ABSTRACT

The soul characters in Disney/Pixar’s Soul have a stylized appearance that sets them into a unique world, which introduced many new challenges. Everyone from the art department and character modelers and shaders to the technical directors and developers in the effects, lighting, and software groups collaborated to bring this new visual style to screen. The soul world is abstract and ethereal; this needed to be balanced with visual clarity and design appeal.

As our character rigging and animation tools use rigged surfaces, a key challenge was presenting a new representation derived from this data that meets our visual goals. To achieve softness of volumetric form and dynamically changing linework, we built a system to procedurally generate this data in Houdini. Significant numerical computation was required to create this data at the fidelity required. We developed an automated system for managing this computation in a configurable way, while keeping data for downstream renders in sync with changes to character performances.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGGRAPH ’20 Talks, August 17, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7971-7/20/08.
https://doi.org/10.1145/3388767.3407361
We use Houdini to create per-frame volumes and volumetric signals for the IDriver shading node. To make sure the animation reads well with semi-transparent volumes, most of our workflows are directed towards camera. Each character is made of several individual volumes; most of these are partially or completely culled by self occlusion, creating a cleaner read with no layered shapes. For example, the teeth and tongue are precisely cut to camera to match the open mouth shape. The eyes are surface geometry, not a volume, so we create a masking texture instead of cutting them.

Every shading signal (eye shadow, cheeks, lips, etc.) is also a volume; and many are projected from the camera. We use these to adjust the color and shader lobe properties in the IDriver and shading network. We build the normals volume by projecting the surface normals through the character from camera. For the soft normals volume, we convert the surface to an SDF, dilate it, apply an iterative mean value blur, and convert it back to a surface to project the normals of this smoothed geometry.

For all the background and secondary souls, we generalize the houdini network to work for all of their designs (Figure 1, two right characters). They only need a handful of parameters for specific control. While each soul is modeled as their own unique shape, most share the same topology, which greatly streamlines the process. The same network also creates rest pose volumes for our soul crowd accessories. Additional portions of the soul look development process are described in [Fong et al. 2020].

2 BUILDING VOLUMES

We use Houdini to create per–frame volumes and volumetric signals for the IDriver shading node. To make sure the animation reads well with semi-transparent volumes, most of our workflows are directed towards camera. Each character is made of several individual volumes; most of these are partially or completely culled by self occlusion, creating a cleaner read with no layered shapes. For example, the teeth and tongue are precisely cut to camera to match the open mouth shape. The eyes are surface geometry, not a volume, so we create a masking texture instead of cutting them.

Every shading signal (eye shadow, cheeks, lips, etc.) is also a volume; and many are projected from the camera. We use these to adjust the color and shader lobe properties in the IDriver and shading network. We build the normals volume by projecting the surface normals through the character from camera. For the soft normals volume, we convert the surface to an SDF, dilate it, apply an iterative mean value blur, and convert it back to a surface to project the normals of this smoothed geometry.

For all the background and secondary souls, we generalize the houdini network to work for all of their designs (Figure 1, two right characters). They only need a handful of parameters for specific control. While each soul is modeled as their own unique shape, most share the same topology, which greatly streamlines the process. The same network also creates rest pose volumes for our soul crowd pipeline that uses volumetric deformation [Ouellet et al. 2020]. Volumes for facial linework (Figure 4, left and center) are generated by extruding the rigged flat meshes away from camera to form the volume. For silhouette lines that are procedurally generated, we create ribbons perpendicular to camera along the curves before we extrude those also away from camera (Figure 4, right).

3 PIPELINE

Our Houdini networks generate multiple passes of high fidelity volume data. This is computationally intensive and needs to happen regularly; we designed a system called hexport (houdini export) that automatically generates the data by parallelizing it on our renderfarm. The system is highly tunable, giving departments different parallelization and scheduling behavior based on turnaround needs.

Early in production, character artists used our existing effects pipeline. Effects artists also work with computationally expensive volumes, and we have developed tools for parallelization and large data storage. Other departments export animated USD assets in a single file with a time sample per frame, but this does not allow for immediate parallelization, as a race condition exists on file writes. Instead, our effects artists export a file per frame and use the utility usdstitchclips to assemble per frame data into USD Value Clips. On a USD stage, this appears identical to time samples in a single file 1.

We used this for early look development, but this did not scale to character assets that are shared among multiple users and departments. We developed a versioning system that includes multiple unpublished working directories, which we store in high performance storage for parallel renderfarm use. Publishing a working directory for use in downstream departments involves only a symlink and database change. In addition, we moved from storing houdini network data in shot level hip files to character asset HDA (houdini digital asset) files that are shared among relevant shots. When exporting a shot, the hexport system creates a shot level hip file and references the current character HDAs into it; these HDA subnetworks read shot–specific cached surface animation, convert it to volumes and linework, and export results to disk (Figure 5).

An important part of hexport is partitioning and scheduling network computation. Effects artists parallelize networks, but the farm scheduler does not scale to a task for every character on every frame. Most of our computation is not history dependent; this increases partitioning and scheduling flexibility. When exporting a shot, it is initially split into subsets of characters, and the network for each character is partitioned into three classes of compute tasks: expensive but parallelizable across frames, tasks for aggregating data from prior tasks, and independent single process tasks. Portions of each network’s computation are assigned to appropriate tasks.

Hexport also schedules time–dependent character simulations. We assign these to the independent single–process task and rely on houdini for thread parallelization. We tune thread allocation and resources for each task type based on turnaround needs and experimentally measured data. As we create tens of terabytes of high fidelity volume data per night that downstream renders read in parallel, we use a cached filesystem designed for heavy write loads and many parallel reads. To avoid copying data, we use symlink labels to indicate whether data is IP or published for rendering.

ACKNOWLEDGMENTS

We thank George Nguyen, Junyi Ling, and Ling Tu for character development, Matt Kuruc and Elliott Cattell for pipeline development, and Boris Dalstein, Tolga Goktekin, and Fernando de Goes for silhouette software development.

REFERENCES


1 graphics.pixar.com/usd/docs/USD-Glossary.html#USD-Glossary-ValueClips