NeuralPVS: 『グランツーリスモ 7』における ニューラルネットワークを用いた 次世代オブジェクトカリングシステム

于承中

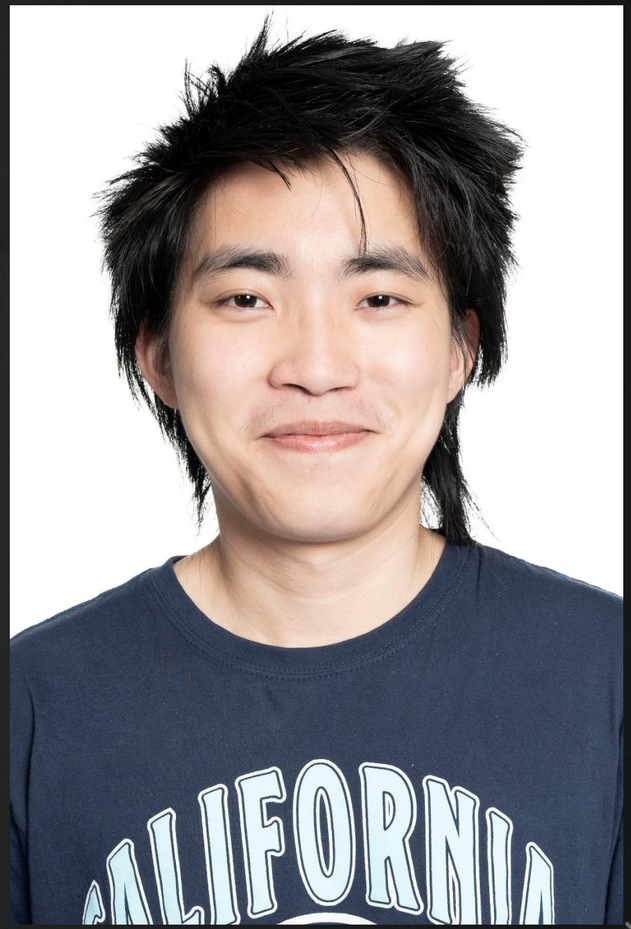
内村創

POLYPHONY™ DIGITAL

自己紹介 / 于承中

- 東京理科大学卒
- 2024年入社

- マシンラーニング
- ニューラルネットワーク
- レンダリング



POLYPHONY™ DIGITAL

自己紹介 / 内村 創

- 画像処理
- 色彩工学
- 事前計算カリング

POLYPHONY™ DIGITAL

gran-turismo.com/jp/gt7/top/

GRAN TURISMO LIVE

NEWS GT7 PRODUCTS

My Page

トップ ニュース マイページ ショーケース スポーツモード マニュアル 製品情報

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR

Agenda

- 前半
  - PVSとは
  - 基本的なアイデア
  - 具体的な実装と実データにおける工夫
  - クラスタリングの詳細
  - 手法の限界
- 後半
  - Neural PVS
  - トレーニングデータ
  - ネットワークストラクチャ
  - 量子化
  - 性能評価

POLYPHONY™ DIGITAL

PVS(Potentially Visibility Sets)とは

POLYPHONY™ DIGITAL

PVSとは

- オブジェクトカリングを事前計算しておく手法
- Potentially Visibility Sets [Nirenstein02]
- 可視になる可能性のあるモデルの集合

