

ゲーム開発のための レイトレーシング

Polyphony Digital Inc.



小澤祐樹

ozawa@polyphony.co.jp

目次

- はじめに
- レイトレーシングとは
- 弊社でのレイトレーシングの歴史
- 弊社での使われ方
- 事故、デバッグ、対策
- 結び

目次

- **はじめに**

- レイトレーシングとは
- 弊社でのレイトレーシングの歴史
- 弊社での使われ方
- 事故、デバッグ、対策
- 結び

はじめに

イントロダクション

自己紹介

- 組み込み業界 → Tango Gameworks → Polyphony Digital
- 趣味はレイトレです。



<https://github.com/qatnonoil>

話すこと、話さないこと

- 主に**レイトレーシング**を**製品**で使った時に起こった事について触れます。
- 仕事でレイトレを書く予定があれば**何かのヒント**になるかもしれません。
- **2018年中頃現在まで**の話です。
- “リアルタイム”レイトレーシングの話ではありません。

目次

- はじめに
- **レイトレーシングとは**
- 弊社でのレイトレーシングの歴史
- 弊社での使われ方
- 事故、デバッグ、対策
- 結び

レイトレーシングとは

簡単な復習

全ての始まり

- 弊社のタイトルは**フォトリアル**な映像を目指しています。
- フォトリアルな映像を目指すのであれば、**レンダリング方程式**を解かなくてはなりません。

レンダリング方程式

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

レンダリング方程式

表面の位置 出る方向

出る光

物自体の発光

物質の反射率

入る光 傾きによる減衰

入る方向

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

全ての方向を足す(積分)

レンダリング方程式

表面の位置 出る方向

出る光

物自体の発光

物質の反射率

入る方向

入る光 傾きによる減衰

①積分大変

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

全ての方向を足す(積分)

レンダリング方程式

表面の位置 出る方向

①積分大変

②再帰大変

入る方向

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

出る光

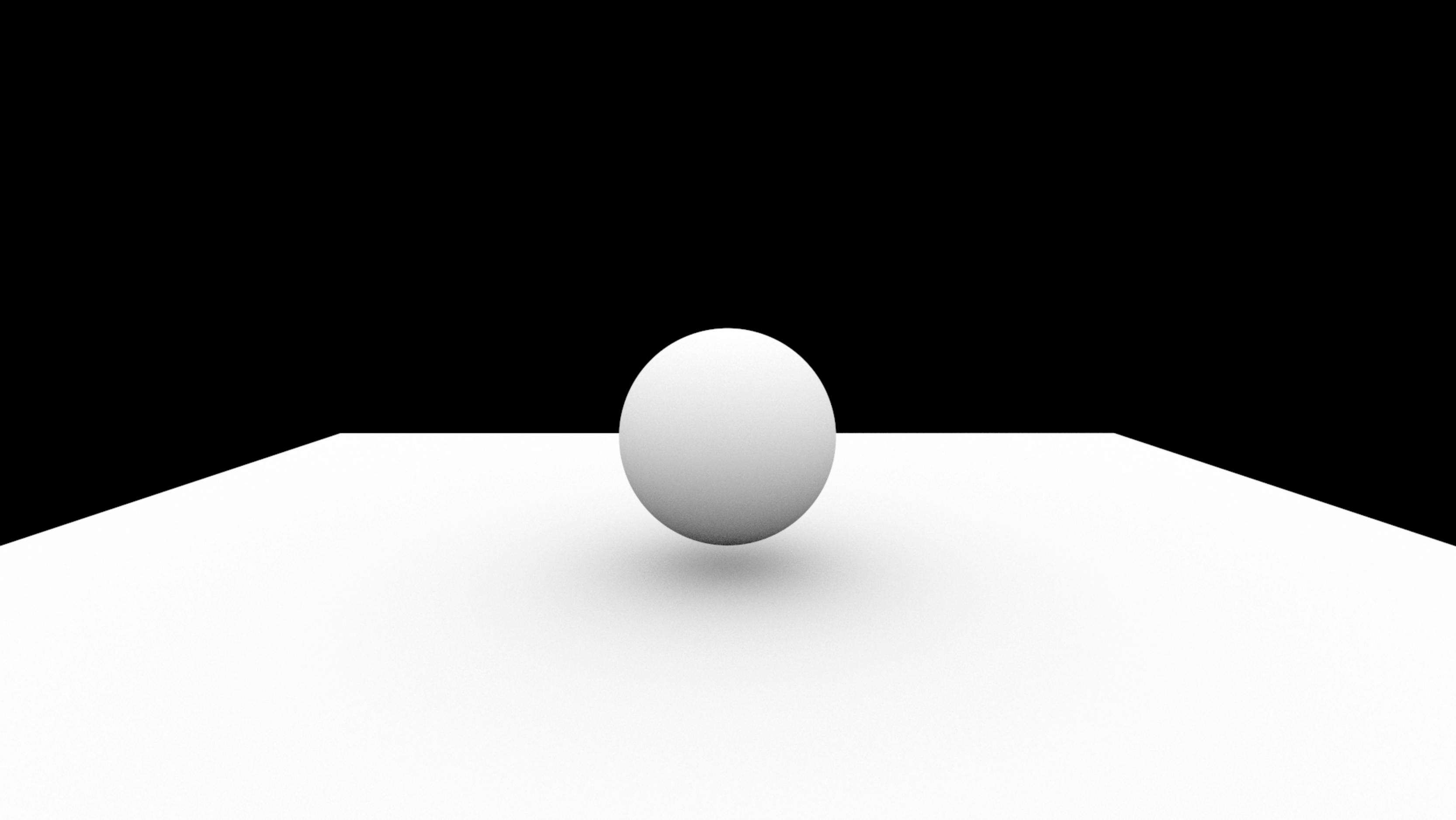
物自体の発光

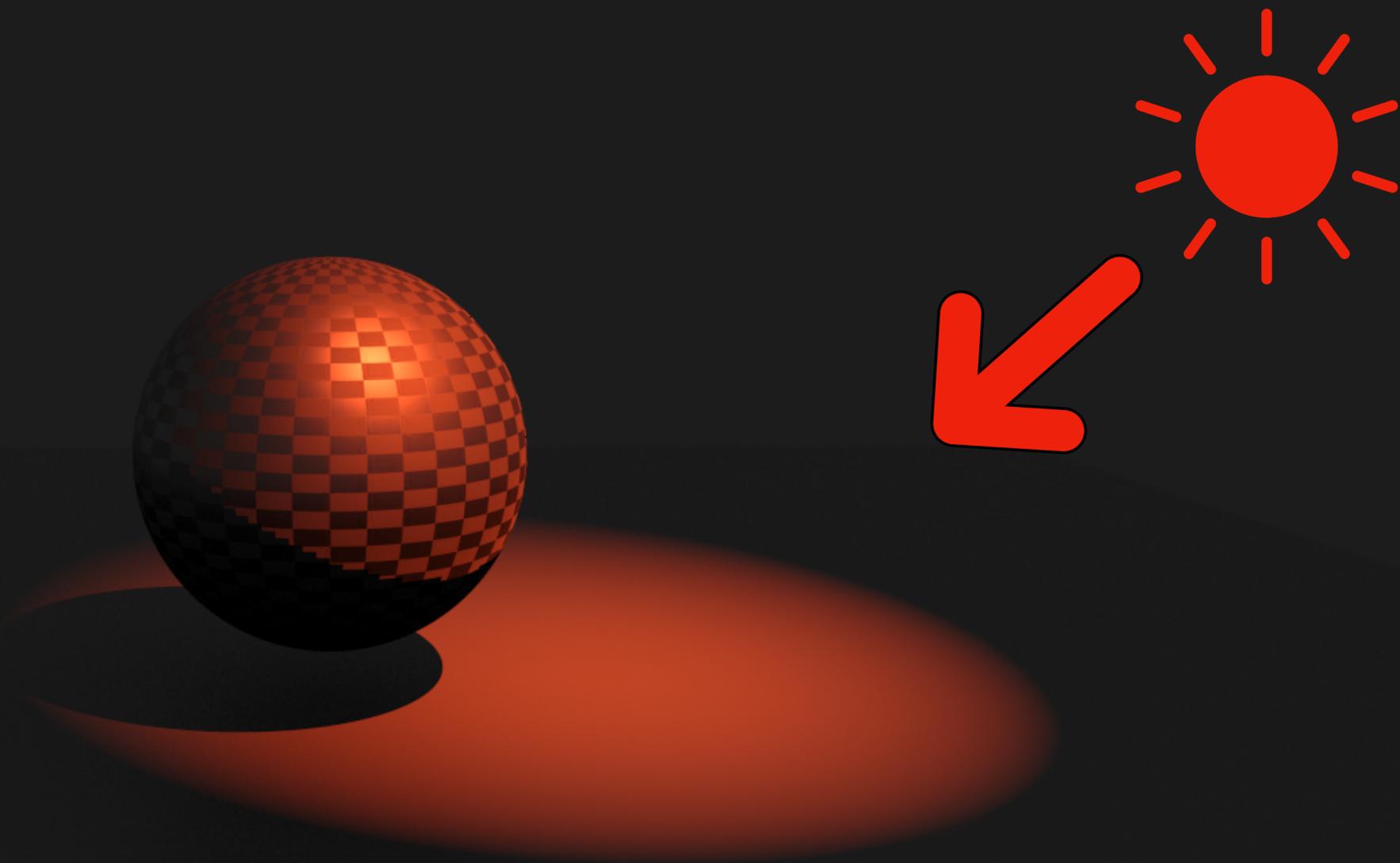
物質の反射率

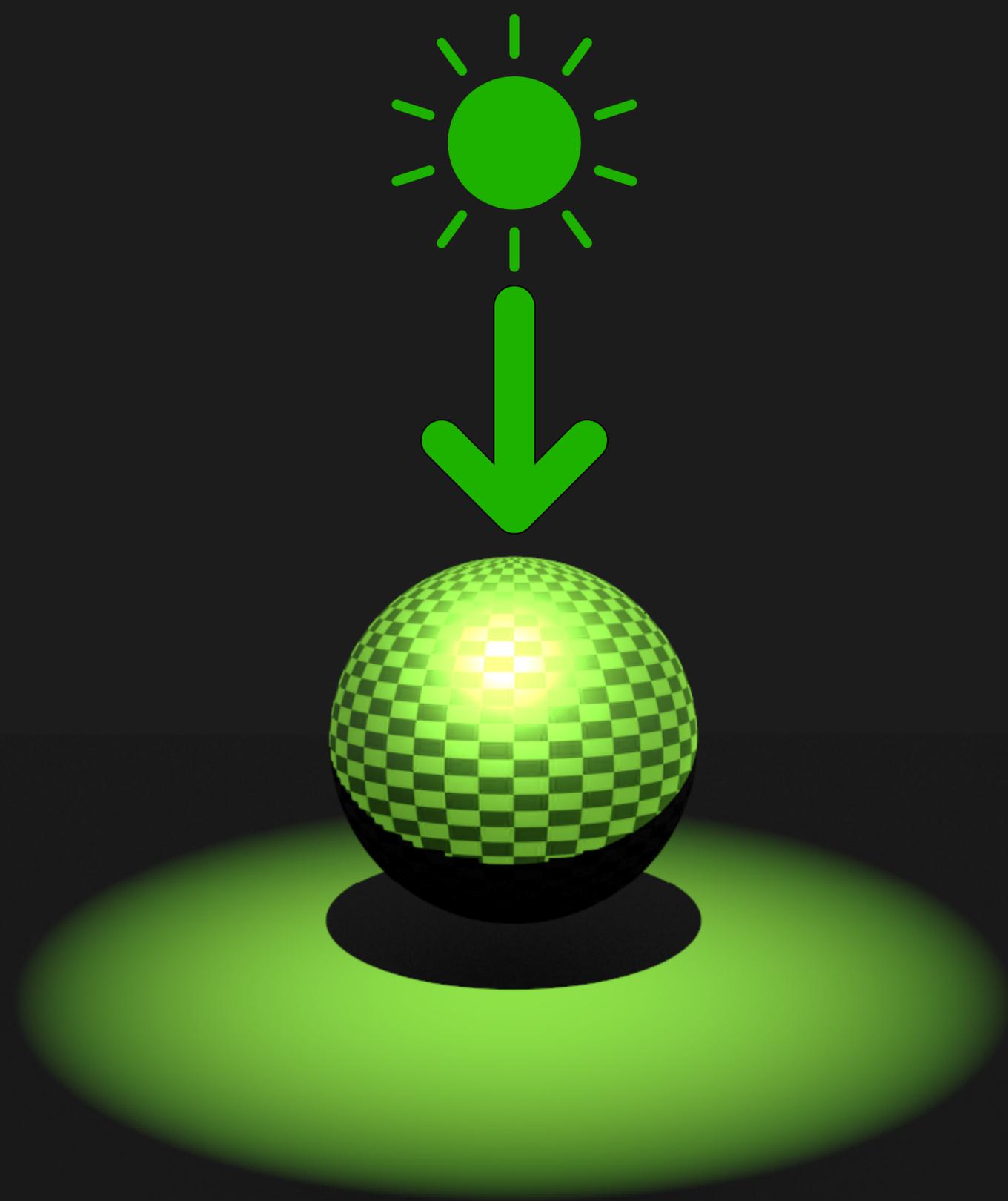
入る光

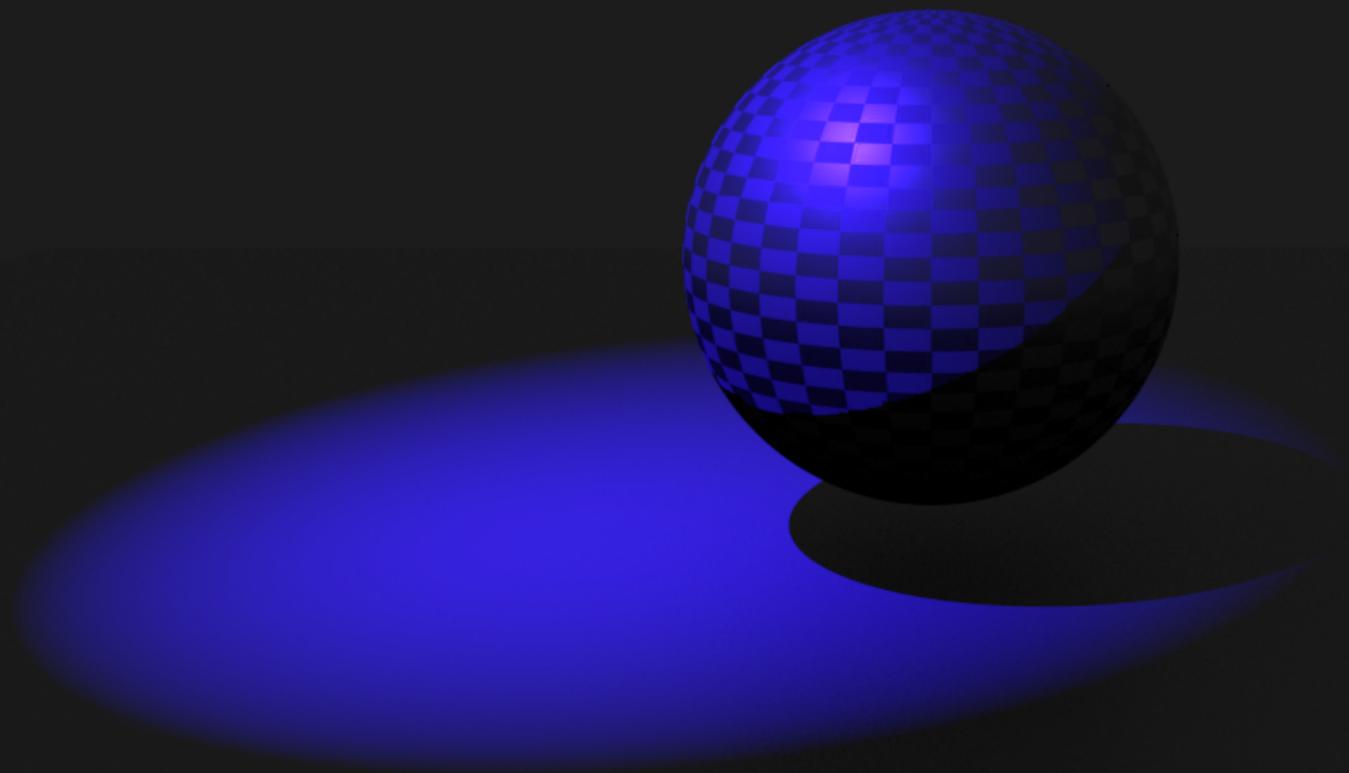
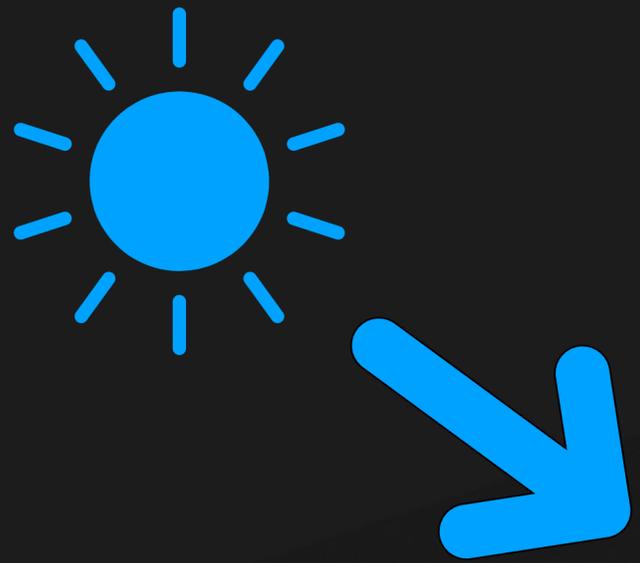
傾きによる減衰

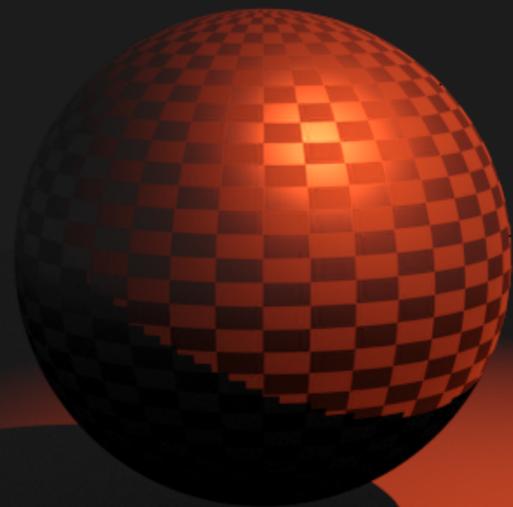
全ての方向を足す(積分)

