Transparency Rendering in Cyberpunk 2077

Peter Sikachev
peter.sikachev@gmail.com
Independent
Warsaw, Poland

Szymon Ślęga
szymon.slega@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

Kamil Nowakowski
kamil.nowakowski@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

Karol Kowalczyk
karol.kowalczyk@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

ABSTRACT

This talk discusses in-detail the transparencies pipeline of Cyberpunk 2077. We present a high-level overview of the system, and then provide details on individual components. Particularly, we discuss our take on decoupled particle lighting (DPL), our distortion approach, and a parallel slab refraction approximation. We also provide performance details on target platforms for these features.

CCS CONCEPTS

• **Computing methodologies** → **Reflectance modeling**; *Rasterization*; *Image processing*; • **Applied computing** → **Computer games**.

KEYWORDS

transparency, real-time rendering, particles, refraction, game development

ACM Reference Format:
Peter Sikachev, Szymon Ślęga, Kamil Nowakowski, and Karol Kowalczyk. 2021. Transparency Rendering in Cyberpunk 2077. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '21 Talks), August 09-13, 2021.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3450623.3464629

1 INTRODUCTION

In the highly vertical, futuristic setting of Cyberpunk 2077, a lot of transparent objects are visible in a typical frame. These can include: particles from ambient smoke, VFX from a gunfight, glass windows, clothing made from semi-transparent plastic, etc. Additionally, being a first-person game, a higher level of graphical fidelity is required, as more props appear closer to the player.

With all those requirements in mind, we needed to create a robust and efficient workflow for our transparent objects. Firstly, a solution was needed for particle lighting. Per-vertex lighting yields results of too low quality, while per-pixel lighting is too slow. [Persson 2012] suggested using tessellation for particles and computing lighting at tessellated vertices, but this results in overheads from using the tessellation pipeline and too many small triangles. [Sousa and Geffroy 2016] suggested precomputing particle lighting in texture space and applying it later in the pixel shader, but few implementation details were given and there are some caveats along the way.

Next, an effective solution for glass and other transparent materials was needed. We needed to be able to render multiple layers of refractive and/or "frosty" surface fast and have a high-quality lighting on it.

Unlike [Geffroy et al. 2020], being an open-world game, we could not allow a solution that would impose strict limits on number of visible layers. Therefore, we developed rendering tech allowing arbitrary number of transparent surfaces of any type (mesh with any custom material or particle of any type) with refraction, distortion, frosted (blurry) glass effect or any combination of thereof. Figure 1 shows one of many possible scenarios that our system needed to handle seamlessly.

Figure 1: Looking out onto a complex city from a car window covered with refractive water droplets.

As a baseline refraction technique, we used a single bounce approach proposed in [Sousa 2005]. This method, however, does not particularly well approximate the plane-parallel plate refraction that occurs in flat glass. Therefore, we proposed an alternative parallel slab refraction approximation technique that allows for better representation of this phenomena (see Figure 2).

SIGGRAPH '21 Talks, August 09-13, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8373-8/21/08.
https://doi.org/10.1145/3450623.3464629

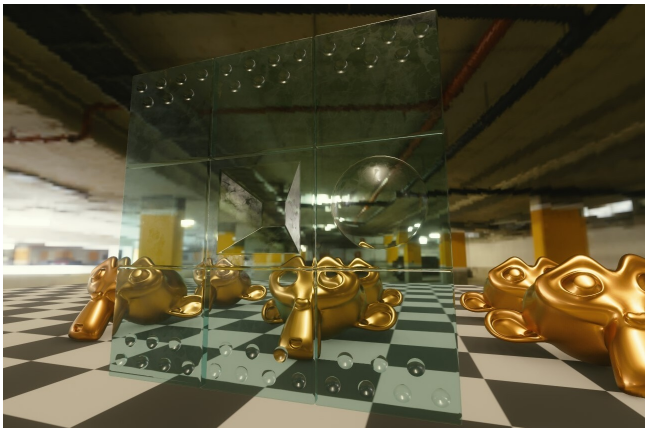

Figure 2: Parallel slab refraction. Notice how refraction in the flat regions of the glass only shifts the image without distorting it too much.

Finally, we discuss several custom transparent effects implementation. We chose vehicle skid marks and holograms as interesting case study examples.

2 IMPLEMENTATION AND RESULTS

Particle Lighting. We used different lighting techniques for particles, depending on their material and type: per-vertex, per-pixel, and DPL (see Figure 3). For DPL, we support multiple particle types by reading in compute shader the particle vertex buffer and replicating parts of the vertex shader that affects particle position.

Figure 3: A scene with lots of particles and distortion. Smoke billboards are lit using DPL, smoke trails are lit per-vertex.

We improved previous DPL implementations by better DPL texture utilization, efficient batching, and culling. We split DPL pass into two main stages: allocation and lighting.

The allocation stage included culling invisible particles using hierarchical depth buffer, computing the number of tiles of each size in advance, and assigning them to each particle. This allowed us to pack tiles efficiently into the DPL texture for any size distribution without making any prior assumptions. In the lighting stage, we simply launched an indirect dispatch call for each of the tile sizes.

In addition, we introduced a simple-yet-efficient empirical lighting model for particles that simulates particle backlighting with plausible results. Our implementation runs within 0.5 ms budget on PlayStation 4 with 1024×1024 DPL texture for most of the scenes.

Glass Blending. Unlike smoke and other participating media, glass tints the objects behind it, in addition to "additive blending effects". Therefore, traditional alpha blending does not work: we would need to sacrifice either one or the other. To address that, we utilized dual-source blending. This allowed us (with some limitations) to have the best of both worlds with little to no overhead.

Deferred Distortion Buffer Pipeline. In order to create both a high-performing and versatile distortion solution, we introduced a deferred distortion buffer, where we create an off-screen buffer which is filled during an additional pass, with screen-space offset introduced by distortion.

This distortion is accumulated for all the objects along with the blurriness. During this process, we mark tiles of the screen affected by it for optimization. Afterwards, a postprocess is ran on those affected tiles. For a very distortion/frosted glass-heavy frame, the distortion cost yielded under 1 ms for the post process resolve and 0.3 ms for accumulation pass on Xbox One X at 2160p resolution. Figure 4 shows an example of frosted glass used in game.

Figure 4: Frosted glass in the window of a diner.

The main limitation of this technique is that front transparent objects do not blur or distort back ones. We found that thoughtful approach to content creation greatly mitigates this problem.

REFERENCES

Jean Geffroy, Yixin Wang, and Axel Gneiting. 2020. Rendering the Hellscape of Doom Eternal. In *SIGGRAPH 2020 Advances in Real-Time Rendering in Games*. Online.

Tobias Persson. 2012. Practical Particle Lighting. In *GDC 2012*. San Francisco, CA, USA.

Tiago Sousa. 2005. Generic Refraction Simulation. In *GPU Gems 2: Programming Techniques For High-Performance Graphics And General-Purpose Computation*, Matt Pharr and Randima Fernando (Eds.). Addison-Wesley.

Tiago Sousa and Jean Geffroy. 2016. DOOM: The Devil is in the Details. In *SIGGRAPH 2016 Advances in Real-Time Rendering in Games*. Anaheim, CA, USA.