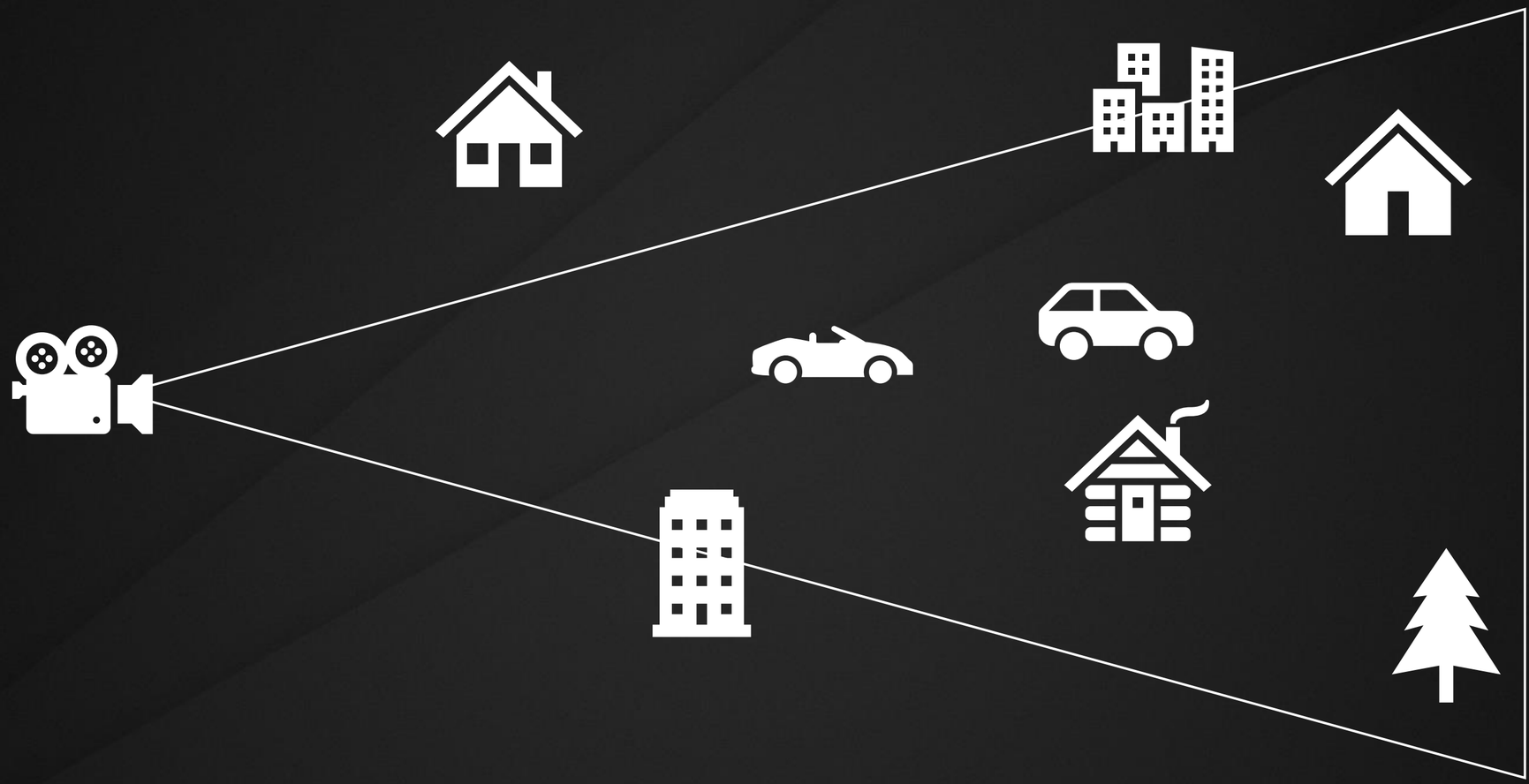
オブジェクトカリングとは

- オブジェクトは、描画するモデルの単位
- カリングは、見えないモデルを除去すること