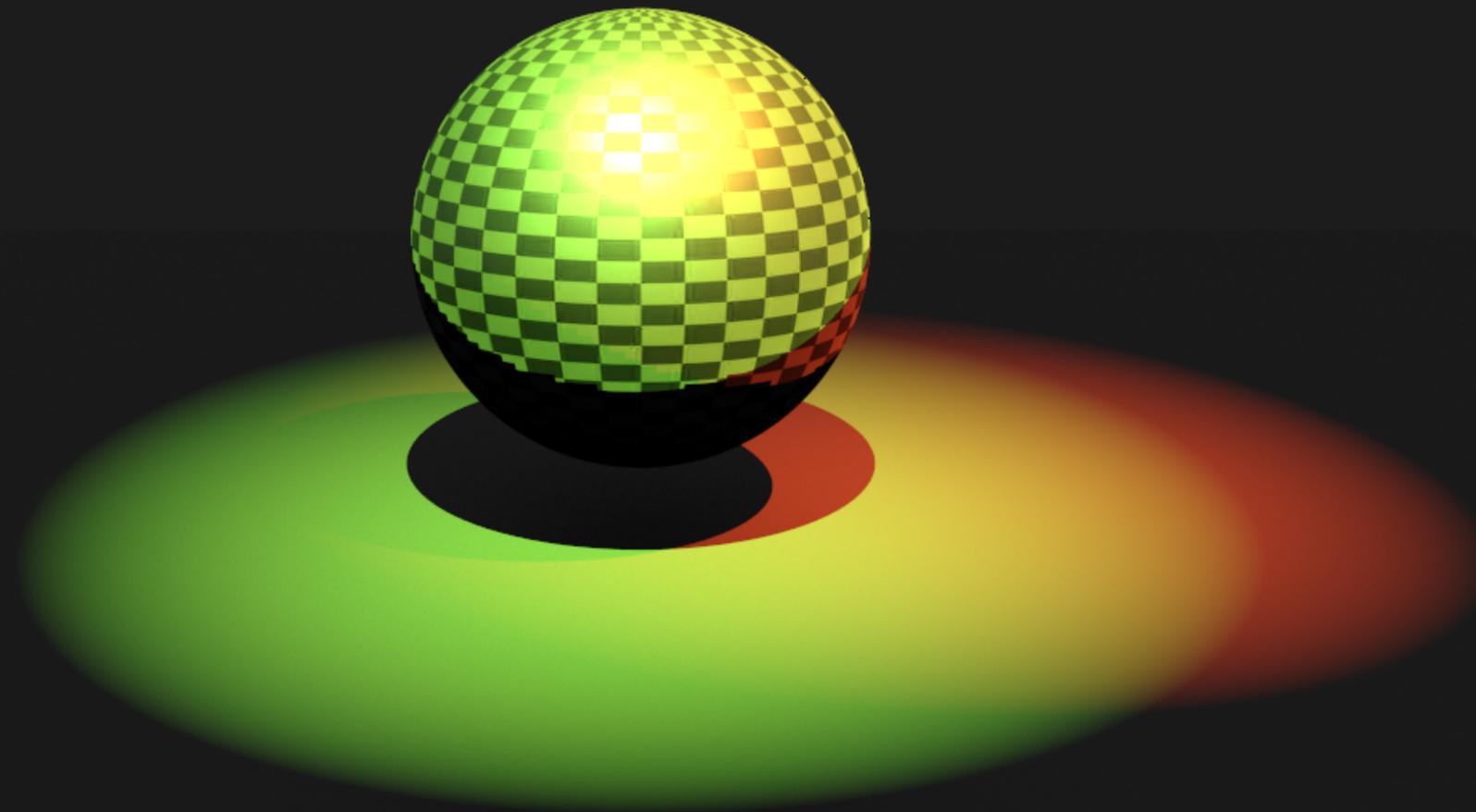


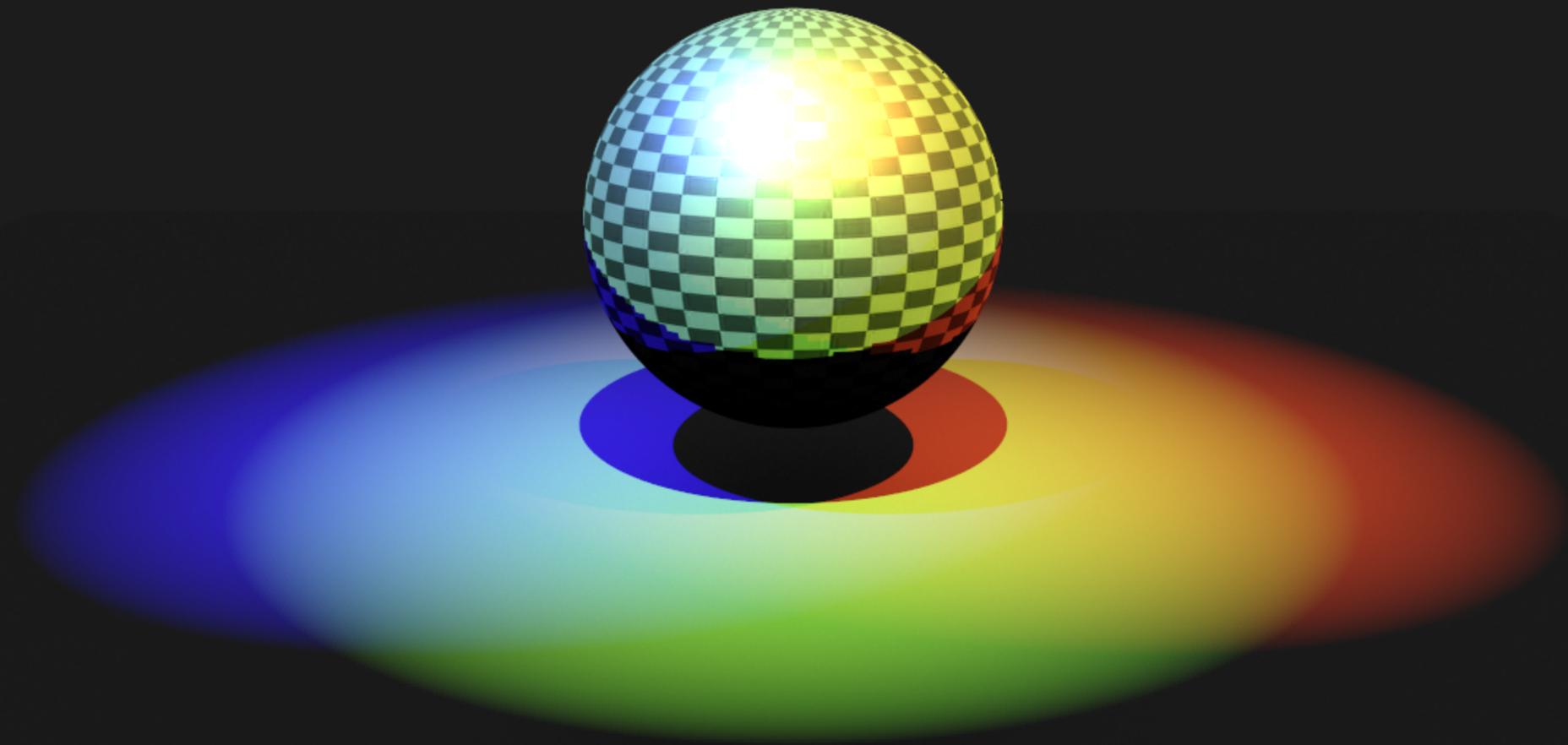


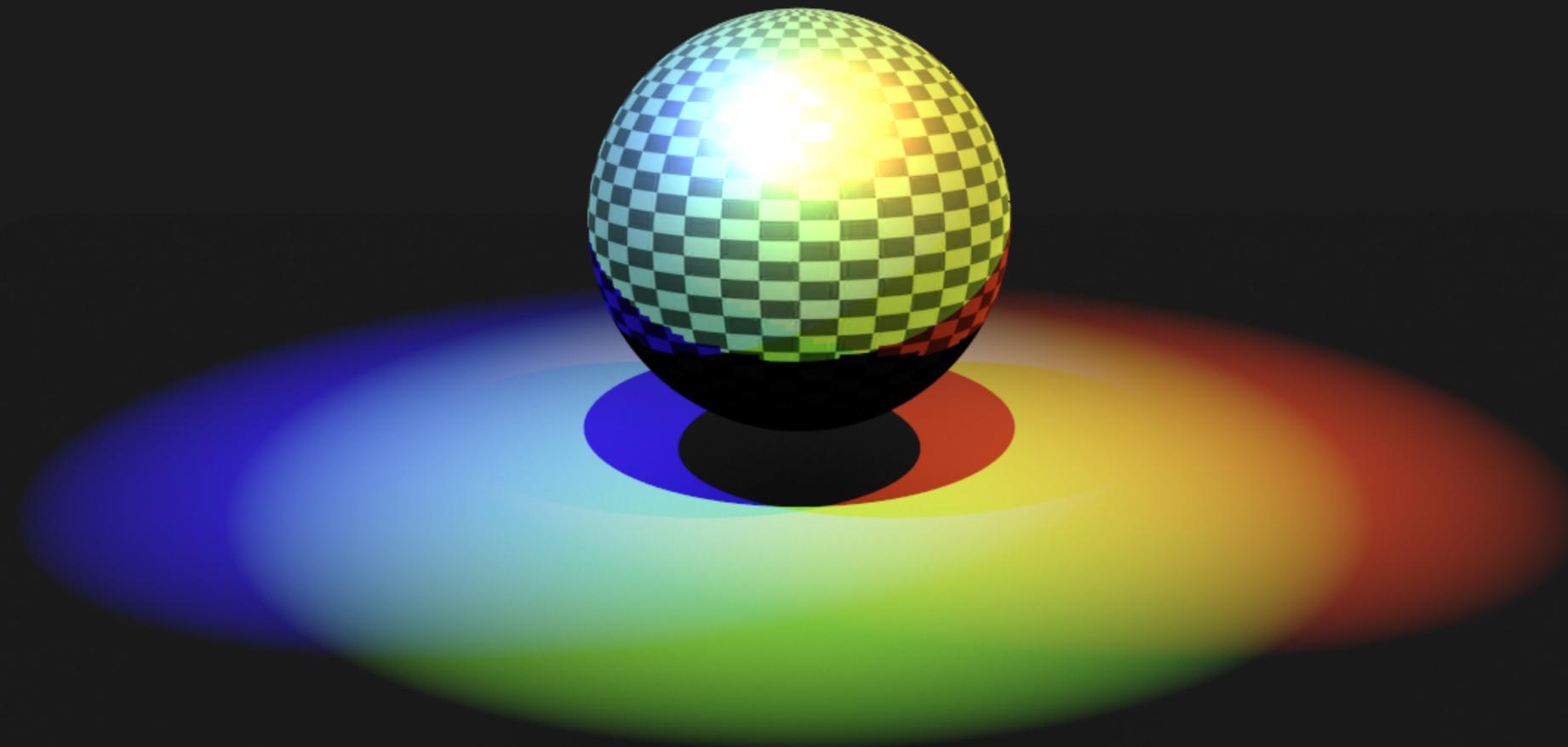




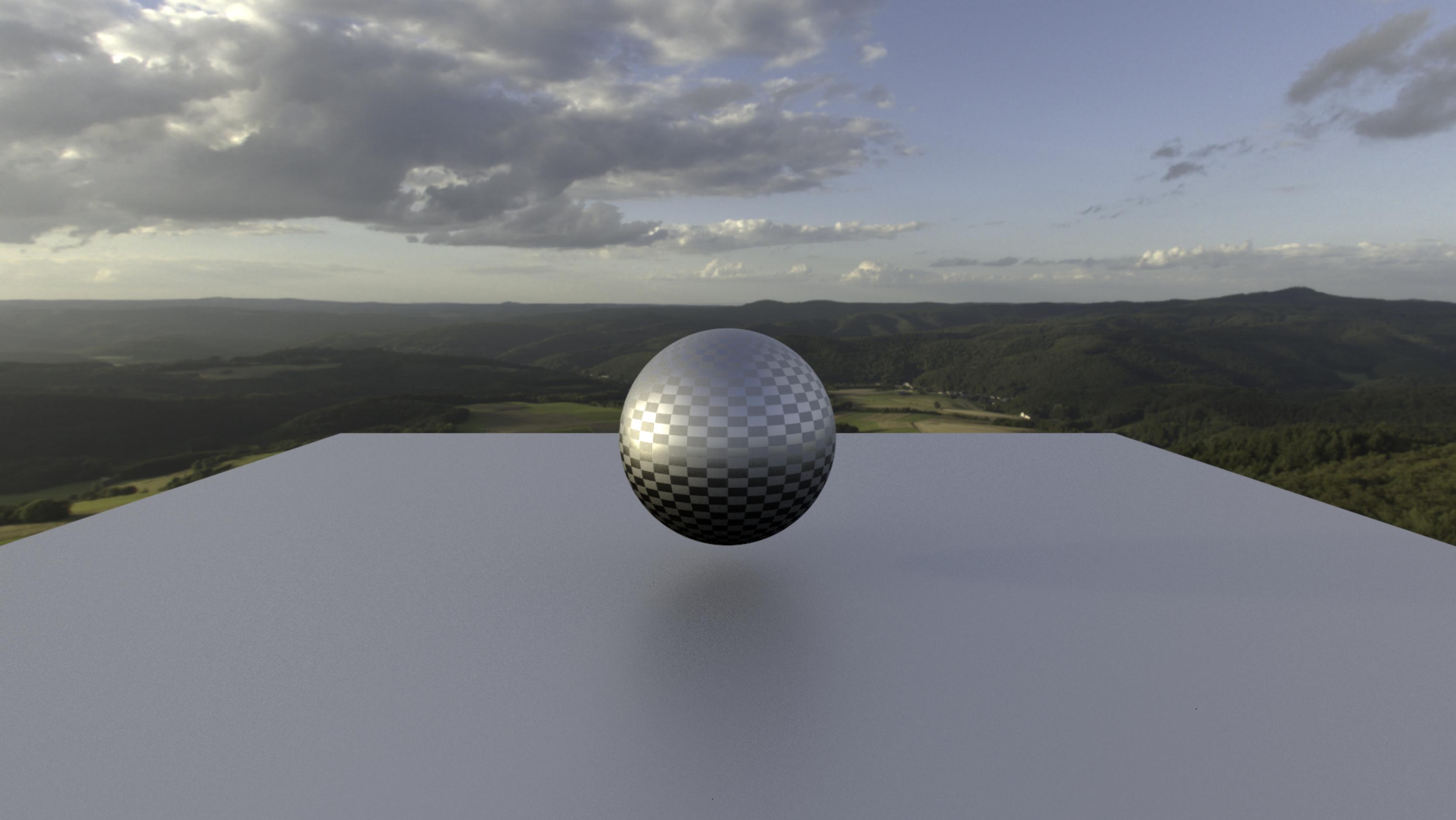


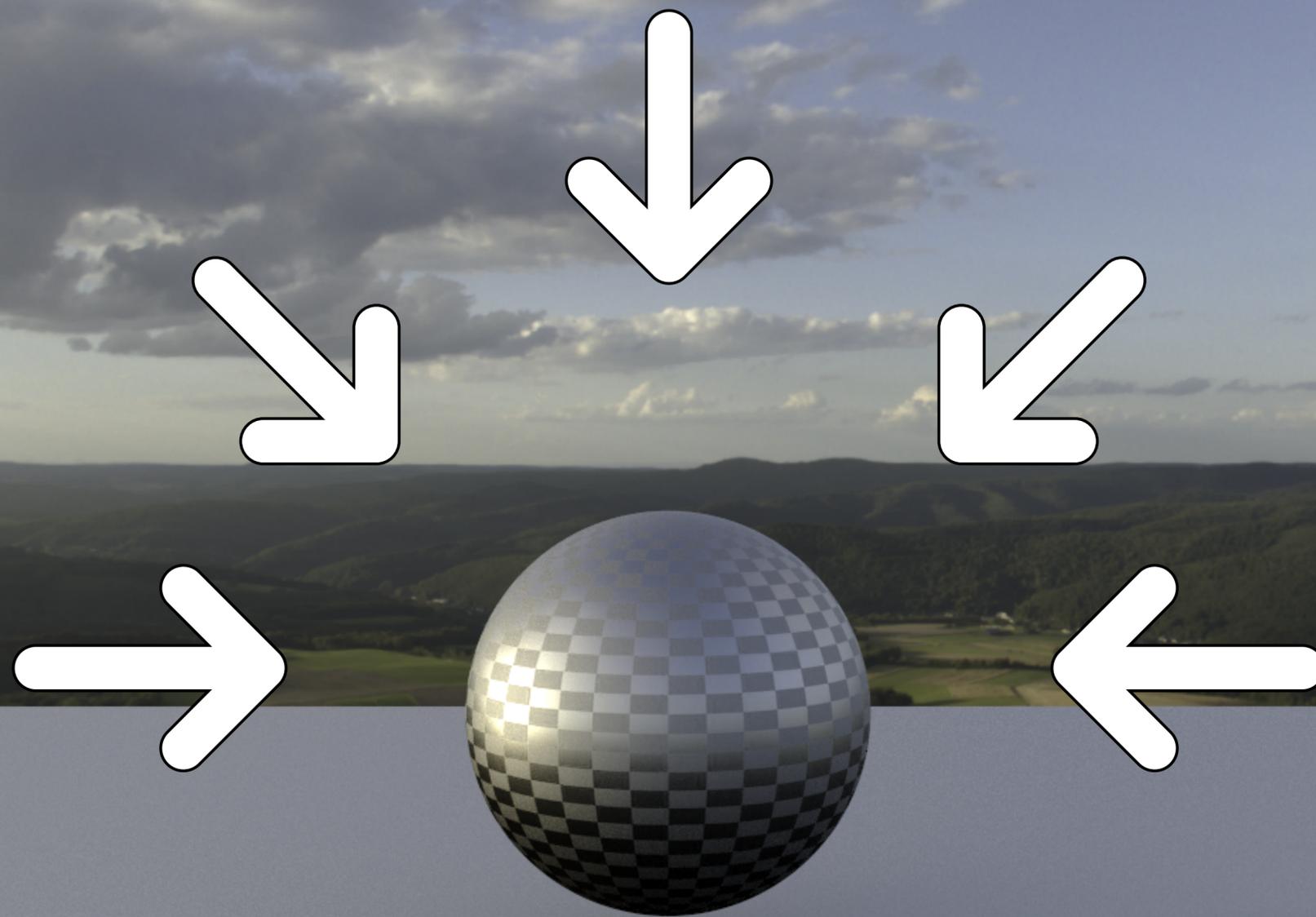




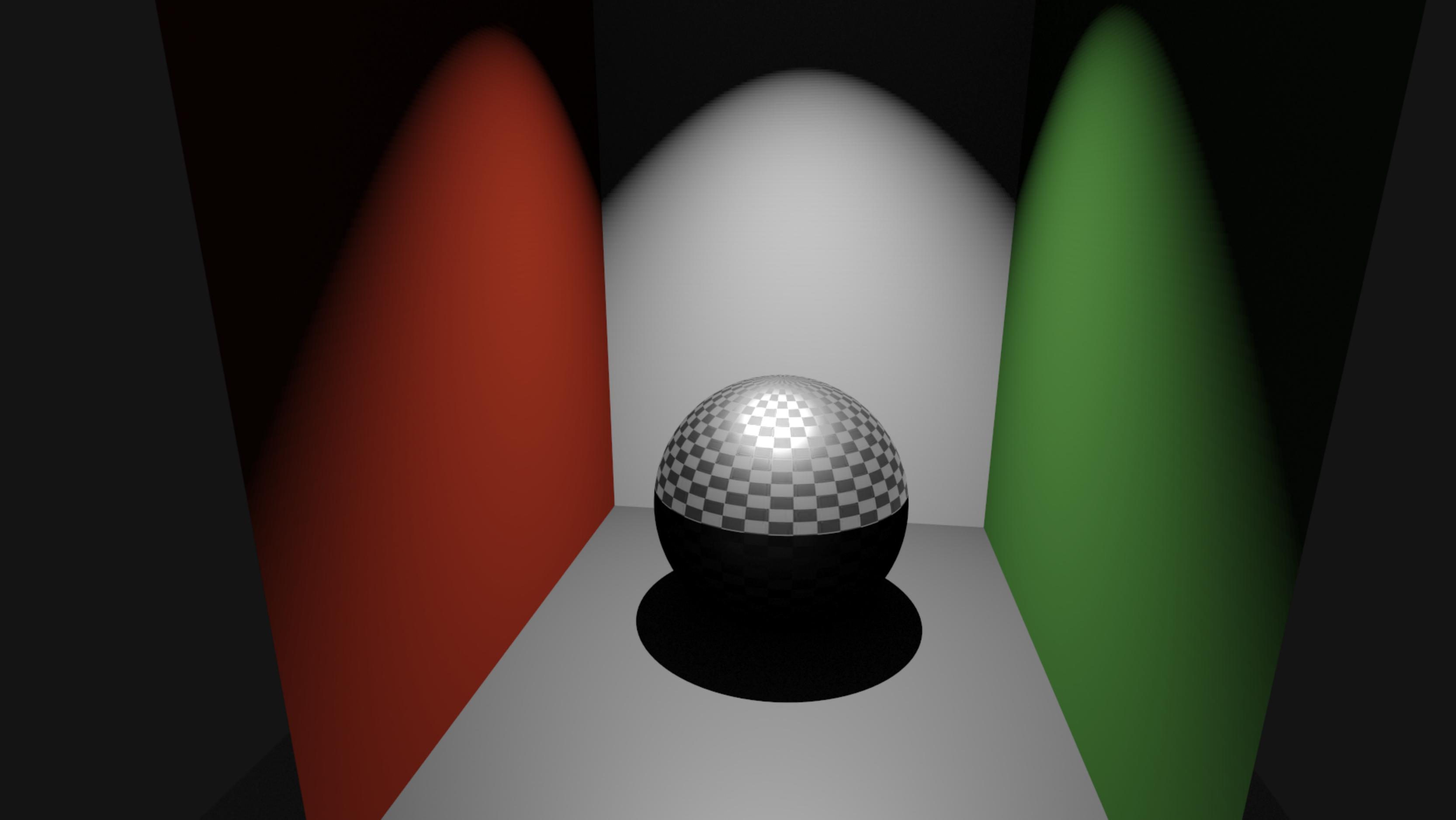


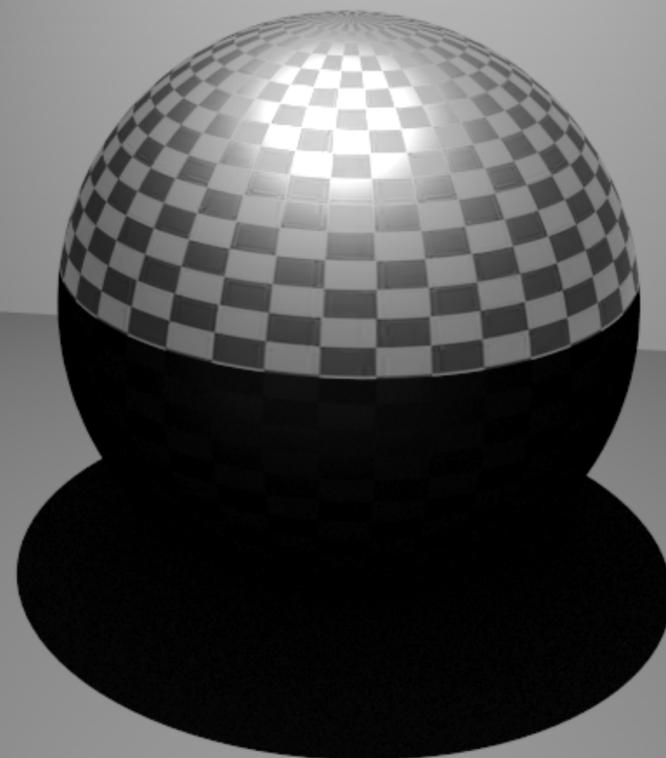
全ての方向の積分 = 三つのライトを足す



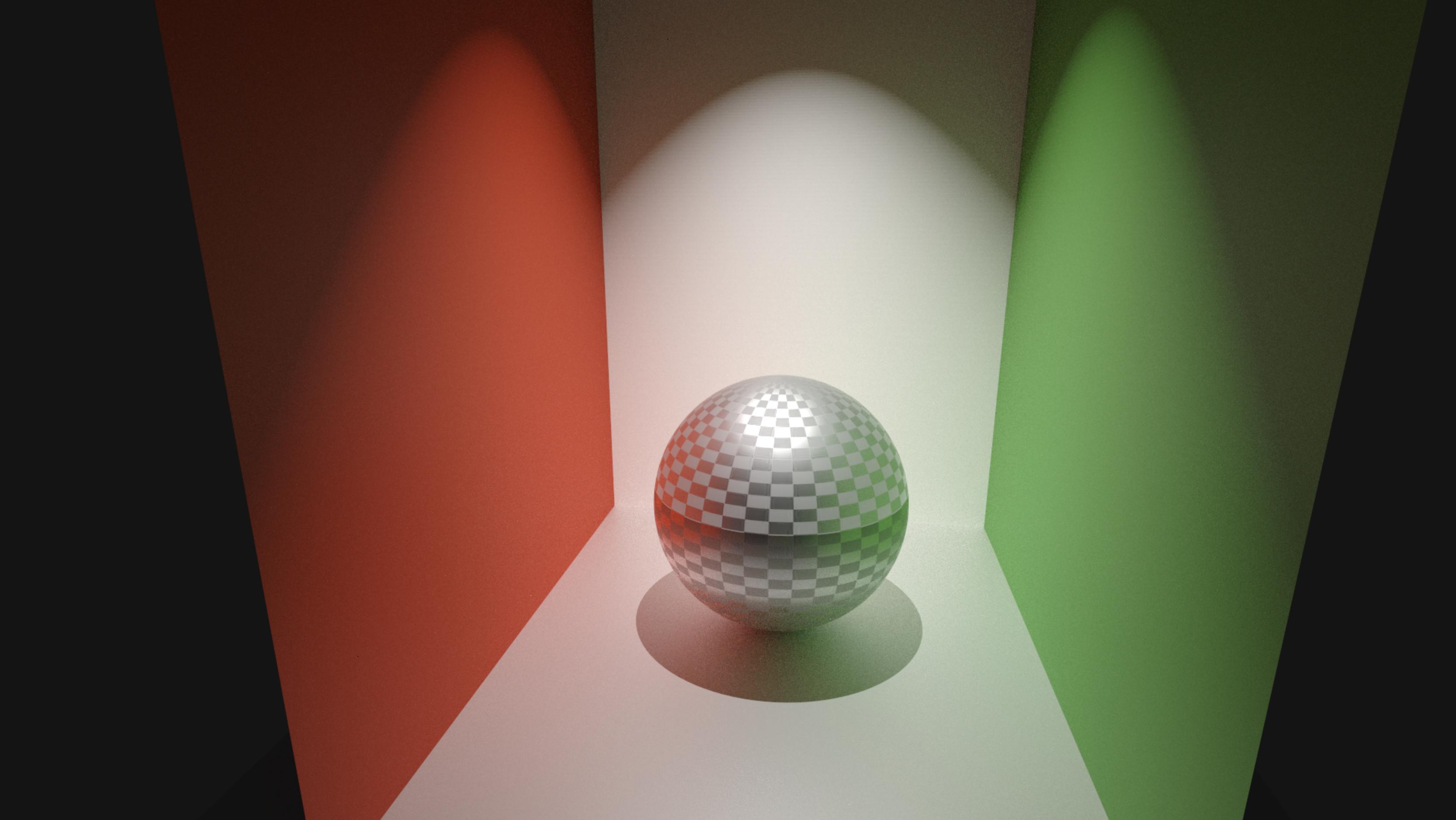


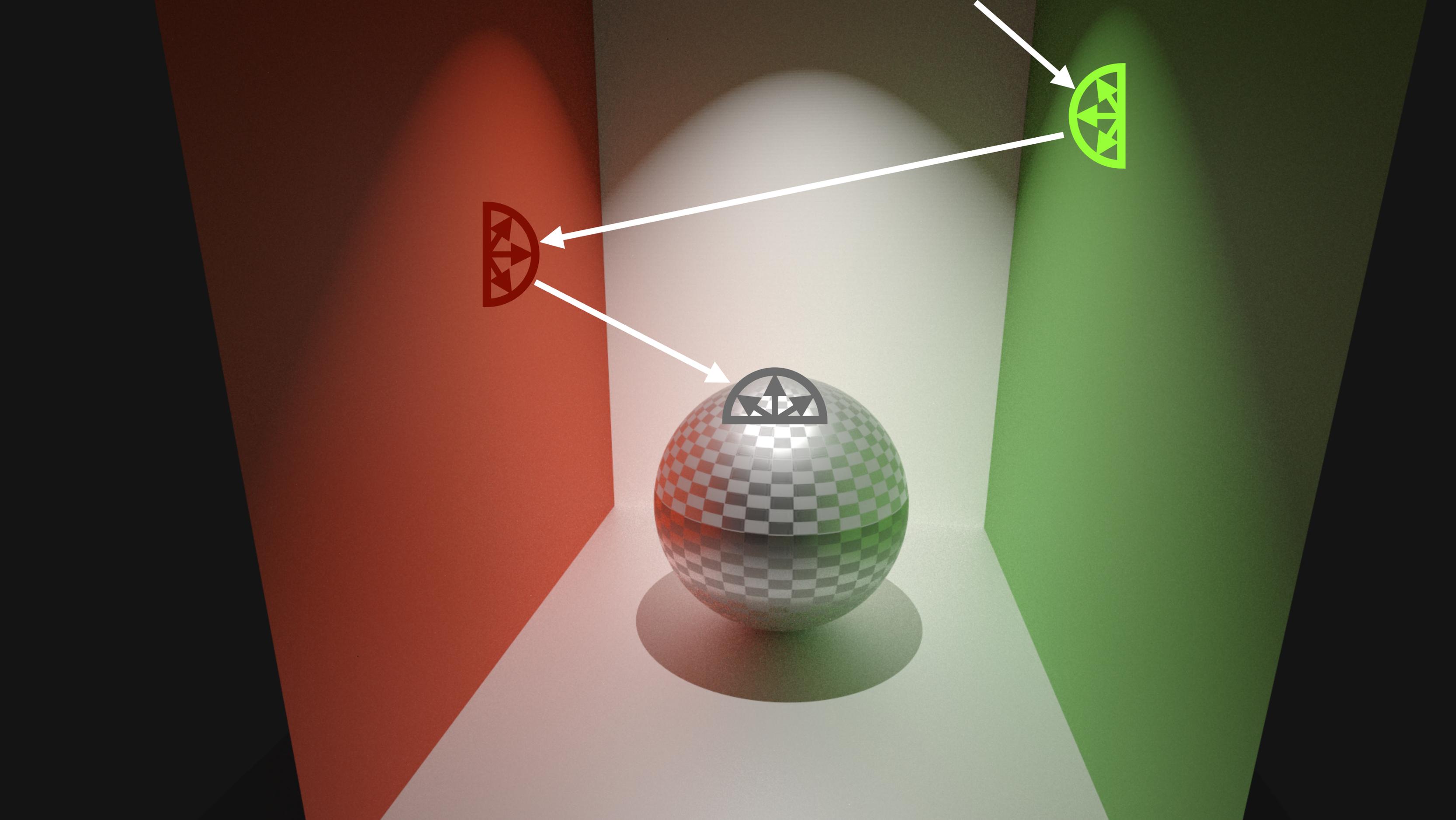
無数の方向からの光を足す = ①積分大変

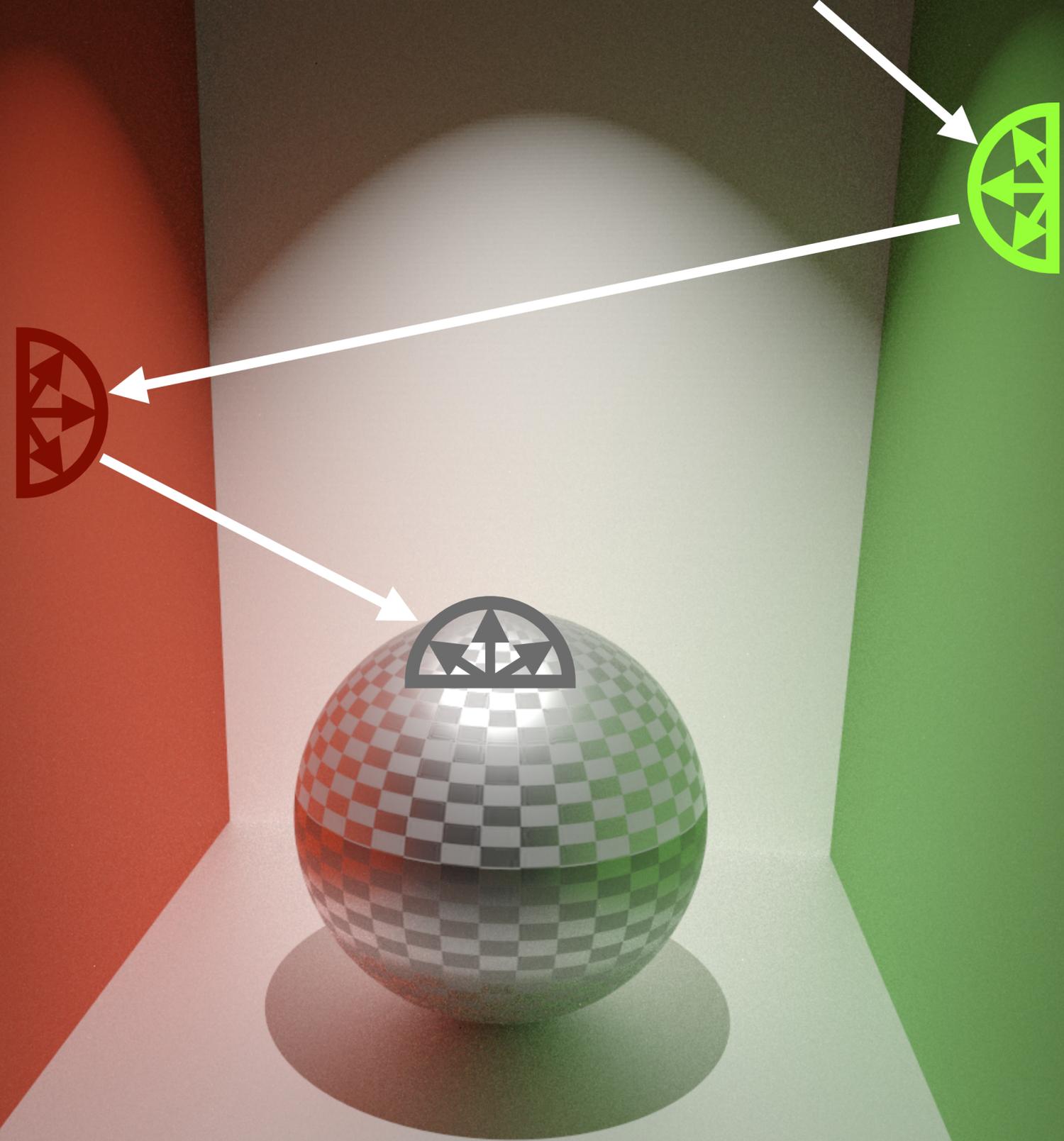




1回だけの反射



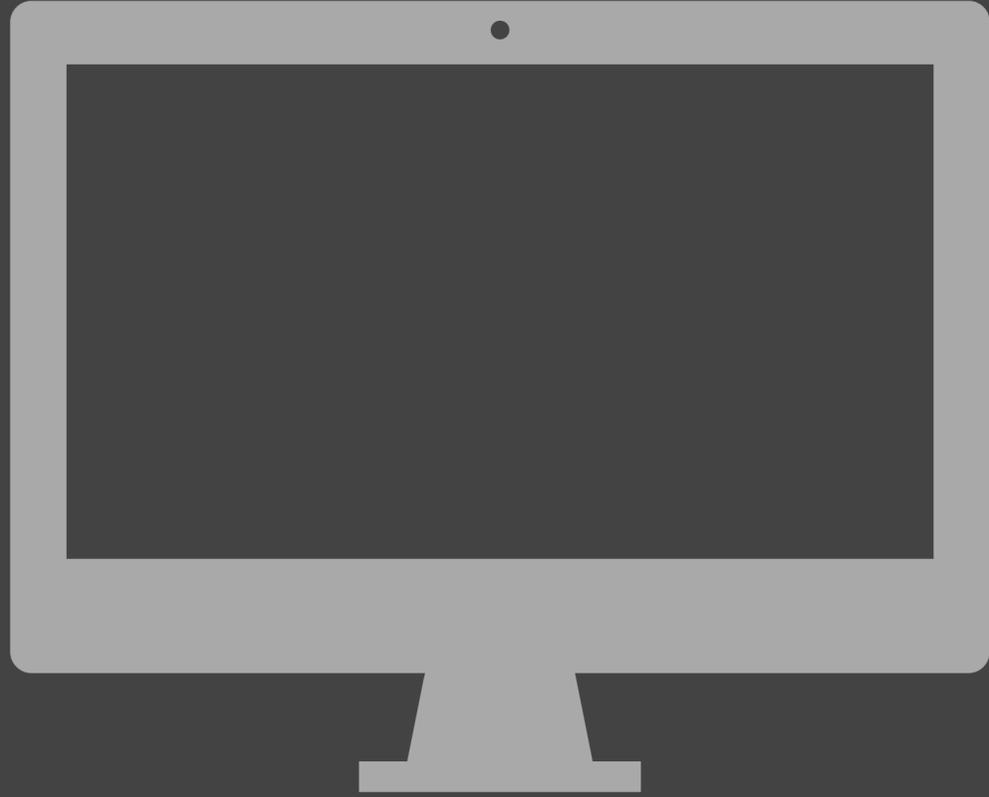




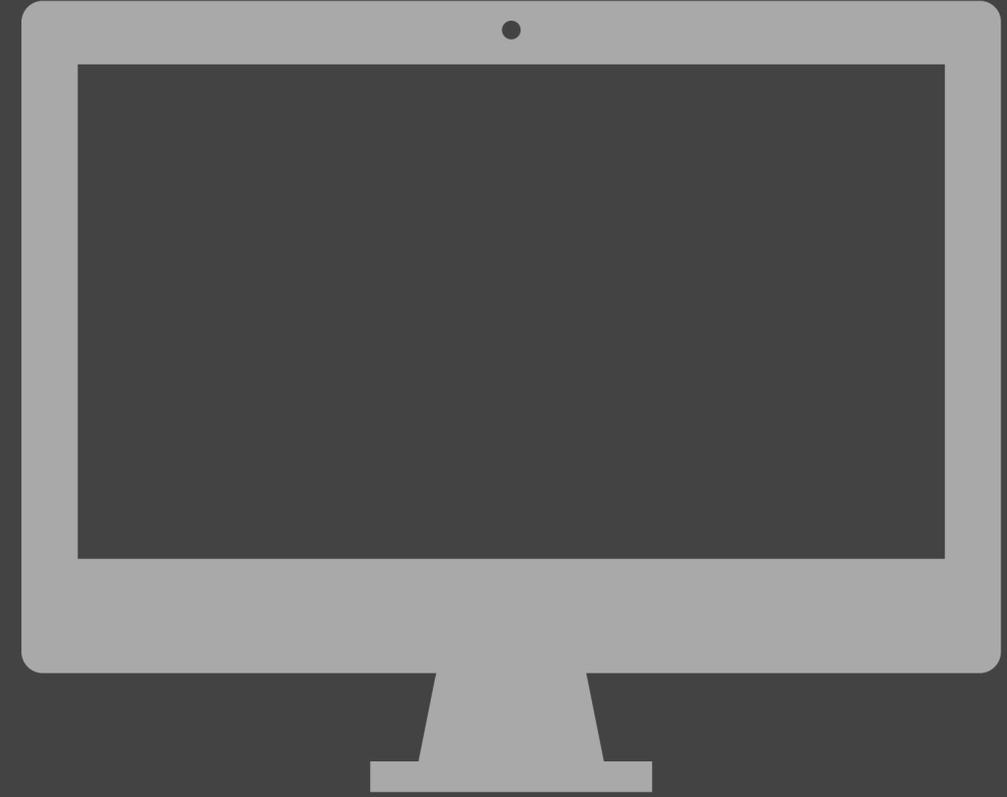
複数回の反射をする = ②再帰大変

ラスタライズの限界

- **フォトリアルな絵を出すためには、レンダリング方程式を解く必要があります。**
- **ゲームのようにラスタライズだけで完結しているシステムでは、レンダリング方程式のこの大変な部分を解けない場合があります。**
- **しかしレイトレーシングであれば解くことができます。**
レイトレーシングとは何でしょうか？



ラスターライズ



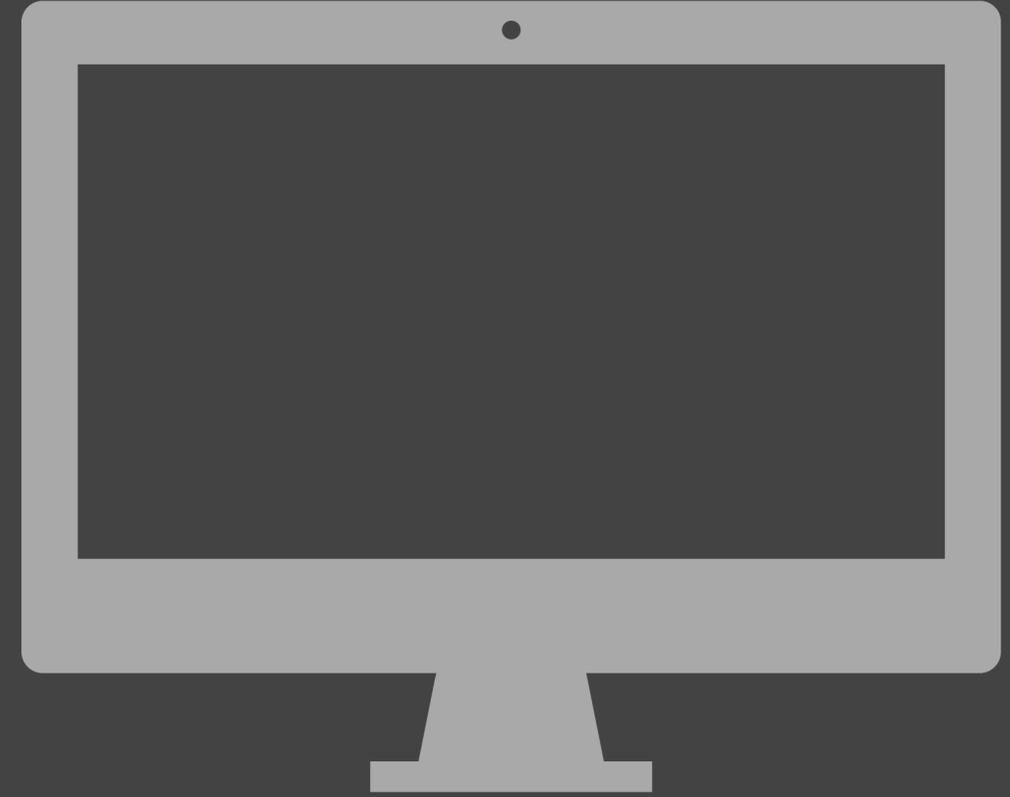
レイトレーシング



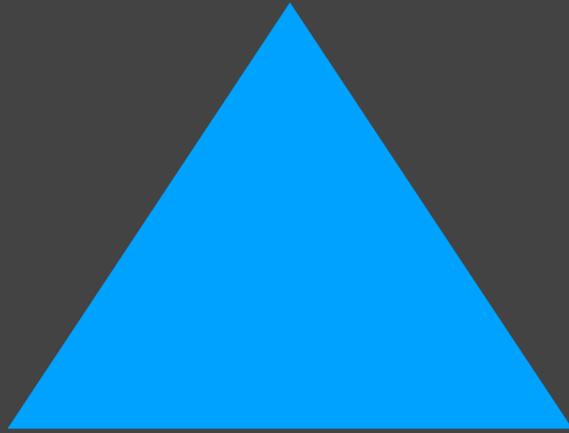
三角形を描画しよう



ラスタライズ



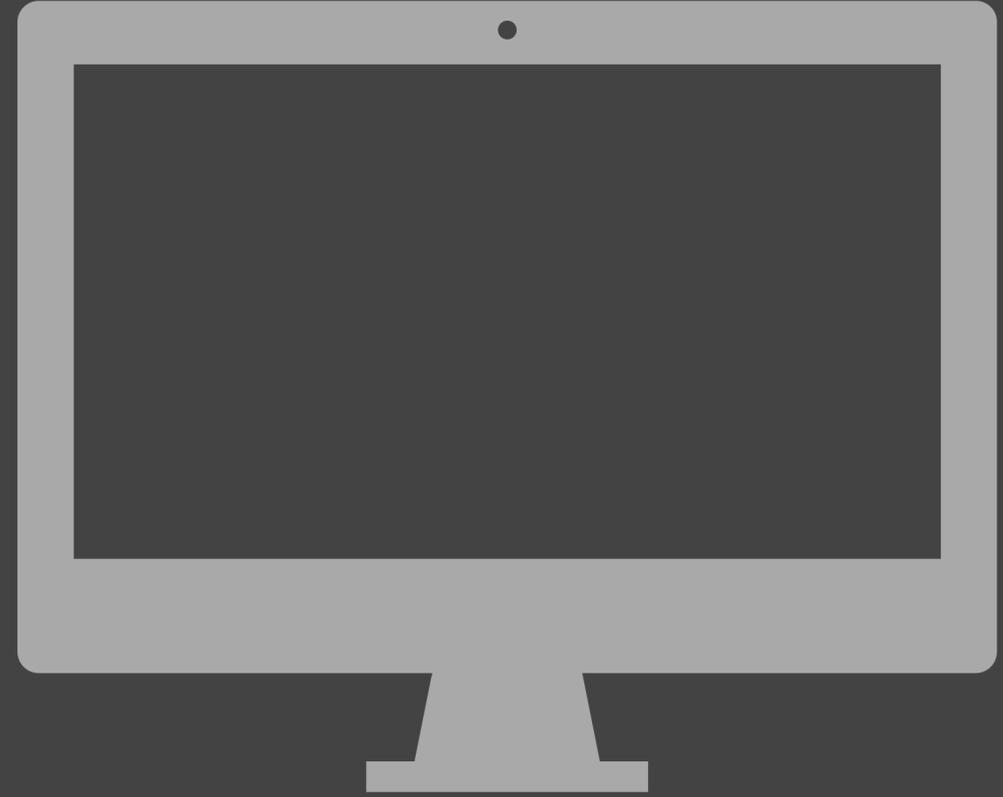
レイトレーシング



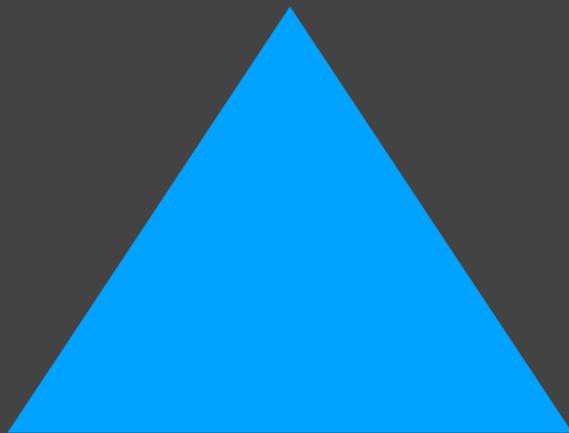
三角形を描画しよう



ラスターライズ



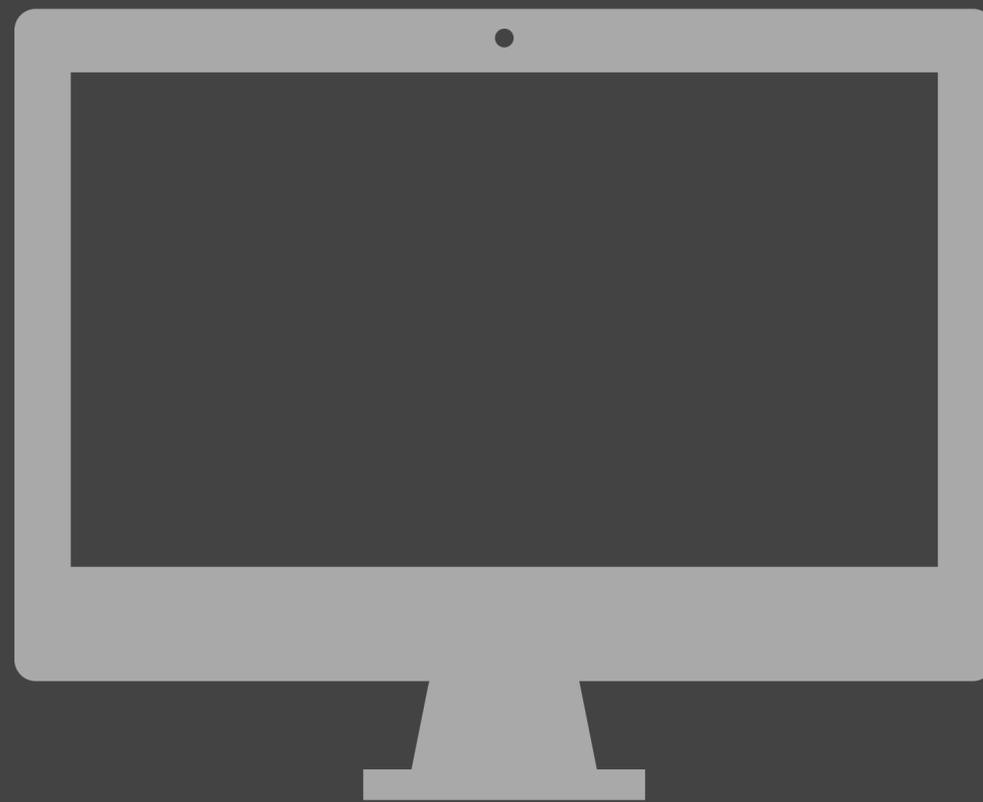
レイトレーシング



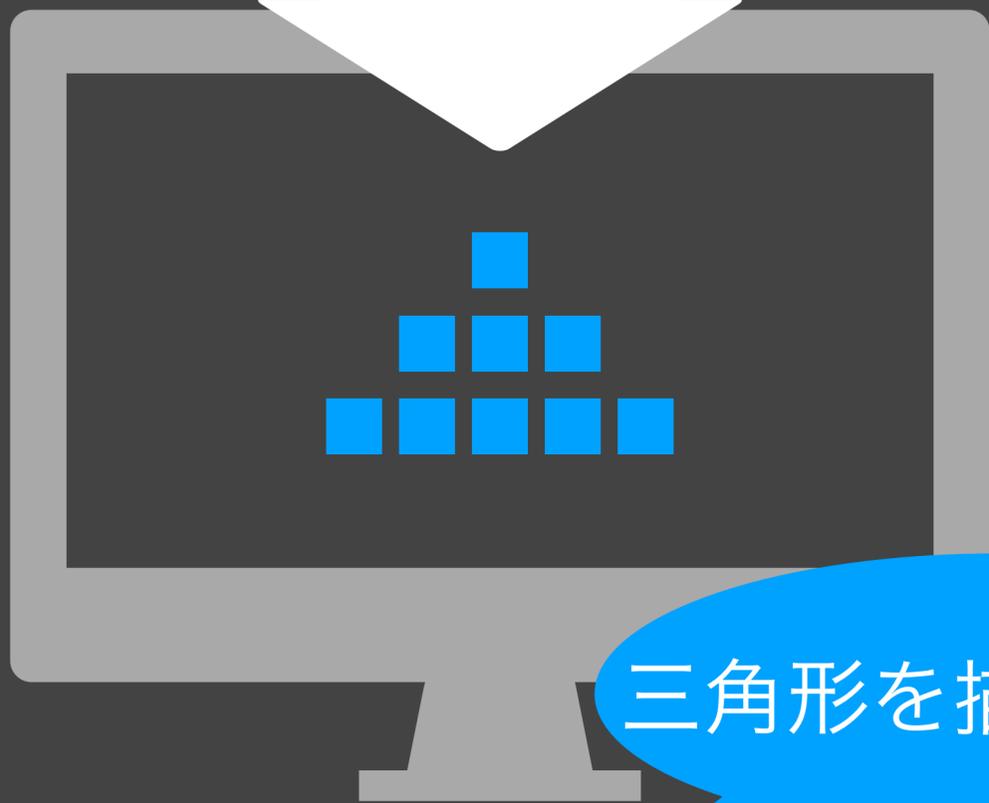
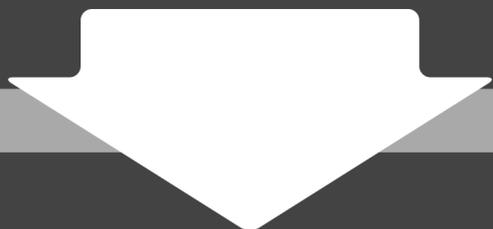
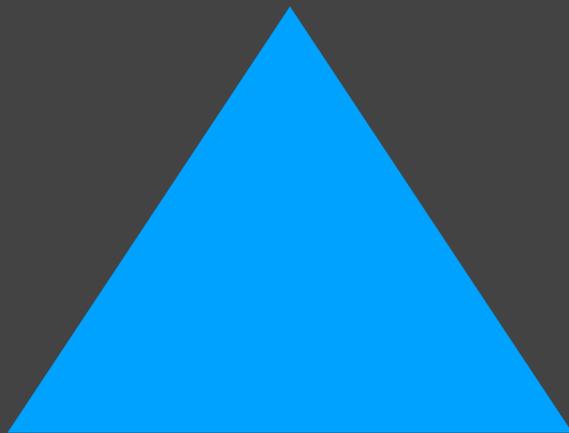
三角形を描画しよう



ラスターライズ



レイトレーシング



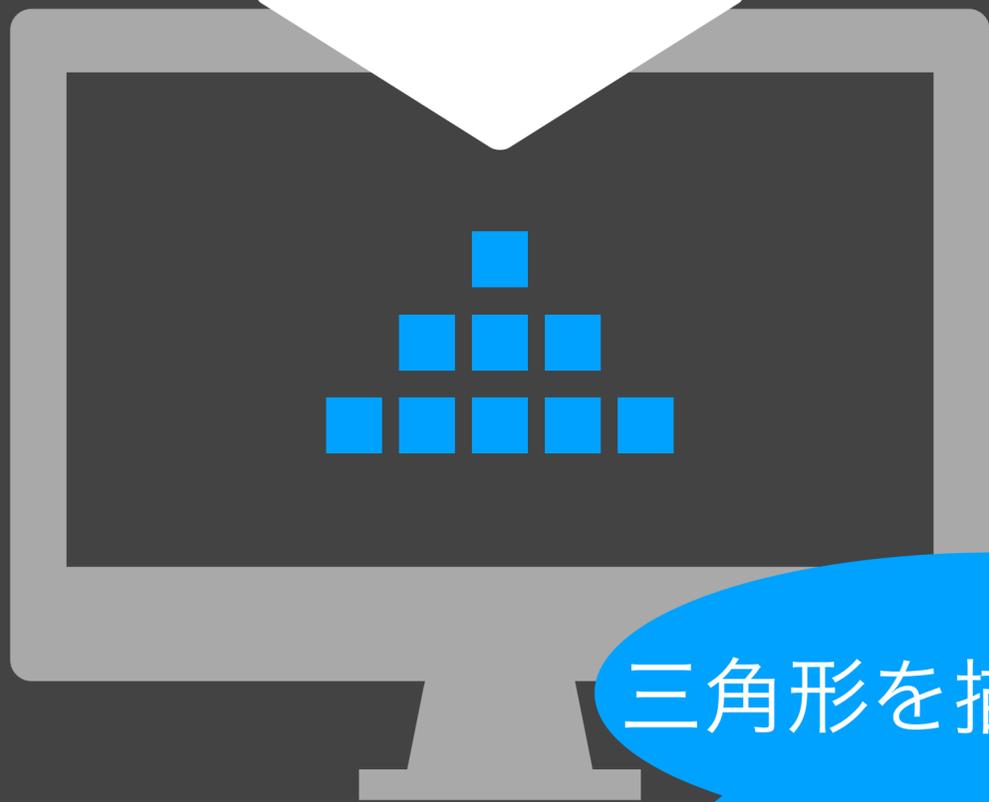
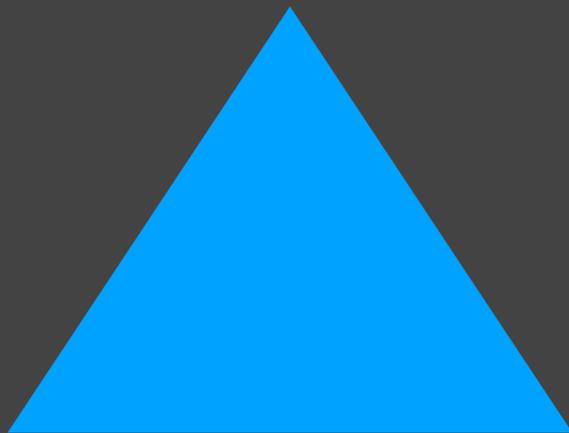
三角形を描画しよう



ラスターライズ



レイトレーシング



三角形を描画しよう



ラスターライズ



レイを飛ばそう



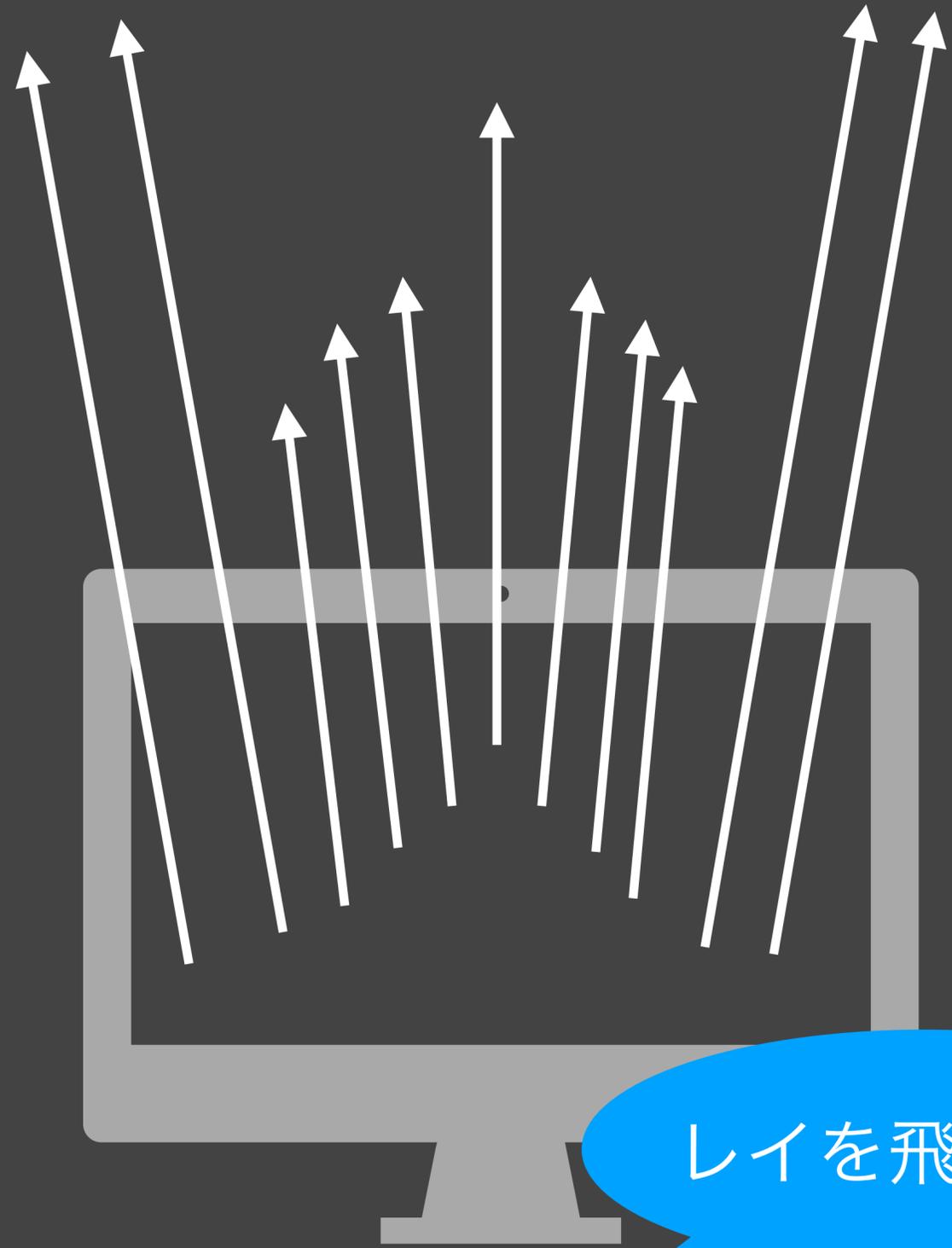
レイトレーシング



三角形を描画しよう



ラスターライズ



レイを飛ばそう

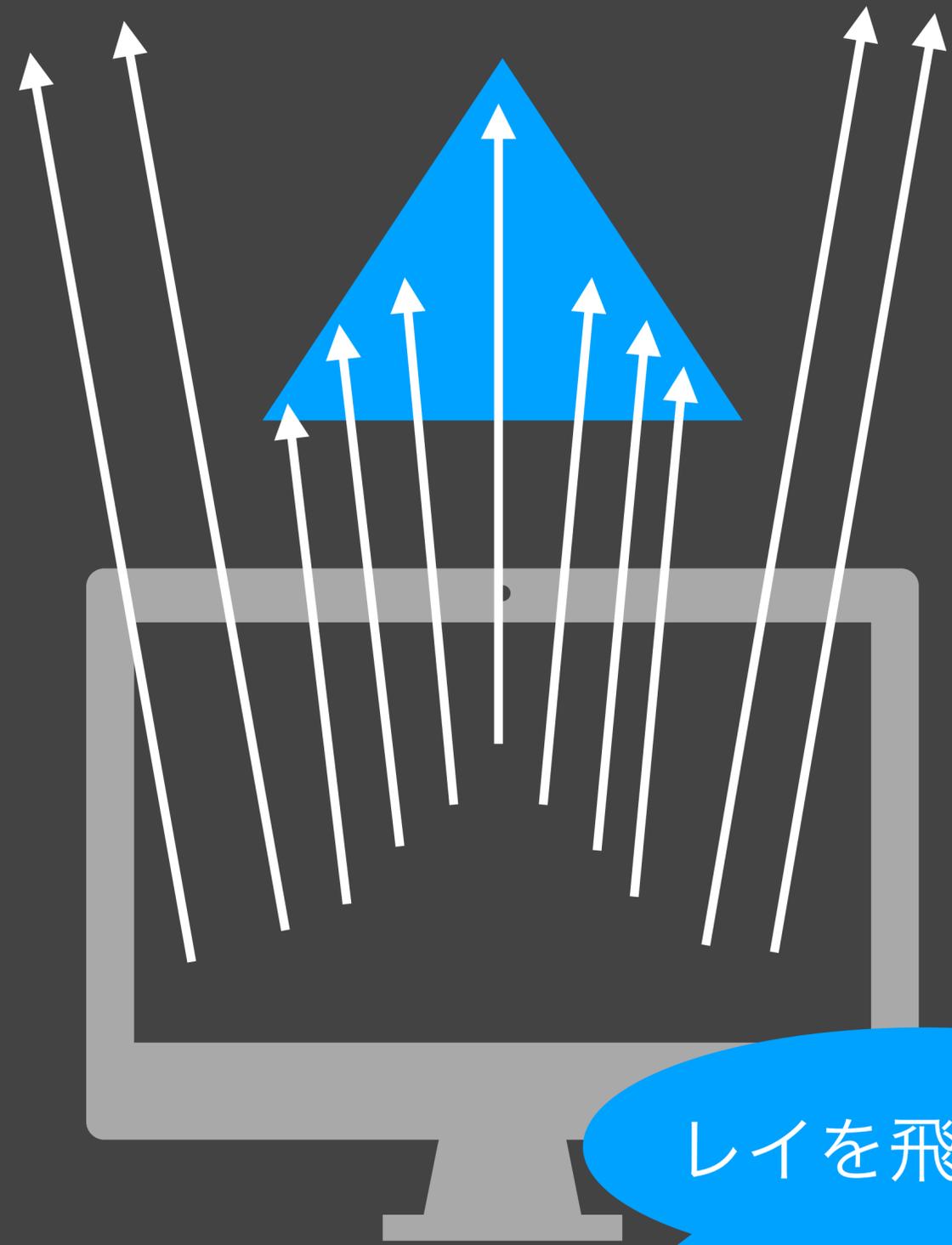


レイトレーシング



三角形を描画しよう

ラスターライズ



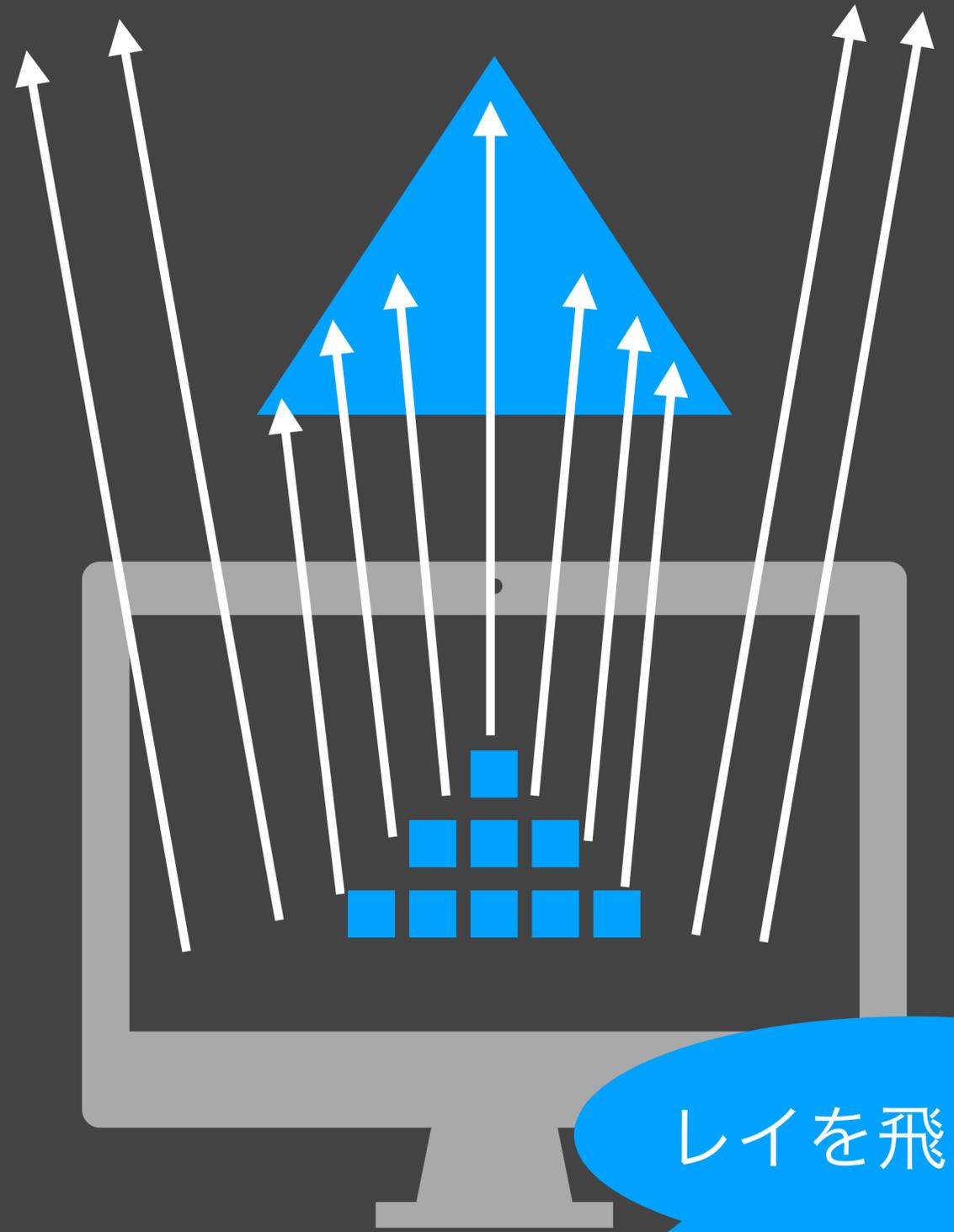
レイを飛ばそう

レイトレーシング



三角形を描画しよう

ラスターライズ



レイを飛ばそう

レイトレーシング

ラスタライズ vs レイトレーシング

ラスタライズ

- ・ **「プリミティブをピクセルにする」**が基本的な操作です。
- ・ ピクセルにするときにシェーディングで複雑な計算を行います。
- ・ レイトレと比較して**ハードウェアの支援**もあり**非常に高速**で動作します。
- ・ しかし**レンダリング方程式は直接は解けません**。
ハックを重ねて近似しています。

