Real-Time Refraction Shader for Animation

Antoine Domon
Animal Logic
Sydney, NSW, Australia
antoine.domon@al.com.au

Ankit Sinha
Animal Logic
Sydney, NSW, Australia
ankit.sinha@al.com.au

Zhicheng Ye
Animal Logic
Sydney, NSW, Australia
zhicheng.ye@al.com.au

Valerie Bernard
Animal Logic
Vancouver, BC, Canada
valerie.bernard@animallogic.ca

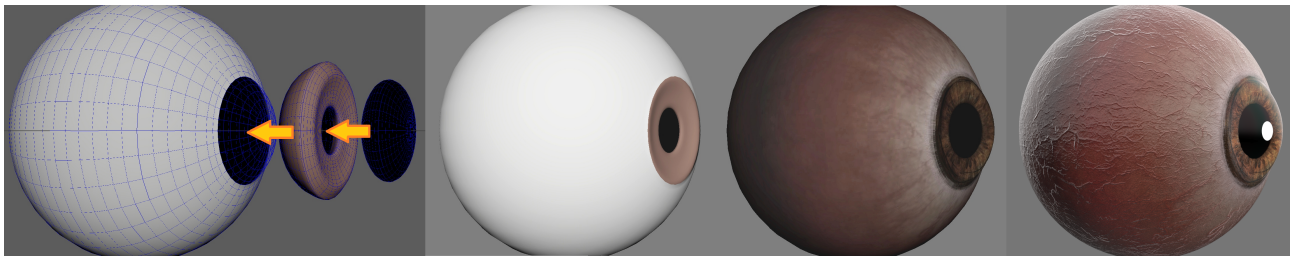

Figure 1: a) The cornea mesh encompasses the iris and pupil and has transparent faces on the tip acting as a refractive surface; b) Default viewport render; c) Viewport render with textures and refraction shader; d) Render of the same frame with *Glimpse*;

ABSTRACT

We present Animal Logic's solution to simulate light refraction in deformable characters' eye corneas in Autodesk® Maya®'s viewport, with a result close to the final render, significantly reducing iterations for facial animation workflow. Our approach is tightly integrated with Animal Logic's GPU-based deformation engine for a minimal impact on playback. The refraction is generated automatically from the scene's geometries and a simplified shading definition exported by the lookdev department using Pixar® Universal Scene Description. It has proven to give reliable results, allowing for its adoption in production for all current and upcoming shows.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**; **Animation**.

KEYWORDS

animation rig deformation USD Maya OptiX CUDA

ACM Reference Format:
Antoine Domon, Ankit Sinha, Zhicheng Ye, and Valerie Bernard. 2024. Real-Time Refraction Shader for Animation. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '24 Talks), July 28-31, 2024.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3532836.3536247

1 INTRODUCTION

Rasterization graphics pipeline has always been a standard in the animation industry. While it has demonstrated strong abilities for fast playback, it often lacks rendering features that might be critical for validating a shot at the animation stage. One of them is the refractive nature of transmissive materials that cannot be represented in the viewport of most animation software.

Light refraction in the cornea plays a significant role in the perceived shape of the iris and the pupil, which are crucial elements for characters' expressiveness. Integrating it improves productivity for facial performance because it allows animators to have a close overview of the look of the eyes downstream in the pipeline, thus reducing iteration cycles between lighting and animation departments, in particular for shots where the camera angle has an impact on the look of the eyes.

2 RELATED WORK

Simulating refraction in a rasterization graphics pipeline is commonly achieved through screen space methods involving two render passes with a depth and a color buffer [Tipprasert and Kanongchaiyos 2006]. These approaches are limited as they can lead to artifacts, and are unsuitable for the Maya VP2 rendering pipeline because handling multiple render targets can only be achieved at the renderer level, while we want to manage it at the object level.

In our initial approach, we assumed the iris and pupil to form a circular plane, allowing us to estimate the refraction ray intersections with the inner shapes and thus shade it instead of reading screen buffers. This method has a high maintenance cost because TDs must tweak the shader parameters per show per character and update them whenever the model changes. It also needs extra parameters to reflect the deformation of the inner shapes.

Many techniques exist for producing high-quality results in the viewport, but either they suffer from the abovementioned issues, or they do not integrate seamlessly into our pipeline. Overlaying an offline render of the eyes in the viewport [Lopez and Richards 2017] could be a solution, but we could not achieve it in real-time as the animator manipulates the rig. We want to generate it automatically from the live scene and basic shading information, while still providing reliable results and not impacting interactivity. Most of all, it must support eye deformation to not put limits on the rigs and address any animation style. We also wanted to leverage our

SIGGRAPH '24 Talks, July 28-31, 2024, Denver, CO, USA

ACM ISBN 978-1-4503-9371-3/22/08.
https://doi.org/10.1145/3532836.3536247

deformation engine *Bond* [Baillet et al. 2020], which has native support for GPU processing, to keep the geometry data on the graphics card from the deformation to the rendering stage. As its evaluation relies on NVIDIA® CUDA® library, this was an opportunity to simulate refraction with raytracing using NVIDIA® OptiX®, which is also CUDA-based, thus facilitating data transfers for rendering.

3 PIPELINE INTEGRATION

The eye's materials as defined by the surfacing department usually involve complex shading networks designed to be ingested into our in-house renderer *Glimpse*. The first challenge was to build a system that turns this shading definition into a simpler representation more compliant for real-time rendering, and make it part of the asset's USD data so it can be consumed by downstream departments.