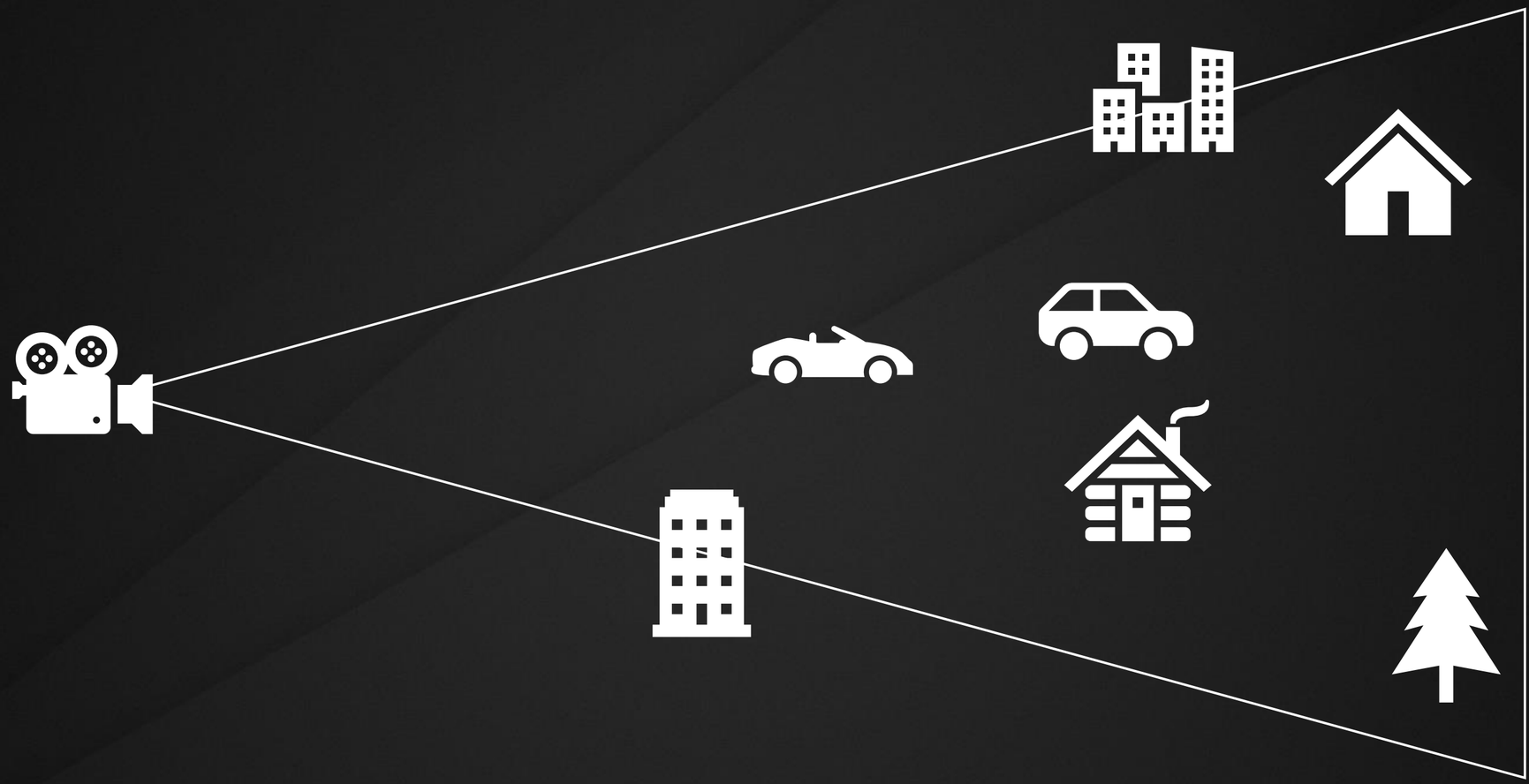
なぜ除去するのか？

- 軽量になるから
  - 処理が減るので、CPUが軽くなる