ラスタライズ vs レイトレーシング

レイトレーシング

- ・ **「シーンに対してレイを飛ばす(交差判定)」**が基本的な操作です。
- ・ **モンテカルロレイトレーシング**(レイトレーシングの一種)は、**レンダリング方程式を解くことができます。**
- ・ しかしMCレイトレーシングは**確率的にしか解が出ません。**
- ・ また専用ハードウェアもなく、ラスタライズに比べ**非常に低速**です。

特殊効果とその実現方法

| | ラスタライズ | レイトレ |
|------------|-----------|----------|
| シャドウ | シャドウマップなど | レイトレ |
| 反射効果(鏡等) | 環境マップ | レイトレ |
| 屈折効果(ガラス等) | 環境マップ | レイトレ |
| 被写界深度 | ポストエフェクト | たくさんレイトレ |
| モーションブラー | ポストエフェクト | たくさんレイトレ |
| プリミティブ | 基本的に三角形 | なんでも |
| 大域照明 | 色々なやり方 | たくさんレイトレ |

レイトレーシングは一般化されたもの

- 本来であれば一つの方程式から導かれる特殊効果が
ラスライズのハックの歴史で**近似、分断**されてきました。
- しかしレイトレーシングは、それらを**正しくかつ統一的に扱えます**。
- 「**PlayStation®2の時代**なら全ての要素の結果を**把握できた**。
しかし現代のラスライズは**ハック**が多く、
結果の**妥当性がわかりづらい**。
一般化された方法で解かれているレイトレは安心できる」(弊社TA談)

まとめ

| | ラスタライズ | レイトレ |
|-----------|----------|---------|
| 基本操作 | プリミティブ描画 | レイを飛ばす |
| 解の性質 | — | 確率的 |
| 速度 | 高速 | 低速 |
| レンダリング方程式 | 解けない | 解きやすい |
| 特殊効果 | ハックを重ねる | 統一的に解ける |

小話

「レンダリング方程式を解けば物理的に正しい」？

- 現実世界を**近似**した物理学のモデルが最初にあります。
そこからさらに**取捨選択**されたものがレンダリング方程式です。
欲しい絵にとって必要十分なので大抵これが採用されています。
- 構造色、二重スリット、赤方偏移、ブラックホール…などを表現したくなったら別の手段が必要です。

例) 映画「インターステラー」のブラックホールのレンダリング

<https://www.wired.com/2014/10/astrophysics-interstellar-black-hole/>

小話

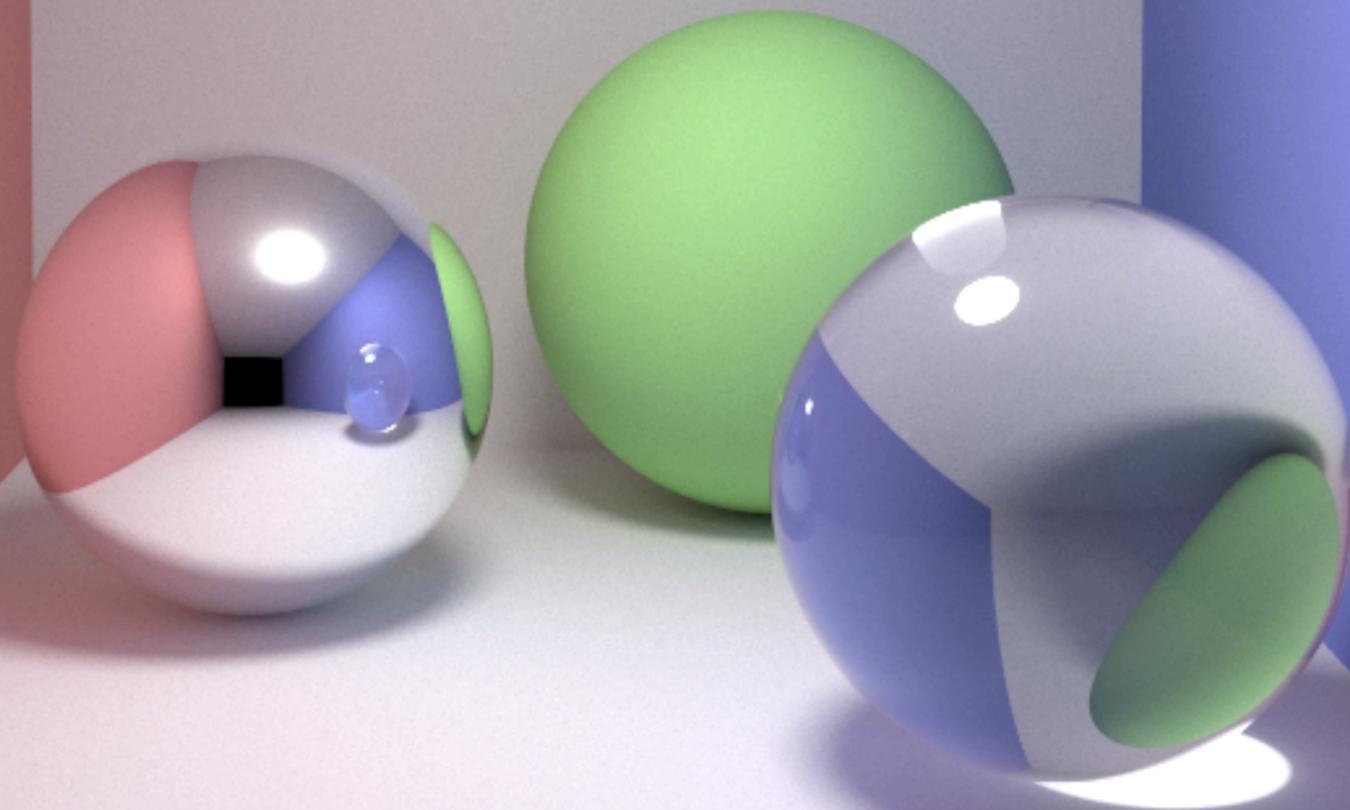
「レンダリング方程式を解けば物理的に正しい」？

- 「物理的」 ≠ 完璧な正解。
「たまたま結果の見栄えがいい方程式」位の接し方がいいかもしれません。
- “All models are **wrong**, but some are **useful**.”
George E. P. Box

学習リソース

最初はどう始めればいいのか

edupt



<http://kagamin.net/hole/edupt/>

過去のCEDEC

- “GPGPUによる高速なグローバルイルミネーションベイクツールの作り方”
https://cedil.cesa.or.jp/cedil_sessions/view/753
- “モンテカルロレイトレーシングの基礎からOpenCLによる実装まで”
https://cedil.cesa.or.jp/cedil_sessions/view/1051
- “モンテカルロレイトレーシングの基礎からOpenCLによる実装まで（実装編）”
https://cedil.cesa.or.jp/cedil_sessions/view/1134

RAY TRACING

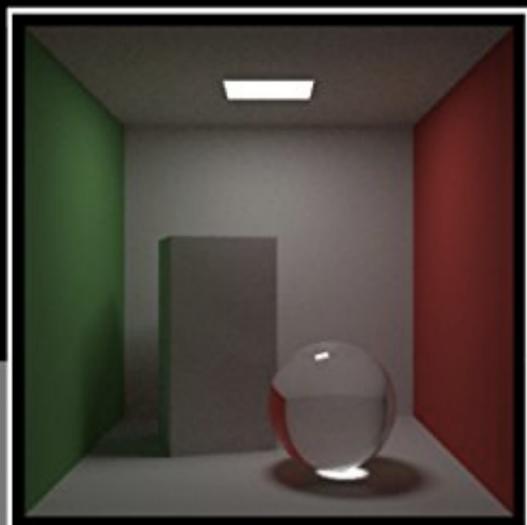
IN

RAY TRACING

THE NEXT WEEK

RAY TRACING

THE REST OF YOUR LIFE



PETER SHIRLEY

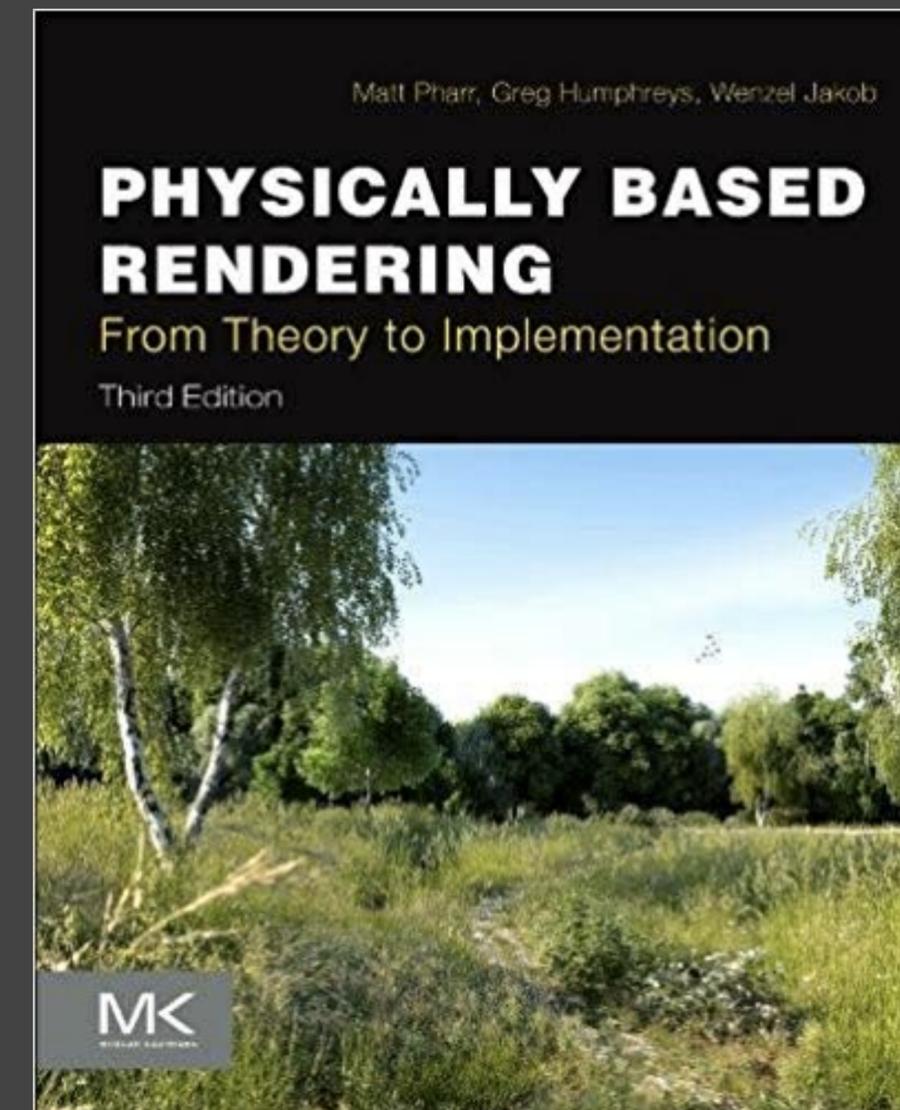
One weekend

一冊当たり**60ページ**



CG Gems JP 2015

第1章 **27ページ**



PBR book(**辞書として**)

1226ページ

この章のまとめ

- レイトレーシングの概要、長所、短所についてお話ししました。
- 入門のための学習リソースについて軽く触れました。

目次

- はじめに
- レイトレーシングとは
- **弊社のレイトレーシングの歴史**
- 弊社での使われ方
- 事故、デバッグ、対策
- 結び

歴史

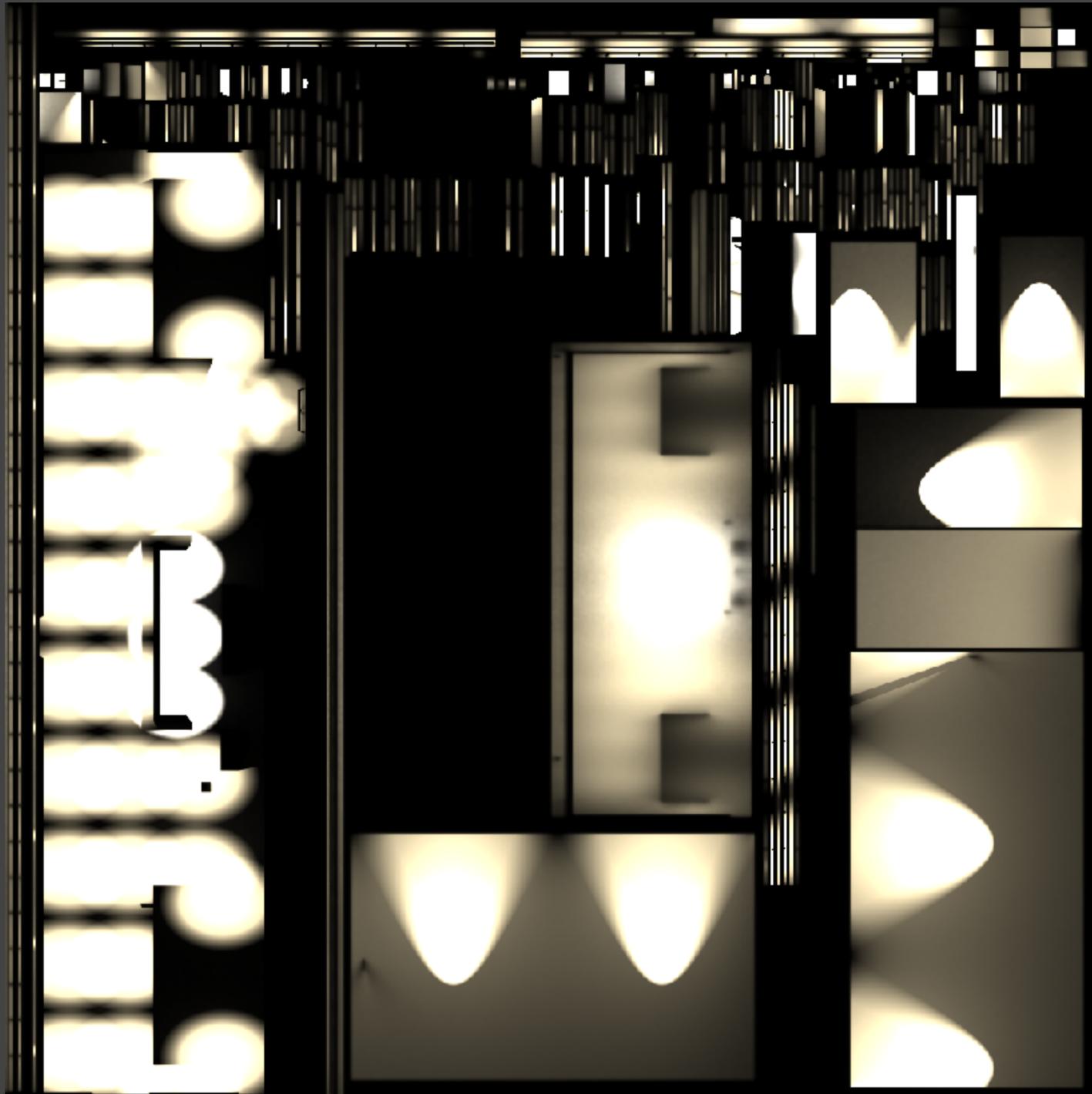
弊社のレイトレーシングの歴史

さかのぼること11年前…

2007年～

- Mayaの**MentalRay**で**ベイク**をしていました。
- しかし**汎用さゆえの使いづらさ**がありました。
特に**カスタムシェーダー**の取り扱いなどに
難があり、**内製のレンダラー**を作ることになりました。
- **Kasuri(飛白)**レンダラーが開発されました。
- **シャドウ、オクルージョン、ライト**のベイクなどが行われていました。





ライトベイク



シャドウベイク

そもそもなぜベイクが必要なのか？

- 2018年中頃現在、リアルタイムレンダリングでは**限定的**にしかレンダリング方程式は解かれていません。
 - レンダリング方程式の**一部分のみ解決**にとどまっています。
 - もしくは**高価で高性能なハードウェア**が必要です。
- 「その**解かれていない部分を事前に計算**すればいいのではないか？」これがベイクをする動機です。

解かれていない部分を事前に計算する

- ・ **市販のレイトレーサー**には**ベイク機能**がついています。
そのベイク結果をリアルタイムレンダリング中に参照することで**事前に解いておいた結果**を利用していることとなります。
- ・ よくあるベイク手法は一体何を事前に計算していたのでしょうか？

下準備

話を簡単にするためにレンダリング方程式に次のような修正を加えます。

- L_e (発光)を取り除きます。
- 一次光に限定します。
- ある方向にあるライトが可視かを V とします。
- 対象のBRDFを完全拡散反射だとする。 f_r が $\rho(x)/\pi$ になります。

$$L_o(x) = \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

下準備

話を簡単にするためにレンダリング方程式に次のような修正を加えます。

- L_e (発光)を取り除きます。
- 一次光に限定します。
- ある方向にあるライトが可視かを V とします。
- 対象のBRDFを完全拡散反射だとする。 f_r が $\rho(x)/\pi$ になります。

$$L_o(x) = \rho(x) \frac{1}{\pi} \int_{\Omega} \underline{V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n)} d\omega_i$$

その方向にあるライトが見えているか？

ライトマップとは何だったのか

ライトマップとは何だったのか

簡略化した式の左辺には x (表面上の位置)しかないので
右辺の要素が変化しなければ先に**積分**ができます。

ライトマップとは何だったのか

簡略化した式の左辺には x (表面上の位置)しかないので
右辺の要素が変化しなければ先に**積分**ができます。

$$L_o(x) = \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

ライトマップとは何だったのか

簡略化した式の左辺には x (表面上の位置)しかないので
右辺の要素が変化しなければ先に**積分**ができます。

$$L_o(x) = \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

事前に計算する

ライトマップとは何だったのか

簡略化した式の左辺には x (表面上の位置)しかないので
右辺の要素が変化しなければ先に**積分**ができます。

$$L_o(x) = \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

事前に計算する

リアルタイムレンダリング

ライトマップとは何だったのか

簡略化した式の左辺には x (表面上の位置)しかないので
右辺の要素が変化しなければ先に**積分**ができます。

$$L_o(x) = \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

事前に計算する

リアルタイムレンダリング

ライトマップ = 「完全拡散面の仮定した積分結果」

AOとは何だったのか

AOとは何だったのか

ライトマップの式の **L_i** (入射光)が方向に寄らない定数 **L_c** だったとします

AOとは何だったのか

ライトマップの式の **L_i (入射光)**が方向に寄らない定数 **L_c** だったとします

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= L_c \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) (\omega_i \cdot n) d\omega_i \end{aligned}$$

AOとは何だったのか

ライトマップの式の **L_i (入射光)**が方向に寄らない定数 **L_c** だったとします

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= L_c \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) (\omega_i \cdot n) d\omega_i \end{aligned}$$

ここがAO

AOとは何だったのか

ライトマップの式の **L_i (入射光)**が方向に寄らない定数 **L_c** だったとします

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= L_c \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) (\omega_i \cdot n) d\omega_i \end{aligned}$$



ここがAO

リアルタイムレンダリング

AOとは何だったのか

ライトマップの式の **L_i (入射光)**が方向に寄らない定数 **L_c** だったとします

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= L_c \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) (\omega_i \cdot n) d\omega_i \end{aligned}$$

↑
リアルタイムレンダリング

ここがAO

AO = 「完全拡散反射と一様な輝度の光源を仮定」

シャドウベイクとは何だったのか

シャドウベイクとは何だったのか

簡略化した式からVを外に出そうとすると…

シャドウベイクとは何だったのか

簡略化した式からVを外に出そうとすると…

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= \rho(x) \frac{1}{\pi} \int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \cdot \frac{\int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i}{\int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i} \end{aligned}$$

シャドウベイクとは何だったのか

簡略化した式からVを外に出そうとすると…

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= \rho(x) \frac{1}{\pi} \int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \cdot \frac{\int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i}{\int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i} \end{aligned}$$

ここがシャドウベイク

シャドウベイクとは何だったのか

簡略化した式からVを外に出そうとすると…

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= \rho(x) \frac{1}{\pi} \int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \cdot \frac{\int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i}{\int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i} \end{aligned}$$

リアルタイムレンダリング

ここがシャドウベイク

シャドウベイクとは何だったのか

簡略化した式からVを外に出そうとすると…

$$\begin{aligned} L_o(x) &= \rho(x) \frac{1}{\pi} \int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \\ &= \rho(x) \frac{1}{\pi} \int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i \cdot \frac{\int_{\Omega} V(x, \omega_i) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i}{\int_{\Omega} L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i} \end{aligned}$$

リアルタイムレンダリング

ここがシャドウベイク

シャドウベイク = 「ライトの影響度合い」

レンダリング方程式の近似

- ライトマップ、AO、シャドウマップは
レンダリング方程式の近似を与えていました。
- これだけでは、やりたい表現ができない時はどうしたらいいのでしょうか？

システムの刷新

- ・ **ラスターライズでフォトリアルな絵を出すために、レンダリング方程式をより高い精度で近似する必要性が増加しました。そのためデータを供給する新しいレンダラーが必要です。**

もっと詳細な情報が欲しい

レンダリング方程式

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

もっと詳細な情報が欲しい

レンダリング方程式

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

この部分の

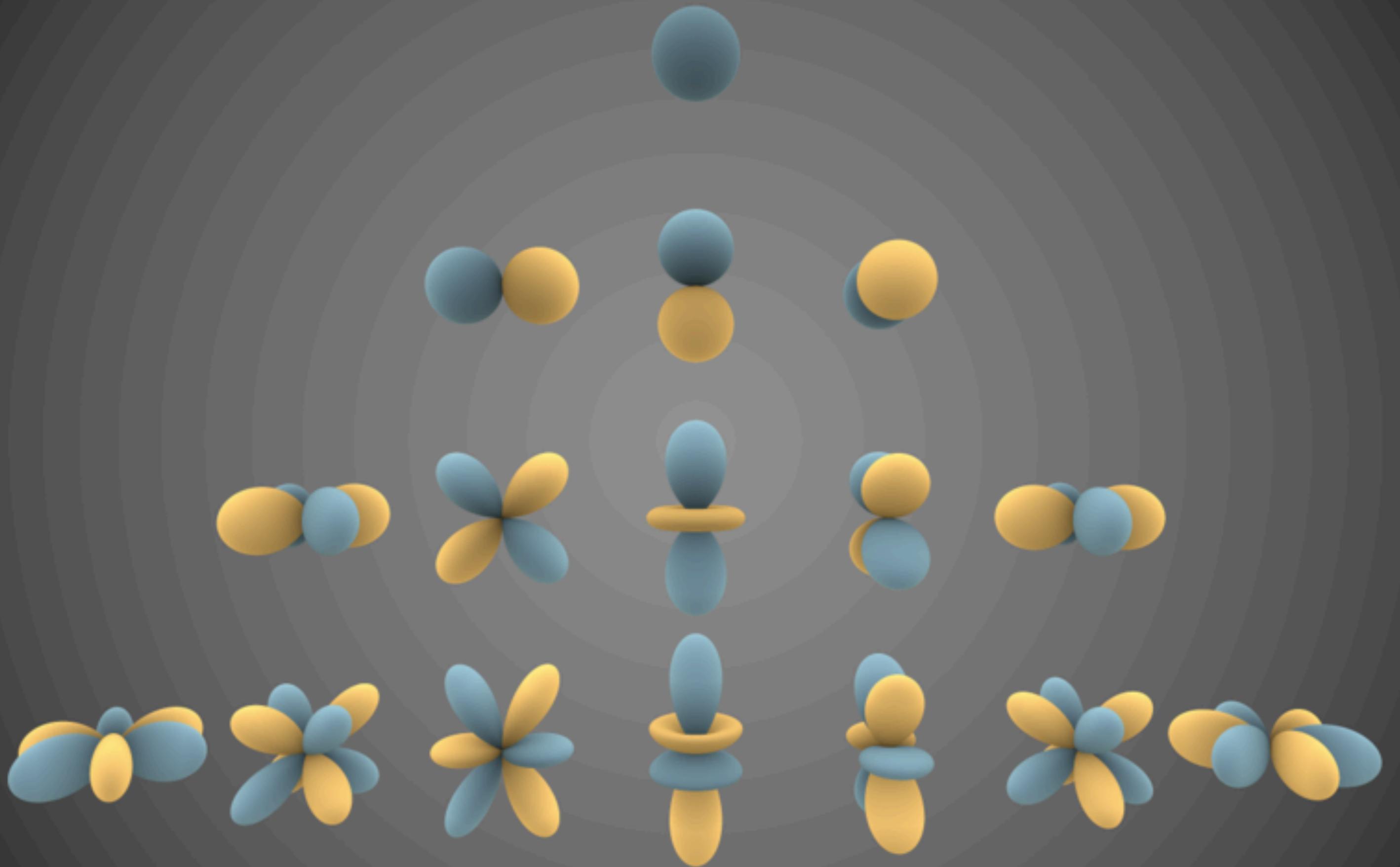
- ・ 二次反射以降の輝度情報
- ・ 二次反射以降の遮蔽情報
- ・ 光源の主成分方向情報

...

などなど**方向を含む情報**を取り出したいです。

Spherical harmonics(復習)

- **SH(Spherical harmonics)**とは**全周方向の関数のエンコード方法**
低周波にエネルギーが集中している場合はコンパクトに表現が可能です。
- SHは積算が容易 = **2つの関数の畳み込みが容易です。**
そのため、事前計算の分割の仕方に幅が出ます。
例) 環境マップのSH・遮蔽項のSH = イラディアンス(照度)
環境マップのSH・相互反射を考慮したSH = PRT
- 弊社では様々なベイク用のデータを**SHとして保持**しています。
PRTをベースに相互反射を考慮したSHをベイクしています。



引用: https://en.wikipedia.org/wiki/File:Spherical_Harmonics.png

PRT(Precomputed Radiance Transfer)

- 過去のCEDECに詳細がありますのでこちらを参照してください。

“ゲームプログラマーのための初級PRT”

http://cedil.cesa.or.jp/cedil_sessions/view/10

“使える最新PRTのススめ ～おまえのPRTはもう死んでいる～”

http://cedil.cesa.or.jp/cedil_sessions/view/17

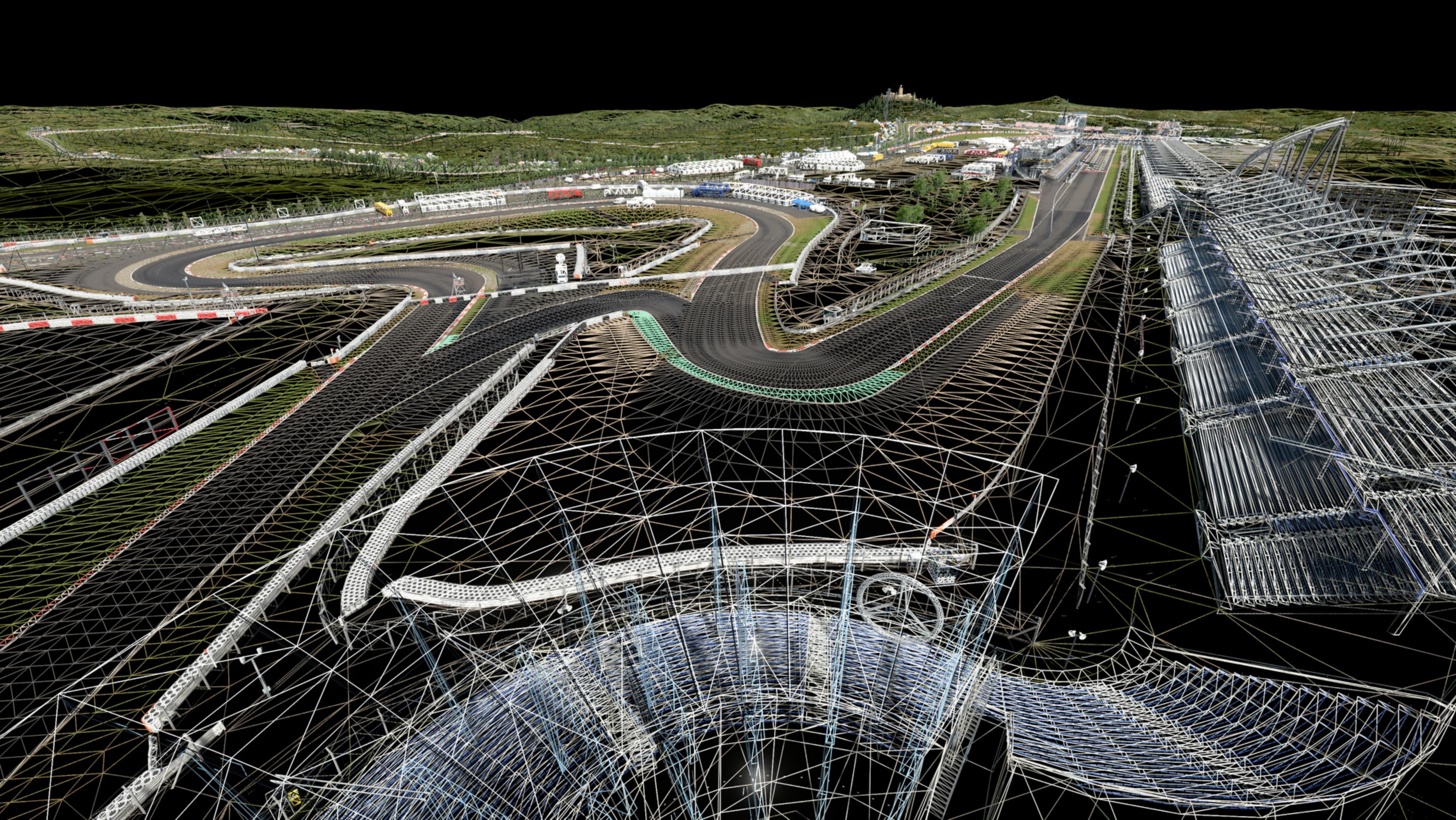
もっと詳細な情報が欲しい(続き)

- このレンダリング方程式のさらなる近似を与える情報が何なのかはレンダリングエンジンの思想によって変わってきます。
- つまりリアルタイムレンダリングとベイク環境は両輪となって開発される必要があります。
- ゲームエンジンの中には、市販のレイトレーサーを使わずに、専用のレイトレーサーを持っているものがある理由の一つがこれです。

さらなる要望

- ・ 「ラストライズのための**リファレンスのイメージ**も生成したい」という要件もありました。
- ・ またKasuriは**GPUベース**で**巨大なアセット**を扱うのが困難でした。

巨大なアセット？



An aerial view of the Nürburgring race track, showing the asphalt track, green grass, and surrounding landscape. A complex wireframe overlay is visible, representing a digital model or simulation of the track. The text is centered over the track.

Nürburgring
一周 25.378km
42GB

Iris Renderer

- 新規開発された内製の**MCL**レイトラーサーです。
- 弊社のパイプラインに投入されており、
2017年発売のPlayStation®4用ソフトウェア
『グランツーリスモSPORT』の
全車種、全コースに利用されています。

Raytracing



PlayStation®4



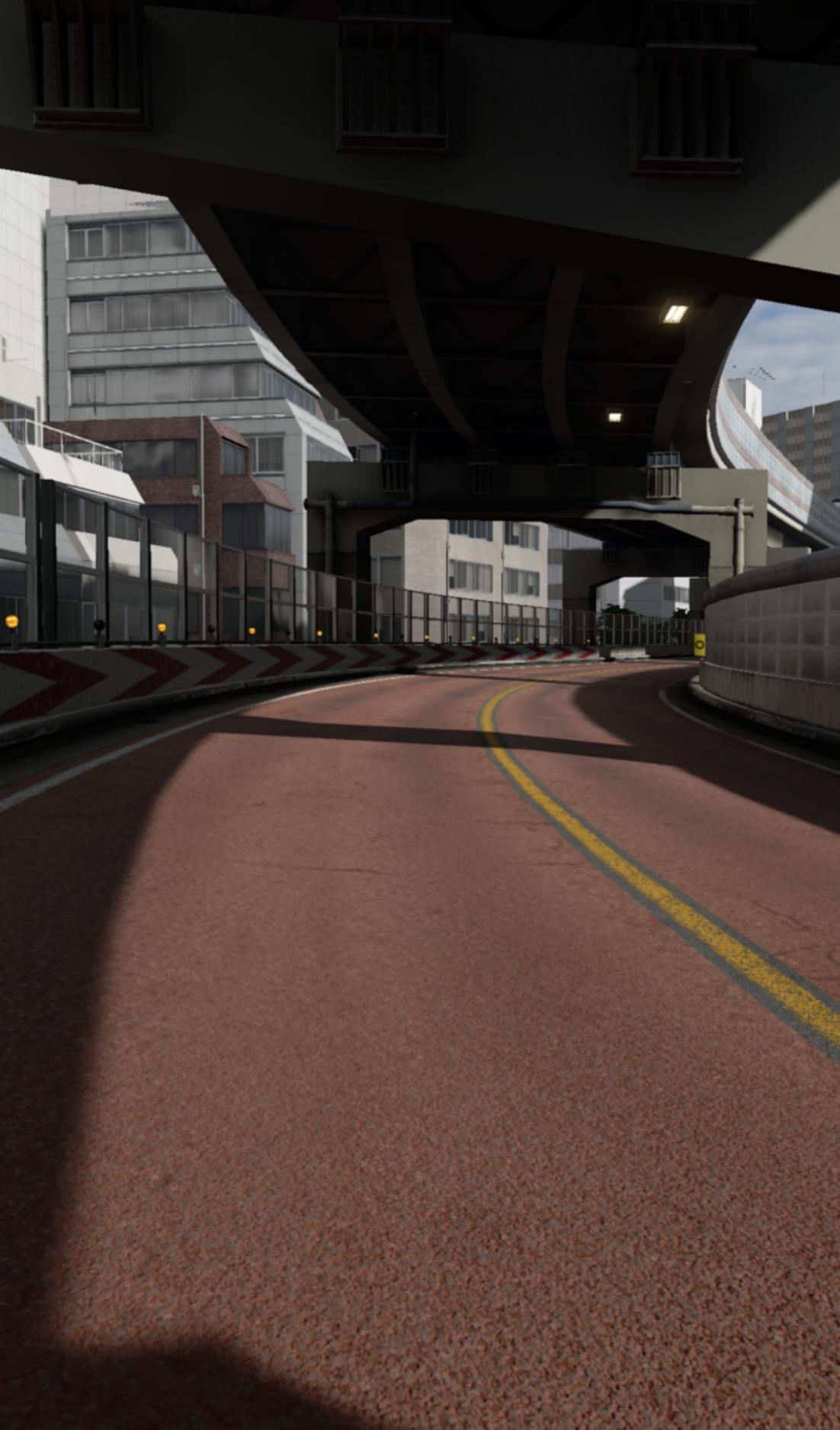
PlayStation®4



PlayStation®4



PlayStation®4



PlayStation®4



GI OFF

PlayStation®4



GI ON

PlayStation®4

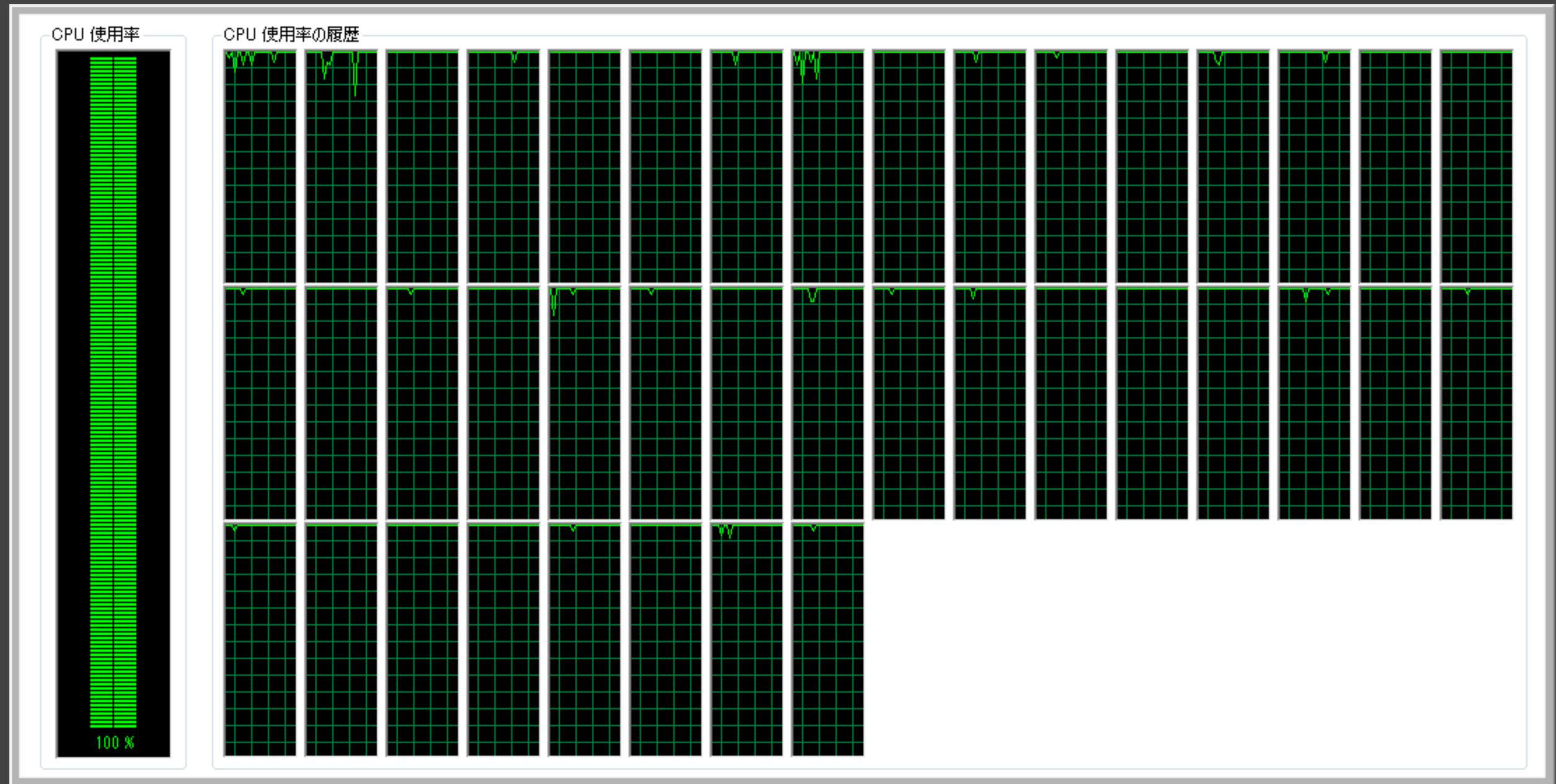


GI ON

GI OFF

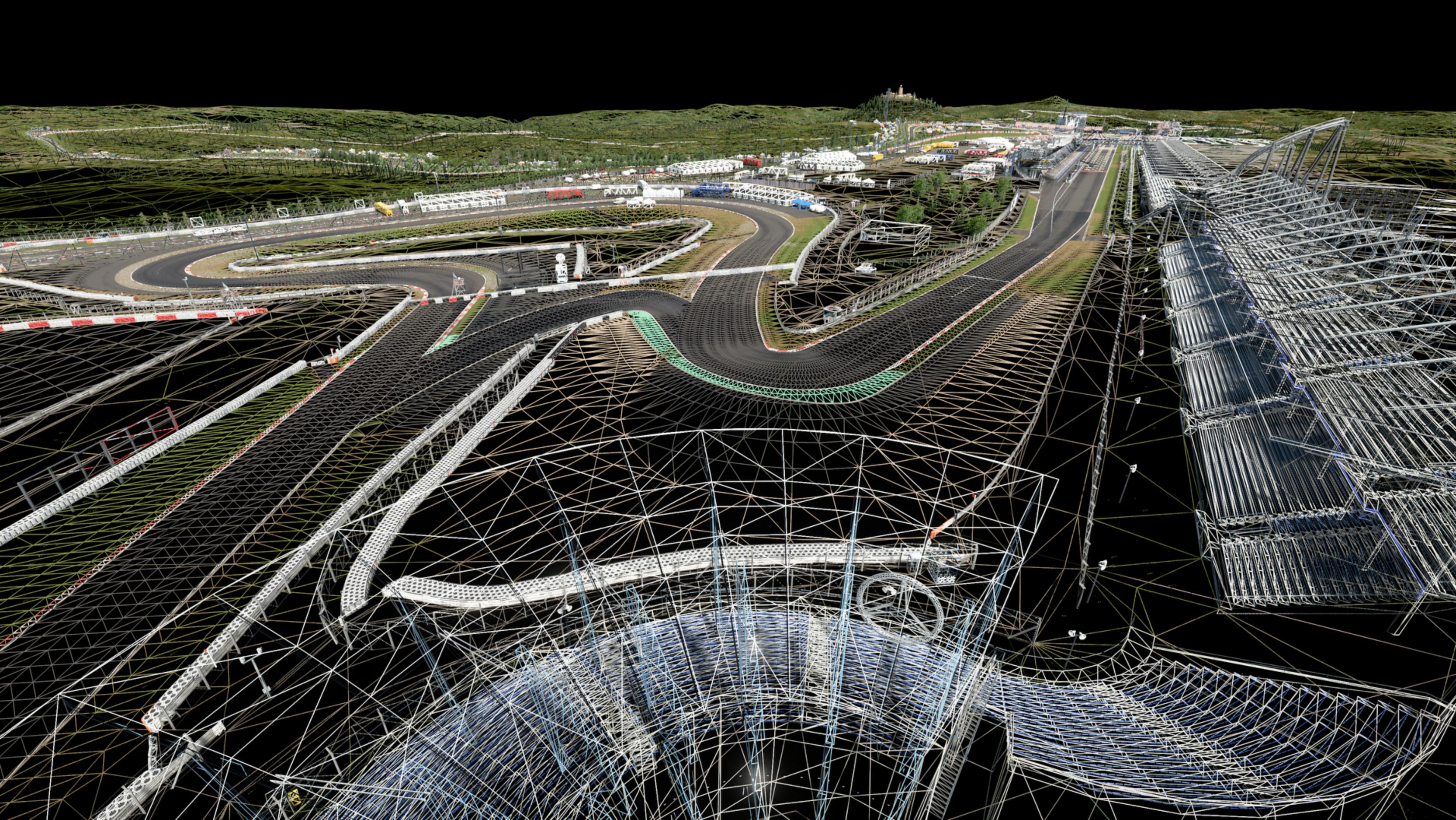
仕様

- Unidirectionalな**Path tracing**(MCレイトレーシングの一種)
- **CPUベース**の実装で、**巨大なコース**も扱うことができます。
- 社内の**カスタムシェーダー**を適切に扱うことができます。
- **複数のマシンに分散**して動作します。



- 一つのバッチで、**8つのサーバーに分散**。40コアx8台 = **320コア**
- 最も巨大なコースの全体ベイクが30分程度で完了します。

最も巨大なコース？

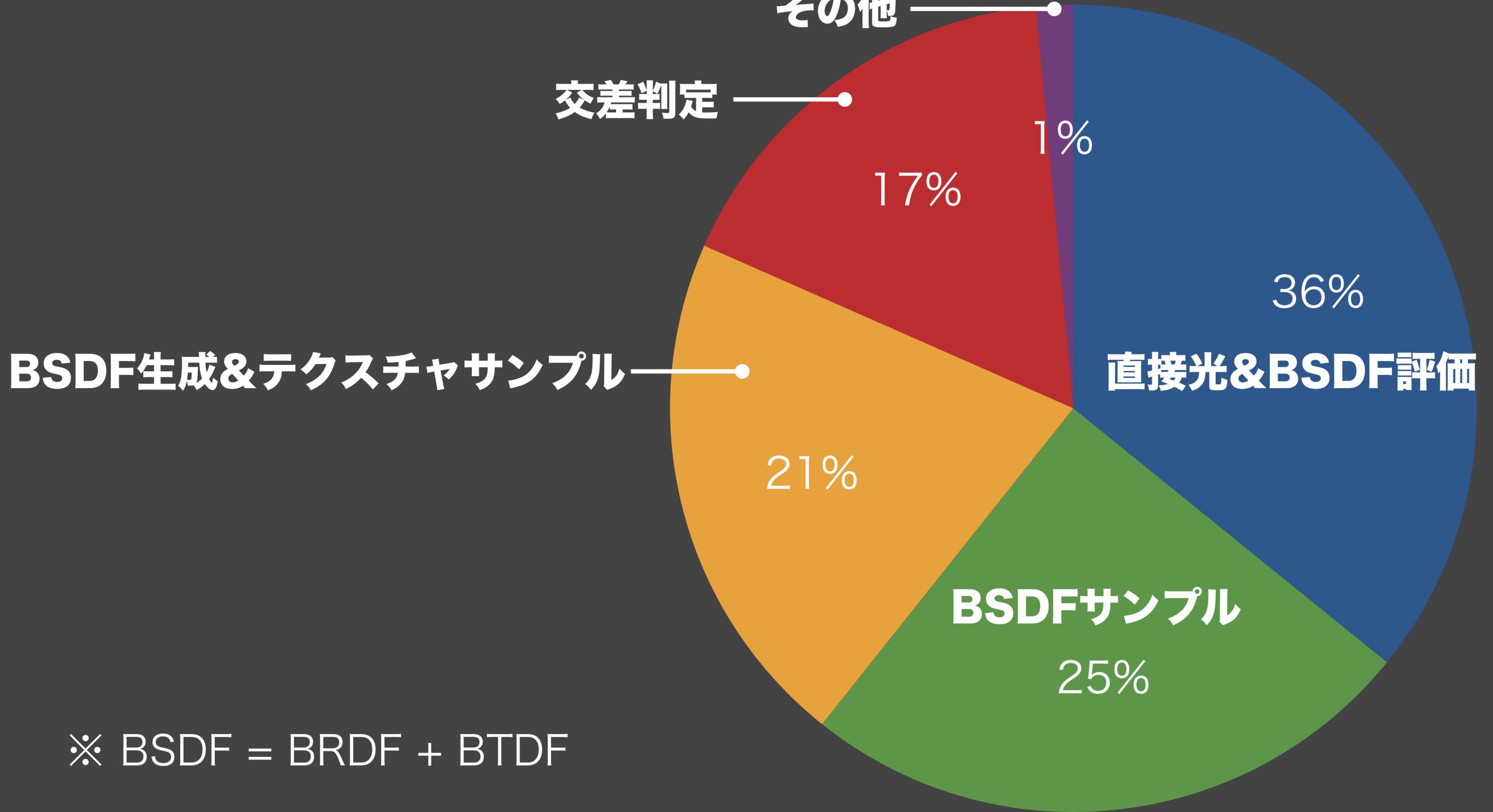


An aerial view of the Nürburgring race track, showing the complex layout of the circuit. The track is overlaid with a dense, grey wireframe mesh, highlighting the intricate geometry of the track and its surroundings. The background shows the green hills of the Nürburgring area under a dark sky.

