Shadows Optimizations in Cyberpunk 2077

Bartłomiej Dybisz
bartlomiej.dybisz@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

Michał Witanowski
michal.witanowski@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

ABSTRACT

This talk presents significant rendering-related CPU and GPU optimizations concerning shadows in Cyberpunk 2077. Given the scale of the game and the variety of platforms that it has to support, different solutions had to be implemented and coupled in order to fulfil the time and memory requirements. We also introduce our take on runtime and offline techniques for object culling and shadow maps caching in order to minimize potential overhead. The article concerns three different shadow types that can be spotted around Night City: cascaded, distant, and local shadows.

KEYWORDS

shadows, rendering, optimization, culling

ACM Reference Format:
Bartłomiej Dybisz and Michał Witanowski. 2021. Shadows Optimizations 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/10.1145/3450623.3464679

1 INTRODUCTION

Efficient rendering of shadows was one one of the bigger challenges we encountered during the development of Cyberpunk 2077. Due to Night City's verticality and density, the amount of meshes and triangles required to be rendered into shadow maps greatly exceeds those we dealt with in the *The Witcher 3: Wild Hunt*. Moreover, Cyberpunk 2077 supports a full day-and-night cycle. Because the city is vast and the shadows need to be visible from far distances in order to maximize believability, we had to implement numerous optimizations to reduce the number of meshes being drawn, which translated to both CPU and GPU times reduction.

2 GLOBAL SHADOWS

Global shadows are used for the Sun and the Moon directional light sources, and are implemented as cascaded shadow maps [Rouslan and Sainz 2007]. Nearby and dynamic objects rely on per-frame rendered shadow maps, referred to as *cascaded shadows* later on. Similarly as in [Kasyan 2013], distant objects rely on shadows refreshed periodically, where rendering is spread over multiple frames. These are referred to as *distant shadows*. We use up to 4 levels of cascaded shadows and 4 levels of distant shadows.

2.1 Distant shadows

In order to handle directional light shadows over the vast area of Night City, rough shadow map data which covers the entire map is needed. Distant shadows also provide shadow information for objects that are out-of-screen, which is crucial for indirect lighting, such as: reflection probes, global illumination, and ray-traced reflections. The shadow cascades span a box around the player, each one 4 times bigger than the previous one. The last cascade covers the whole 256 km^2 map.

Rendering of distant shadows is pretty straightforward and does not require special attention as we can control the per-frame budget we spend on it. Usually, rendering all meshes takes several frames, so the shadow map is refreshed every few seconds. However, care had to be taken while collecting the meshes to be drawn, as this step cannot be spread over frames. With this in mind, we performed multi-threaded frustum culling and sorted the meshes to achieve optimal batching. We also set the limits of how many meshes and triangles can be rendered in one batch.

2.2 Cascaded shadows

Initially, we used a "Map-Based Cascade Selection" approach as described in [S. White and Satran 2018], which means that all the cascades share the same near plane, and during shading we picked the best resolution cascade level. While this approach ensures the best shadow map space utilization, it quickly became evident that it leads to many objects being rendered unnecessarily, and also leads to uneven shadow resolution throughout the screen-space, depending on light and camera angle. Thus, we opted to use "Interval-Based Cascade Selection" which means that we select cascade level in a shader based just on pixel depth. This way, only those meshes with projections that intersect with the camera frustum slice should be considered during shadow map rendering. This enabled us many culling opportunities, as described in the following sections. Although a lot of shadow map texture space is left unused, we alias the memory with HUD and post-process passes.

2.2.1 *Frustum culling.*

Naive implementation of cascaded shadows [Rouslan and Sainz 2007] assumes culling meshes against cuboid-shaped, 6-plane frustum. This can lead to drawing meshes that will never contribute to shadows within the main camera frustum slice. To solve this issue, we constructed a tight convex shape enclosing the frustum slice comprised of up to 8 planes. The box-frustum intersection test is SSE-optimized so, in case of regular 6-plane frustum, two SSE lanes remained unused. This way, switching to the test against an arbitrary 8-plane convex shape was basically free. An example culling shape is presented in the Figure 1.

