"Hustle by Day, Risk it all at Night"

The Lighting of Need for Speed Heat in Frostbite

Kleber Garcia
kgarcia@ea.com
Electronic Arts

Andreas Lindqvist
alindqvist@ghostgames.com
Electronic Arts

Andreas Brinck
abrinck@ea.com
Electronic Arts

Figure 1: In-game image from Need for Speed Heat.

ABSTRACT

Need for Speed Heat is the latest installment in EA's racing game franchise running on the Frostbite engine. The gameplay is centered around daytime and nighttime high speed races with exotic sports cars in a large open world. We improved on the depiction of the cars through a new material system and combination of reflection techniques. For the indirect day and night time lighting, we implemented a sparse GPU irradiance probe grid.

CCS CONCEPTS

• **Computing methodologies** → **Rasterization**; **Rendering**; • **Applied computing** → **Computer games**; *Computer-aided design*; *Media arts.*

KEYWORDS

global illumination, games, reflections, open world, light transport

ACM Reference Format:
Kleber Garcia, Andreas Lindqvist, and Andreas Brinck. 2020. "Hustle by Day, Risk it all at Night": The Lighting of Need for Speed Heat in Frostbite. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '20 Talks), August 17, 2020.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3388767.3407314

1 GLOBAL ILLUMINATION

In previous games the global illumination of dynamic objects used manually placed probe volumes. These were sampled once per object on the CPU using linear interpolation. We developed an automatically placed sparse GPU probe grid which could scale to large worlds similar to [A. Silvennoinen 2015], but using a 3x3x3 node split topology with probes at each corner. We also improved on GPU performance by supporting multi-resolution GPU hardware filtering. Since the player can travel at speeds exceeding 400 km/h, we built the system with the capability to progressively decompress and stream in more detailed lighting.

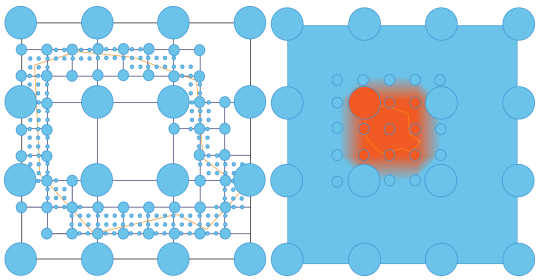

Figure 2: Slice of the irradiance tree in 2D (left), and virtual probe smooth interpolation across levels (right).

The world is split into voxel cells 0.5 km on each side, storing a streamable irradiance tree baked offline using our production path tracer *Flux* [O'Donnell 2018]. The tree decimation process uses a combination of ray tracing and distance transforms [Felzenszwalb and Huttenlocher 2012]. Decimation stops at an artist driven maximum level with the resulting probes being solved and stored in a spherical harmonics (SH) L_1 basis. We use the same pathtracer to produce lightmaps as probes, allowing artists to decide between high quality lightmaps or more flexible but memory expensive probes.

The probes can either be stored in 128 bits using *LogLuv* compression, or at higher quality in 129 bits using *Discrete Spherical Coordinates (DSC)*. In *DSC*, L_0 is stored in 16 bits per channel, $\theta(\vec{L_1})/L_0$ in 9 bits per channel, and $|\vec{L_1}|/L_0$ in spherical coordinates in the remaining bits. At the file system level, Frostbite provides us with *zlib* compression.

We decide which level of the tree to stream in based on a camera location and speed heuristic. Pieces of these levels are uploaded and decompressed on the fly into a volume texture atlas on the GPU.

SIGGRAPH '20 Talks, August 17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7971-7/20/08.
https://doi.org/10.1145/3388767.3407314

Storing the SH coefficients in a texture lets us use hardware filtering when sampling them. We achieved smooth multi-resolution interpolation by placing virtual probes around probe islands on the same level, staying under our streaming, memory, and GPU budgets. These probes are interpolations of their parent level. We sample the probe grid for each pixel of the dynamic objects which significantly improves the indirect lighting detail.

