portals that connect two locations in space. These are typically large enough for objects or characters to pass through, and we often see both openings on screen. To produce this effect, we have introduced support for light transport through connected portals using a path tracer. This allows for a seamless visual connection of light and geometry that can span multiple locations in a set. To allow our layout artists and animators to create compelling shots where characters interact with the effect, we have developed a suite of tools for interactively setting up and previewing portals, as well as for animating characters and objects as they pass through the portals, using our in–house animation system Presto. To visually set the effect in the scene, we have designed animated portal boundary effects to support the idea that they are transient rifts in space.

Each portal is a hole in space whose two sides are separable into a *here* opening and a *there* opening. If *here* and *there* are at the same location, then this is what we normally think of as a hole, but with no teleportation. If *here* and *there* are at separate locations, objects, characters, and light are seamlessly teleported from one location to the other as they pass through the hole. To support this in the 3D software packages we use in our production pipeline, we need to support light transport through the opening, as well as interactive preview renders for users who set up and animate the portals. To see object and character geometry that seamlessly spans the portal

boundary, we set up a copy on each each side portal and constrain these two copies of geometry to maintain a continuous boundary.

While portals have been used in both interactive games, such as the *Portal* series, as well as in live action visual effects by texturing or compositing in footage, supporting path traced teleportation simplifies our production process for fully animated content. The continuous definition of connected space allows for single shot renders and compositing setups and enables us to stage portals such that we can see a portal inside itself, enabling the creation of shots that recursively show repeating copies of the scene, as seen in Figure 1 (left). A picture-in-picture setup could also allow for the filming of another location through portals in space using additional shots, by this would require complex and error-prone coordination of both cameras in the two locations and characters that pass through the portals. In contrast, our approach allows users to interactively update cameras and characters while seeing the view through the portal update live. This also allows our artists to focus on creative animation and staging in a single shot instead of needing to coordinate the content of multiple shots.

2 LIGHT TRANSPORT THROUGH A PORTAL

Since raytracing is a simulation of light transport using rays, it's natural for the teleportal implementation in RenderMan to directly work on these. Our approach uses the *here* and *there* copies of the portal geometry to define a teleportation transform matrix which maps the local space of the *here* portal to that of the *there* portal. During rendering, when a ray hits the *here* portal geometry, the integrator interrupts raytracing and applies the teleportation transform to the ray. This teleports it to the equivalent location and direction relative to the *there* portal, after which it continues tracing into the scene. The portal surface also carries shading signals for the surface normal, refractive index, opacity, and tinting, which give the artist the ability fine tune the look of the view through the portal.

There are a number of advantages to performing the teleportation at the core of the ray tracer. First, the effect "just works" with most RenderMan features—depth of field, recursive hits, motion blur, shadowing, global illumination, volume rendering, and stereo rendering are all immediately compatible with this approach. Second, the workflow it supports is significantly easier than the alternative approach of using pre-rendered textures from another shot for the view through the portal. While shots containing portals are more complicated to animate and wrangle than typical shots, creating a seamless effect is much more easily achieved when everything is kept in-camera as part of a single render pass, especially when the main camera view is moving quickly.

As might be expected, a number of challenges were encountered, both technical and creative, once we moved beyond our development demos and into real production shots:

2.1 View frustum optimizations

The presence of portals in a shot would often visually reveal ways in which either RenderMan or other portions of our pipeline optimize assets to improve efficiency. Typically, the far side of a portal would be off-screen, often nearby, but sometimes a large distance away. Characters and sections of the set needed to be protected from our culling and level–of–detail operations to appear at full quality—or at all—when viewed through a portal. In addition, our lighters and set dressers work to camera. As a result, they often needed to make a second pass on portal shots to add set dressing and visual interest to areas of the scene that the portals revealed.

2.2 Clipping Planes

RenderMan supports arbitrary clipping planes. Our layout team uses these when they need to place a camera inside some geometry within the scene - the clipping plane causes camera rays to ignore that geometry while indirect and shadow rays treat it normally. Rays which have passed through a teleportal are considered by RenderMan to still be camera rays, and the arbitrary clipping planes will cull them. As a result, we avoided the use of these clipping planes in portal shots.

2.3 Geometric Visibility

If a shot calls for an object or character to move through a portal, ray teleportation only provides half of the solution by maintaining the seamless transition to the second copy at the far side. However, the portion of the geometry of the object *behind* the *here* portal is not naturally culled or removed once it moves through the *here* portal, but instead pokes out the back. If the portal is oblique to the camera, that extra geometry can be revealed and would need to be removed. The same situation occurs at the *there* portal, but in reverse–geometry that is visible on the opposite side needs to be hidden.

We considered using animated face culling or shading signals to remove geometry as it passes through the portal plane, but these approaches both involve per-asset work that would need to be redonw as the animation gets refined. Instead, we constrained a volume to the back of each portal which carried a "deathray" shading network. This would be spliced in to the shading network of any object which passes through the portal. When a ray hits the object, if the hit point is inside the volume, the shading network returns a fully transparent response and the ray continues as if the object had not been encountered, effectively rendering that part of the object invisible.