2.2.2 *Main view occlusion culling.*

The occlusion buffer of the main camera is used to discard meshes when their shadows are fully occluded by other geometry. We compute the oriented bounding

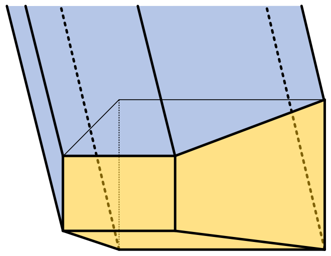

Figure 1: The image shows a shape used for shadow culling. The camera frustum slice is stretched towards the light source direction. Planes highlighted in orange come from the original frustum shape. Blue ones, however, are generated from the frustum edges and direction vector In this case, the final shape is made up of 8 planes.

box of a mesh bounding box stretched along the direction of the light. This is especially helpful in tight indoor areas.

2.2.3 Shadow view occlusion culling. Finally, we capture the first level of distant shadow map and use it as an occlusion buffer from the light's point of view. This helps to reject specific meshes that were not already excluded by the time-of-day visibility mechanism described in the following section.

2.3 Time-of-day visibility

Not all meshes contribute to shadows. Some of them are fully *included* in other objects' shadows (please refer to Figure 2). In order to leverage that fact we introduced an offline system for establishing mesh visibility from global shadows perspective. The information is measured at 12 time intervals (every 2 hours) using objects IDs rasterization and *baked* into mesh instances. This way, depending on the time of day, in the runtime we can record mesh drawcall for global shadows or completely drop it.

Given the vast area of Cyberpunk 2077's world, taking a couple of snapshots of the whole map at different hours is not possible due to texel precision. Instead, we split the game's area into smaller sectors and process them separately. Each sector is further chopped vertically and analyzed including 8 neighboring sectors in order to account for potential inter-sectors occlusion. The process was integrated in nightly builds, so that it catches up with new content submitted by artists.

*T*he time-of-day visibility system proved to be worth implementing. Although stabilising it and educating artists was not an instant process, in the end we were able to decrease the number of triangles rendered for global shadows by millions.

3 LOCAL SHADOWS

Not only does Cyberpunk 2077's world cover a significant area, but it also consists of very dense environments full of artificial light sources. In order to achieve high and consistent visual fidelity, a remarkable number of lights were required to cast shadows (henceforth called *local shadows*). In order to address the issue, several systems were incorporated into the game.

For each frame, artists have a fixed amount of shadow maps to work with. All shadow maps can capture static objects, but only

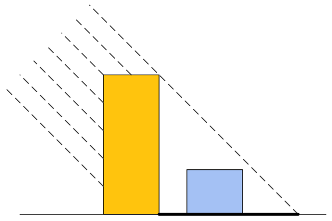

Figure 2: The image shows the blue box not contributing to orange box's shadow. The dotted line represent the Sun ray and bold horizontal line is the produced shadow.

some of them process moving geometry. Shadow maps are picked based on their visibility, and their distance from the player's camera. Visibility is calculated by leveraging the geometry of the already existing light channel system placed around the city by hand. In case a local shadow casting light's frustum intersects the geometry that is visible (tested against CPU occlusion buffer), we consider it as visible. For example, in this way shadows that are encapsulated by a closed room won't be collected unless you can see the room's interior.

Refreshing shadow maps in a frame is throttled, and allows only a certain amount to update per frame (depending on the platform). Notably, resolution differs between platforms. Since most of the artificial light sources do not move, we furthermore split each shadow map into static and dynamic slices in order to cache as many meshes as possible.

On top of that, each shadow casting light has an additional radius and outer angle designated in order to render shadows. We also introduced a system which dynamically changes character meshes rendered to shadows in order to amortize the associated overhead of processing them on less-powerful machines.

Local shadows implementation was a challenging task. Considering the number of shadow maps we wanted to support, and the denseness of the environment overall, we coupled more and more systems in order to reduce the cost. We provided constant assistance to artists, sharing our knowledge with them to help make their work more conscious of, and aligned to, system constraints and interactions.

REFERENCES

Nikolas Kasyan. 2013. Playing with Real-Time Shadows. SIGGRAPH.

Dimitrov Rouslan and Miguel Sainz. 2007. Cascaded Shadow Maps. (Aug. 2007). https://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf

M. Jacobs S. White, D. Coulter and M. Satran. 2018. *Cascaded Shadow Maps.* Retrieved February 12, 2021 from https://docs.microsoft.com/en-us/windows/win32/dxtecharts/cascaded-shadow-maps