Nürburgring
コンバート一回30分 = **160CPU時間**

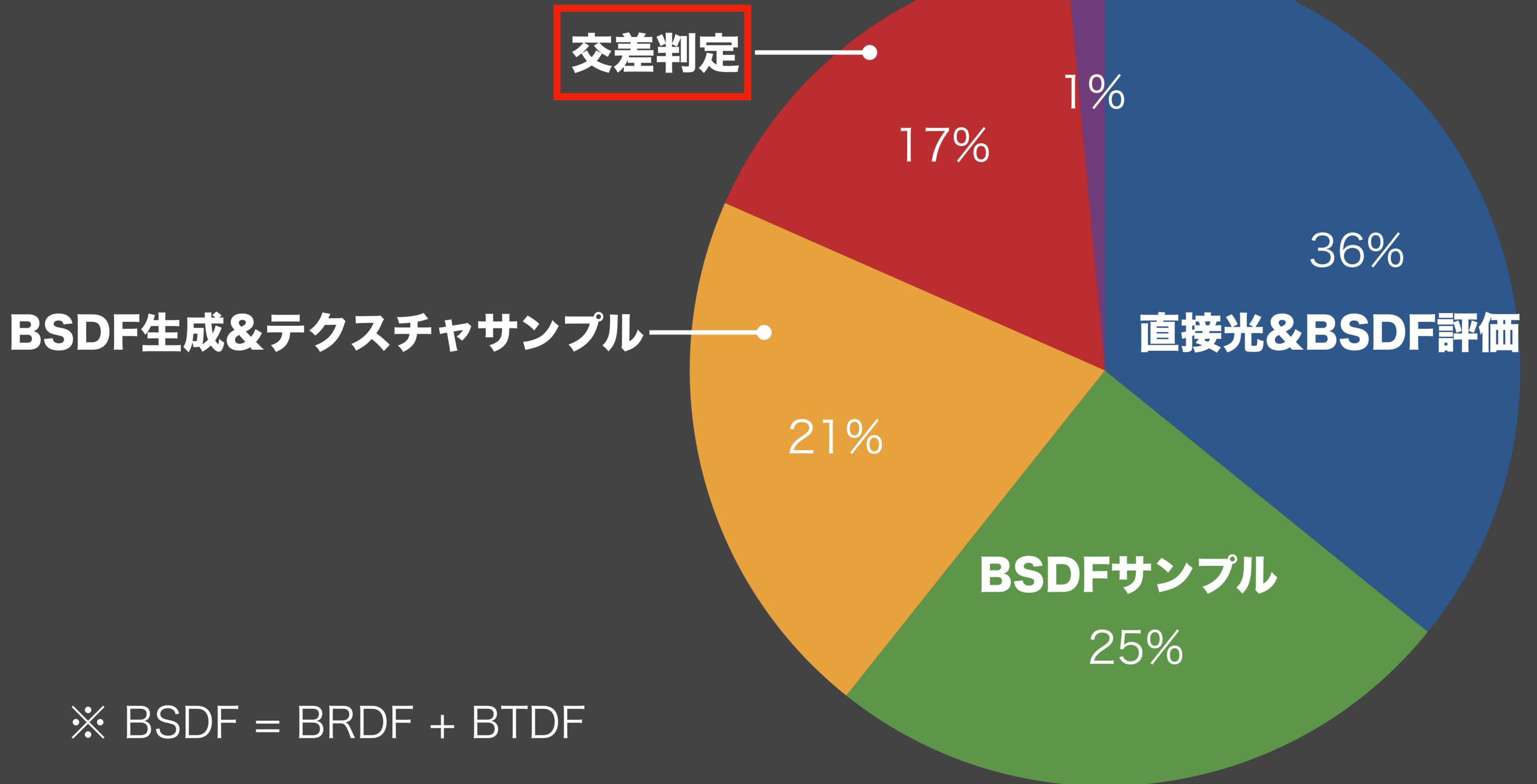
処理時間の内訳

一般的なコースの処理時間内訳



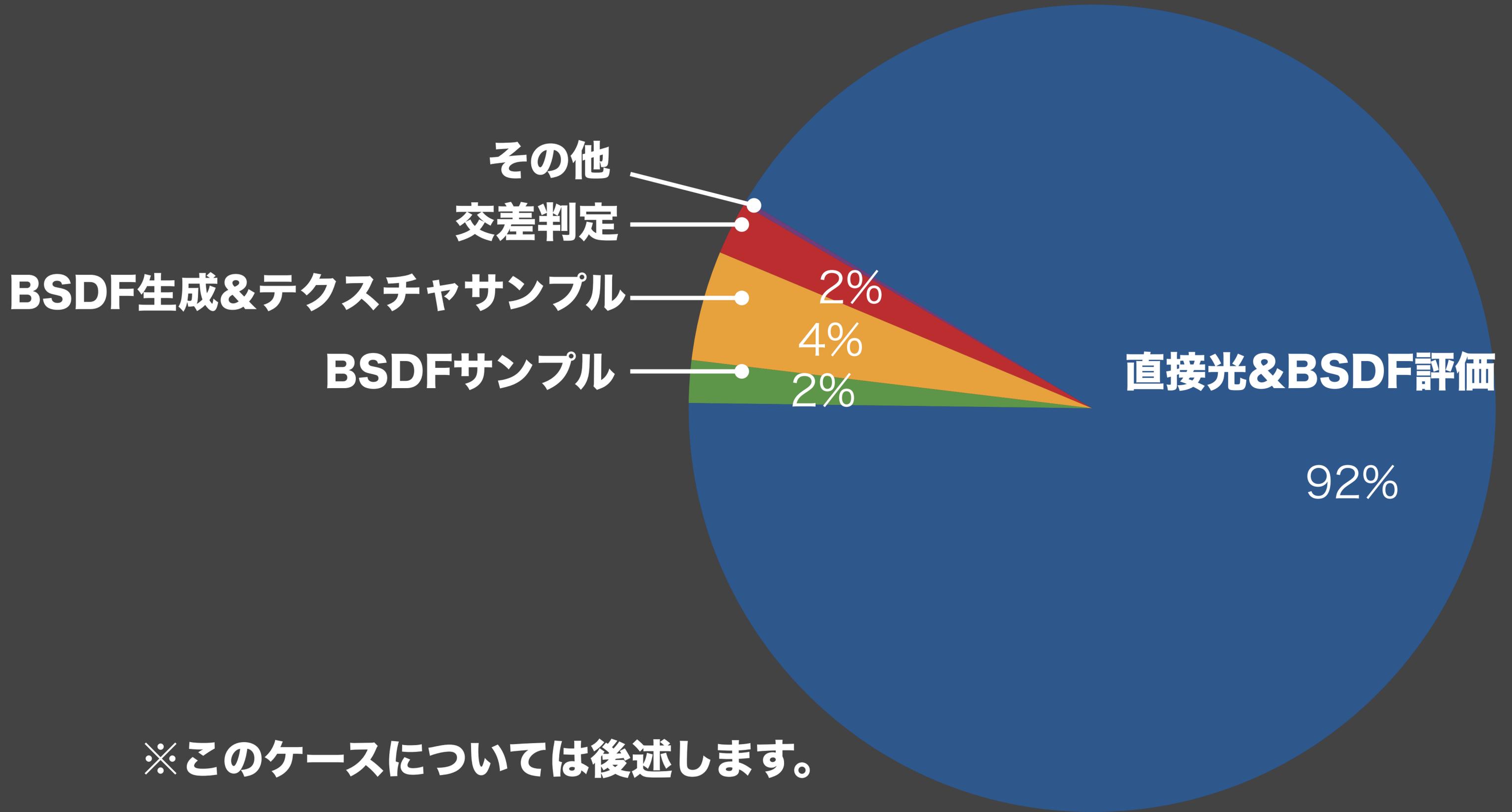
※ BSDF = BRDF + BTDF

一般的なコースの処理時間内訳



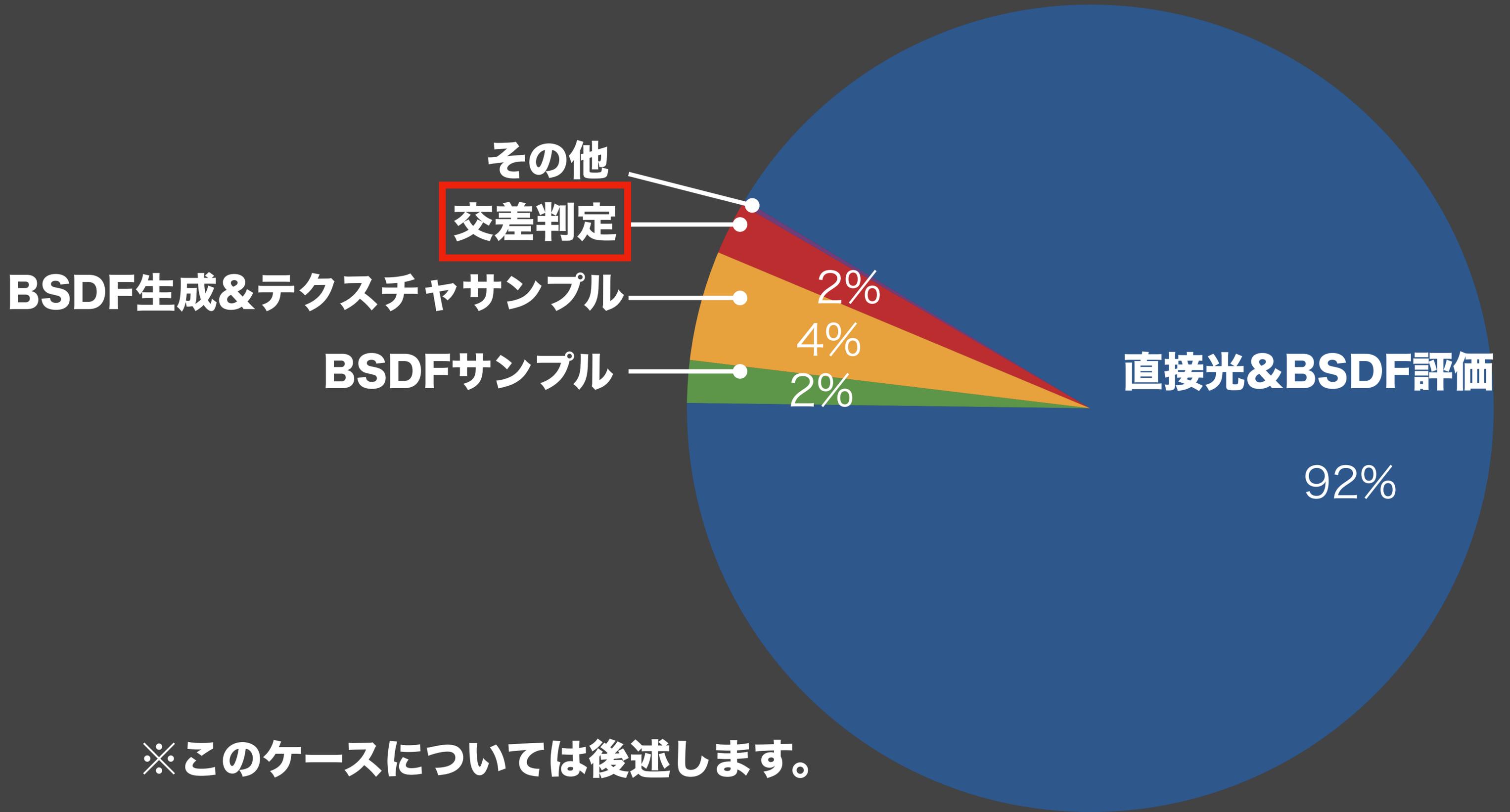
※ BSDF = BRDF + BTDF

光源が非常に多いコースの処理時間内訳



※このケースについては後述します。

光源が非常に多いコースの処理時間内訳



※このケースについては後述します。

小話

「レイトレは交差判定が最も重い」？

- 神話として広く信じられており、簡易なBSDFだと実際そうなります。
- 現代的なシェーダーが使われている場合、**テクスチャの枚数が多く、評価する式も多い**ため、**シェーダーの評価の方が重い**ことが多いです。

小話

「レイトレは交差判定が最も重い」？

- HitSort/RayDifferentials(Mipmap)など
テクスチャアクセスの**コヒーレンシーをあげる手法**があります。
“Sorted Deferred Shading for Production Path Tracing”
<http://www.andyselle.com/papers/20/>
- また光源が非常に多いシーンでは直接光の評価の負荷が大きいです。

小話

「プロダクションレンダラーはCPUベース」？

- GPUは**RAMの制限**が厳しく、実データを投入するのは困難でした。
- 先週発表された **Quadro RTX 8000**では**96GBのVRAM**。
- 今後はもしかしたらGPUレンダラーが増えるかもしれません。

この章のまとめ

- 弊社での**レイトレーシングへの取り組みの歴史**についてお話ししました。
- **ラストライズとレイトレーシングを繋ぐことで**
より高品質な表現が可能になることについてお話ししました。

目次

- はじめに
- レイトレーシングとは
- 弊社でのレイトレーシングの歴史
- **弊社での使われ方**
- 事故、デバッグ、対策
- 結び

弊社での使われ方

実際どう使われているのか

使われ方①

頂点ベイク

周辺情報を頂点に書き込む

テクスチャベイク？頂点ベイク？

テクスチャ

頂点

情報量

高密度

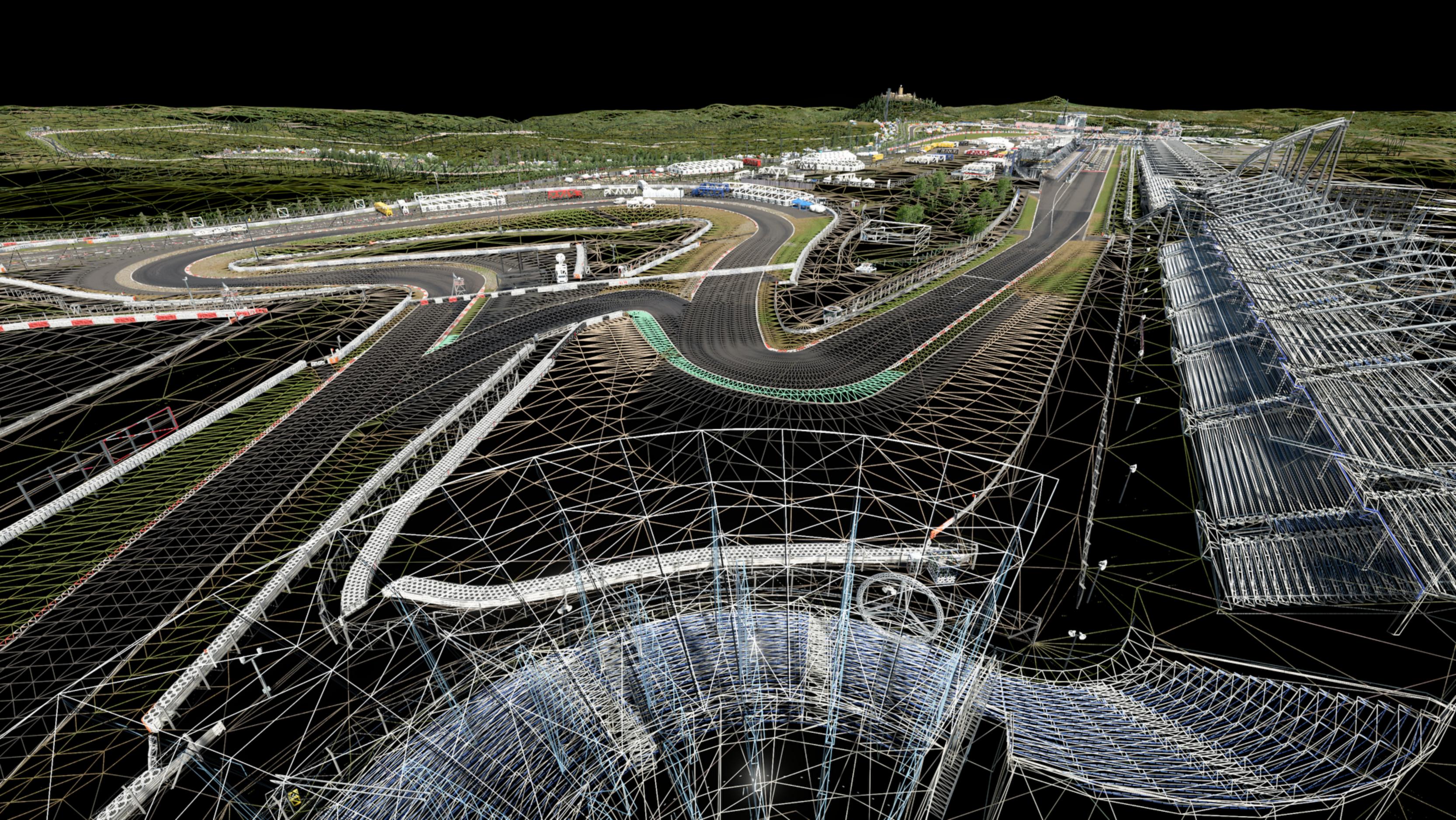
低密度

メモリとストレージ

大量消費

低消費

あのコースで
テクスチャにGI情報をベイクしたら
どれくらいサイズのなのだろう？

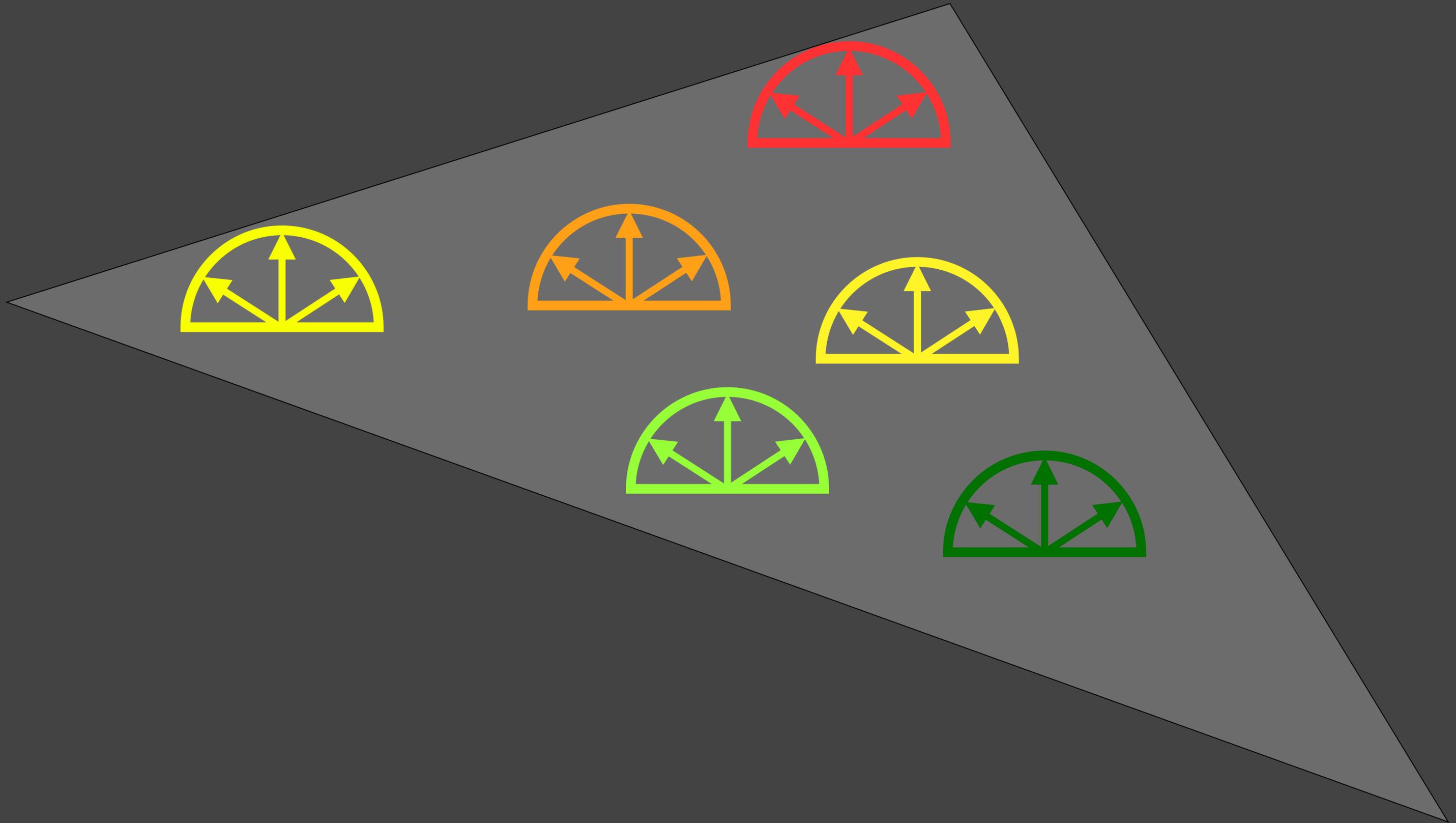


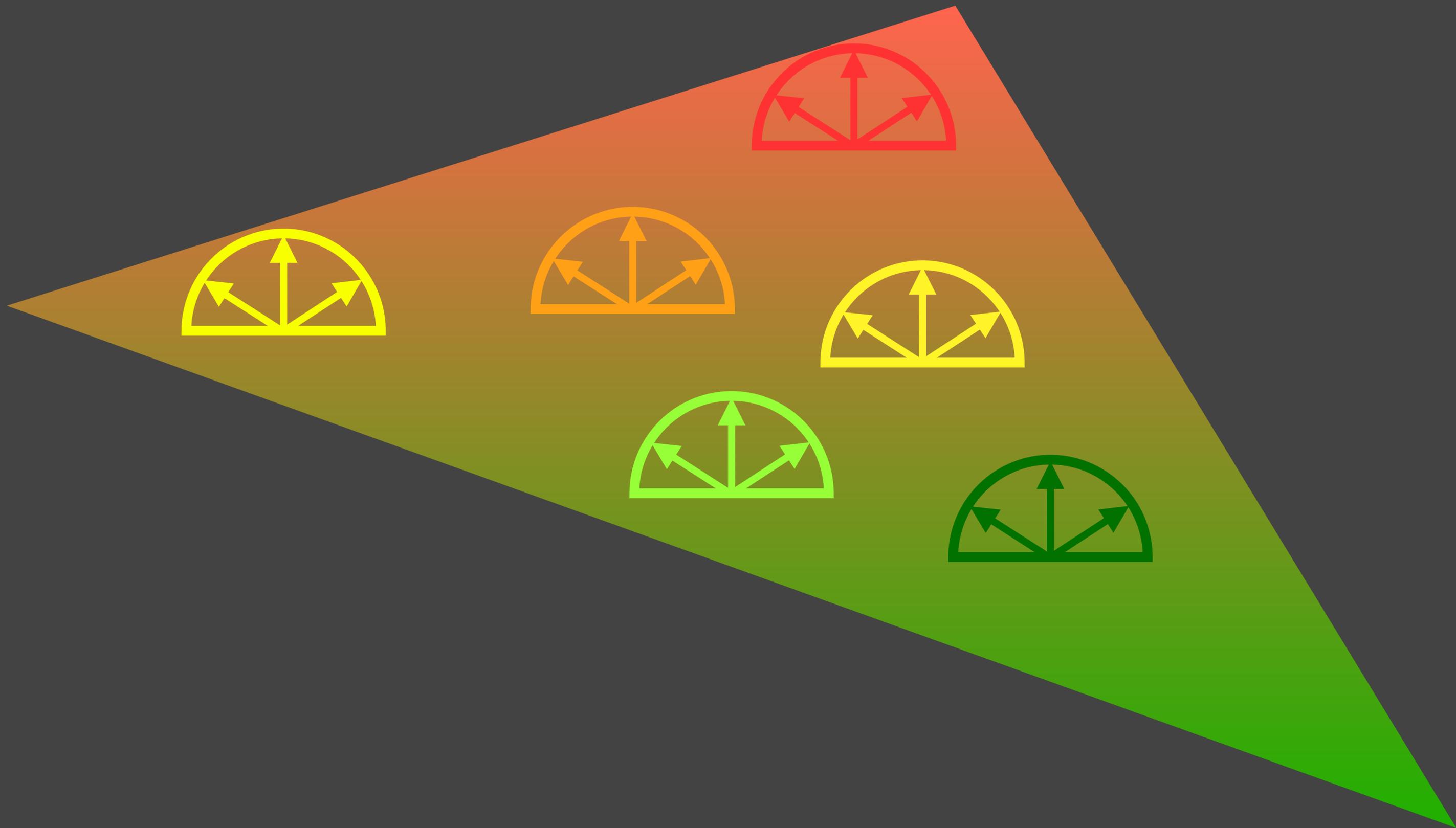


Nürburgring
約50GB(圧縮後)
テクスチャベイクの
実用はほぼ不可能

頂点ベイク

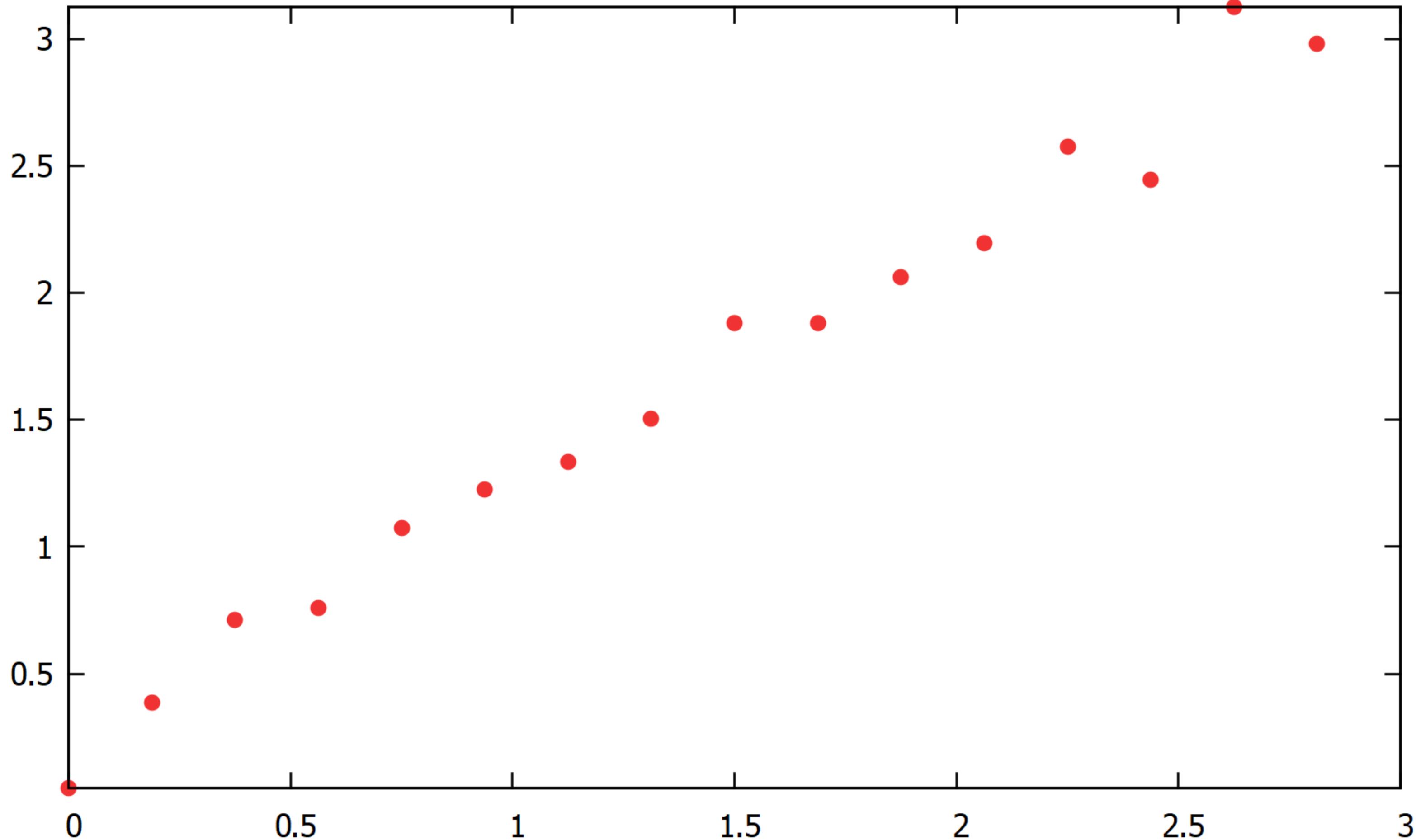
- **頂点付近**のいくつかのGI情報をまとめました。
- テクスチャに焼きこまずに頂点に**いくつかのSH**として焼きこみます。
- SHとして保持することで**方向の情報を残せます**。
- テクスチャは使えませんが、代わりに**シェーディング時の頂点間の補間を考慮**して、**エラーを最小化**するようにしています。

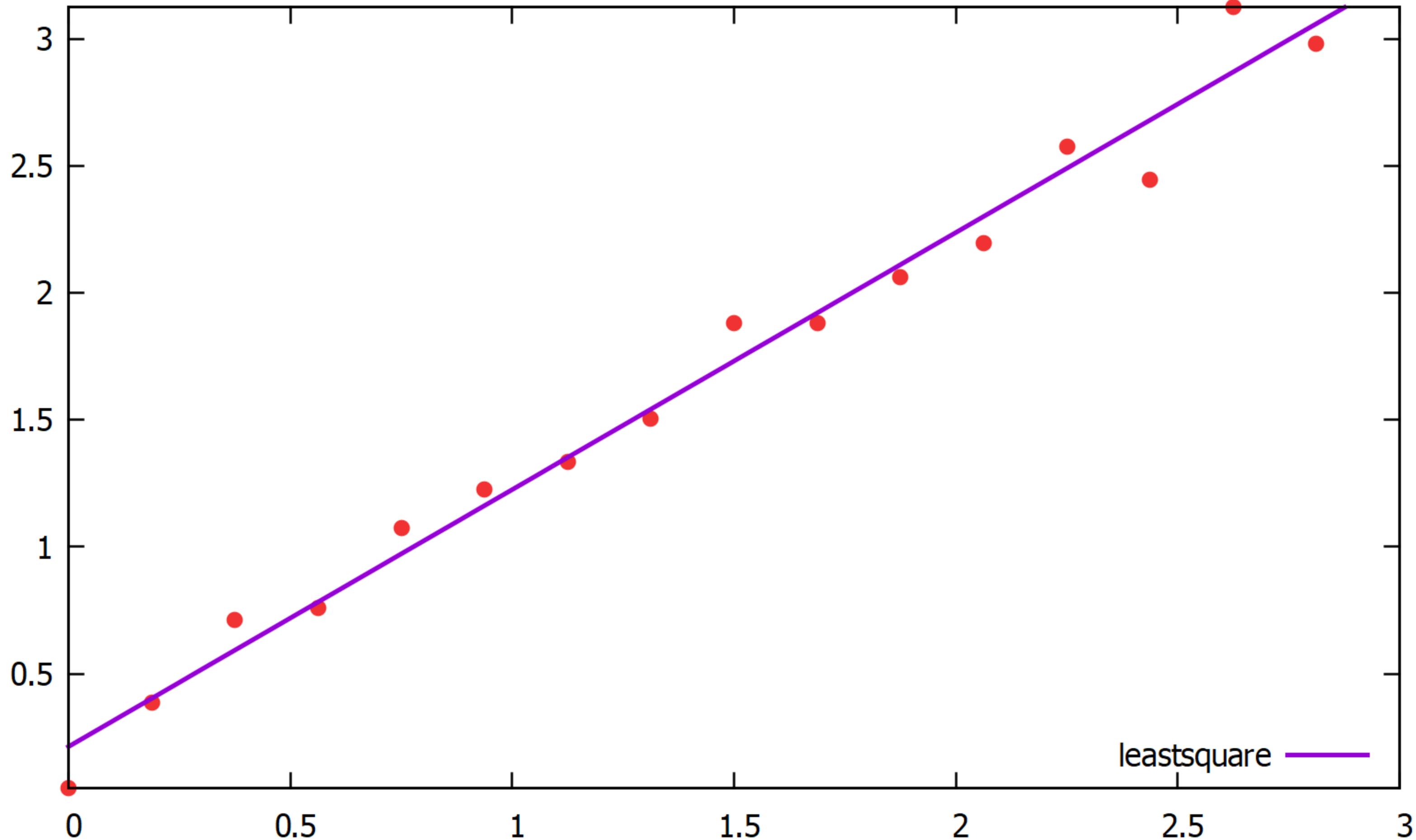




最小二乗法

- サンプルをなんらかの関数に**フィッティング**します。
- **誤差の二乗**を出し、それが**最小になる関数**にします。





極端な外れ値

- 残念なことに最小二乗法だけではうまくいきません。
- MCレイトレーシングは極端な外れ値が発生しやすいです。
「誤差の二乗」を最小化させようと大きく関数がずれます。
- そこで登場するのが正則化です。

正則化

- 最小化させる目的関数を誤差の二乗だけではなく、**パラメーターの二乗**も含めます(L2正則化)。

$$\text{objective} = \text{err}^2 + \text{param}^2$$

- **パラメーターが小さくなる**と、**シンプルな関数**になります。
例) 三次関数の3次のパラメーターが0 → 二次関数

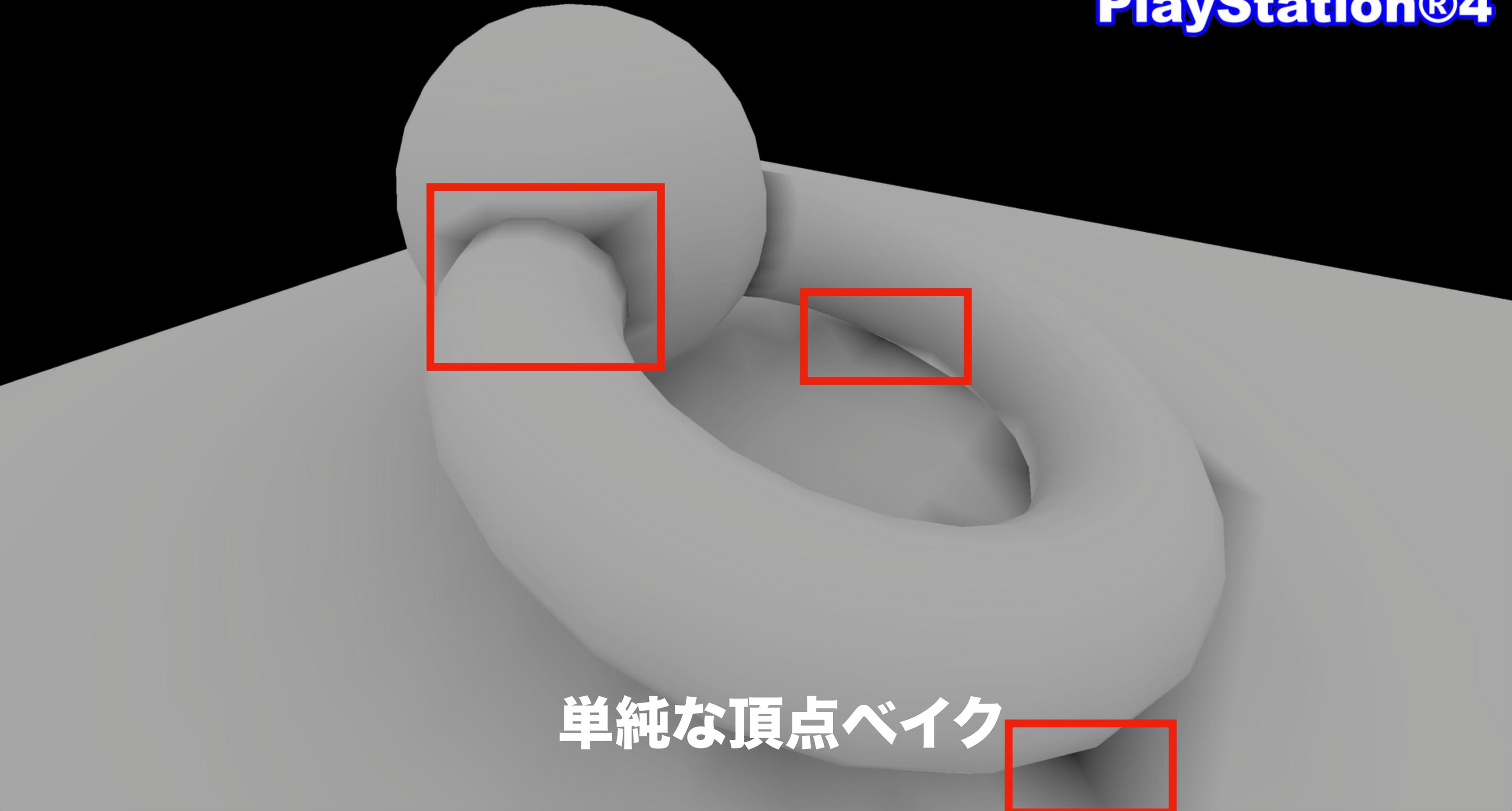
- 詳細は "Least squares vertex baking" を参照してください。

<https://dl.acm.org/citation.cfm?id=2383525>

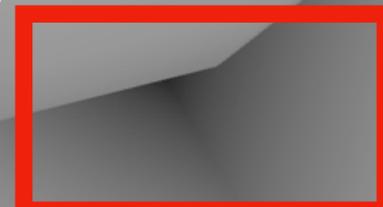
PlayStation®4

テストシーン

単純な頂点ベイク



単純な頂点ベイク



最小二乘法+正則化

使われ方②

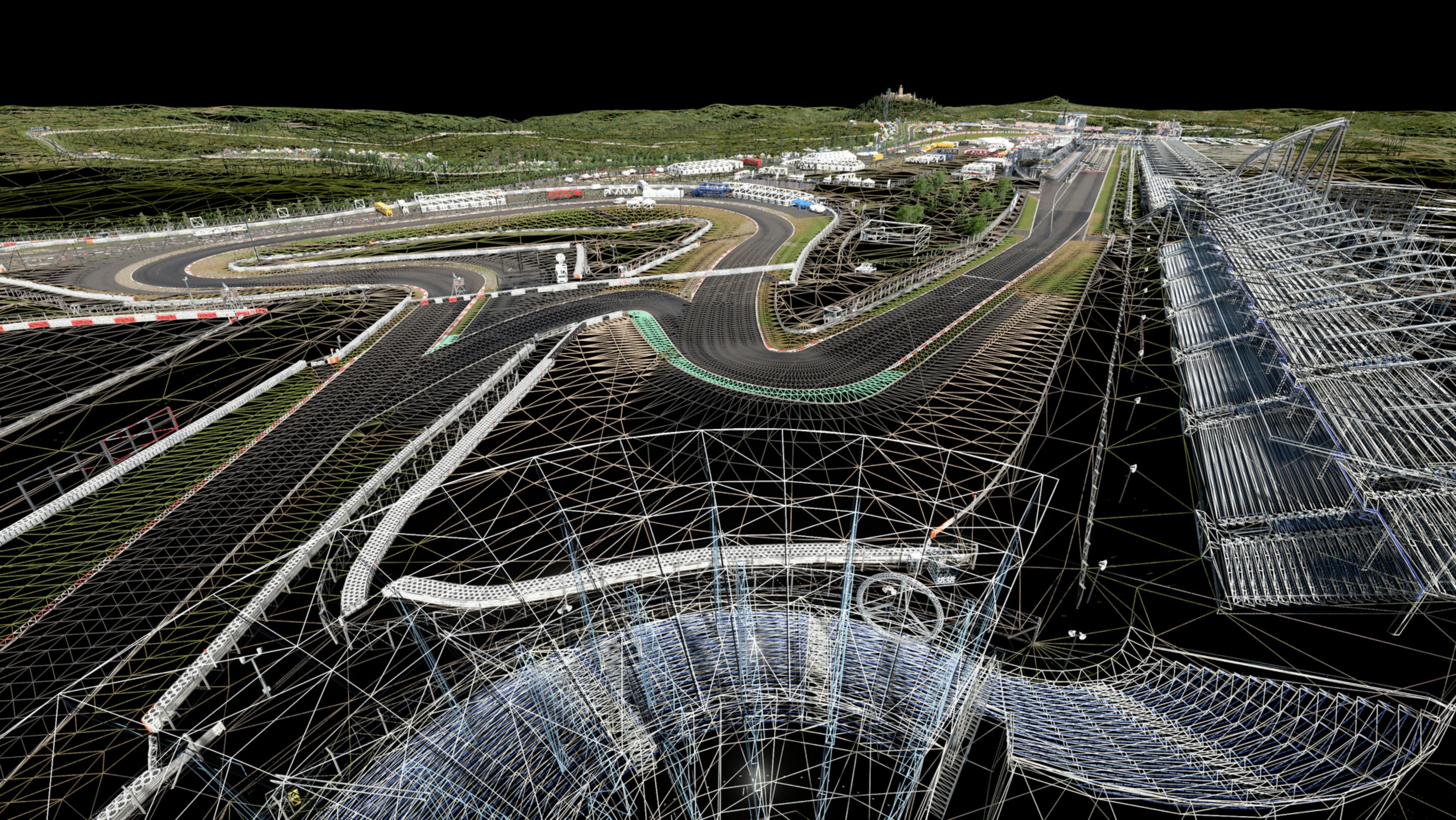
木の処理

木陰の丸み

木の下

- ・ 現実の木の下には**独特な丸みを帯びたシャドウ**が作られます。
- ・ しかし、木は**板ポリゴンの集まり**なので、木の素材データをそのままレイトレしても**よい結果にはなりません**。
- ・ 結果がよくなる都合のいい形状が嬉しいので**楕円体で近似**しました。

ところであのコースの木は
何本生えているのだらう…



An aerial view of the Nürburgring race track, showing the complex layout of the track and the surrounding landscape. The track is overlaid with a dense, grey wireframe grid, which is the subject of the text. The background shows green hills and some buildings under a dark sky.