We created a new HDA in SideFX Houdini® that bakes the result of our *Glimpse* shaders. We apply it to the cornea and iris materials to extract an AOV for both diffuse and IOR channels. Assuming refraction is mostly constant all over the surface, we retrieve the value of the brightest pixel in the IOR's AOV through Open Image IO; thus simplifying the shading definition for real-time rendering.

We export this result as a *USDPreviewSurface* material assigned to eye primitives with a "preview" purpose, which ensures this is only used for viewport display, and not for final rendering. The process then takes care of naming the material and textures. This naming convention follows our Open Color IO rules, which in turn tells consuming DCCs what colour space they are in. The USD material and textures are then checked in along with other lookdev data and follow the asset alongside its usual Lookdev pipeline.

4 MAYA INTEGRATION

At shot load time, our rigging framework translates the *USDPreviewSurface*'s *diffuseColor* and *ior* inputs into attributes in a custom shader node assigned to all the eye shapes. On playback, we retrieve post-deformation vertex buffers generated by *Bond* for those shapes, to make them available in the shader.

Geometry buffers can be copied directly into the appropriate OptiX structure without transferring any memory from device to host because *Bond* also uses CUDA for storing data. None of those operations block the CPU application and can all be evaluated concurrently with the rest of the rig.

We leverage Maya's DG dirtiness engine to synchronize *Bond* and OptiX. Any time a shape node bound to this shader is dirty, we use *Bond*'s API to know if it has re-computed any of the involved buffers since the last update, and if so, we specify which OptiX component must be updated before the next draw call.

5 OPTIX IMPLEMENTATION

We build one Geometry Acceleration Structure (GAS) per eye shape from the deformed positions and triangle indices *Bond* buffers. We associate each GAS with the *Bond* shape's transform to create Instance Acceleration Structures (IAS), defining what to raytrace to the OptiX programs. Vertex normals/colors/UVs buffers and shading data are provided to OptiX through a Shader Binding Table. We also upload the viewport camera and lights using a launch parameter to cast the primary rays appropriately and compute the proper shading. This process is illustrated at the top of Figure2.

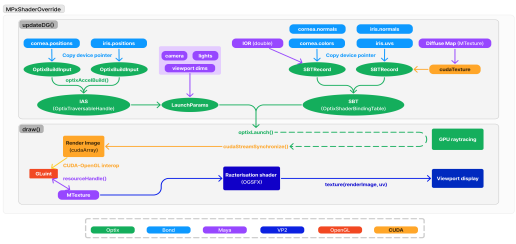

Figure 2: Vertex buffers and USD shading data are uploaded to OptiX. The resulting render is sampled in the VP2 shader.

At each VP2 draw call, we launch the OptiX pipeline to invoke the raytracing on the GPU. The ray generation program casts one ray per pixel in a grid with the same resolution as the current viewport. In the ray-hit program, if the hit occurs on a refractive shape, we throw a new ray computed from Snell's law using the normal on the surface and the IOR defined by the asset's USD data. Otherwise, we calculate a Phong shading from the Maya viewport's lights and the surface normal on the intersection.

The host can then synchronize the GPU evaluation to retrieve the render's image as a CUDA buffer and make use of CUDA-OpenGL interoperability to copy it directly into an OpenGL texture, which is then sampled from the Maya VP2 rasterization shader as shown at the bottom of Figure2. Not transferring this texture from device to host at any stage of the process is crucial to avoid FPS drop.

6 RESULTS, LIMITATIONS AND FUTURE WORK

We support Maya viewport features, such as multi-viewport, 2D panning and zooming, film gate, and cached playback, allowing the shader to integrate seamlessly into the animation workflow.

As we now integrate the post-deformation vertex data in the shader, the result is correct, whatever the deformation applied to the eyes, and it no longer requires any tweak if the eye model is updated upstream in the pipeline.

We profiled multiple shots among three different shows and noticed an FPS overhead of less than 2%, regardless of the number of characters on the screen. For example, we measured a drop of only 0.5 FPS on a complex shot with two animated characters running at 30.4 FPS. GPU memory usage is about 12% higher, and CPU memory is not impacted at all.

Limitations include unsupported textured IOR, limited support of light types, and animators having to turn on the viewport Textured mode. In the future, we plan to generalize the system to apply it to other refractive shapes of the character, such as eyeglasses.

REFERENCES

Aloys Baillet, Josh Murtack, Hongbin Hu, Haoliang Jiang, and Michael Quandt. 2020. Bond: USD-Integrated Hybrid CPU, GPU Deformation System. In *ACM SIGGRAPH 2020 Talks* (Virtual Event, USA) *(SIGGRAPH '20)*. Association for Computing Machinery, New York, NY, USA, Article 34, 2 pages. https://doi.org/10.1145/3388767.3407324

Pilar Molina Lopez and Jake Richards. 2017. The eyes have it: comprehensive eye control for animated characters. In *ACM SIGGRAPH 2017 Talks* (Los Angeles, California) *(SIGGRAPH '17)*. Association for Computing Machinery, New York, NY, USA, Article 24, 2 pages. https://doi.org/10.1145/3084363.3085061

Nuttachai Tipprasert and Pizzanu Kanongchaiyos. 2006. An interactive method for refractive water caustics rendering using color and depth textures. 423–428.