  - 描画が減るので、GPUも軽くなる

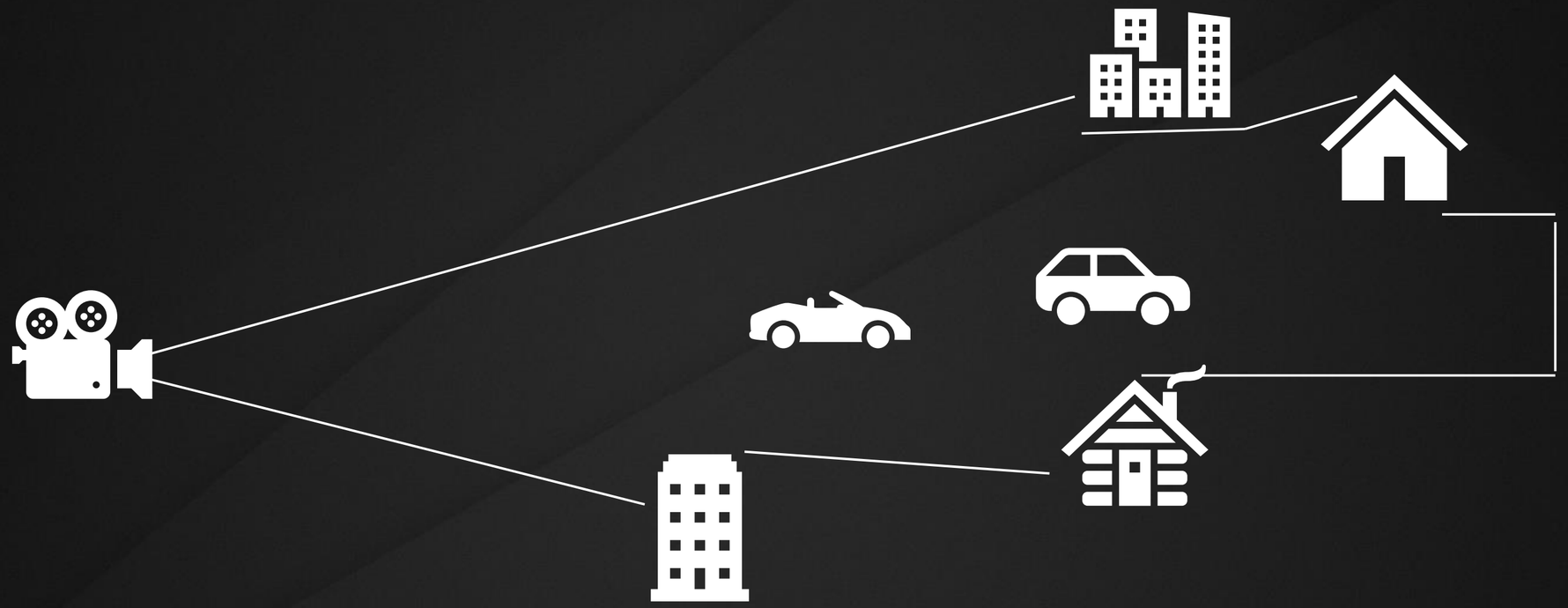
典型的なシーンを考える