2 MATERIALS

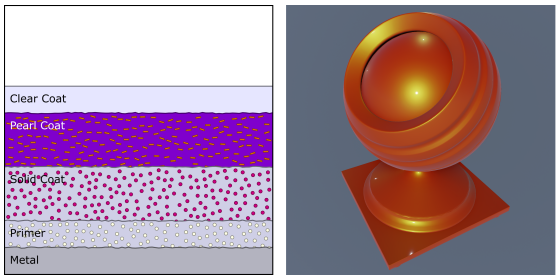

Figure 3: Example of a three stage paint job.

Another aspect we wanted to improve on was our materials, particularly carbon fiber and paint jobs with many layers. Prior to Need for Speed Heat, Frostbite used a fairly standard gbuffer layout with normals, color, smoothness, and metalness. In addition to this there was also an option to cover a material in a thin layer of varnish which shared the normal of the underlying layer and had a hard coded smoothness. To support the new materials we needed to store additional parameters such as a separate color for the specular lobe and a separate normal and smoothness value for the varnish layer.

These extra parameters didn't fit in the existing gbuffer so we introduced the concept of an extended gbuffer which is selectively written to by the advanced materials, and conditionally read by our lighting shader depending on a material identifier. Frostbite uses tiled deferred lighting where the shader used by each tile is determined by which materials and types of light sources are included. Having a permutation of this shader for each combination of materials would lead to an exponential growth of permutations as we add new materials. To avoid this we settled on a model where each material of increasing complexity can also handle all materials with lower complexity. This was a reasonable compromise between memory footprint and performance.

Once we had the extended gbuffer working we found a few other use cases for it. Realistic hair rendering requires a surface tangent as well as parameters describing several specular lobes. With the new system we could fit these within the gbuffer which made it possible to move the opaque pass of our hair rendering to the deferred pass. The previous hair which was completely forward rendered had been too expensive to use in real time cinematics, but with the new deferred approach, this was now possible.

The second use case was retroreflectors such as road signs and number plates. These have an extra specular lobe which is always oriented towards the light source. We didn't need any new parameters for this material, but the infrastructure we built around the extended gbuffer made the material easier to implement.

3 REFLECTIONS

Figure 4: Reflection layers: dynamic reflection volume mask (top-left), static reflection volume mask (top-right), screen space reflections and planar reflections mask (bottom-left), and final reflection composite (bottom-right).

We knew from previous experience that no single reflection technique works well for all of our use cases. For Need for Speed Heat we wanted to combine the strengths of Frostbite's different reflection techniques. This led us to optimize the gbuffer layout and add bits which let our artists specify which reflection system(s) should be used on each object.

The first of these systems is a continuously updated cubemap, centered at the player car and capturing the environment around it (dynamic reflection volume). This system was mainly used on said player car, and select objects in a small area around it.

Second, we pre-generated a set of cubemaps for each time of day, which each captured the look of a distinct area of the world (static reflection volumes). These were used to provide ambient reflections for most of the world geometry.

Third, we used stochastic screen space reflections (SSR) as described in [Stachowiak 2015]. Early and late gbuffer passes were used in combination with two depth layers to achieve artifact-free reflection of foreground objects.

And finally, we rendered the world as reflected through a plane, again centered at the player car. We applied a median filter to a set of vertical ray casts to find a spatially stable plane, and applied a low pass filter to the resulting plane normal to make it temporally stable as well. This planar reflection was used as a fallback solution for SSR. It was applied in areas where there was no valid screen space information available, and on surfaces with roughness values over a certain threshold where the results from SSR would either be too noisy, or require too many rays to be performant.

REFERENCES

V. Timonen A. Silvennoinen. 2015. Multi-Scale Global Illumination in Quantum Break. ACM SIGGRAPH 2015 Avances in Realtime Rendering Course Notes..

Pedro F. Felzenszwalb and Daniel P. Huttenlocher. 2012. Distance Transforms of Sampled Functions. *Theory of Computing* 8, 19 (2012), 415–428. https://doi.org/10.4086/toc.2012.v008a019

Y. O'Donnell. 2018. Precomputed Global Illumination in Frostbite. Games Developers Conference, 2018.

T. Stachowiak. 2015. Stochastic Screen-Space Reflections. ACM SIGGRAPH 2015 Avances in Realtime Rendering Course Notes..