2.4 Light Sampling

While camera and indirect rays pass through a portal intact, RenderMan's light selection component is unaware of them. As a result, lights shining through a portal will not be importance sampled correctly for MIS. This produces noisy results or visible boundaries in the lighting of characters which penetrate the portals. Minor issues of this type were solved with compositing tweaks, while major issues required copies of the relevant lighting rigs to be made, placed at the equivalent location relative to the portal exit, and linked to only one copy of the character. Major issues needing duplicate rigs were only encountered in three shots, so a more robust technical solution was not pursued.

2.5 Temporal offsets

The animators and director found that some shots worked best if the action through a portal is staggered in time. For example, a punch into a portal over *here* might emerge from the portal over *there*

a few frames later. This adds a double–beat to the timing, which increases the visual impact of the shot. It introduces a technical problem, however, because rays which enter the portal can not be time-delayed. To solve this issue, two pairs of copies of the character are needed: one *here* and *there* copy in sync with each other for one portal, and another pair for the time-delayed copy at the *there*. One copy in each pair would have visibility rules set up that restrict their visibility to primary camera rays, this copy corresponds to the geometry that is not seen through a portal. The second copy of each pair would only be visible to camera rays which had passed through the portal; these copies were only visible inside the portals to maintain visual continuity.

2.6 Recursive Teleportation

For shots in which we see a portal inside itself, we want to avoid infinite recursion when the portals line up closely to the camera view. To do so, we track a ray depth that specifically tracks portal hits, and we used this to limit the number of portal transitions a ray could make. This was tracked separately from RenderMan's standard ray depth tracking so that scene lighting would be consistent, regardless of the number of portal teleportations a ray had encountered.

3 INTERACTIVE LAYOUT, VISUALIZATION, AND ANIMATION WITH PORTALS

To allow our camera and staging and animation departments to set up portals, animate them, and animate objects and characters passing through them, we have developed a suite of tools to support interactive hardware rendering through a portal and the rigging tools needed to allow both rigid and deforming geometry to continuously span the boundary of the portal. As we sometimes film a single opening that connects to another region of the set, and at other times we film both connected openings, we consider each opening to be a separate portal rig with its own *here* portal and *there* portal. When we do film both ends of a connected portal, we constrain each *there* portal to the position and location of the opposite *here* portal. This approach allows for flexibility to modify composition through each portal if needed and to create timing on portal motion that slightly deviates from the logical idea that space is connected to increase visual appeal.

To provide an interactive hardware render of the view through a *here* portal, we render an OpenGL pre–pass from another view that maintains the illusion of connected space. We then use a hardware shader to sample the pre–pass and map it to the correct location on the *here* portal. To maintain this connected space illusion, we solve for an additional *there camera* to render the pre-pass texture.

We've developed a teleportal constraint solver that maintains the *there* camera's spatial relationship to the *there* portal's transform, such that it is equivalent to our primary camera's relationship to the *here* portal's transform. By doing so, and additionally constraining the perspective projection parameters to be equivalent to those of the primary camera, we can interactively manipulate or animate our primary camera or either portal's transform while correctly maintaining the illusion of connected space through the *here* portal boundary. Finally, we solve for arbitrary clipping plane parameters on the *there* camera to prevent objects in front of the *there* portal from appearing in the pre–pass texture.

When props or characters pass through portals, we set up two copies, one on each side of the portal, to ensure that both light transport and the preview hardware renders appear continuous across the boundary. We first use the teleportal constraint solver to determine the top-level transform on the *there* copy of the object. Users can then manipulate the primary object, near the *here* portal, to see both copies moving in sync and appearing to align at the portal boundary. For deforming characters, we can constrain the deformation parameters to be equal, or we can copy values from the primary copy to the *there* copy; we've found cases where each workflow is useful. We can also procedurally cull portions of an object that pass through a portal to prevent seeing this portion of the geometry when we film a portal from an oblique angle. While geometric instancing could be used to reduce memory overhead in both our animation software and at render time, the limited gain from removing a single additional copy did not justify the additional pipeline complexity.

Figure 2 demonstrates how we set up and constrain a second copy of the character to maintain visual continuity of geometry across the portal boundary. At left, as seen from an interactive view above the scene, the primary copy of the character (screen right), reaches through one portal and her arm should appear to exit the other side. The second copy of the character (screen left) is constrained using our teleportal constraint solver and responds interactively as we modify the primary copy of the character or either portal, and the procedurally geometry culling prevents the portion of the character behind the portals from being visible. When seen from the primary camera (bottom), the teleportal constraint solver positions cameras to create the pre-pass textures for the portals that maintain the illusion of visual continuity into the portals. This interactive view allows us to ensure that the geometry will appear continuous at the boundary for final renders.