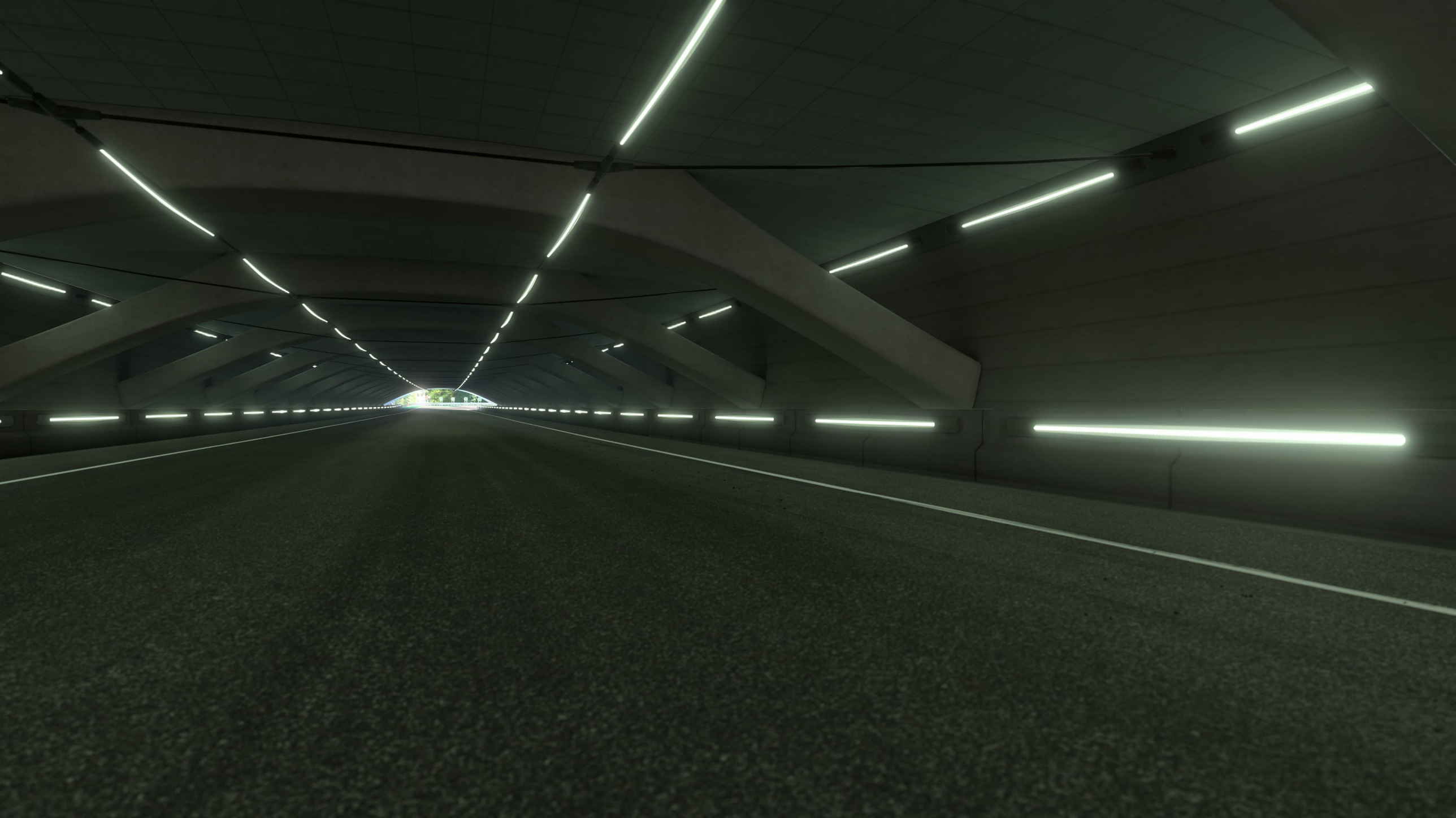


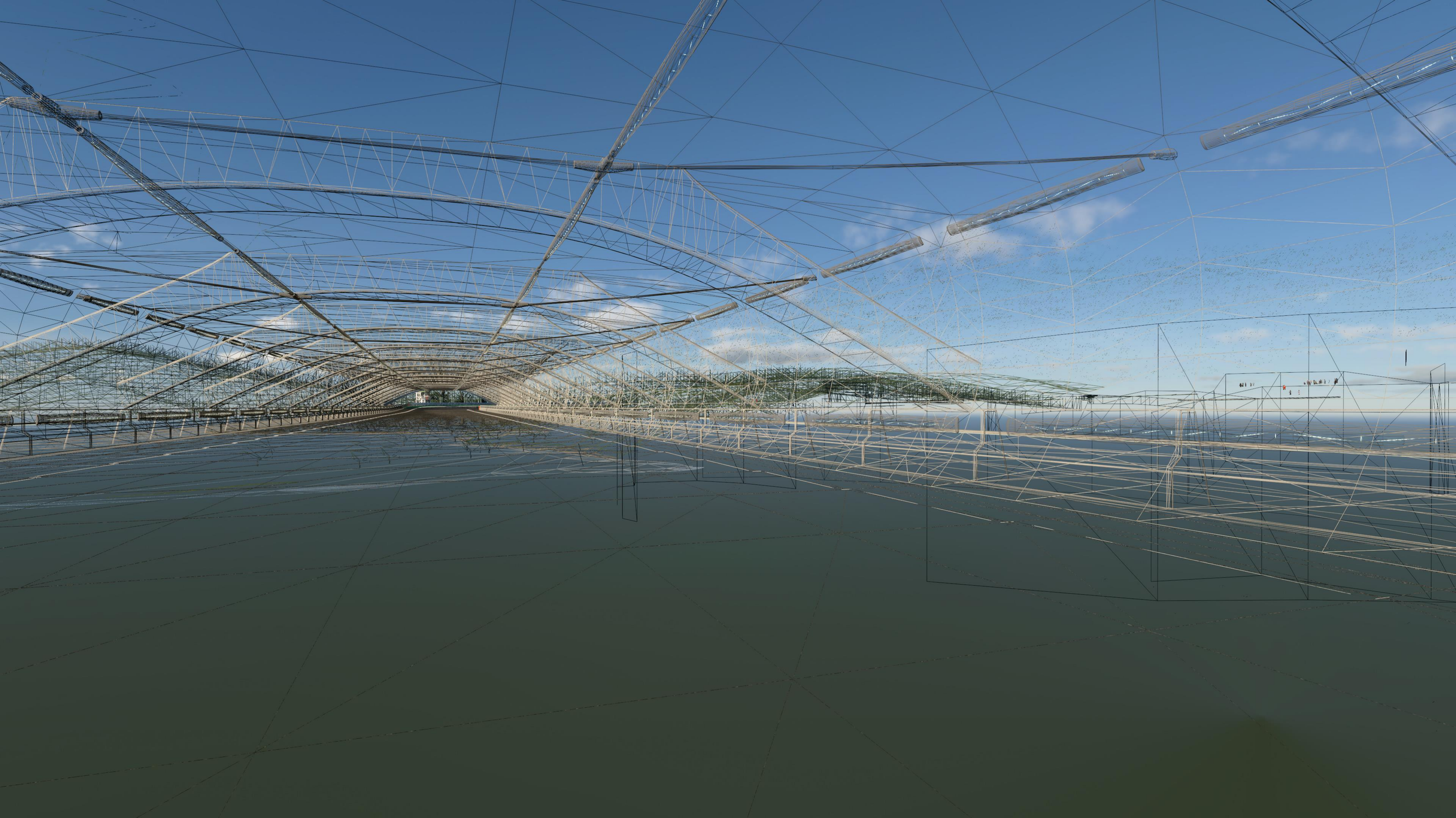
カメラの範囲外は描画しない (フラスタムカリング)