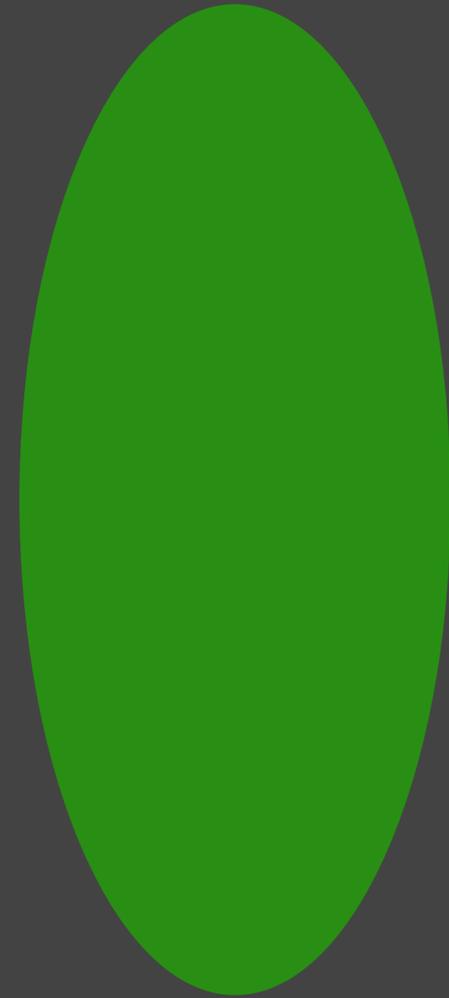
Nürburgring
10674本

PlayStation®4



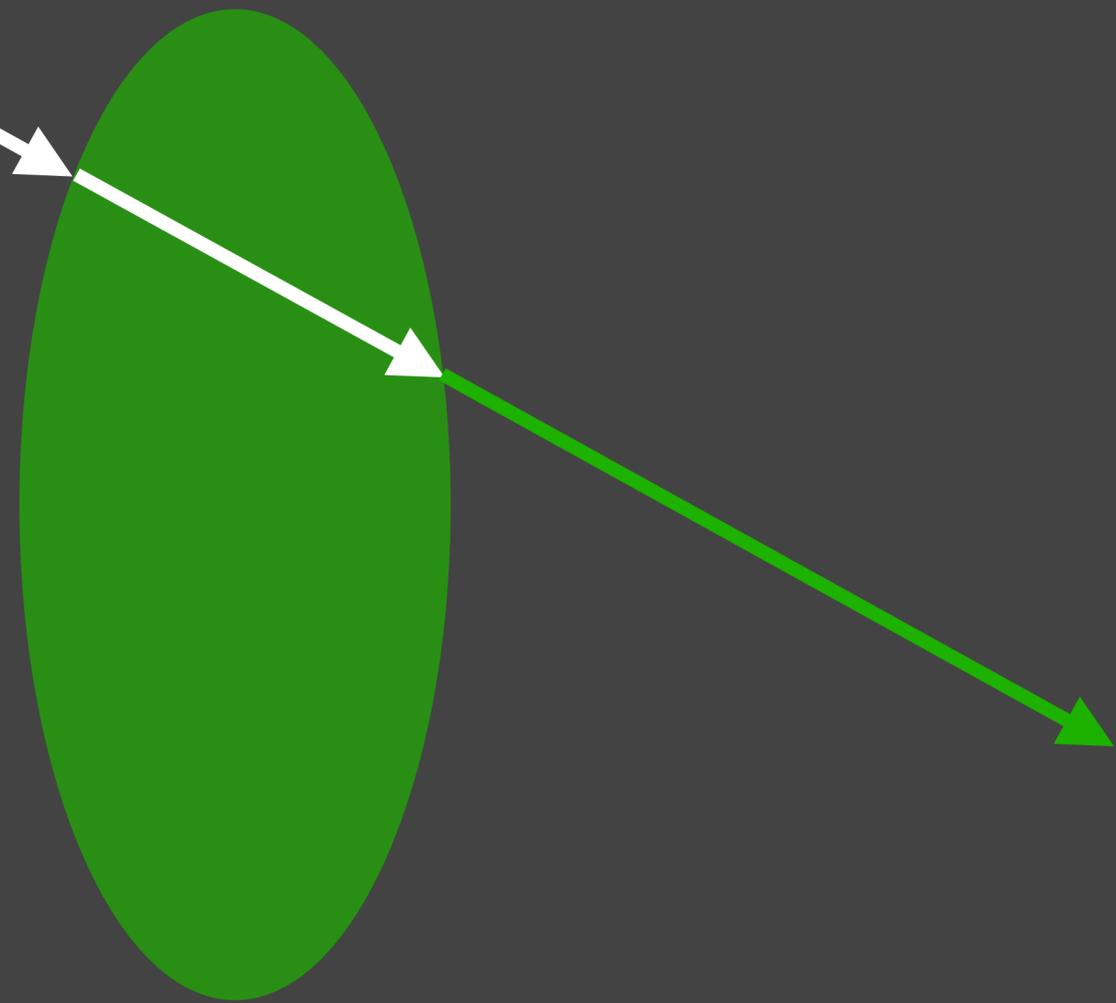
PlayStation®4





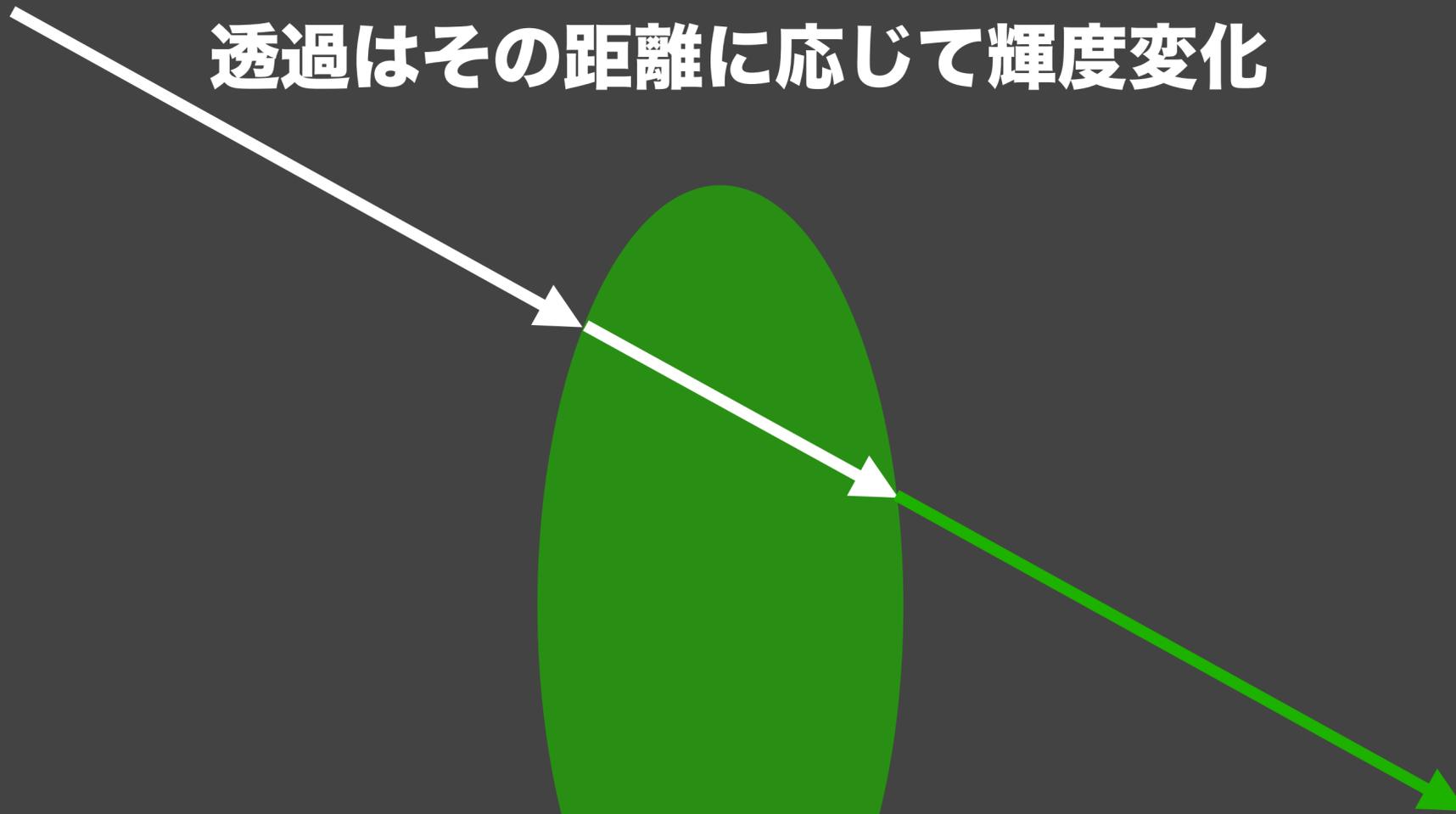


透過はその距離に応じて輝度変化

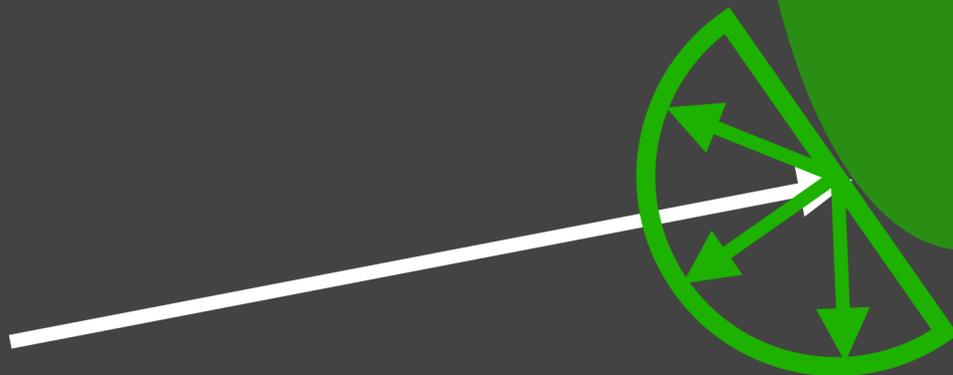




透過はその距離に応じて輝度変化



反射は拡散反射



PlayStation®4



使われ方③

手作業工程の自動化

アーティストの負荷軽減

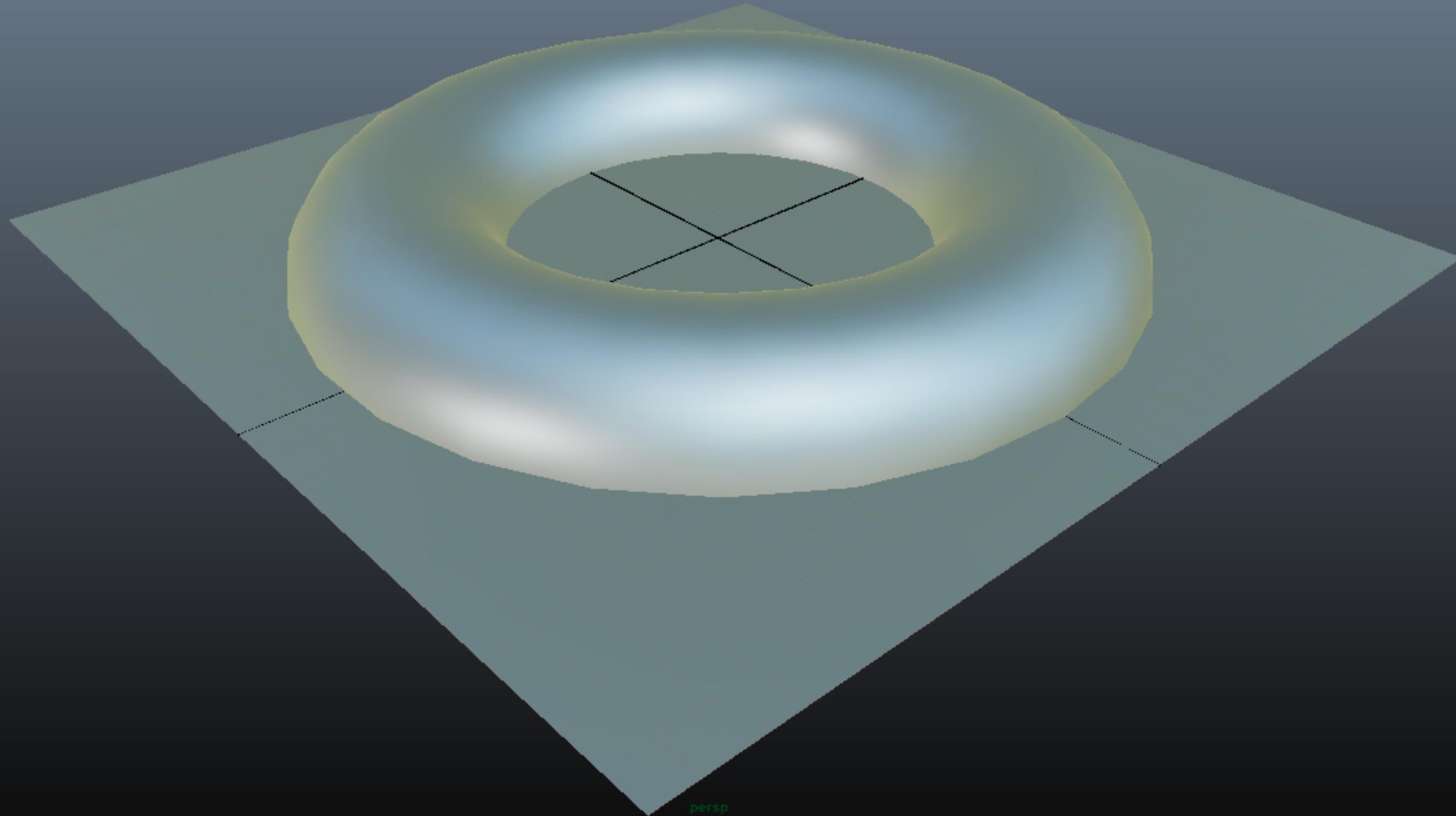
手作業工程の自動化

- ベイク時に以前は**手作業**で行われていた工程を**自動化**しました。
「周囲をHideしてからベイク」 「ちょっと浮かしてベイク」
「ベイク用の遮蔽物を置いてからベイク」
- 手作業で行っていた工程の情報をオブジェクトやシーンに埋め込んでいます。

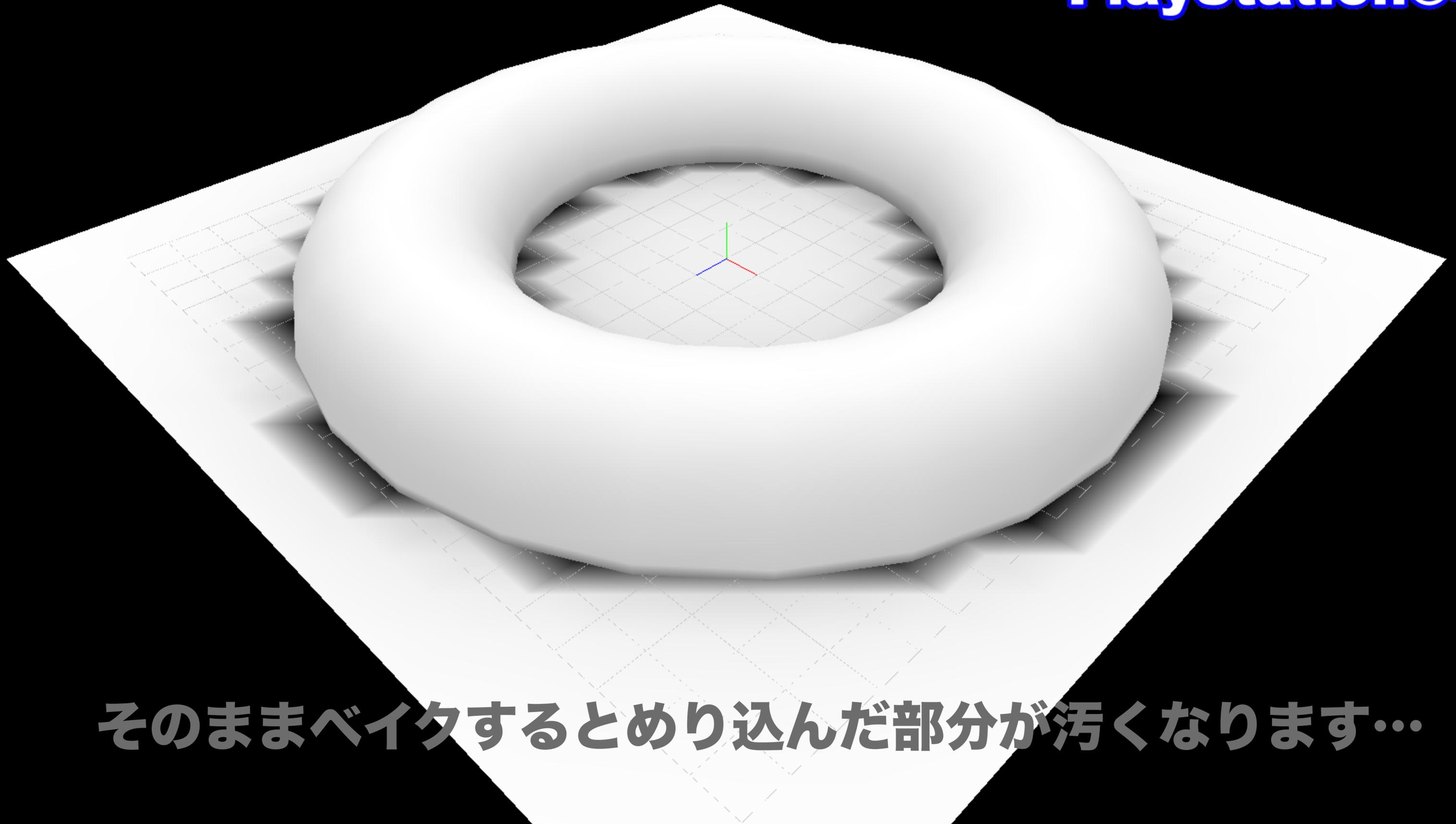
自動化① シェイプオフセット

- いわゆる”ぶっ刺し”で埋めているものを**指定量浮かせて**ベイクします。
- 交差による暗がりや焼きムラを避けます。

Maya

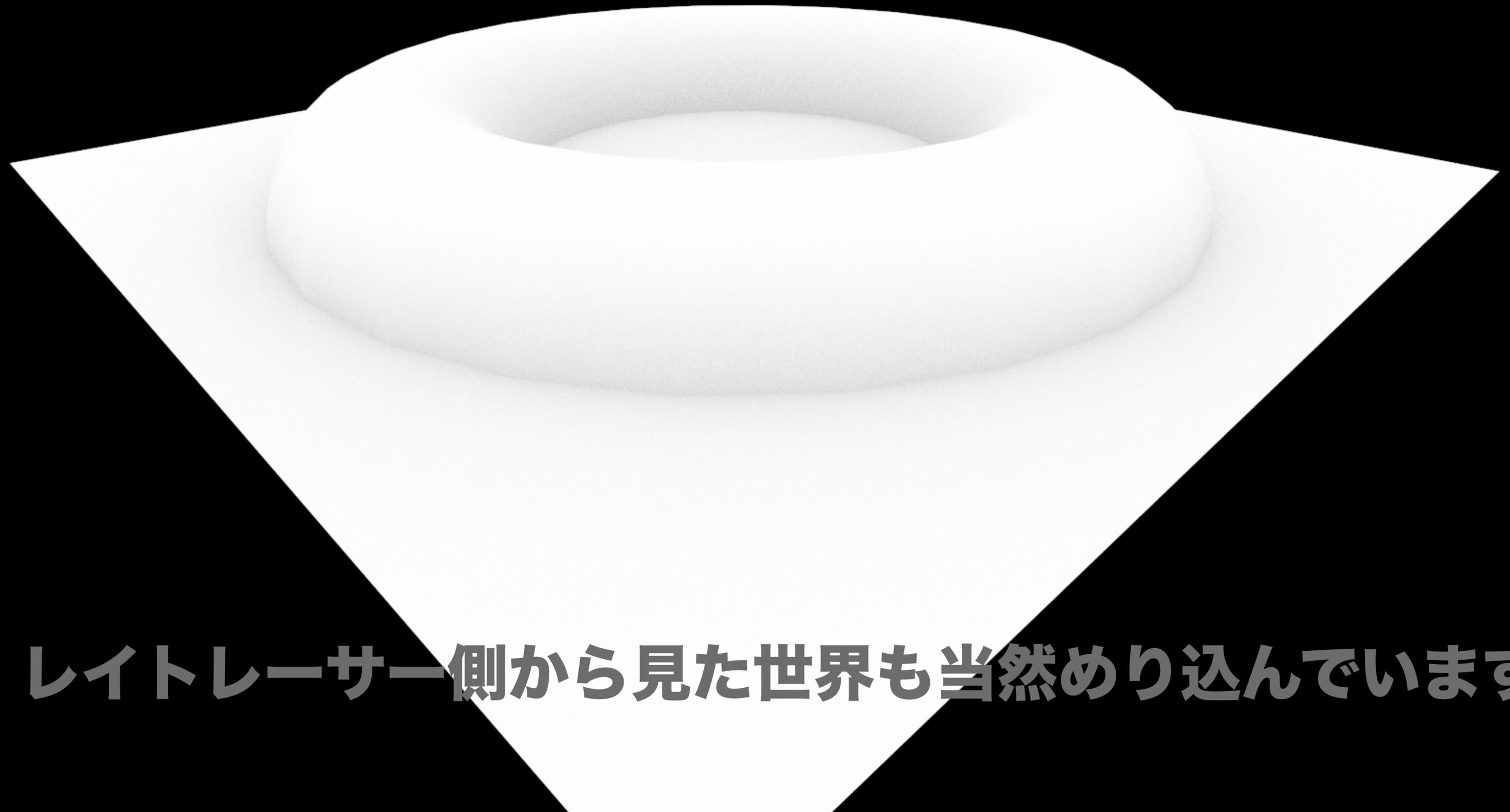


persp



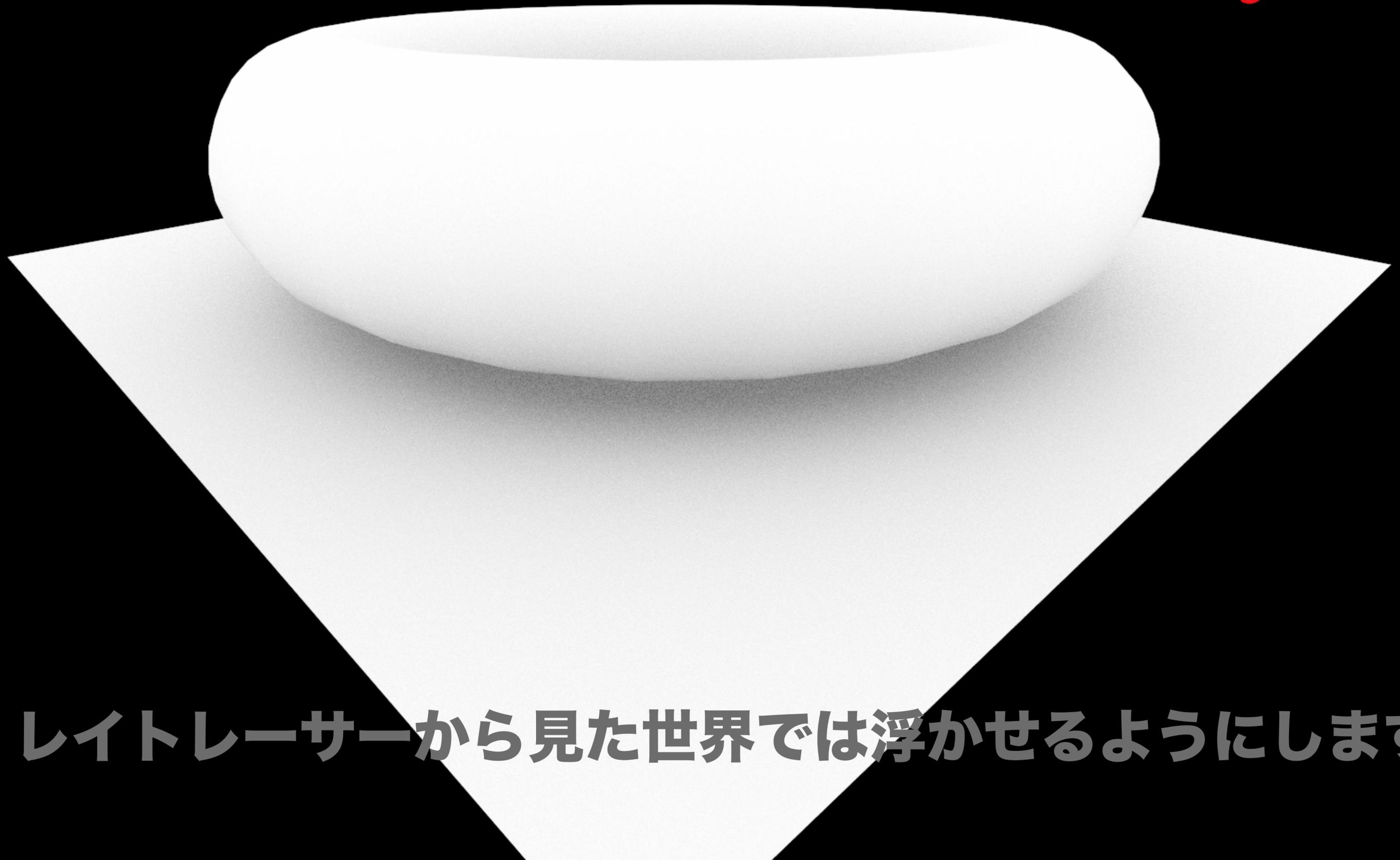
そのままベイクするとめり込んだ部分が汚くなります...

Raytracing



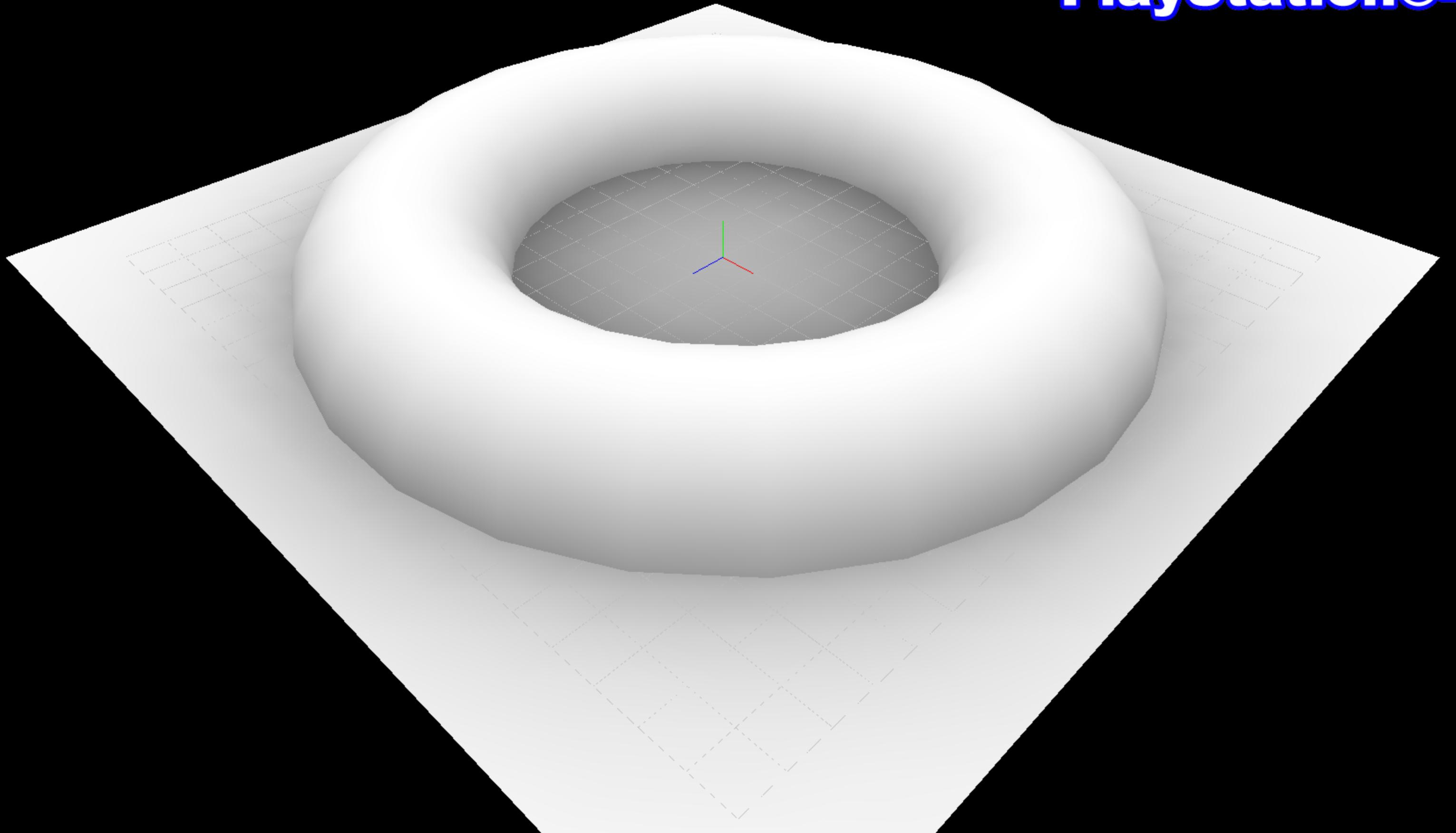
レイトレーサー側から見た世界も当然めり込んでいます

Raytracing



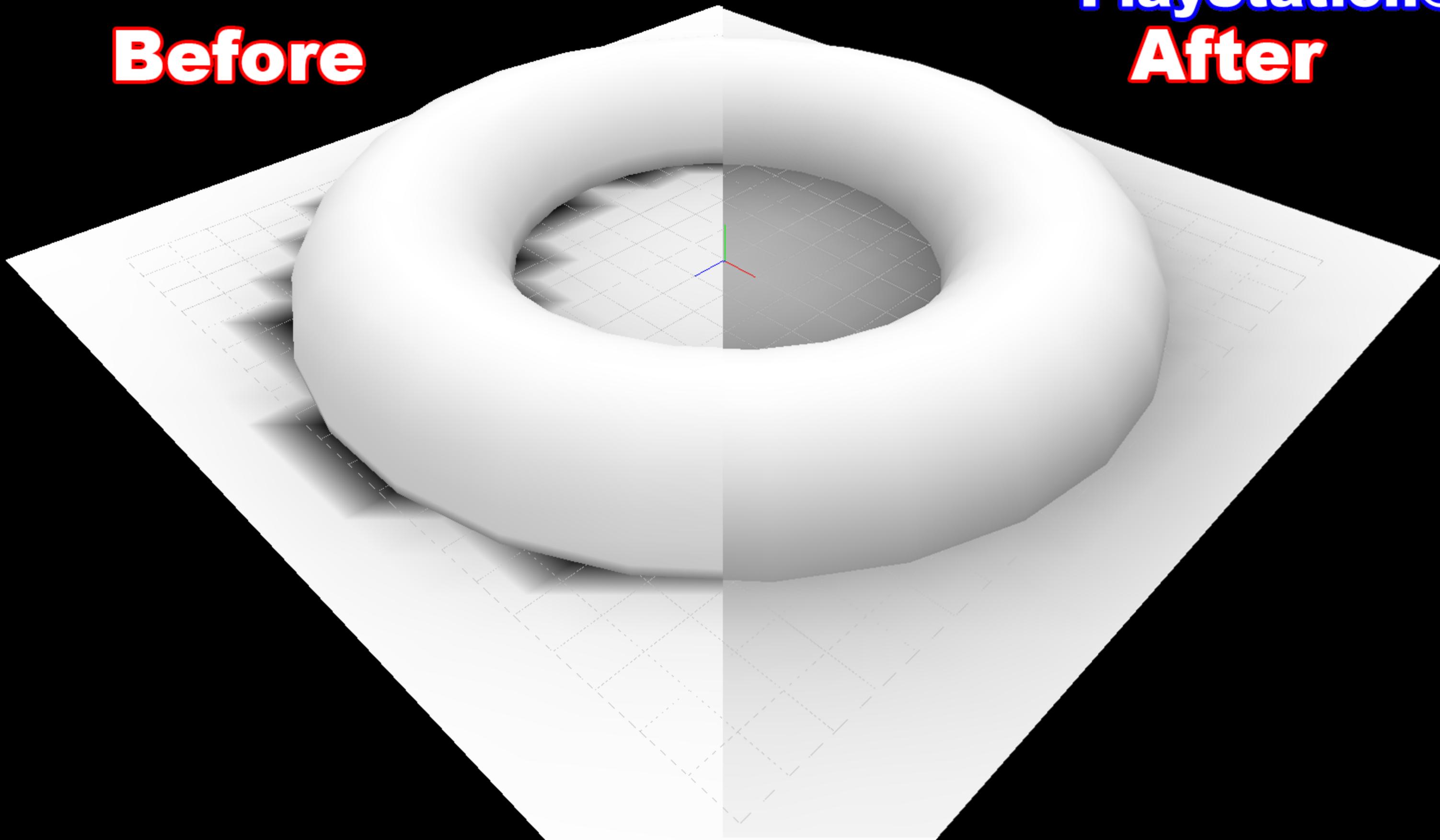
レイトレーサーから見た世界では浮かせるようにします

PlayStation®4



Before

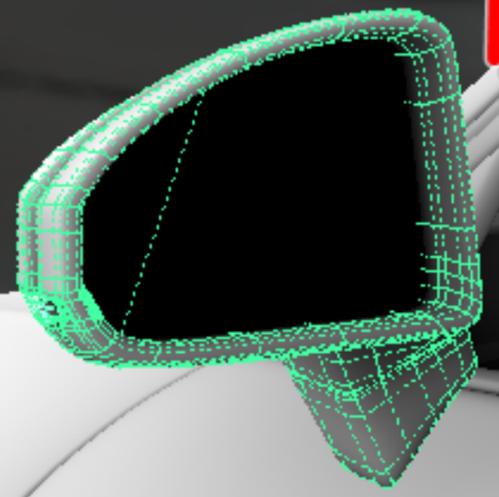
PlayStation®4
After



0
0
0
0

Maya

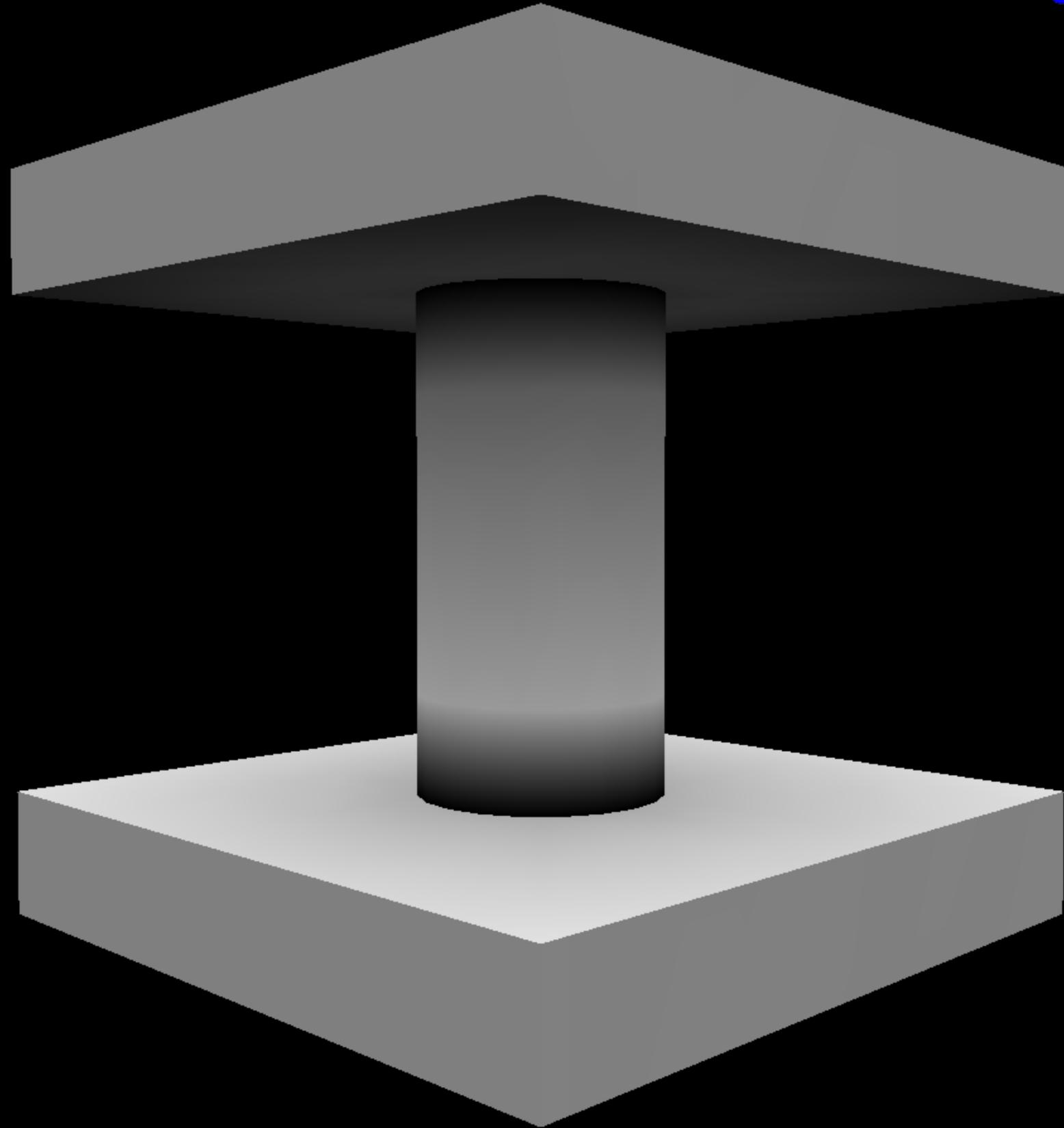
+Y に 0.03 浮かす



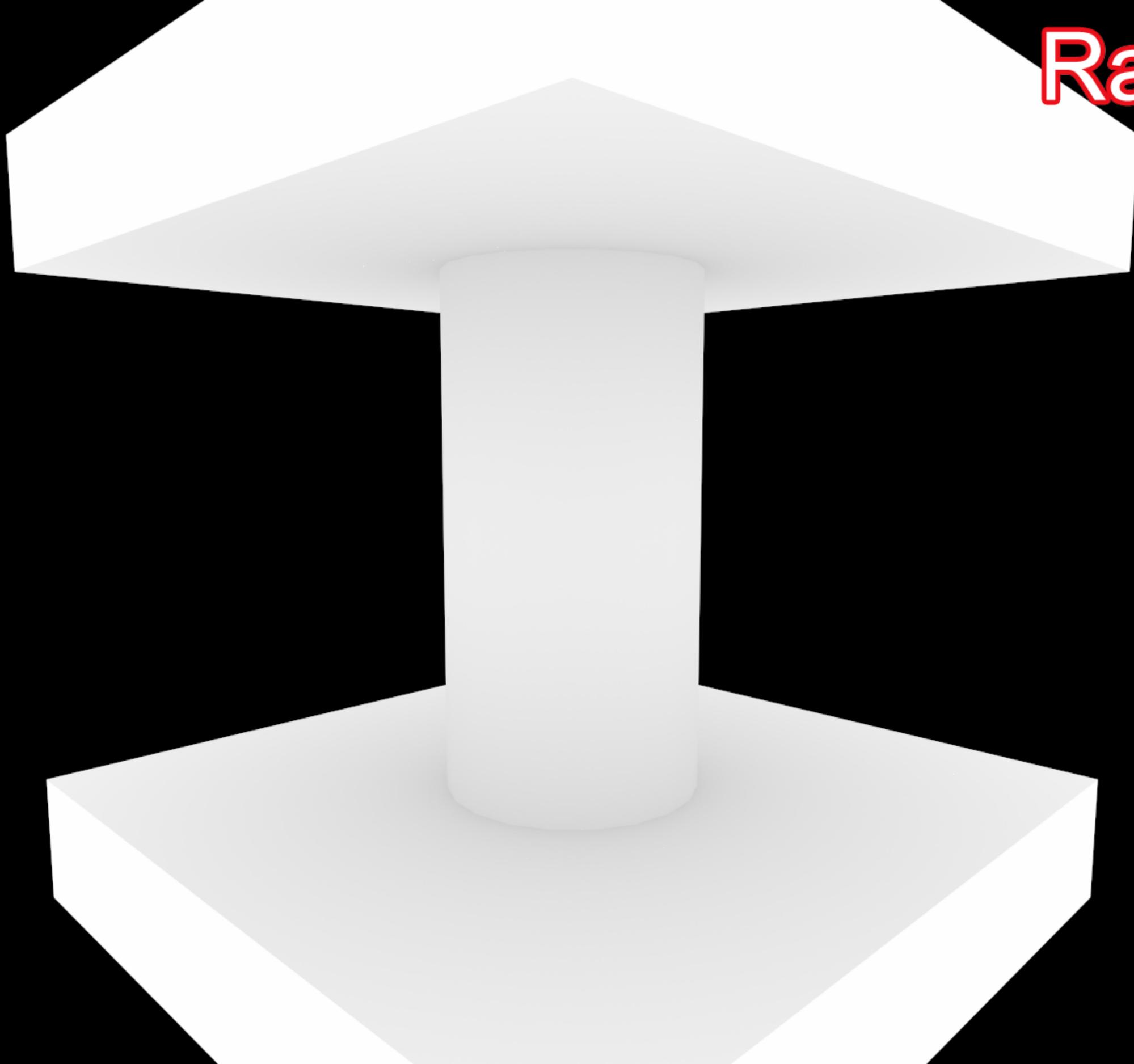
自動化② シェイプスケール

- ベイク時に**指定量スケール**してベイクします。
- 2方向以上が埋まっており1方向のオフセットでは対応できない場合、下に出る影が強すぎる場合などに使用します。