The portal constraint solver takes a reference frame on the *here* side of the portal, $\mathbf{M}_H$, and determines a corresponding reference frame on the *there* side, $\mathbf{M}_T$, that maintains visual continuity of both camera views and geometry that span the portal boundary. If the portal reference frames are specified as $\mathbf{P}_H$ and $\mathbf{P}_T$, the solver evaluates the result as $\mathbf{M}_T = \mathbf{P}_T\mathbf{P}_H^{-1}\mathbf{M}_H$. For interactive cameras, M_T determines the transform of the camera for the prepass render. For characters, $\mathbf{M}_T$ determines the top-level transformation. To maintain visual continuity, as well as consistency with the teleported rays in the renderer, we keep remaining parameters in sync for each copy; this includes parameters for the camera's field of view and for the character's deforming pose.

When characters grab objects or other characters through a portal, we need to ensure that our standard constraint solvers and inverse kinematic rigs appear to function correctly through the portal to connect to constraint targets and end effectors on the *there* side. If we were to use these solvers without accounting for the portals, the solvers would create implausible states in which a character is stretched across normal space to the area of the *there* portal, instead of appearing to pass through the portal. To account for this, we define a second *here* copy of the constraint target or end effector near the *here* portal, which mirrors the original copy near the *there* portal. We then use an inverted version of the portal

constraint solver to position and orient the constraint target or end effector such that it maintains a spatial relationship to the *here* portal that is equal to that of the original to the *there* portal.

As described in Section 1, in a handful of shots, characters pass through portals with a few frames of temporal offset, effectively experiencing a fraction of a second of time travel. While not a story point in itself, allowing for this kind of flexibility has given animators greater freedom to create compelling motion that reads clearly on screen. To support this for animation, we create a character pair for each unique time, and each pair is then constrained across the portal boundary. While we could also constrain body pose parameters between the temporally offset pairs, animators preferred to copy these values to allow them to cheat the pose of the temporally offset pair for visual appeal.

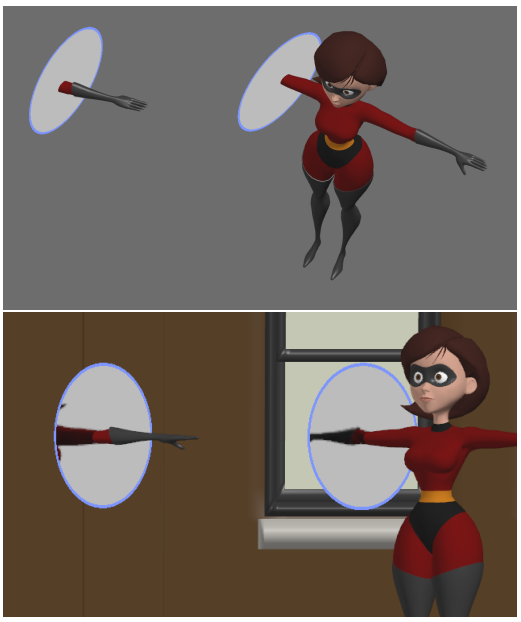

Figure 2: An interactive view of a portal (top) and the resulting view from the primary camera (bottom), maintaining the illusion of visual continuity into the portal. © Disney/Pixar.

4 PORTAL BOUNDARY EFFECTS

The portal boundary was visually conceived as a rotational and unstable ring connecting two locations in space (Figure 1). The boundary is also luminous in nature, emitting light and affecting the objects and characters on both sides of the portal. This design presented two challenges for our effects artists. First, we need to create a procedural animated object that could be quickly tailored to any individual shot, including modifications to scene composition and portal motion. Second, we need the portal boundary to be a luminous and graphic object that would interact with the scene and characters. Furthermore, the object needs to be generated as film renders progress to allow the same boundary to be visible as multiple copies when portals align to the camera, allowing us to set up visually recursive compositions.

To create the boundary shape, we use Houdini to generate curves at the origin, and we modify the frequency and speed to maintain a consistent visual perception of motion, rather than a constant speed. Angular velocity can be modified to account for the distance to the portal, the portal's orientation, and the length of time that it is visible. Counter rotational curves are included and adjusted to create the appearance of instability. Finally, the boundary can be exported to our Presto animation software to share the actual geometry and motion our to animators and camera and staging artists, as the effect can affect shot composition and character motion.

To render the boundary, shaders are authored for RenderMan to enable the rendering of the boundary, transport geometry, and environment in a single pass. Since we sometimes need to align portals, to allow us to recursively see copies of the portal inside itself at various depths, we are limited in our practical ability to use traditional compositing methods. We cache motion blur values into the curve geometry to save render time and to avoid the complexity of setting up accurate curved motion blur for curves moving at a variety of speeds. Color and intensity are also baked into the geometry to support reflections onto characters and the environment. The boundary also determines the edge of the region where we apply ray teleportation, so we export moving curve data in point clouds that we can access in the shader for the portal surface.

5 CONCLUSION

By supporting spatial teleportation directly through light transport, we have enabled workflows that allow users to work with single shot setups that natively support the rendering of the view through the portals. This interactive layout and animation toolset has simplified how we can incorporate the view through the portals into an existing shot, allowing users to creatively experiment with interactive feedback. While there is a learning curve for users who work with this approach, it's enabled the creation numerous shots that include complex character interaction with a number of portals.