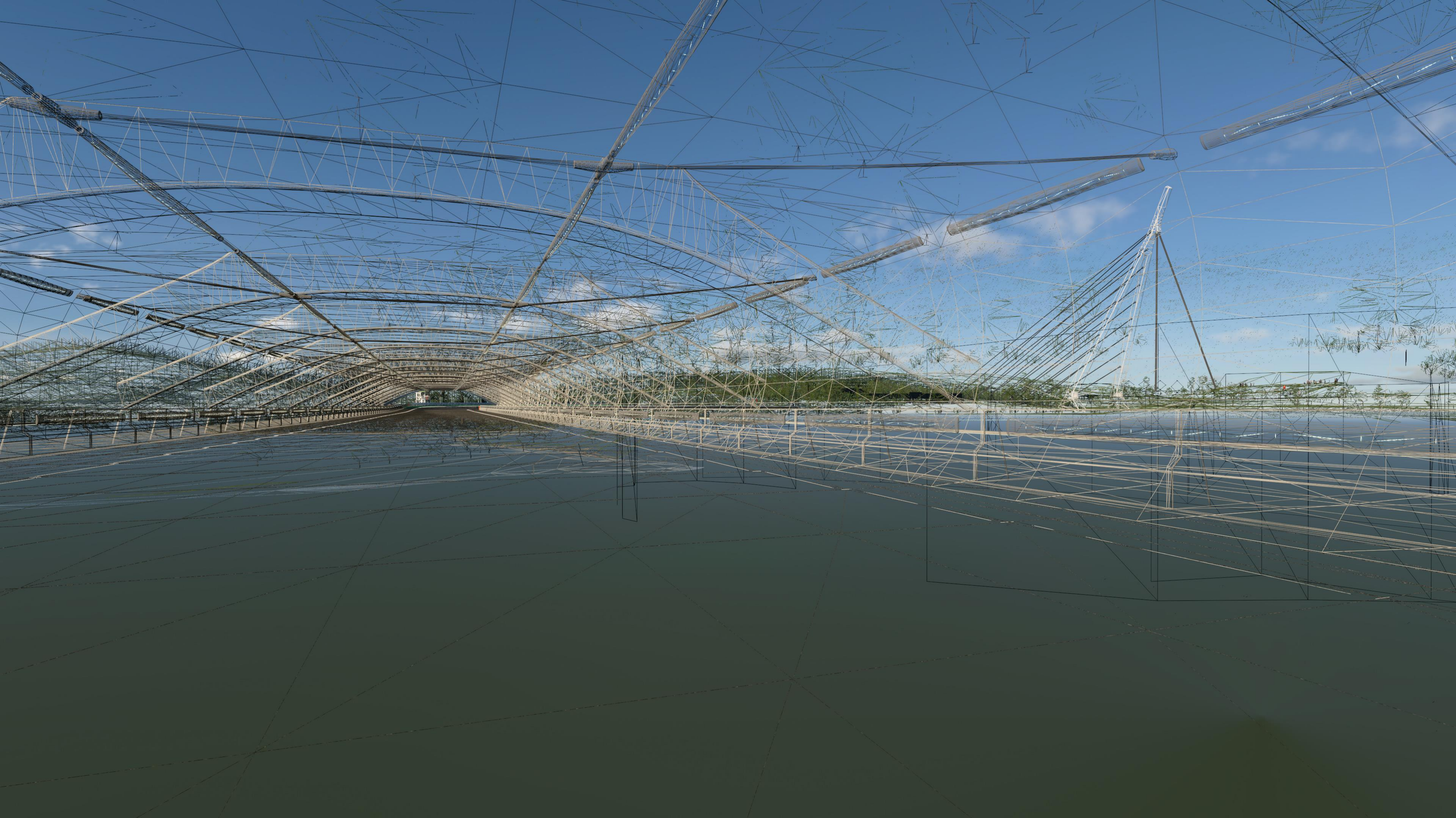


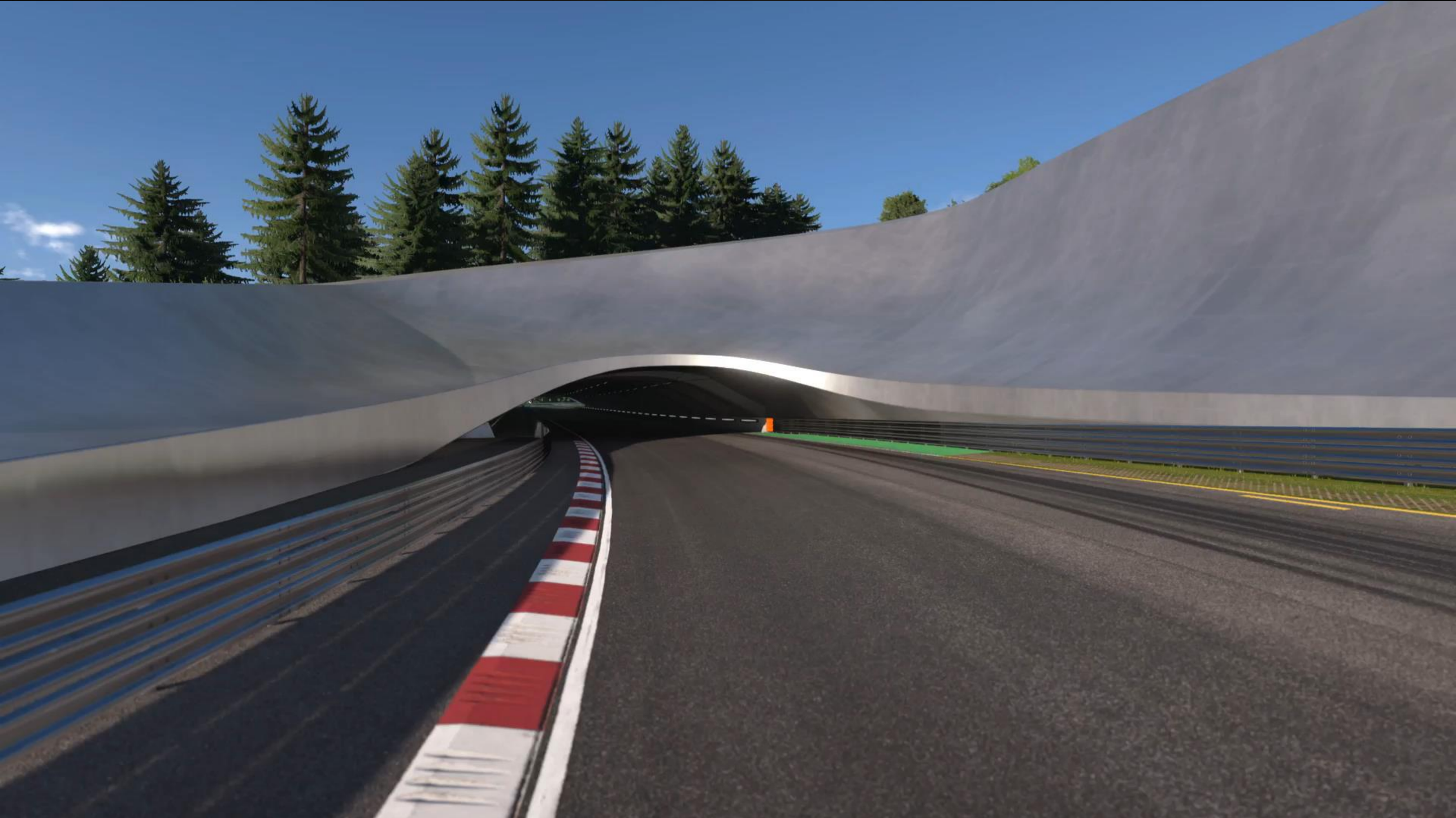
手前のものに隠されるものも表示しない (オクルージョンカリング)












Grand Valley
HIGHWAY-1
WeatherTech
MOTHERS
Exxon
ZURICH
BOSCH
100
200
300

事前計算カリングとは

- 可視情報を事前計算しておくカリングシステム
- 入力：カメラ情報（座標など）
- 出力：可視なモデルのリスト

- UnrealEngineでは”Precomputed Visibility Volume”
- CG用語では”Potentially Visibility Sets”

既存のカリング手法の大まかな比較

事前計算
- Potentially Visibility Sets

リアルタイム
- バックフェースカリング
- フラスタムカリング
- Hierarchical Z-Buffer(HZB) [Shopf 08]