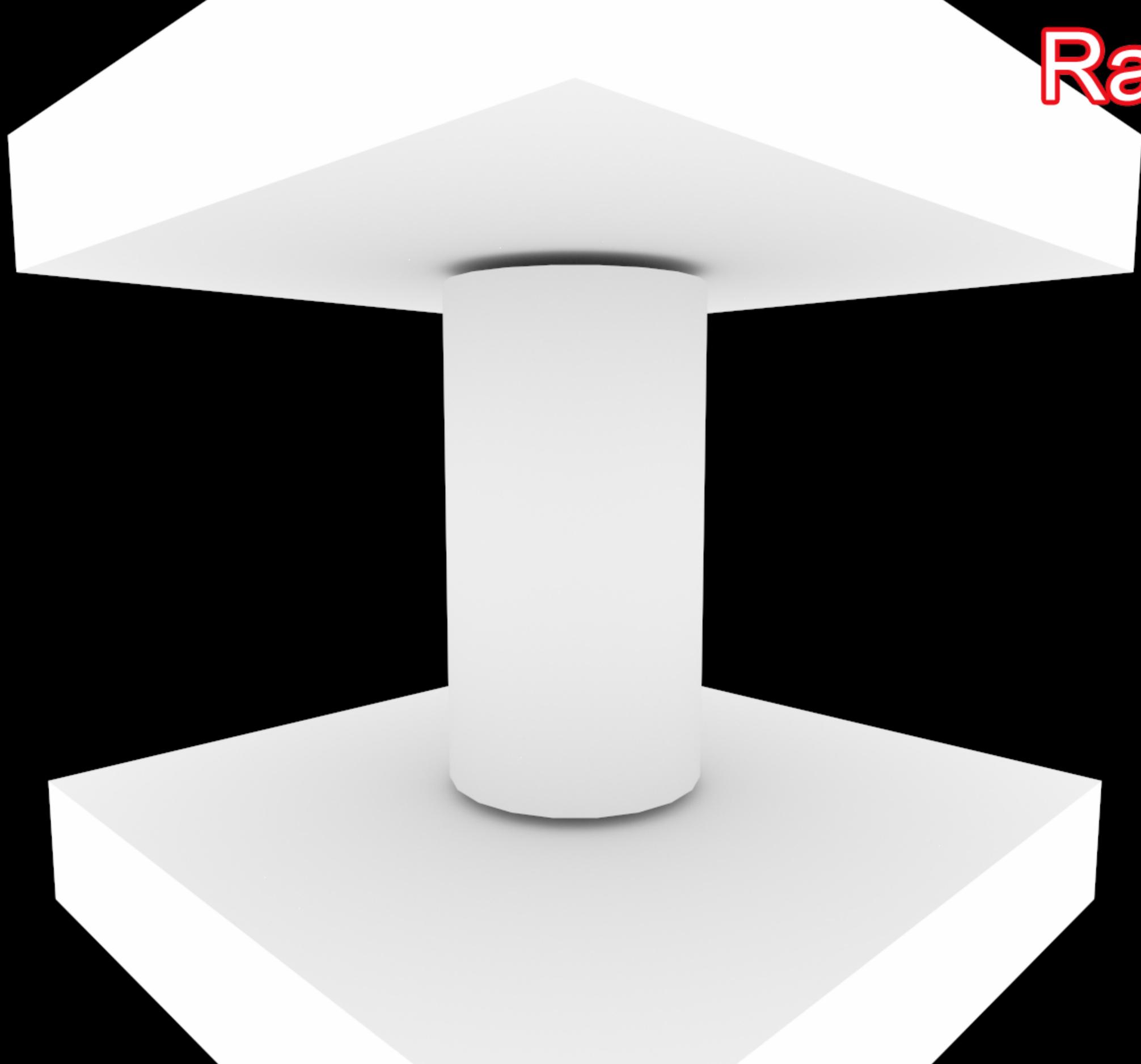
PlayStation®4



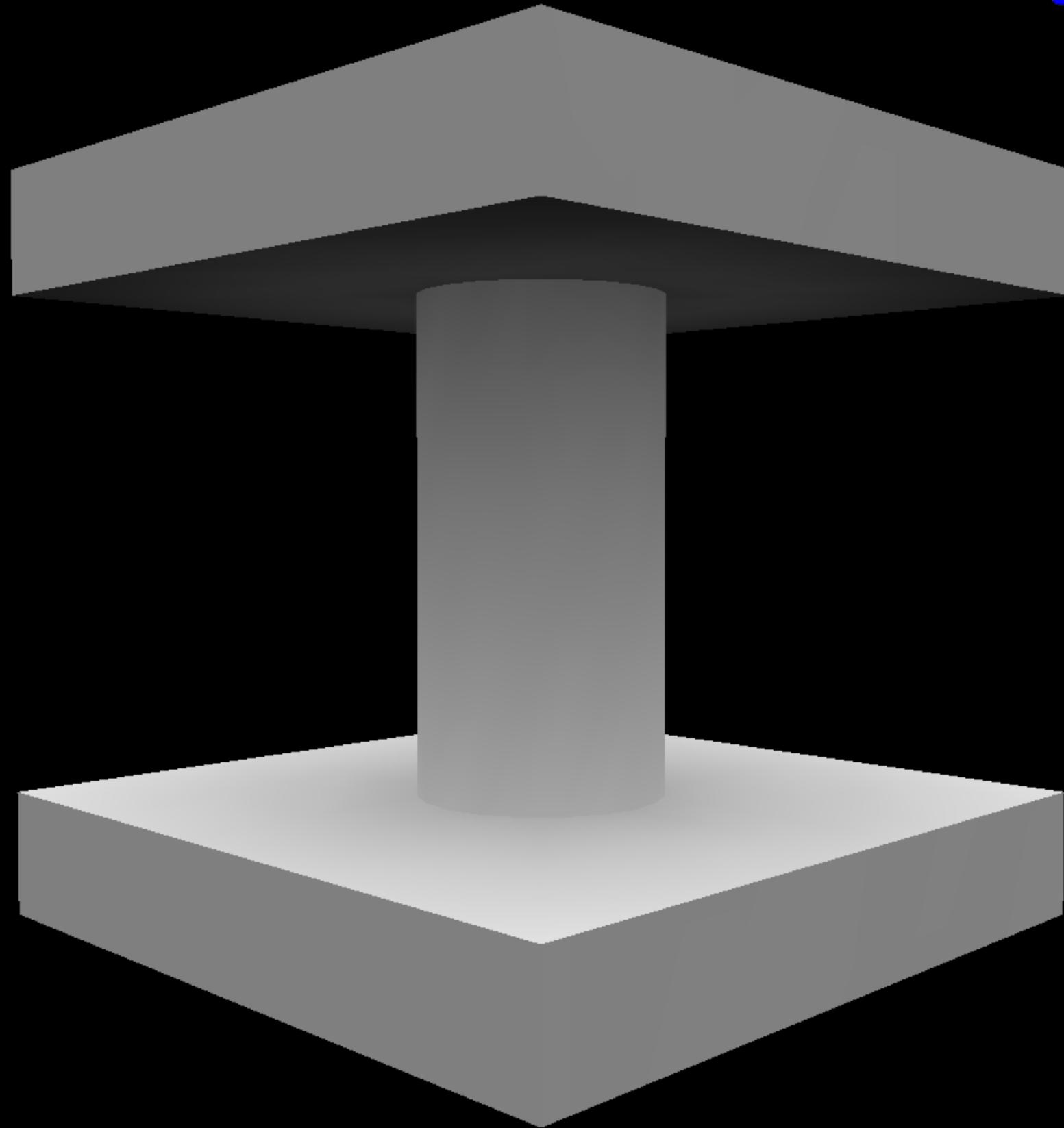
Raytracing



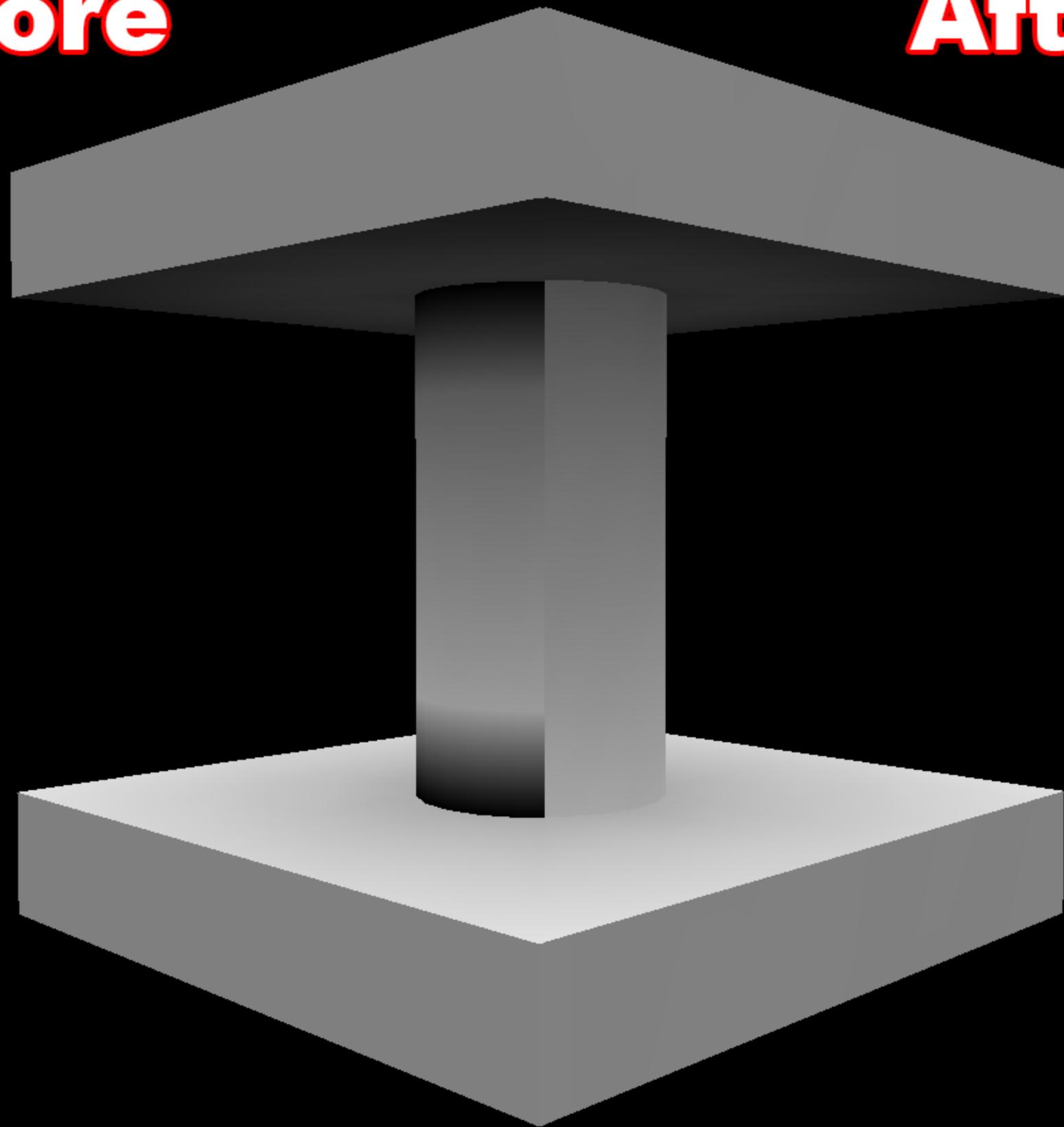
Raytracing



PlayStation®4

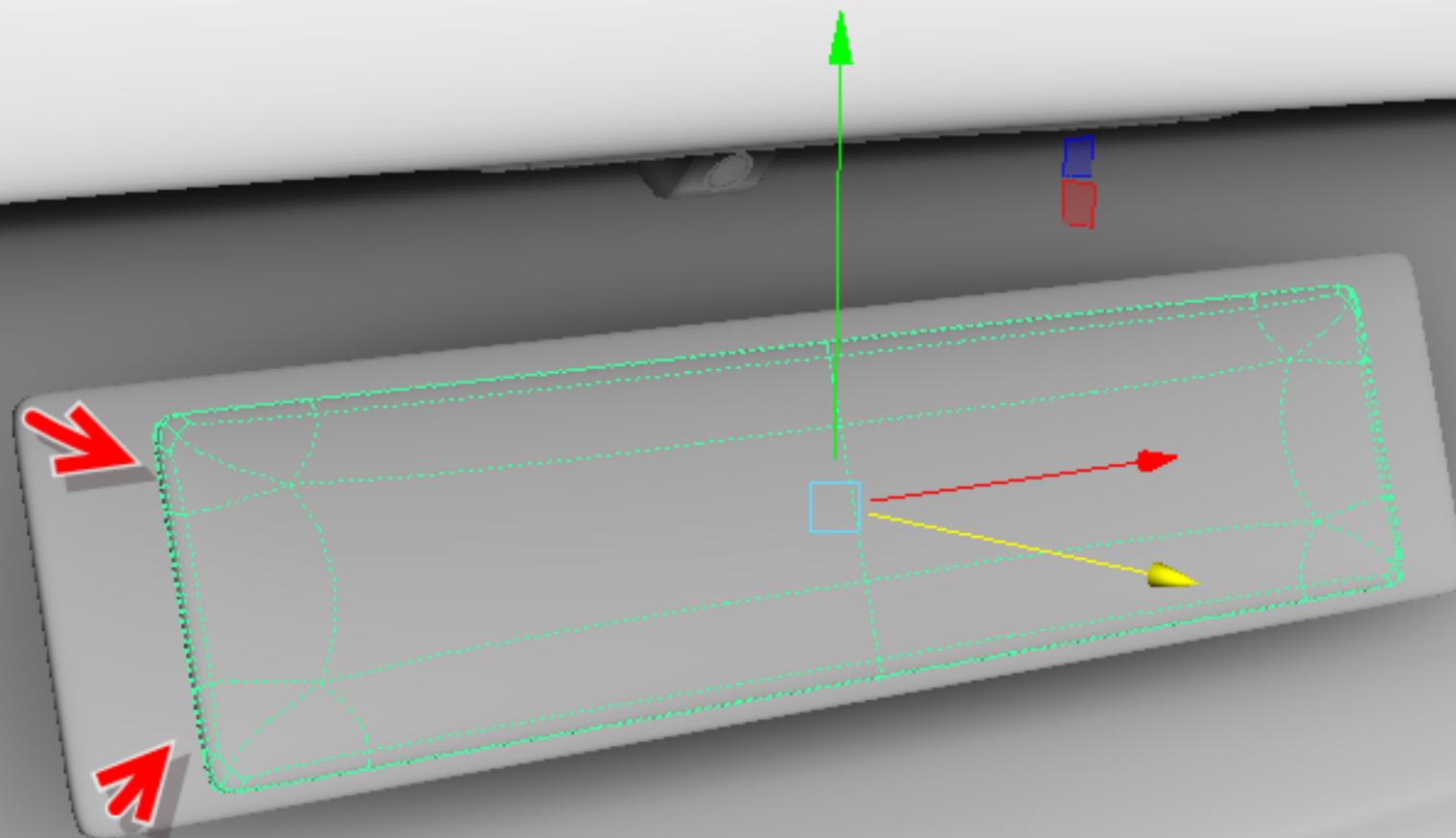


Before



After PlayStation®4

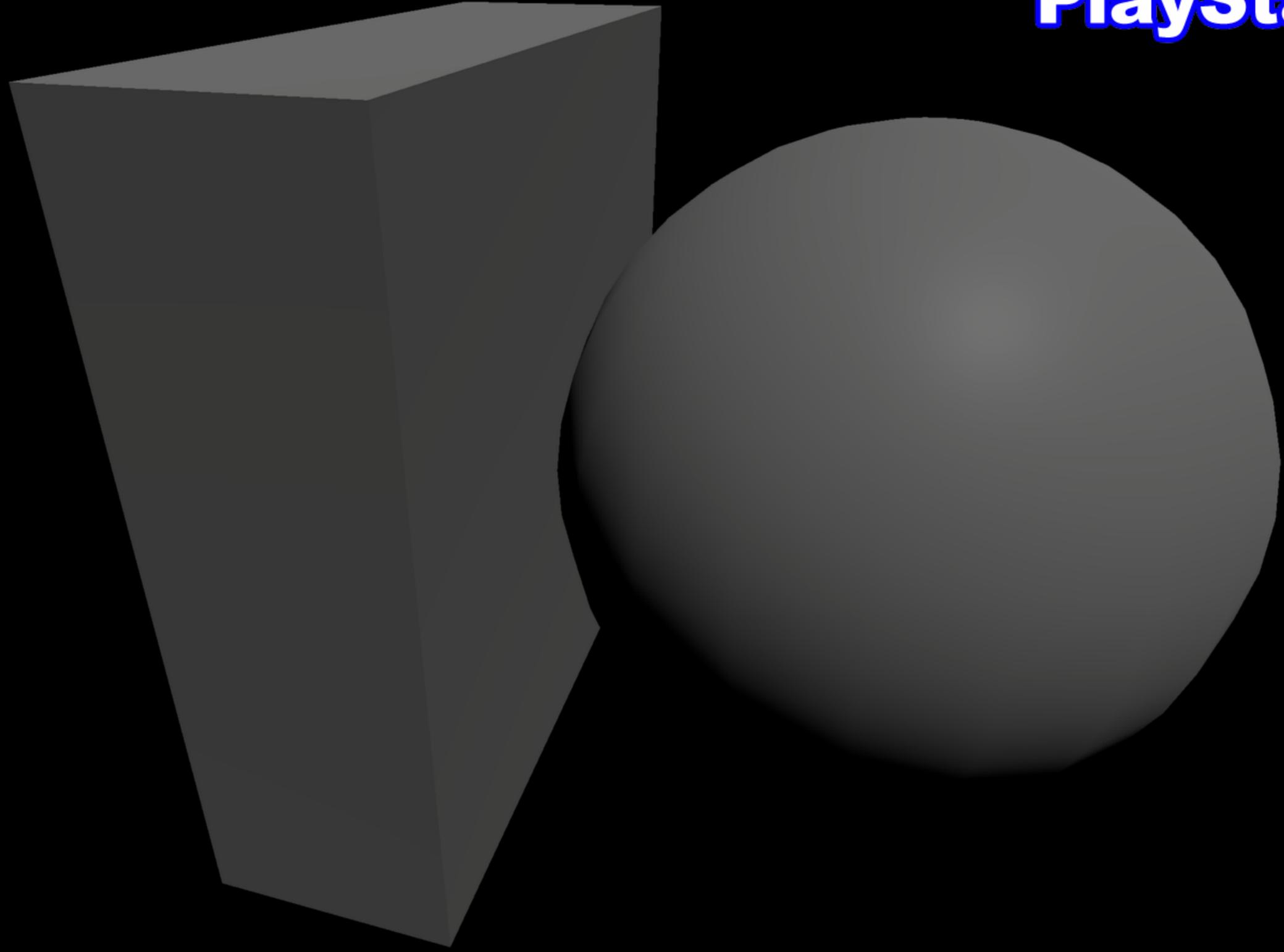
サイズをスケールして小さくしてからベイク



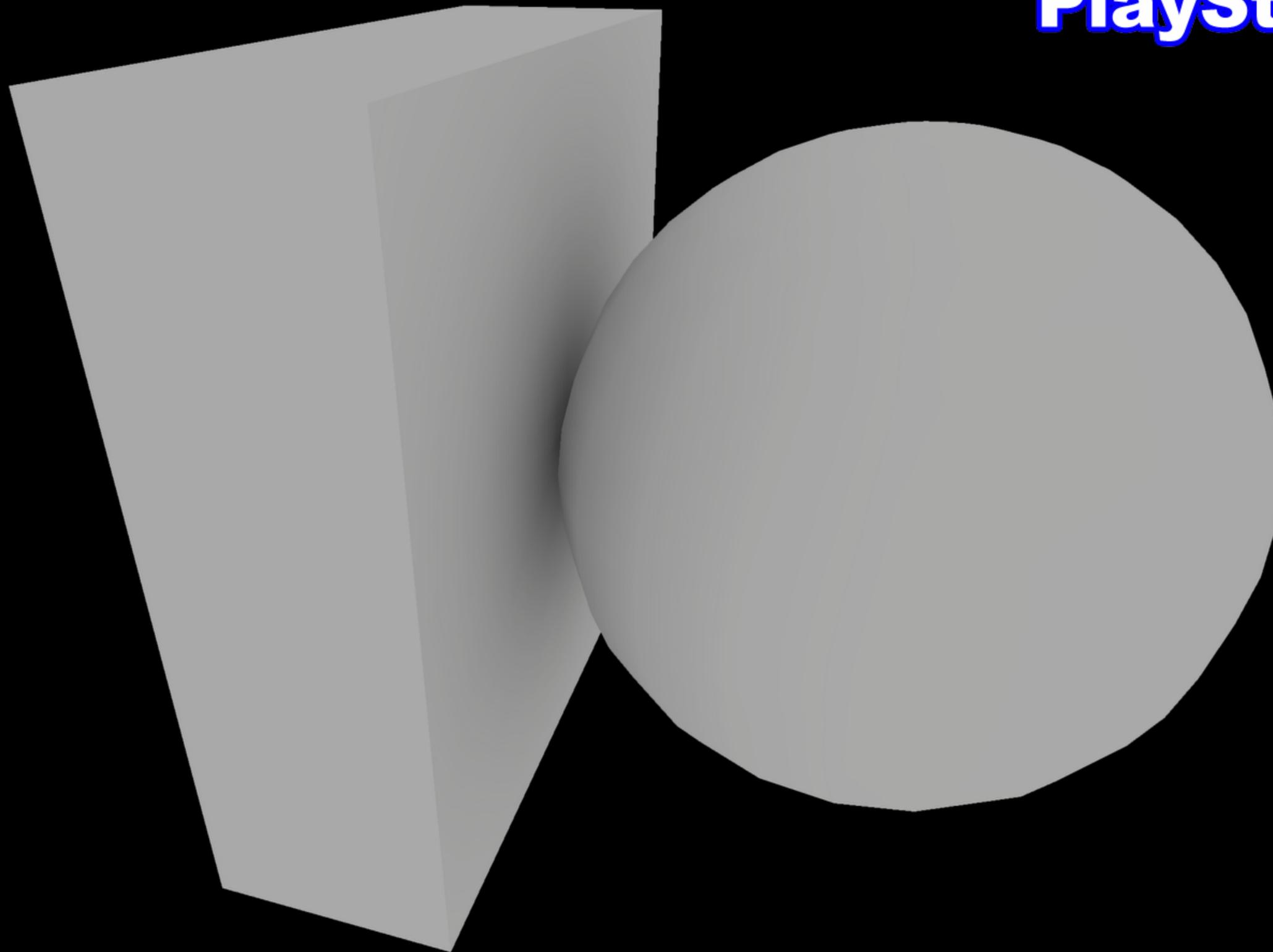
自動化③ ベイクレイヤー

- 大量のオブジェクトが複雑に重なり合う部分を**順番にベイク**します。
- 「**影響を与えるが受けたくない**」のような時に使います。

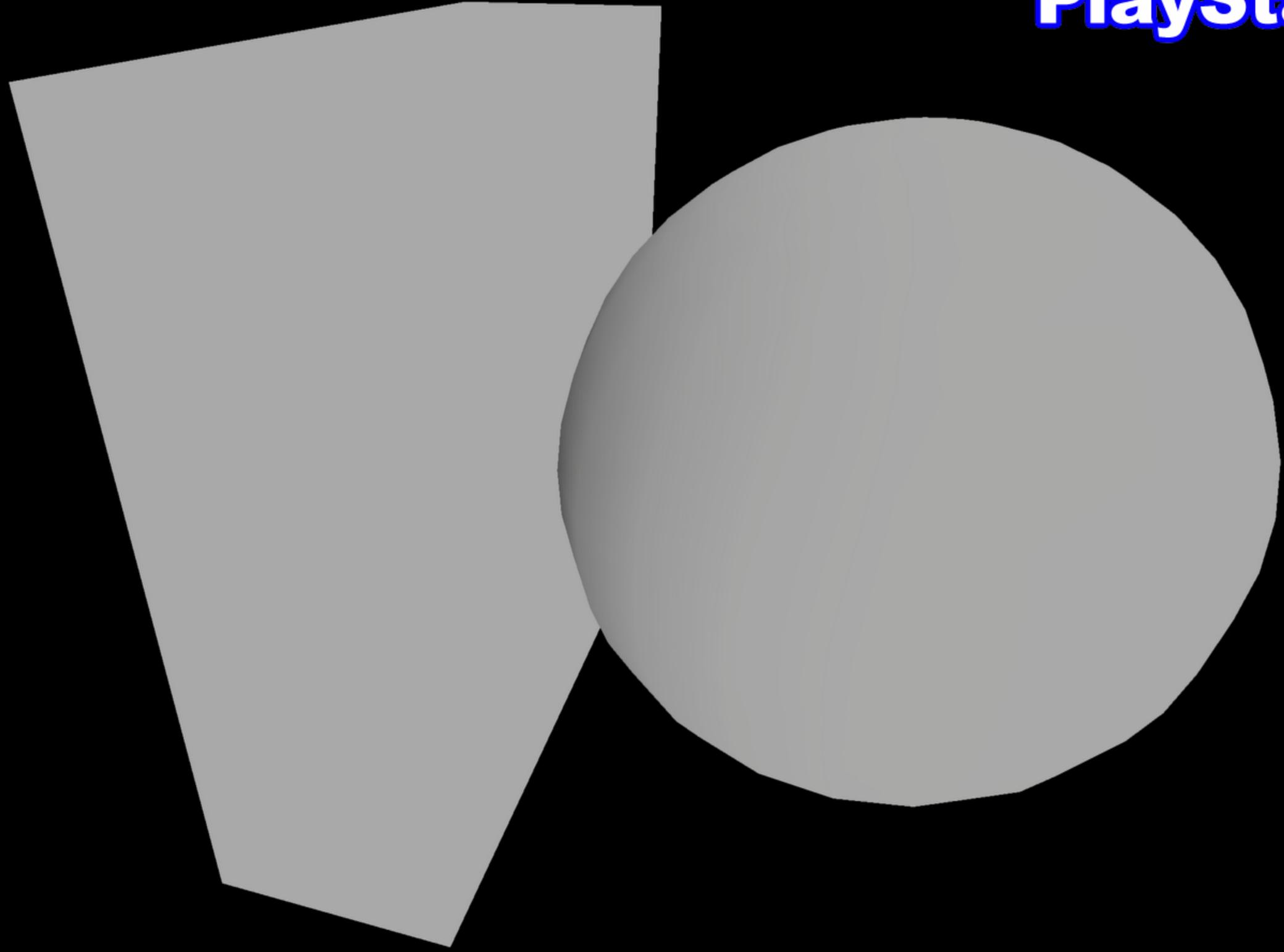
PlayStation®4



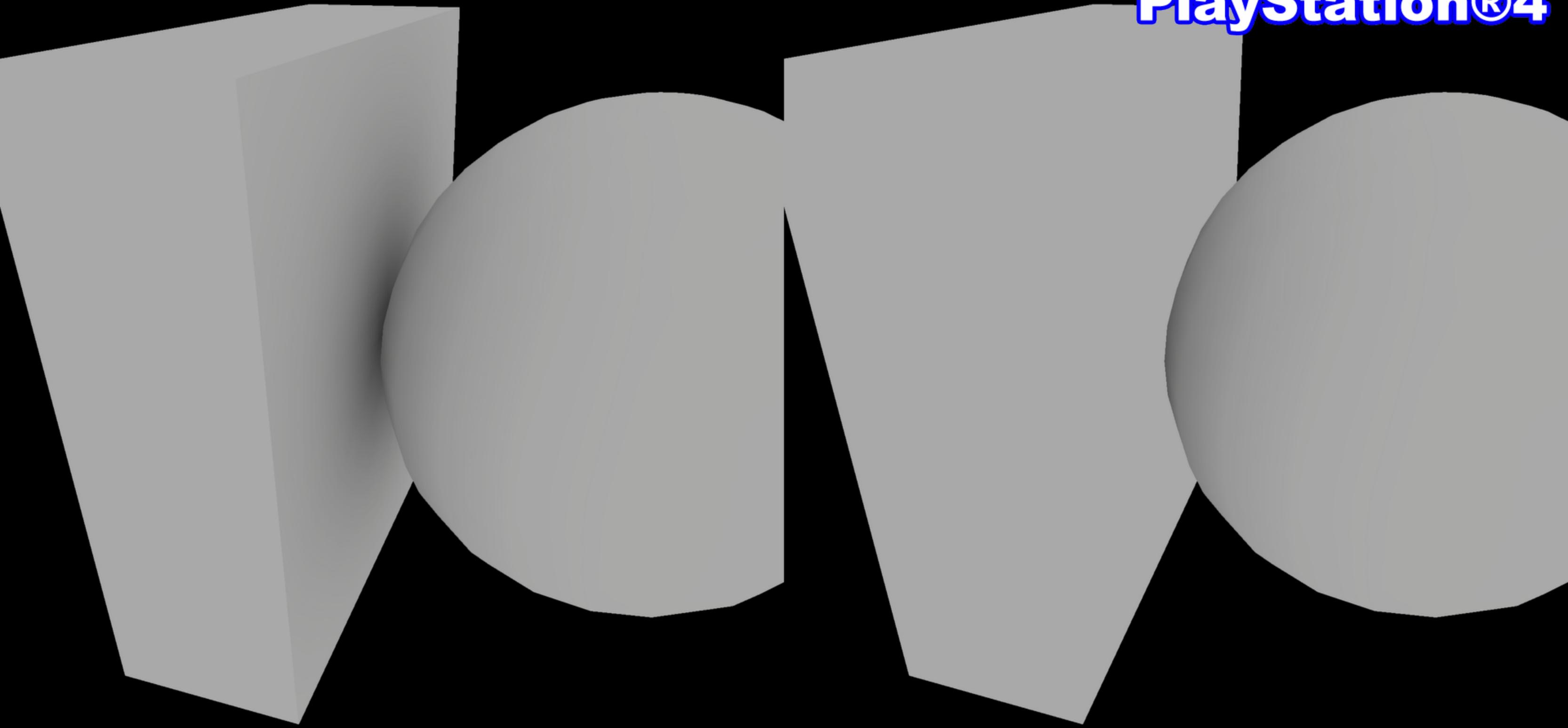
PlayStation®4



PlayStation®4



PlayStation®4



レイヤー-OFF

レイヤー-ON

PlayStation®4

ライトカバーを外してライトの中身をベイク



使われ方④

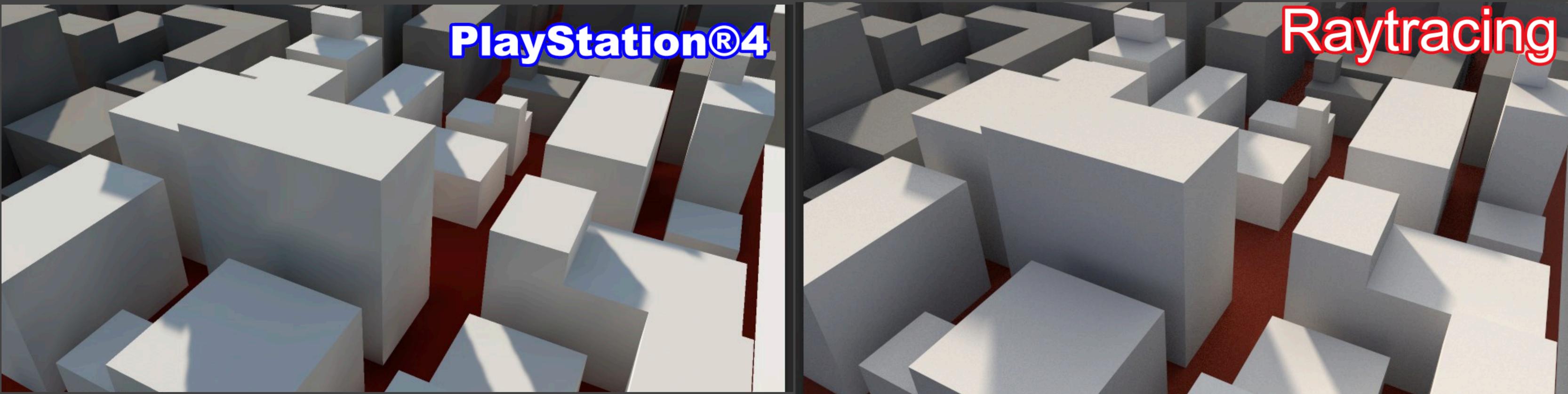
リファレンス

正解との比較

実機とレイトレ

PlayStation®4

Raytracing



実機とレイトレ



実機とレイトレ



リファレンスとして使う

- **レンダリング方程式が解けているMCレイトレーシングと、
実機のレンダリングを比較し、実機用パラメーターの検証、
追い込みに使われました。**
- **結果、ラスタライザでもMCレイトレーシングに迫る
フォトリアリズムを獲得しました。**

この章のまとめ

- どのようなデータがベイクされているかについてお話ししました。
- ベイク以外の用途についてもお話ししました

目次

- はじめに
- レイトレーシングとは
- 弊社でのレイトレーシングの歴史
- 弊社での使われ方
- **事故、デバッグ、対策**
- 結び

事故、デバッグ、対策

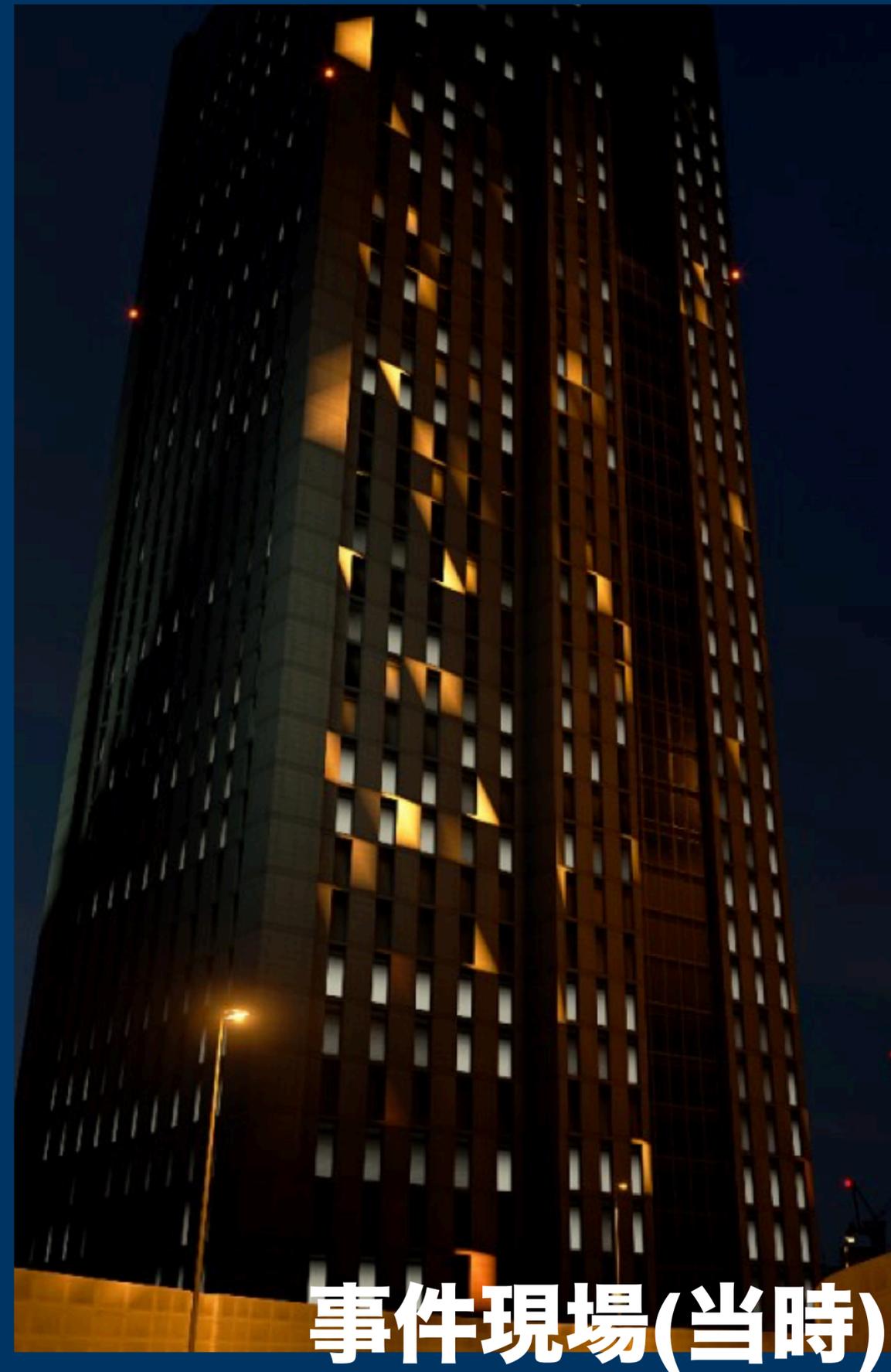
うまくいっていないことをどうしたのか

謎のシミ事件

「とつぜんビルに明るいシミができてます！」

「なんとかしてください！」

PlayStation®4



事件現場(当時)

謎のシミ

- 収束しきっていない…?
- とりあえず**サンプルの数(イテレーション数)**を**1万倍**にしてみました。

PlayStation®4

結果は変わりませんでした

謎のシミ

- 謎はわからないままになっていましたが、あるときアーティストがあることをすると綺麗になることを見つけました。

PlayStation®4



ビルにライトを当て、その二次反射に照らされています

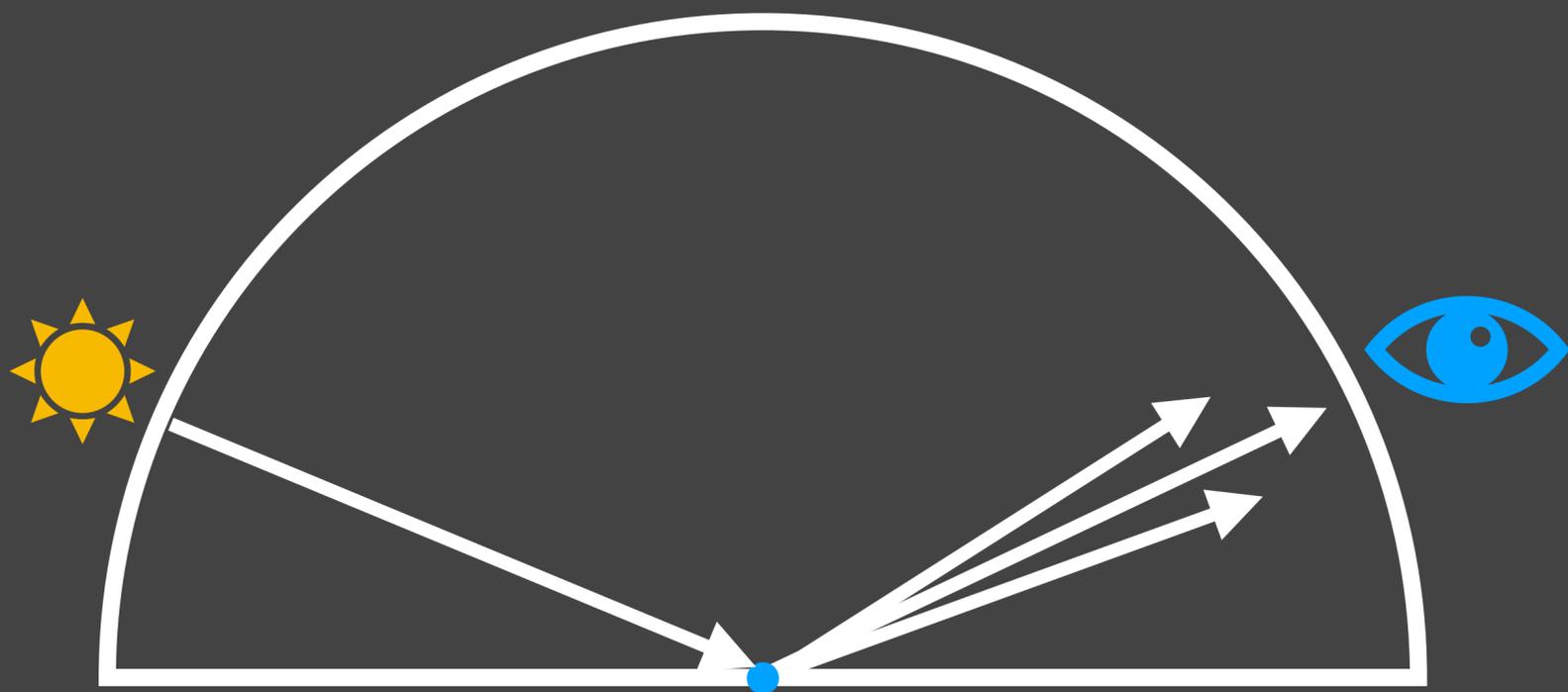
PlayStation®4



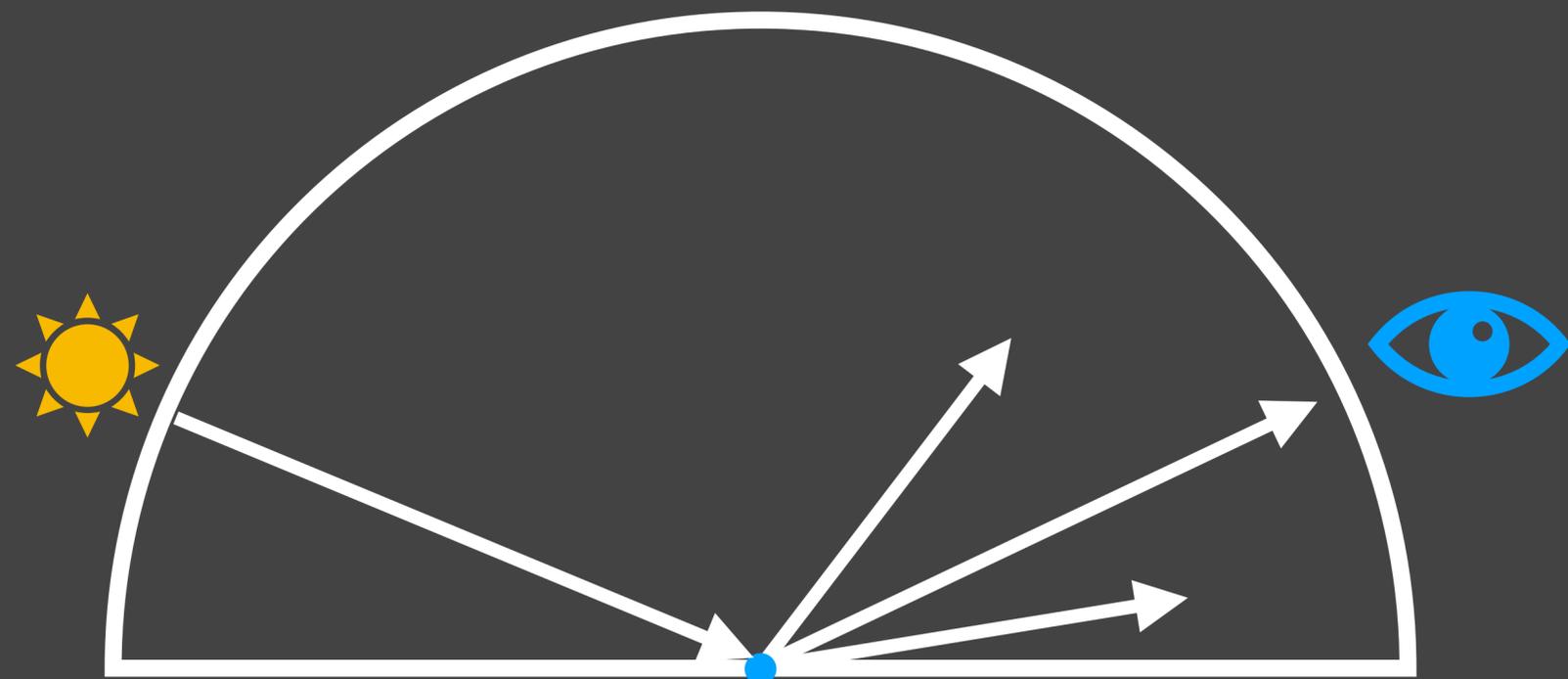
壁のガラスについている法線マップを取り除きました

謎のシミ

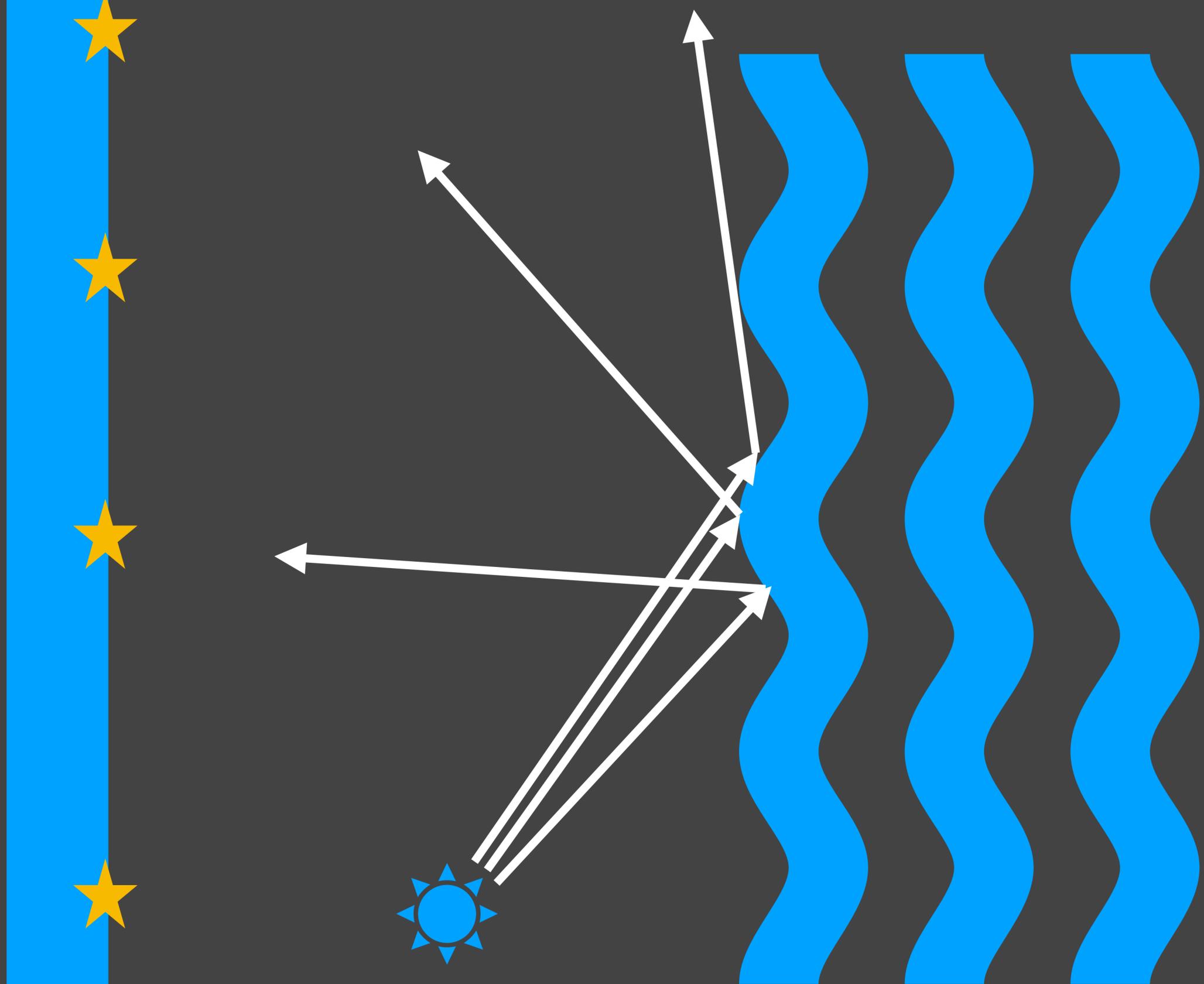
- 犯人は**ガラス**に適用される**法線マップの有無**でした。
- しかしなぜでしょうか？
- ビルの窓ガラスに鋭いスペキュラーがあるのがヒントでした。



鋭いハイライト
(狭いローブ)



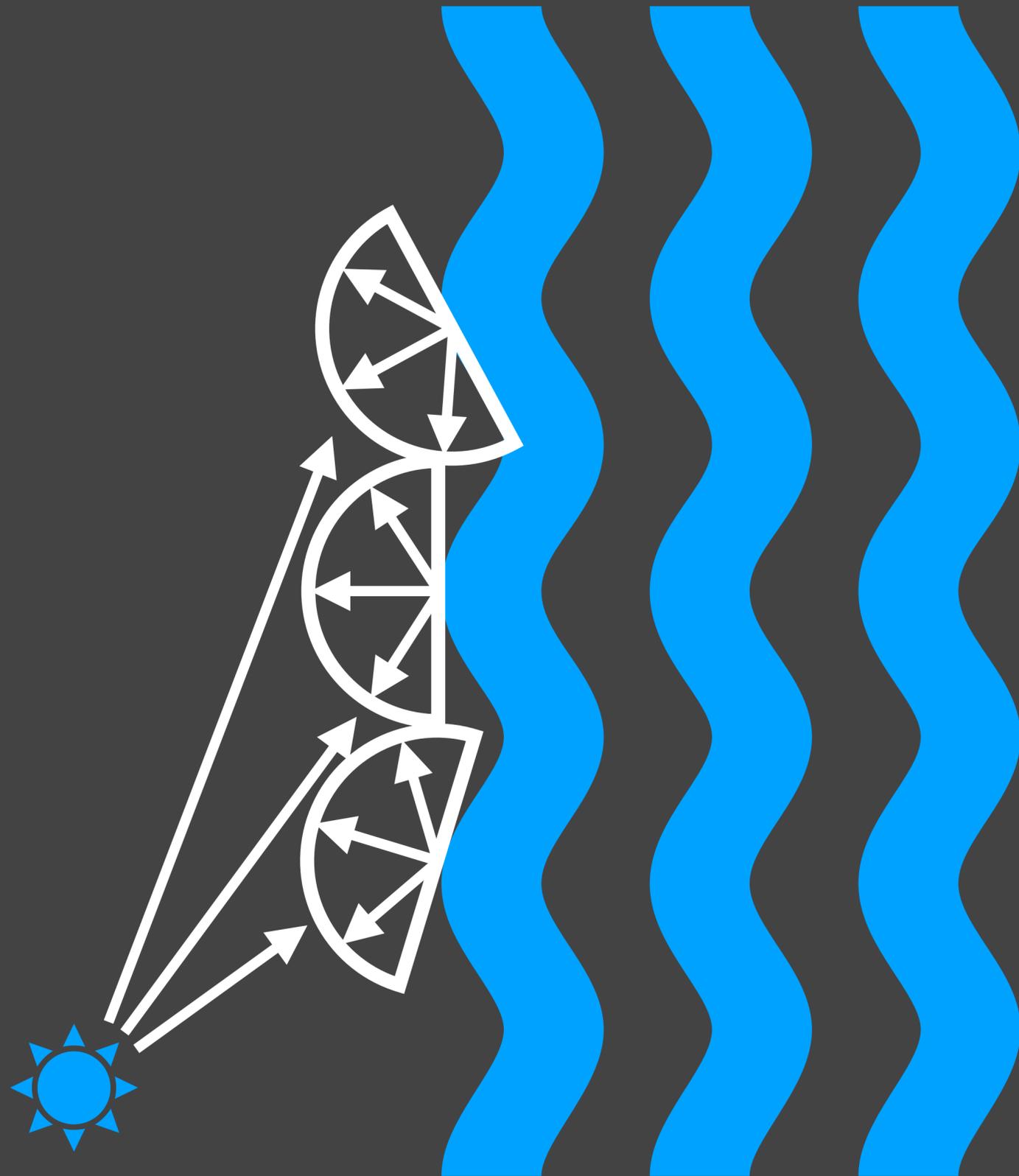
鈍いハイライト
(広いローブ)



細かいローブ+法線マップ=高輝度が飛び飛びに

極めて細かいローブ

- 問題は**ビルの窓**についている**スペキュラーの鋭く(ローブが狭く)**そこに**法線マップ**がついていたからでした。
- **スペキュラーを鈍く(ローブの太く、ラフネスを高く)**すれば…?



広いローブ+法線マップ=安定

PlayStation®4

事件現場(解決後)

犯人



ガラス+NormalMap

正しくはないが…使える

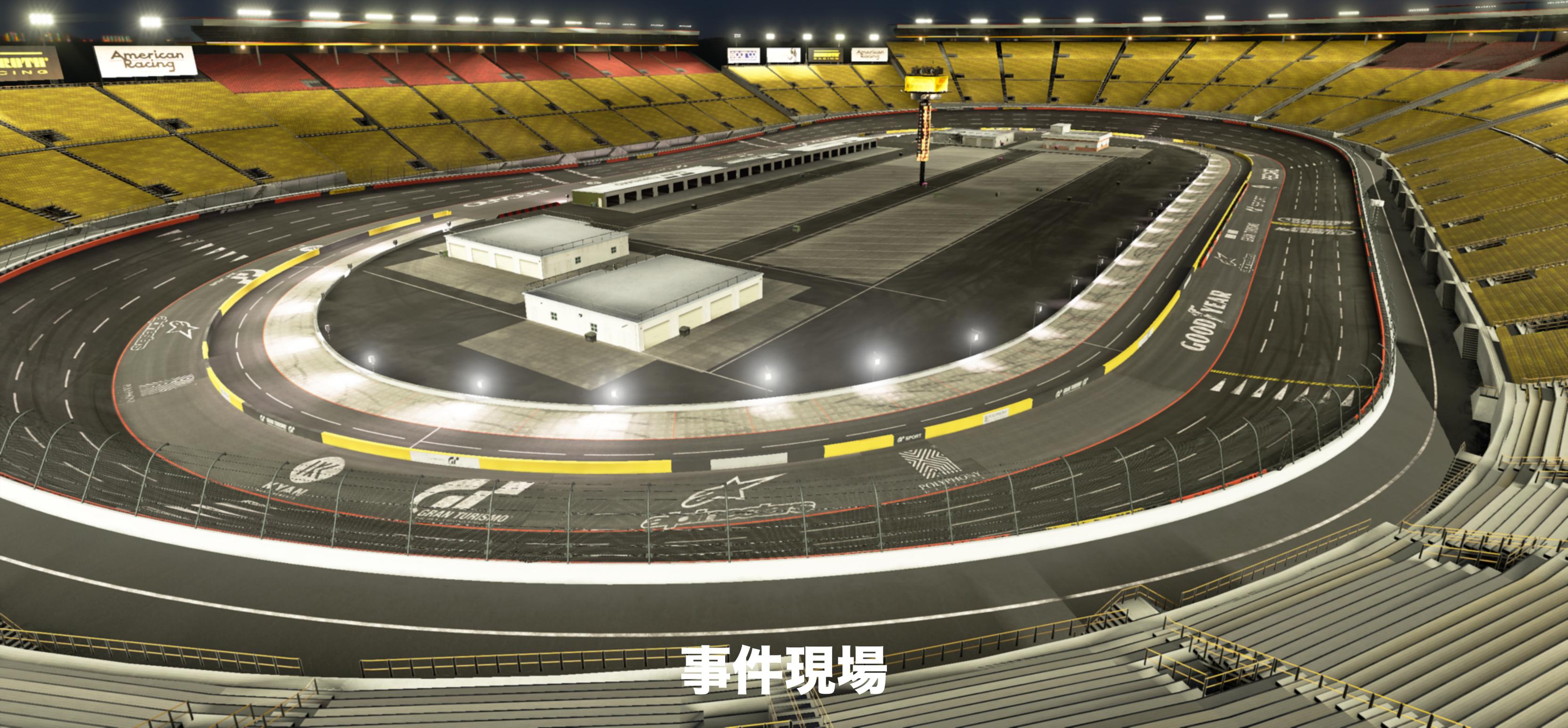
- **ベイク時のラフネスの下限値**という設定項目を用意しました。
- このような**ベイク専用のハック**は**結果を変えて**しまいます。
- しかし**現実のアセット**に対応するには工夫も必要です。

ライト激重事件

「ベイクが激重です！」

「なんとかしてください！」

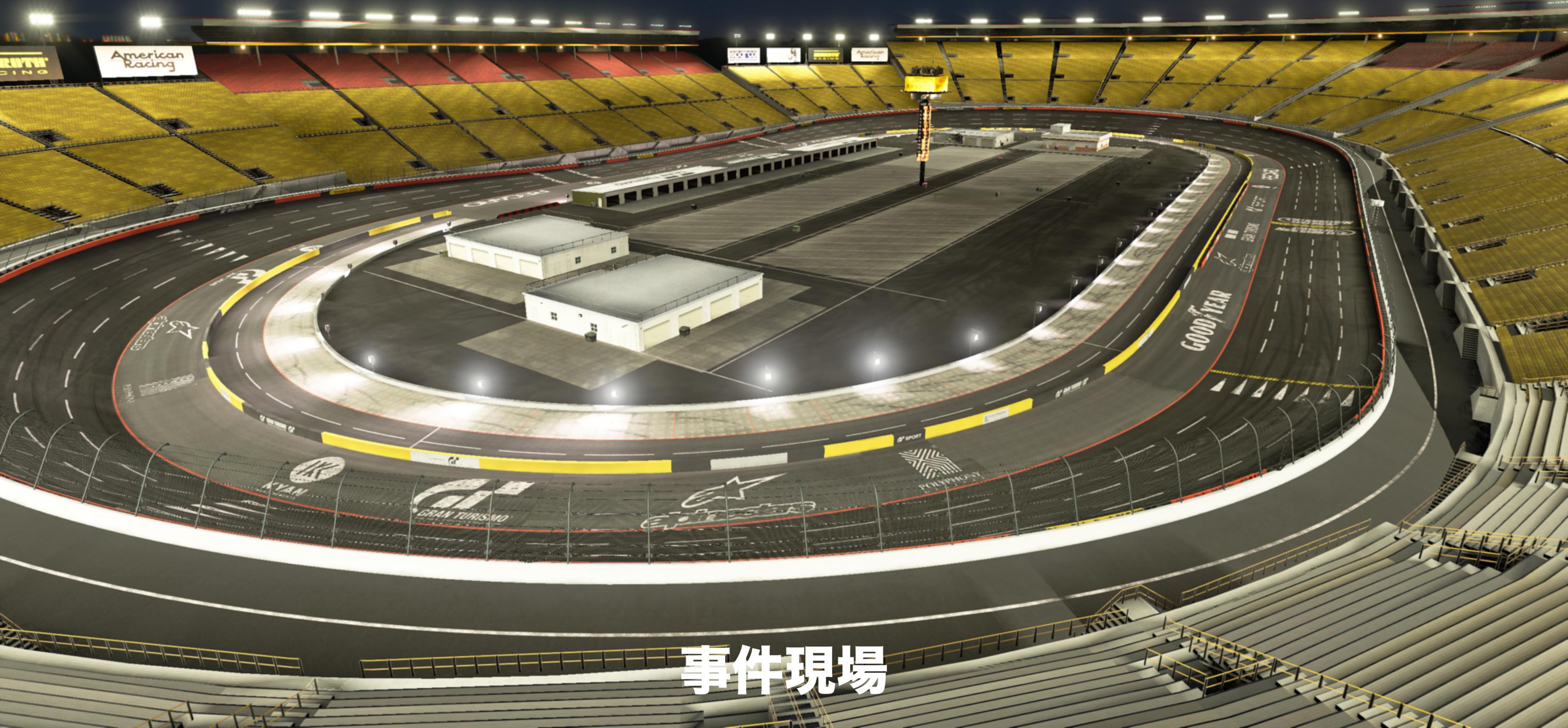
PlayStation®4



事件現場

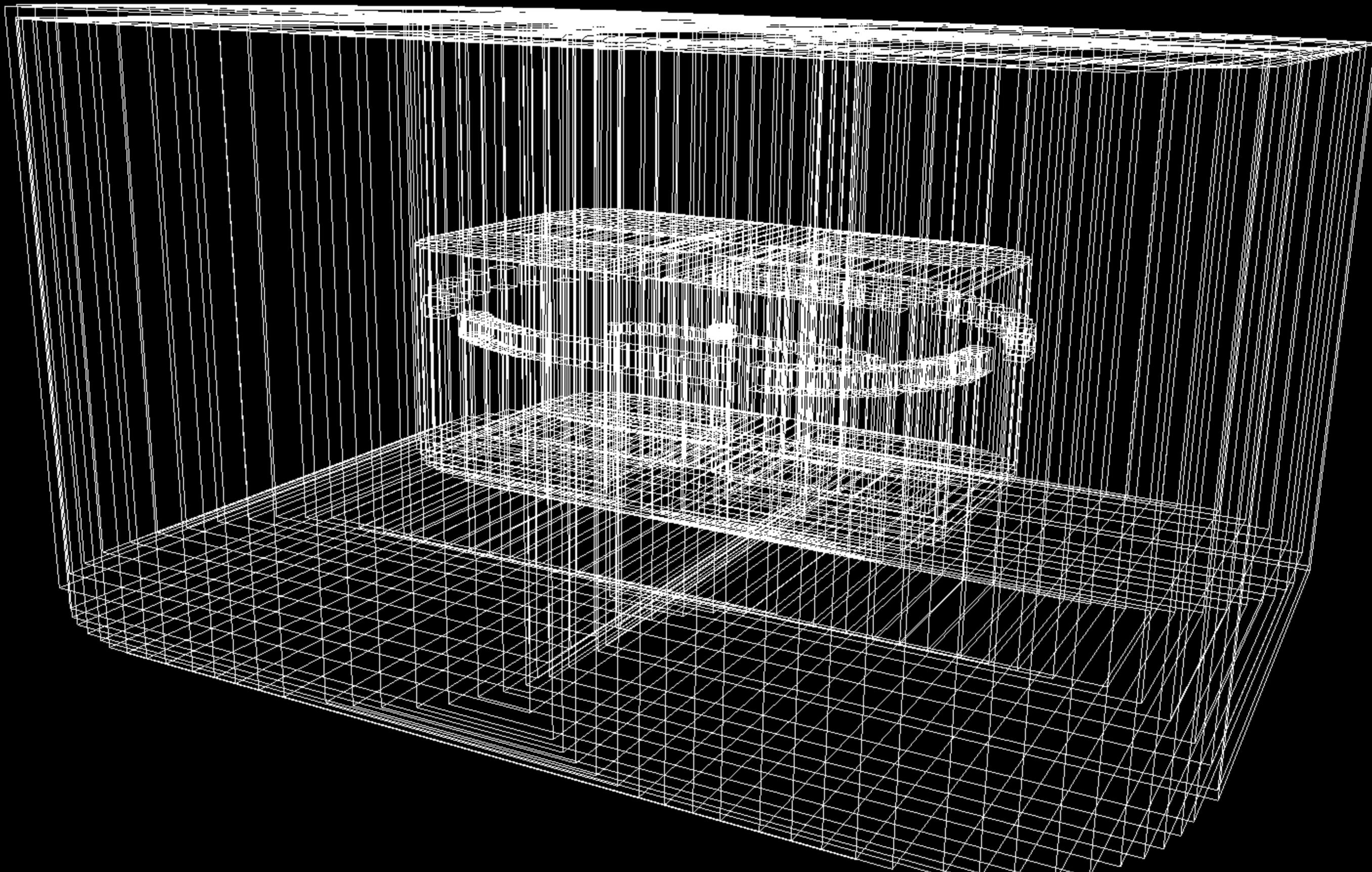
PlayStation®4

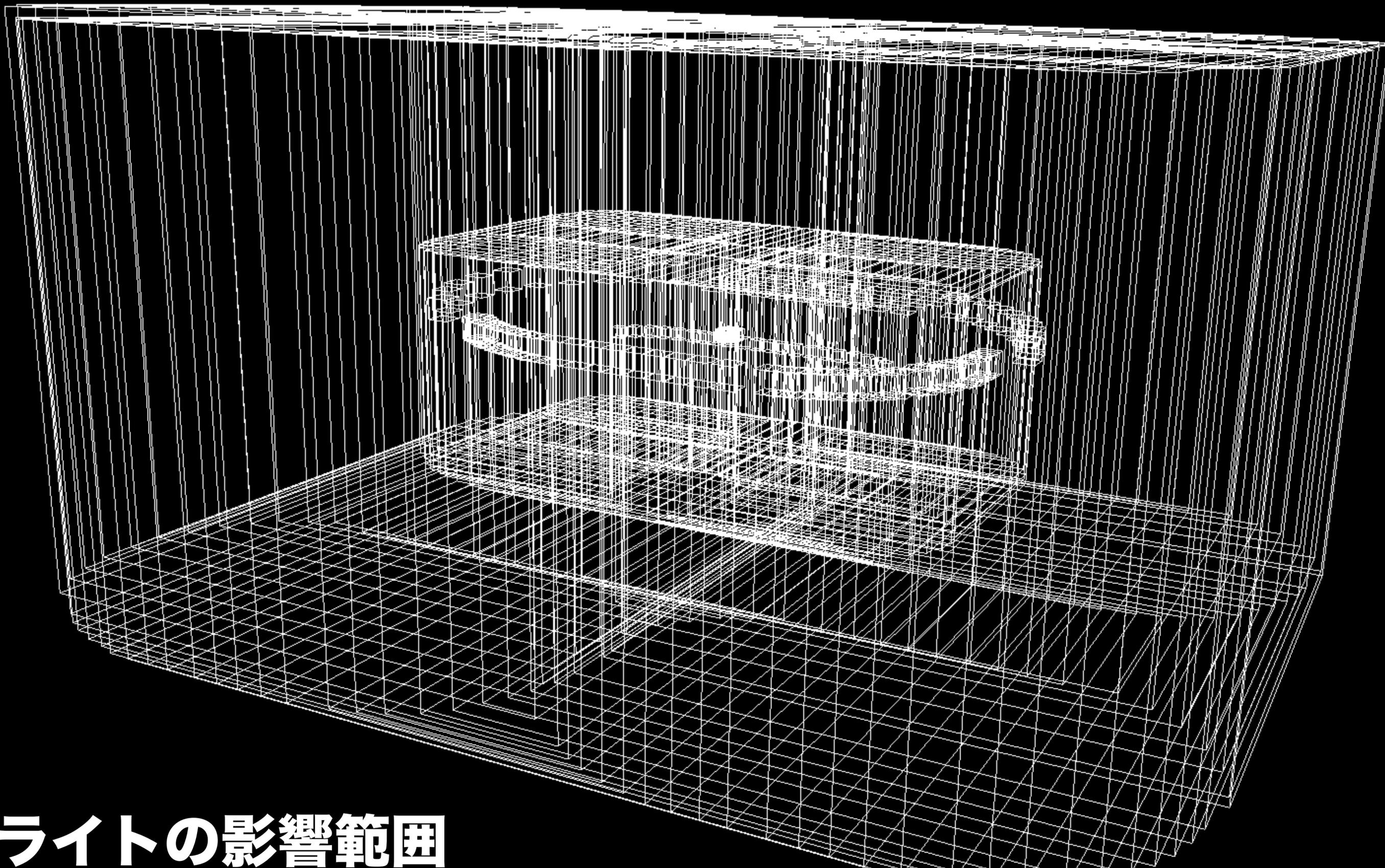
大量のライト！



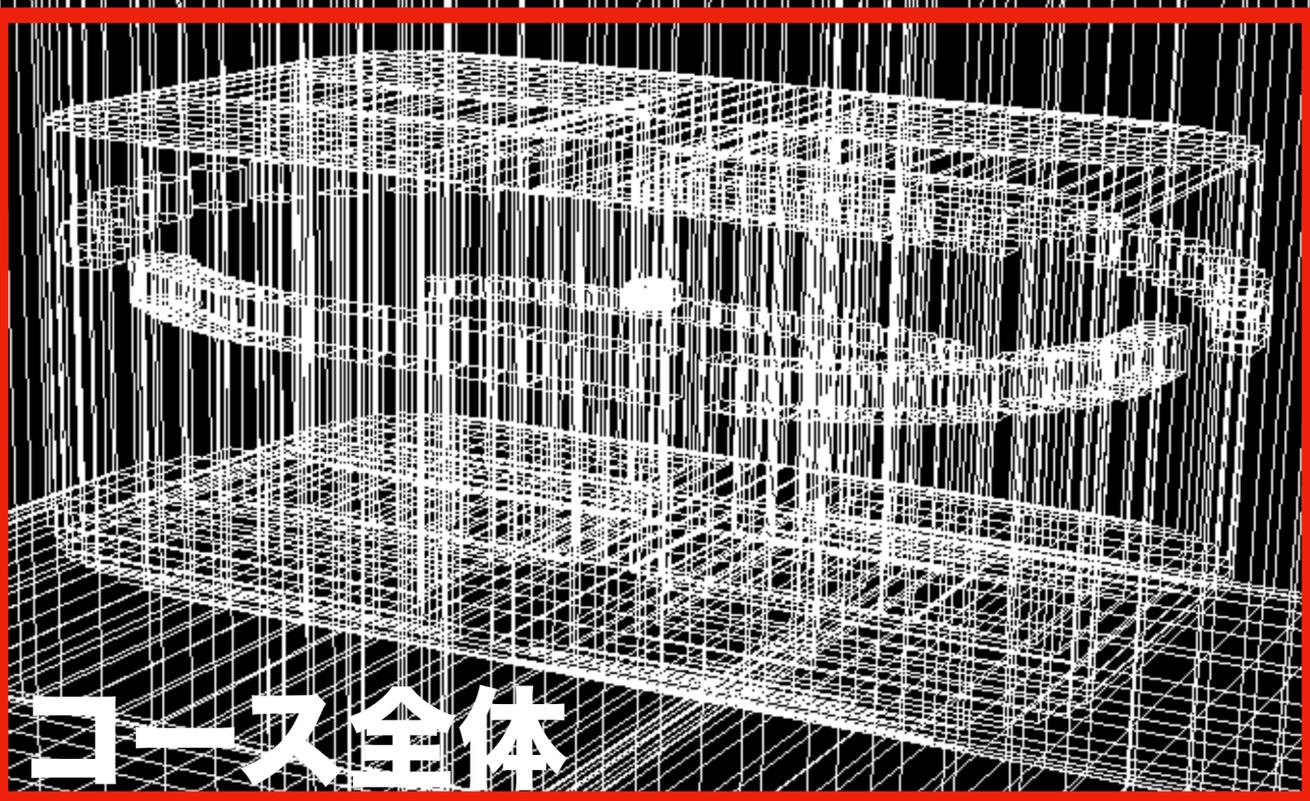
事件現場

ライトの影響範囲を可視化



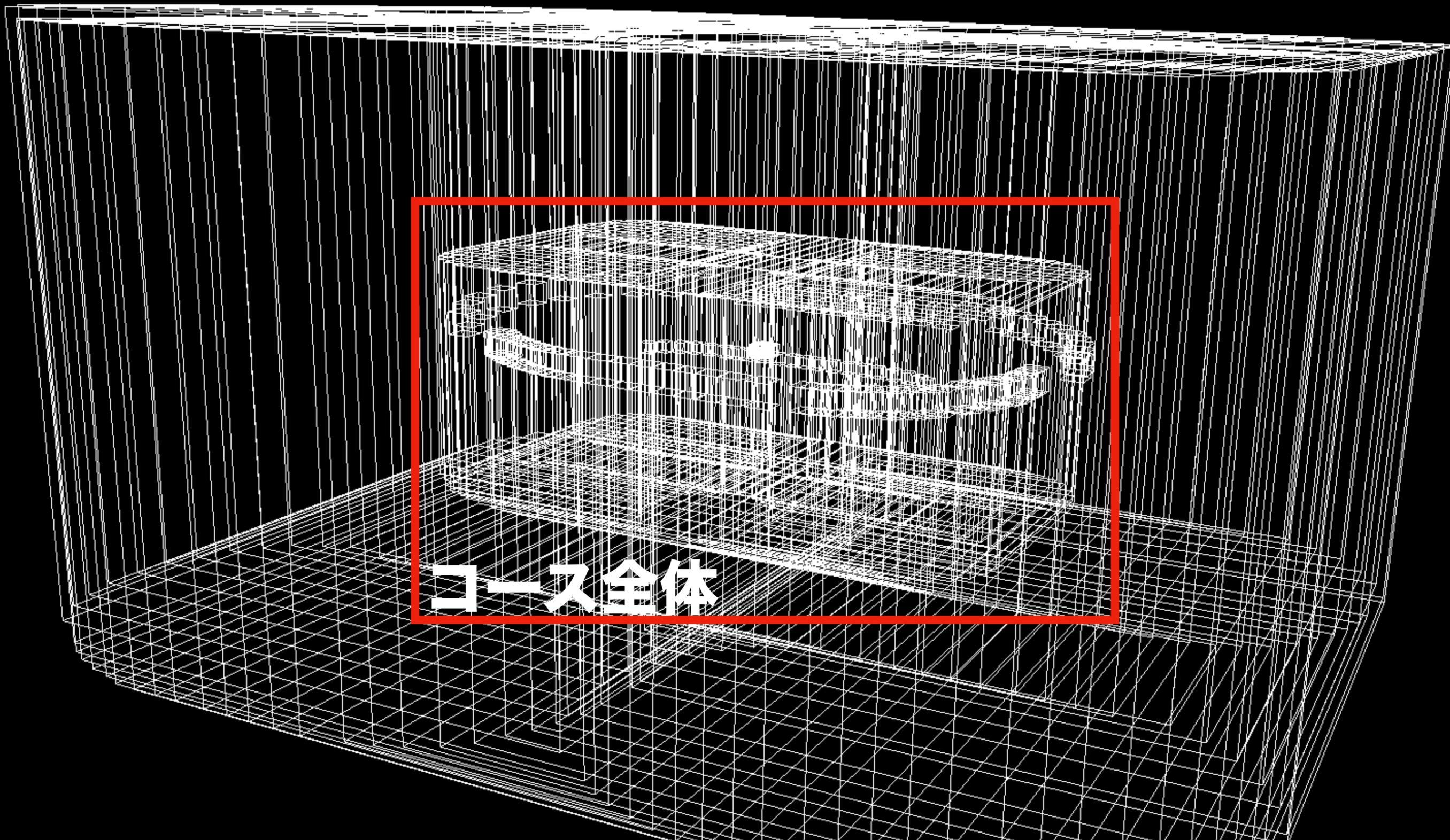


ライトの影響範囲



コース全体

ライトの影響範囲



コース全体

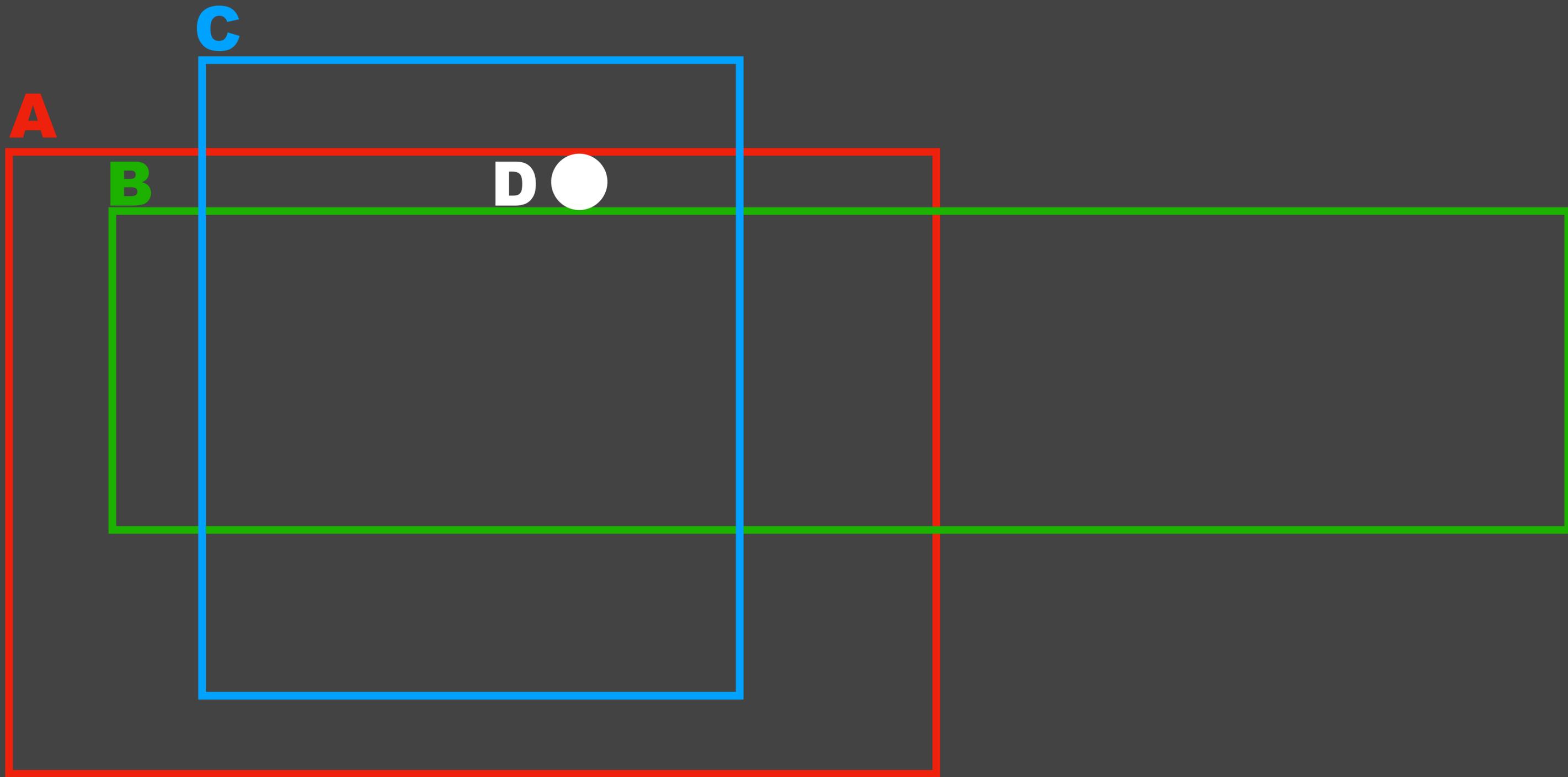
ライトの影響範囲

206個のライト

ライトのサンプル

- **広範囲に影響**を与えるライトが**大量**に配置されていました。
- 当初は**全てのシェーディングで全てのライト**を見ていました。
- 「**影響度範囲付き**」のライトであったのでそれを利用して**BVH(Bounding volume hierarchy)で枝刈り**を試してみました。
- BVHとはボリウムを階層化する空間データ構造の一つで「**ある点を含むボリウムの列挙**」のような操作を**高速に実行**できます。

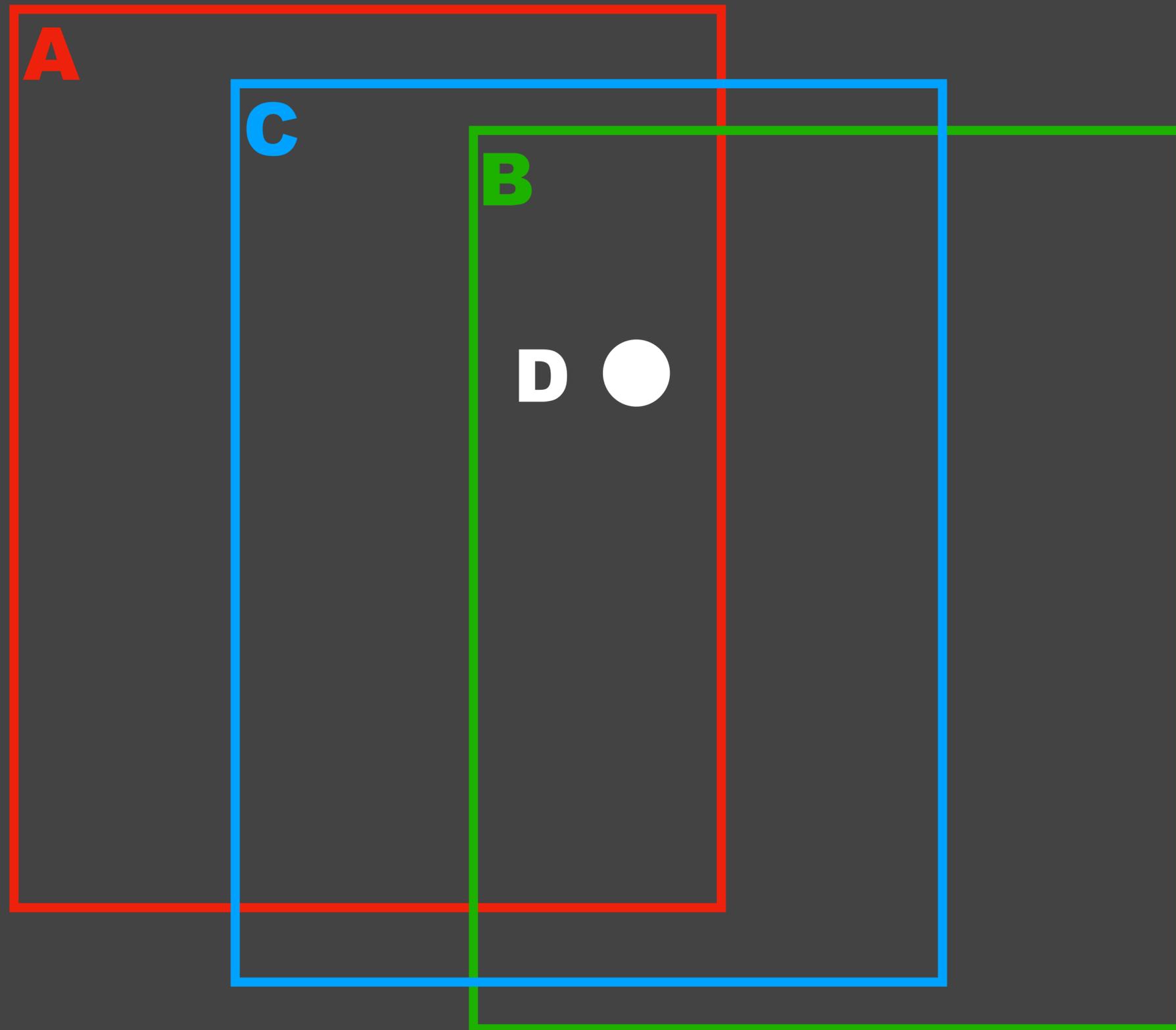




Dを含むボリ्यूームは？ → **A**と**C**

ライトのサンプル

- それでもまだまだ低速でした。
- さらにライトを**確率的に選択**するようにしました。
- 三つのライトから一つを選択した場合、輝度が $1/3$ になってしまうので結果を3倍しました。
- 確率的な輝度の**期待値は変化しません**。
しかし輝度の**分散は上がります(ムラが出やすい)**。



DがAとBとCに含まれる。AとBとCから一つを選択し3倍。

ライトのサンプル

- コンバート時間が**19023秒** → **970秒 (x20)** に短縮されました。
- 同じイテレーション数では、よりムラはできやすくなりましたが、許容範囲内でした。

よりスマートな方法

- 多光源を効率的に扱う発展的な手法に

「**確率的ライトカリング**」や「**Lightcuts**」があります。

“確率的ライトカリング -基礎から動的コースティックスの描画まで-”

<https://2018.cedec.cesa.or.jp/session/detail/s5ab3333907663>

明日8月24日(金) 11:20 ~ 12:20

“確率的ライトカリング -理論と実装-“ (CEDEC 2016)

https://cedil.cesa.or.jp/cedil_sessions/view/1531

- 今年のHPGで発表されたこちらにも注目です。

“Importance Sampling of Many Lights With Adaptive Tree Splitting”

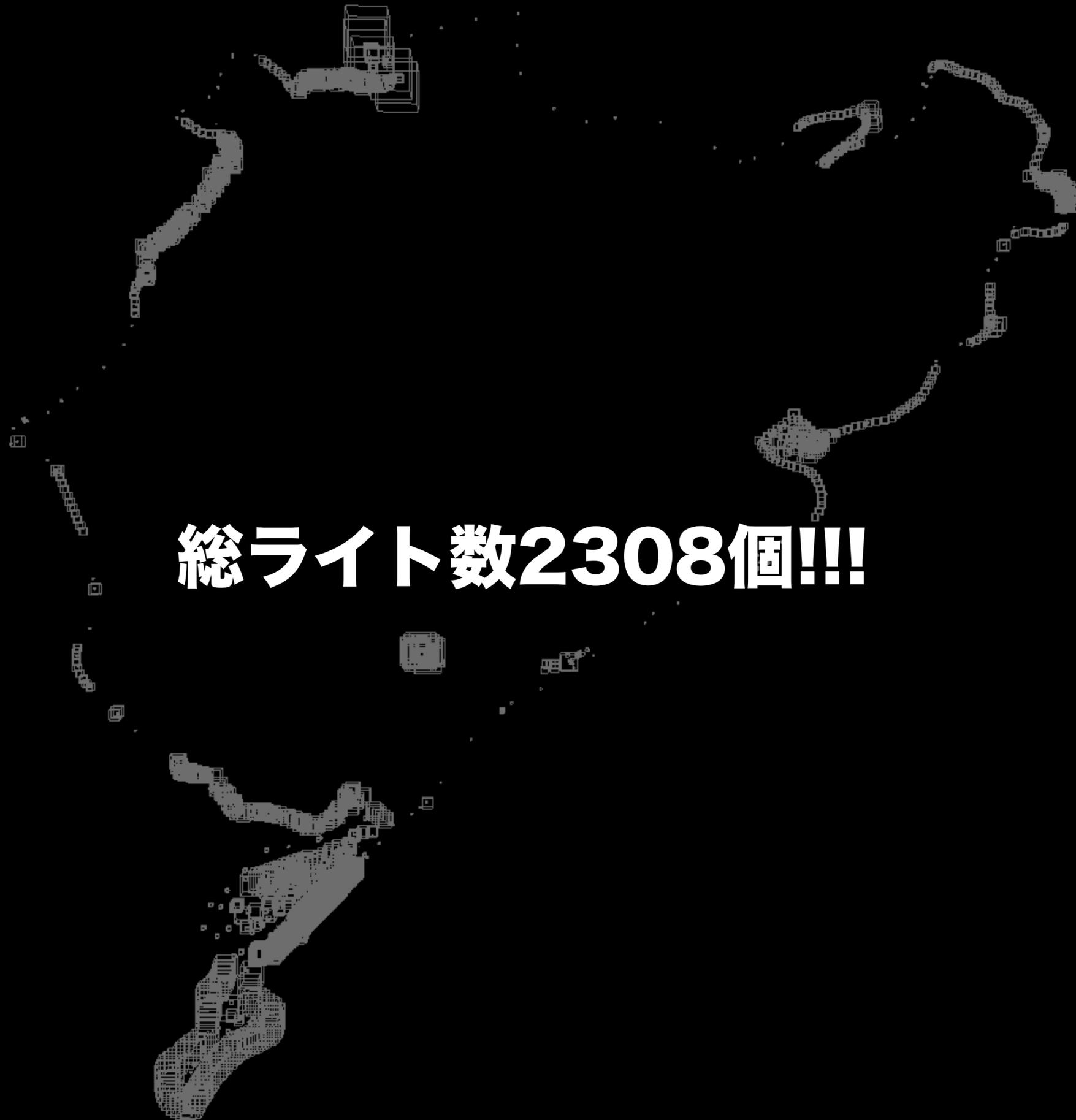
<https://fpsunflower.github.io/ckulla/>

ところで、あのコースは
どうだったんだらう…

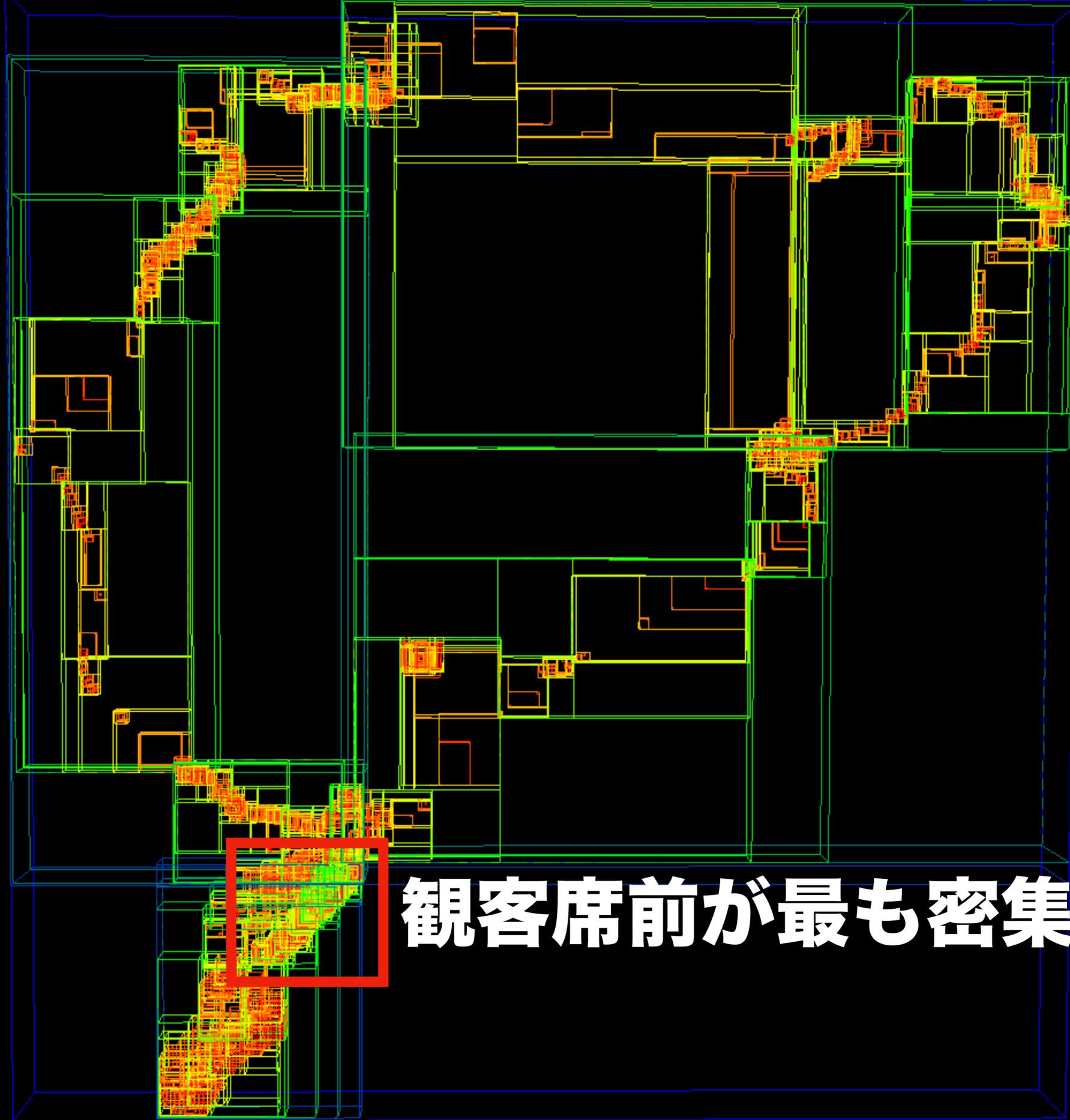


Nürburgring



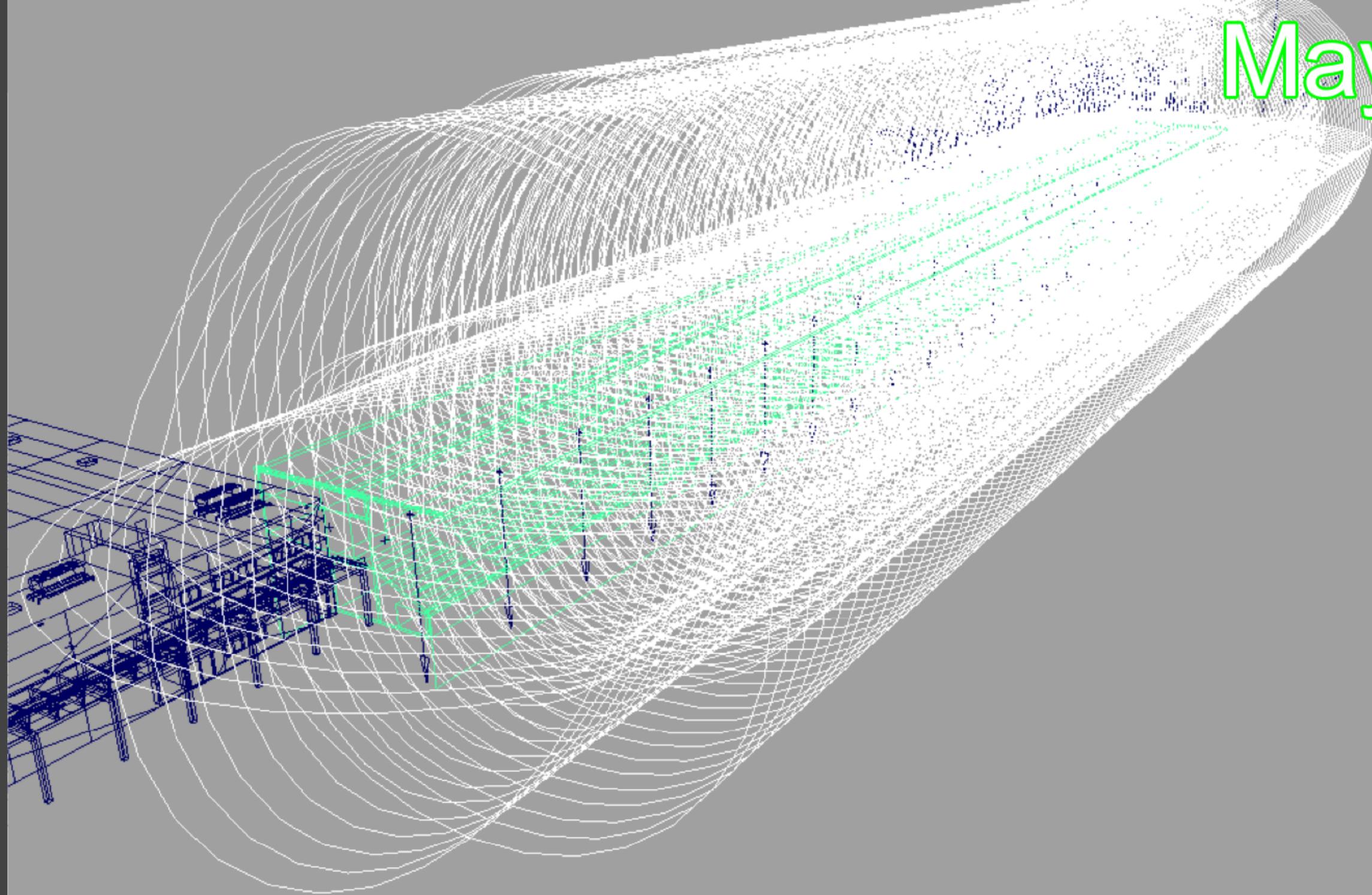


総ライト数2308個!!!

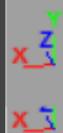


観客席前が最も密集

Maya



観客席の恐ろしい密度の光源



PlayStation®4

結果

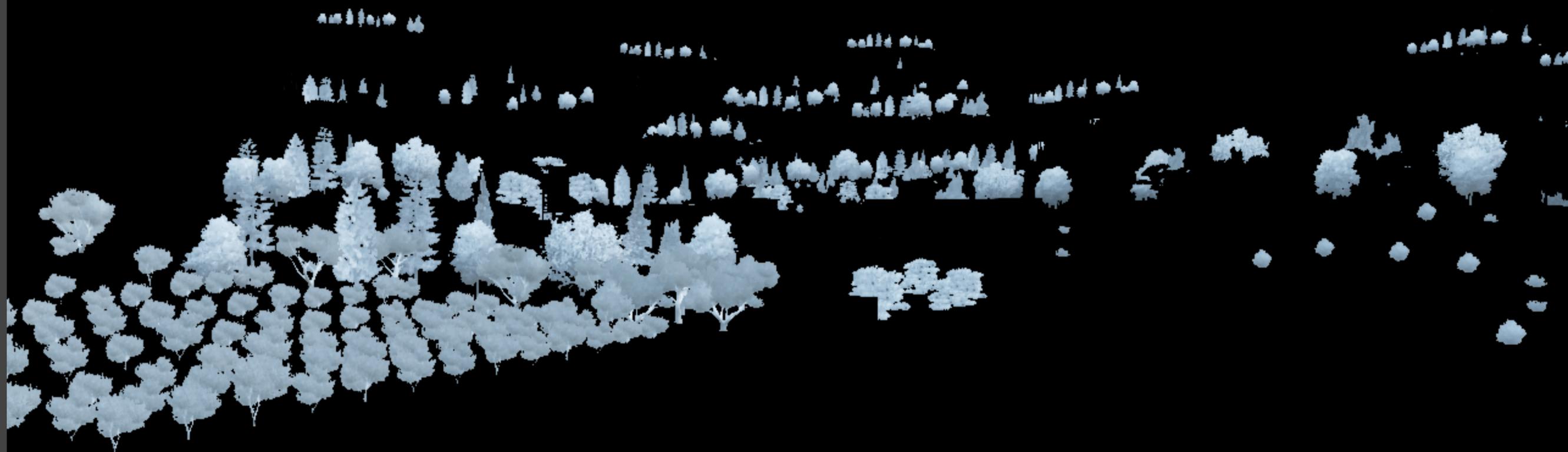
高密度な光源+短時間でもクオリティができるように

真っ黒事件

「とつぜんGIが黒くなりました！」

「なんとかしてください！」

PlayStation®4



事件現場(当時)

調査

- **printf()デバッグなどしてみましたがさっぱりわかりませんでした。
レイが何かあったって外に出れていない...?**

可視化してみました

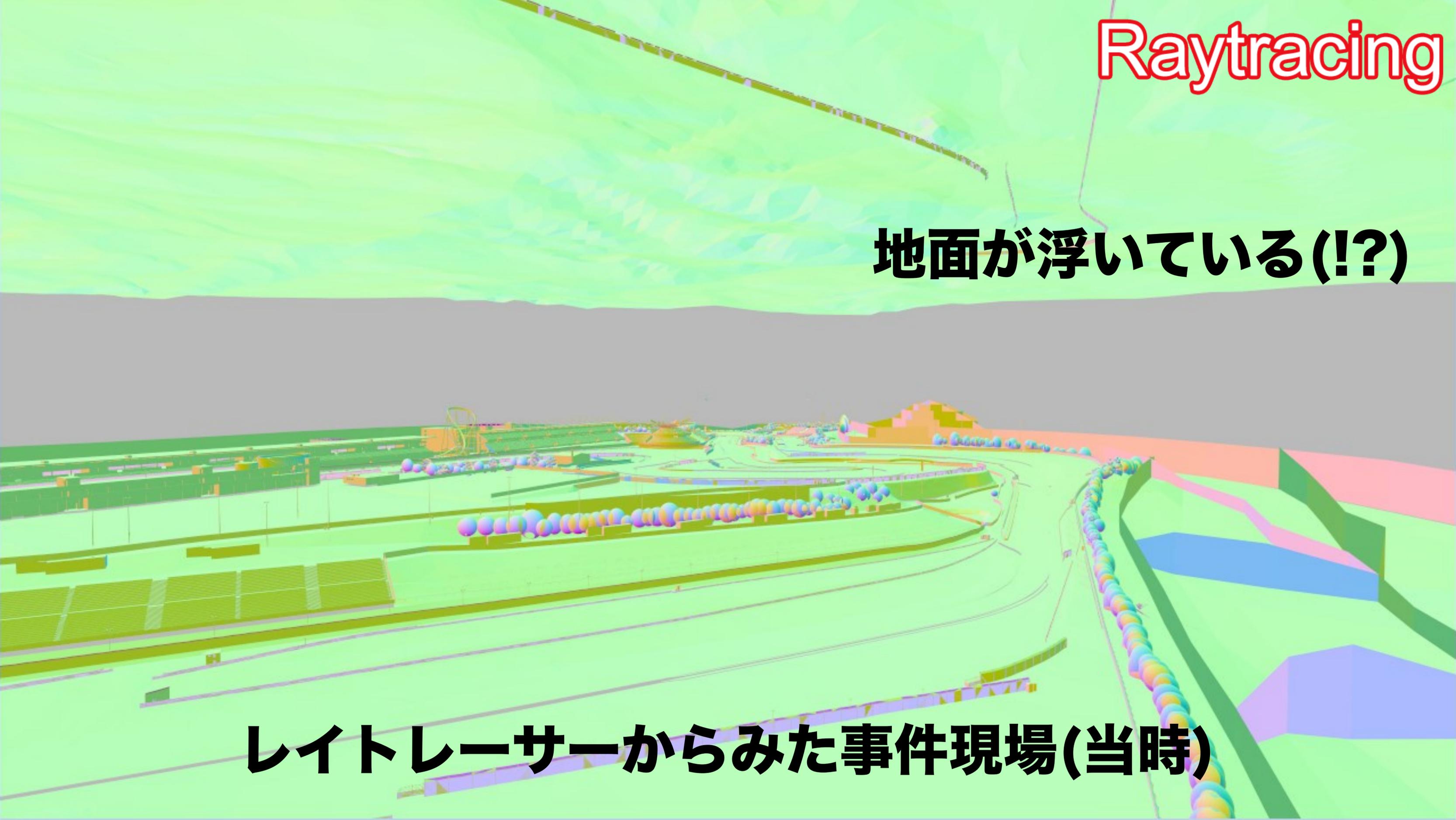
Raytracing

レイトレーサーから見た事件現場(当時)

Raytracing

地面が浮いている(!?)

レイトレーザーからみた事件現場(当時)



原因

- ・ **レイトレーサーから見た世界**で地面が浮いていました。
- ・ これは先ほど説明した**ベイクオフセット設定の誤設定**が原因でした。

真っ黒事件②

「とつぜんコース中央に暗い影が出始めました」

「なんとかしてください！」

PlayStation®4



事件現場(正しい姿)

可視化してみました

Raytracing

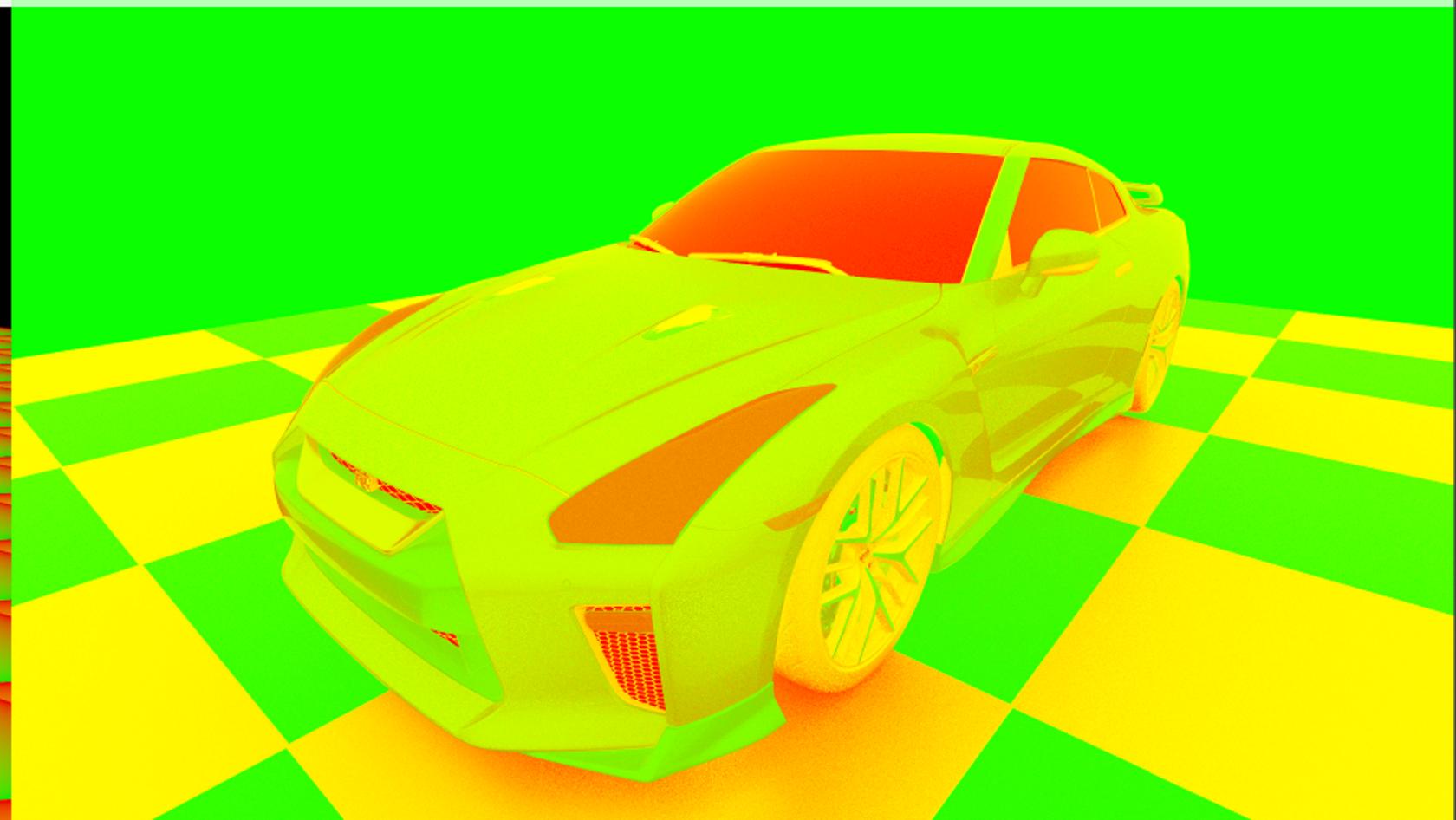
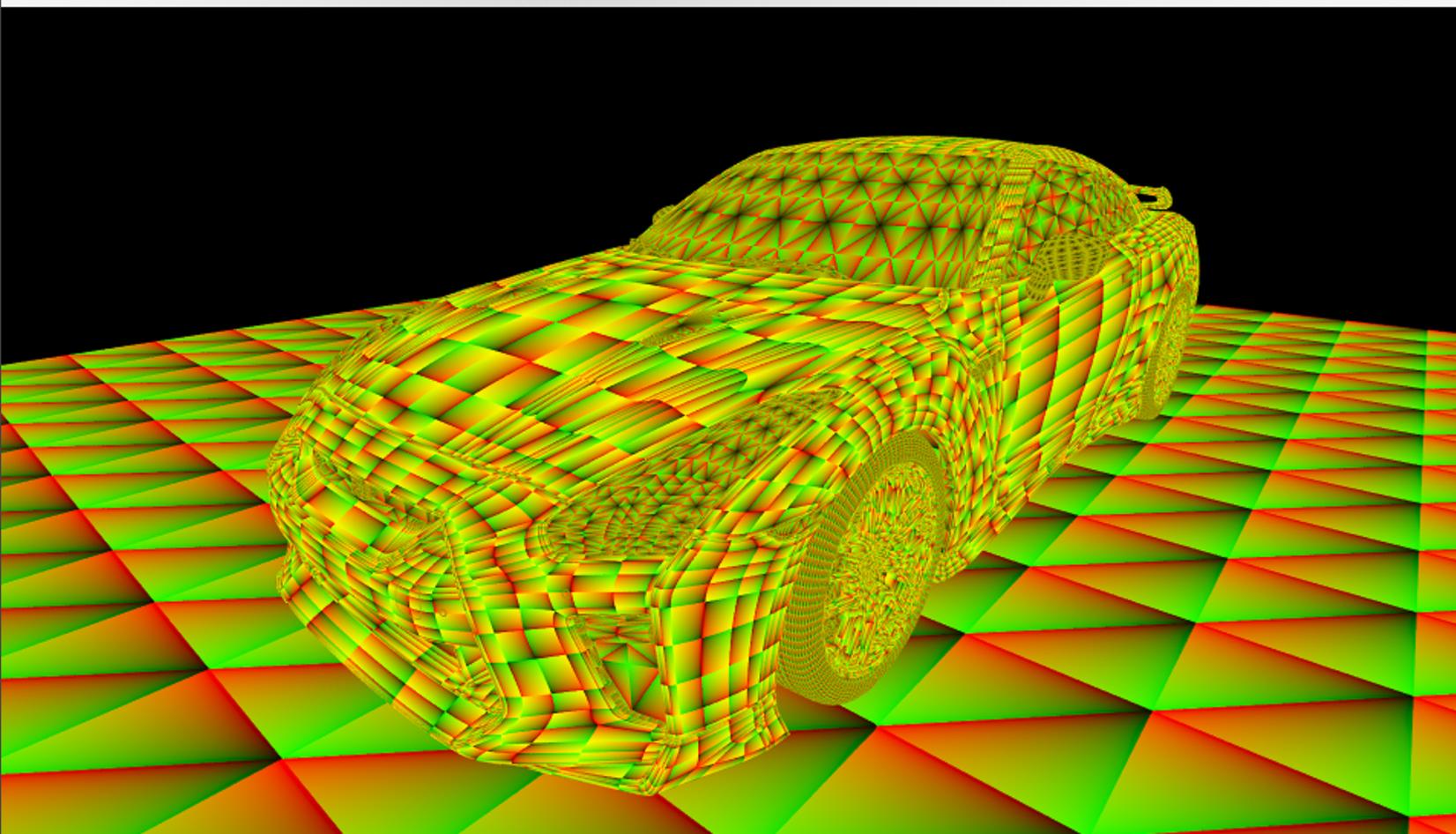
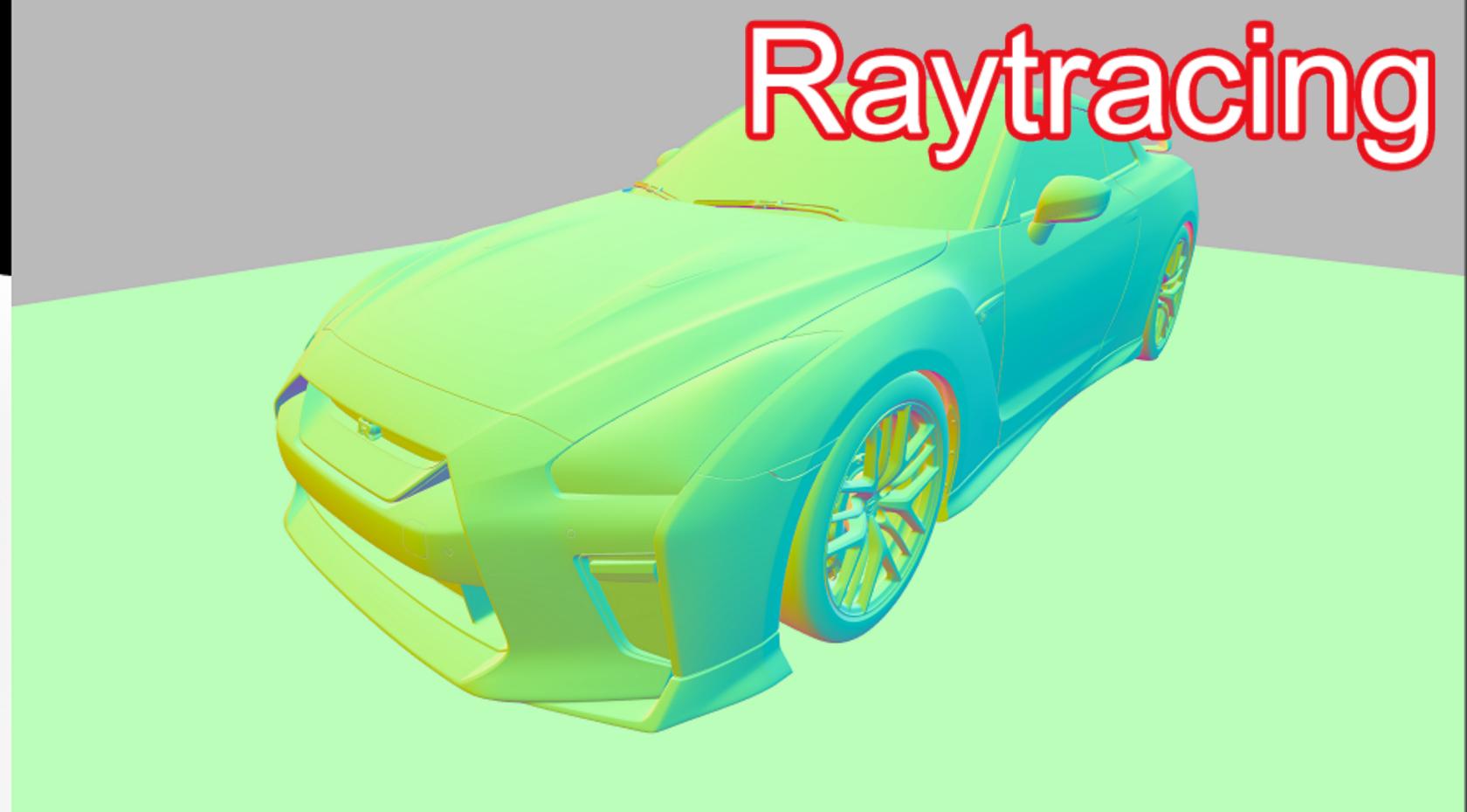
レイトレーサーからみた事件現場(当時)

原因

- ・ **レイトレーサーの世界**でだけ古いアセットが読まれていました。
- ・ コンバートパイプライン上の不具合でした。

デバッグプレビュー

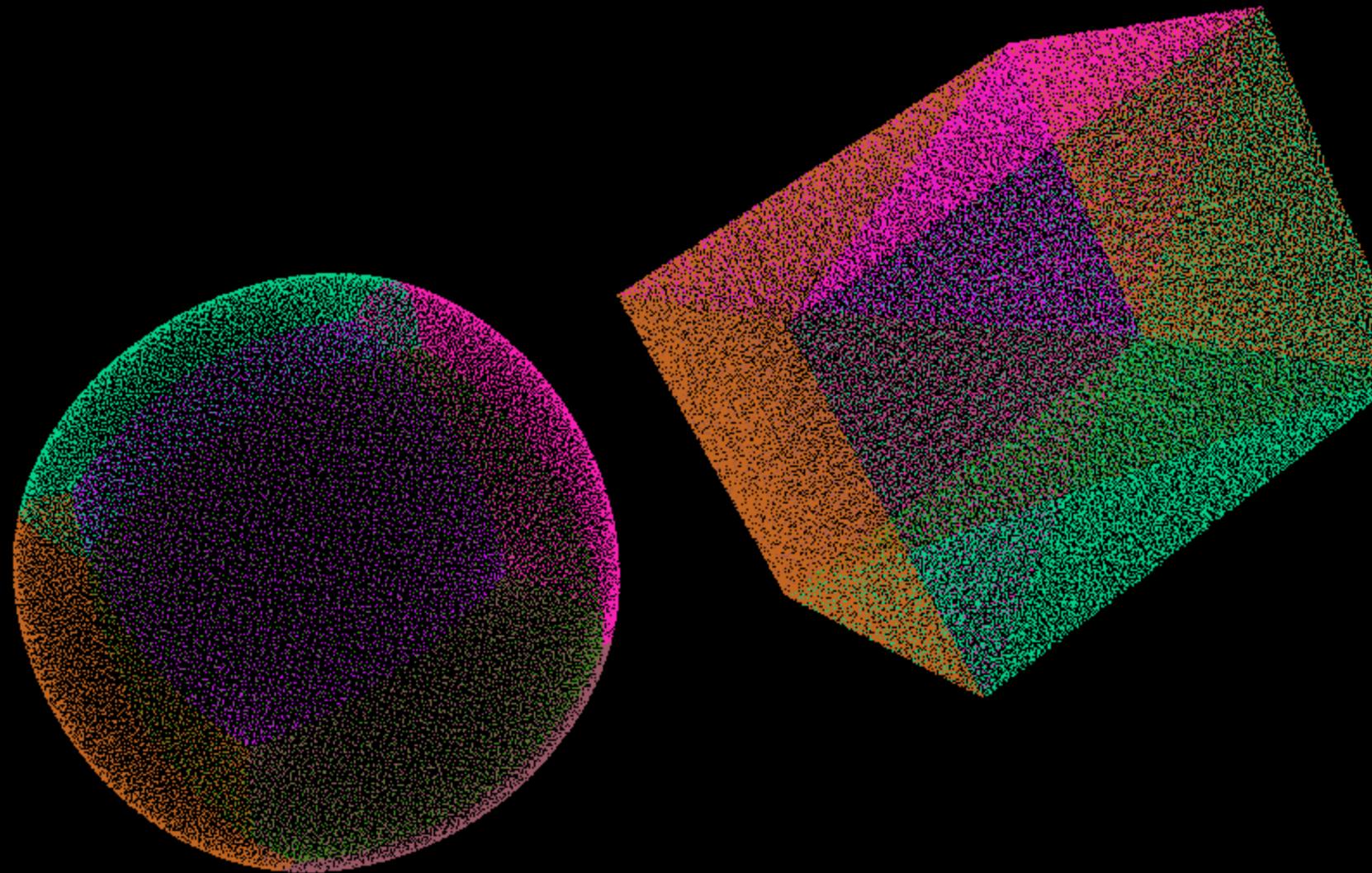
- ・ **可視化をするプレビューがないとデバッグは不可能**です。
- ・ 法線、UV、AO、ヒートマップ…などなど
必要なものを必要なだけ出せると便利です。
- ・ コア部分の**デバッグが捗ります**。



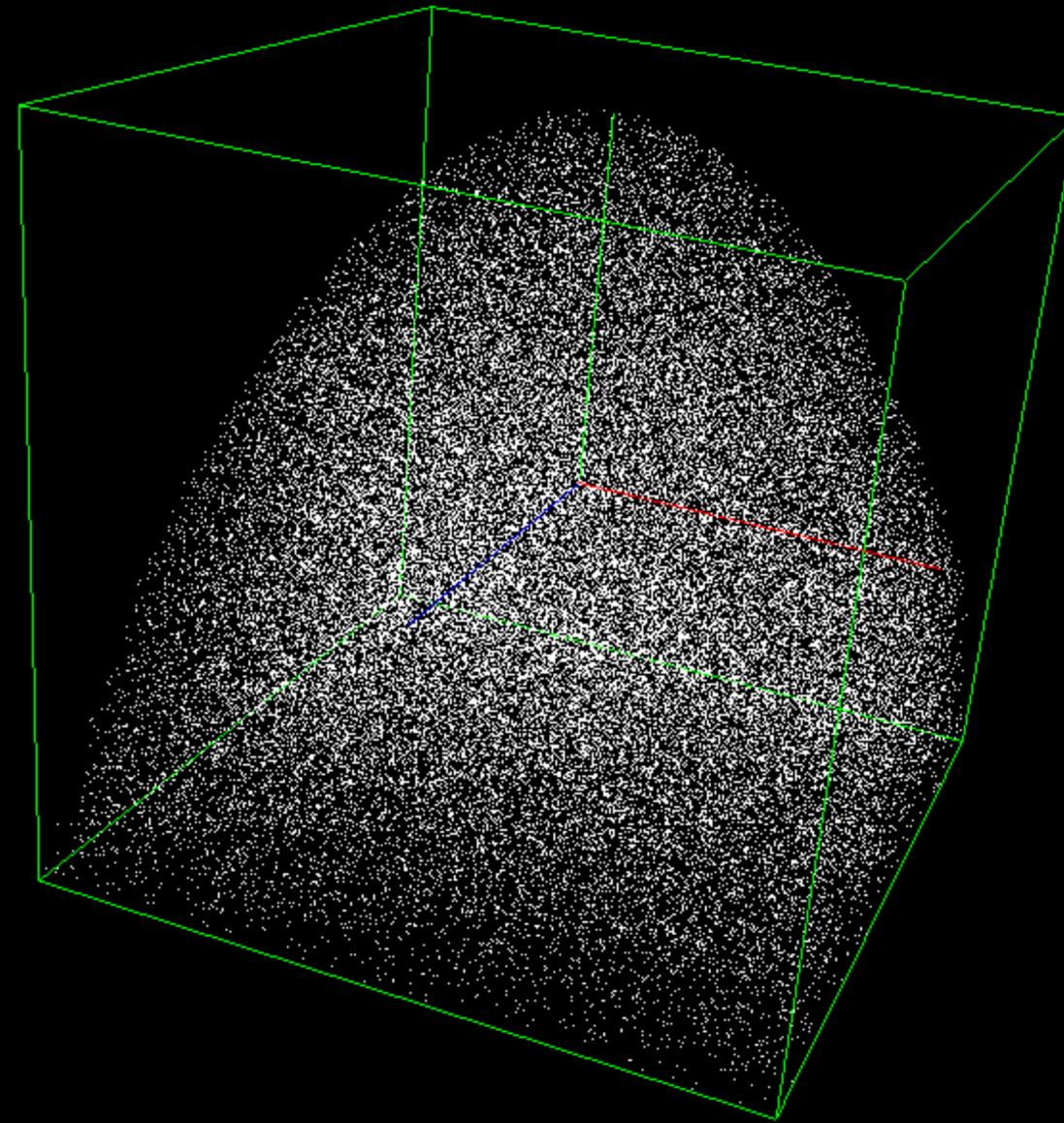
VDB: the printf of visual debugging

- デバッグ用のUIを作るのすら面倒であればVDBがおススメです。
Zachary DeVito氏によって製作された可視化環境です。
<https://github.com/zdevito/vdb>
- **printf()デバッグの感覚**で書いて3Dに可視化できます。
`vdb_point(x,y,z);`
`vdb_line(x0,y0,z0,x1,y1,z1);`
`vdb_triangle(x0,y0,z0,x1,y1,z1,x2,y2,z3);`

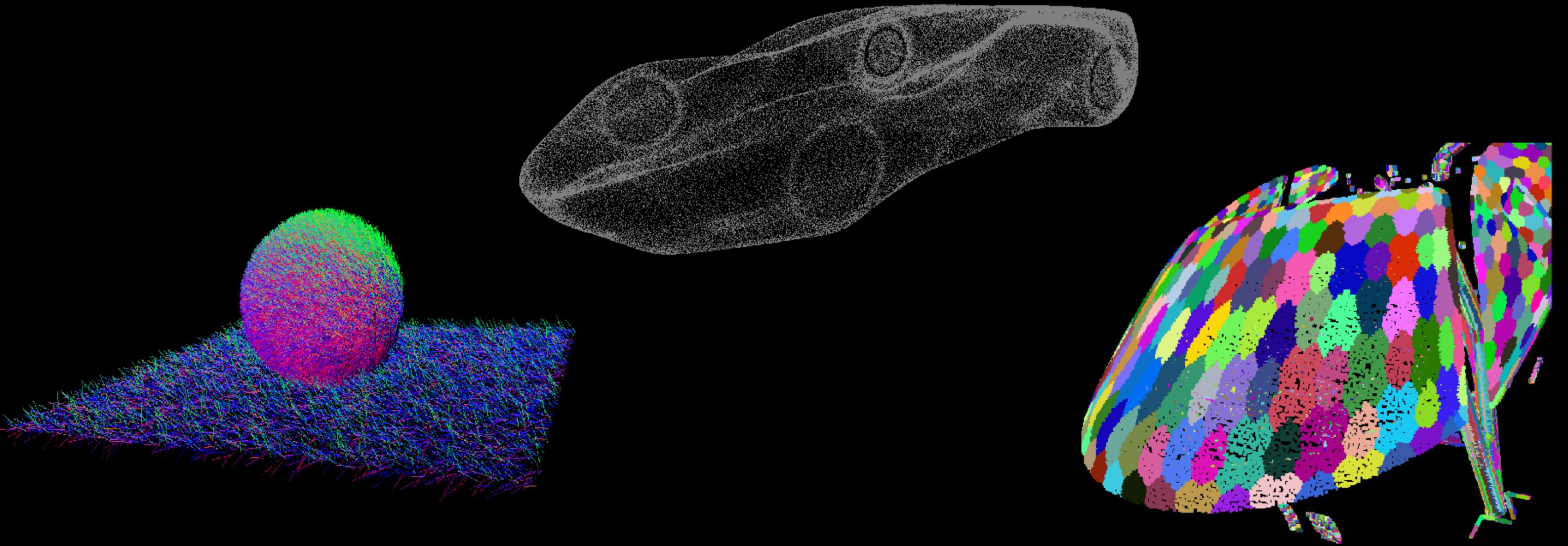
本当に曲面ができてる？



本当に想定した分布になってる？



そのほかにも色々



鉄塔が緑発光事件

「とつぜん鉄塔が緑に発光しはじめました！」

「なんとかしてください！」

PlayStation®4

1 HITOMI_crs 

YAG 0'3 1.597

2
1 10'08.019

FL HITOMI_crs 
10'08.019

TOTAL TOTAL TOTAL



事件現場(当時)



SH SH
000016.3



10'39.6 16

PlayStation®4

遠くの森から高輝度が飛来！



1 HITOMI_crs

0'3 1.597

2
1 10'08.019

FL HITOMI_crs
10'08.019

TOTAL TOTAL TOTAL

10'39.6 16

事件現場(当時)



テスト

- 原因は木専用のBSDFの**コード上のバグ**でした。
- $\cos(\theta)$ で1回割るところを不用意に2回割っていました。
キワのレイ($\cos(\theta)$ が小さくなる)のエネルギーが大増幅していました。

テスト

- レイトレは、**一箇所壊れると全体が壊れる**類のソフトウェアです。
- **単体テスト**はできるだけ書いたほうがいいです。
幸運なことに**レイトレはモジュール性が高く**単体テストがしやすいです。
- **White Furnace Test**や、**球の作るAO、一様な輝度の球体の内部**など解析的な解が知られているものから固めるのがオススメです。
- **単体テスト+ソフトウェアテストでCI**を回しています。

| | |
|---------------------------------|---------|
| ✓ AABB basic | Success |
| ✓ BVH basic | Success |
| ✓ Distribution1D::Args | Success |
| ✓ Distribution1D::basic1 | Success |
| ✓ Distribution1D::basic2 | Success |
| ✓ Distribution1D::basic3 | Success |
| ✓ Distribution1D::basic4 | Success |
| ✓ Distribution1D::basic5 | Success |
| ✓ Distribution1D::basic6 | Success |
| ✓ Distribution1D::basic7 | Success |
| ✓ Distribution1D::Fill1 | Success |
| ✓ Distribution1D::Fill2 | Success |
| ✓ Distribution1D::integralTest1 | Success |
| ✓ Distribution2D::Basic1 | Success |
| ✓ Distribution2D::integralTest2 | Success |

この章のまとめ

- ・ **実際にあった事故**とその**解決策**についていくつかお話ししました。

目次

- はじめに
- レイトレーシングとは
- 弊社でのレイトレーシングの歴史
- 弊社での使われ方
- 事故、デバッグ、対策
- **結び**

結び

- **レイトレーシング**を使ったゲーム開発の一端をお話しました。
- 色々ありましたが、**弊社の目標とするビジュアル**を達成できました。
- 今後も**高品質なレンダリング**に繋がる方法を探っていきたい。
- **レンダリングの未来はレイトレーシングの中にあります！**
まだやったことがない方も一緒に未来を作りましょう！

謝辞

本スライドはたくさんの人たちの協力によって作成されました。

ここで皆様への感謝の意を表します。

エンジニア(敬称略) アーティスト(敬称略)

鈴木 健太郎

曾根原 勉

手島 孝人

高杉 悟

高野 修一

安富 健一郎

内村 創

及びアーティストの皆様

安田 廉

遠藤 忠

杉原 正通

小林 征生

竹歳 正史

中川 展男

及びエンジニアの皆様

Q&A