既存のカリング手法の大まかな比較

事前計算
- カリング効果を高くできる
- 事前計算時間がかかる
- カメラ移動範囲に制限

リアルタイム
- カリング効果が低下しがち
- ポッピングなどが発生する
- カメラ移動範囲が自由

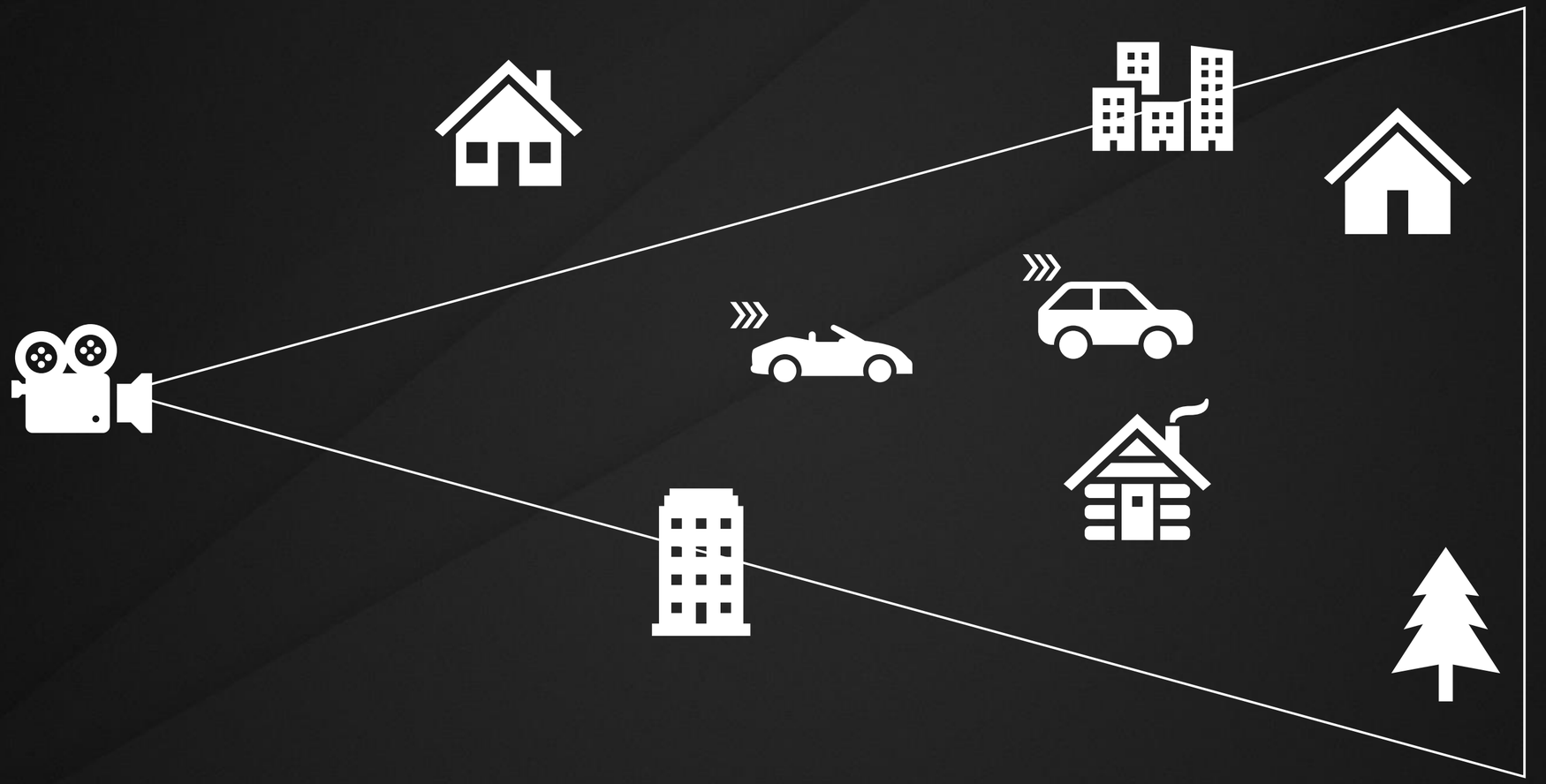
基本的なアイデア

POLYPHONY™ DIGITAL

基本的なアイデア

1. コース全体をいろいろな位置と角度からレンダリングする
2. レンダリング結果から、可視オブジェクトのリストを得る
3. 可視オブジェクトのリストを、シンプルな実機データにする

先ほどのシーン



POLYPHONY™ DIGITAL

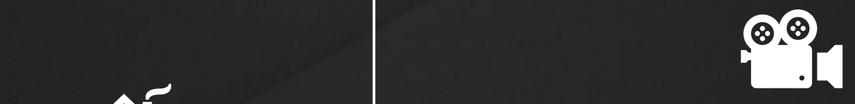
事前計算は静止物のみを対象とする

可視判定する

可視判定結果の表

モノ	可視？
	✓
	✓
	✗
	✓
	✗

可視リストにする

カメラ	リスト
	

オブジェクトは
IDで管理しておく

モノ	ID
	1
	2
	3
	4
	5
	6

カメラは座標情報を持つ

(10, 0, -8)

可視リストの実体
(社内用語「ビジョンポイント」)

座標	IDリスト
(10, 0, -8)	1, 2, 5, 6

実際には可視判定は全方向に行う

カメラの移動範囲をカバーするように

色々な場所で可視判定を行う

半透明物体は後ろを隠さない

演出オブジェクトは全パターン試す

可視リストが完成した
(社内用語「ビジョンリスト」)

座標	IDリスト
(10, 0, -8)	1, 2, 5, 6
(4, 2, 6)	2, 5, 6
(0, 0, -5)	1, 2, 3, 5, 6
…	…

可視リストを用いた場合の素朴なレンダリングステップ

1. カメラの位置から、もっとも近いリストを探す
2. そのリストの内容を描画する
3. 動的なオブジェクトは別に描画する

カメラの位置から
もっとも近いリストを探す

(9, 0, -9)

座標	IDリスト
(10, 0, -8)	1, 2, 5, 6
(4, 2, 6)	2, 5, 6
(0, 0, -5)	1, 2, 3, 5, 6
…	…

IDからオブジェクトを引く

座標	IDリスト
(10, 0, -8)	1, 2, 5, 6

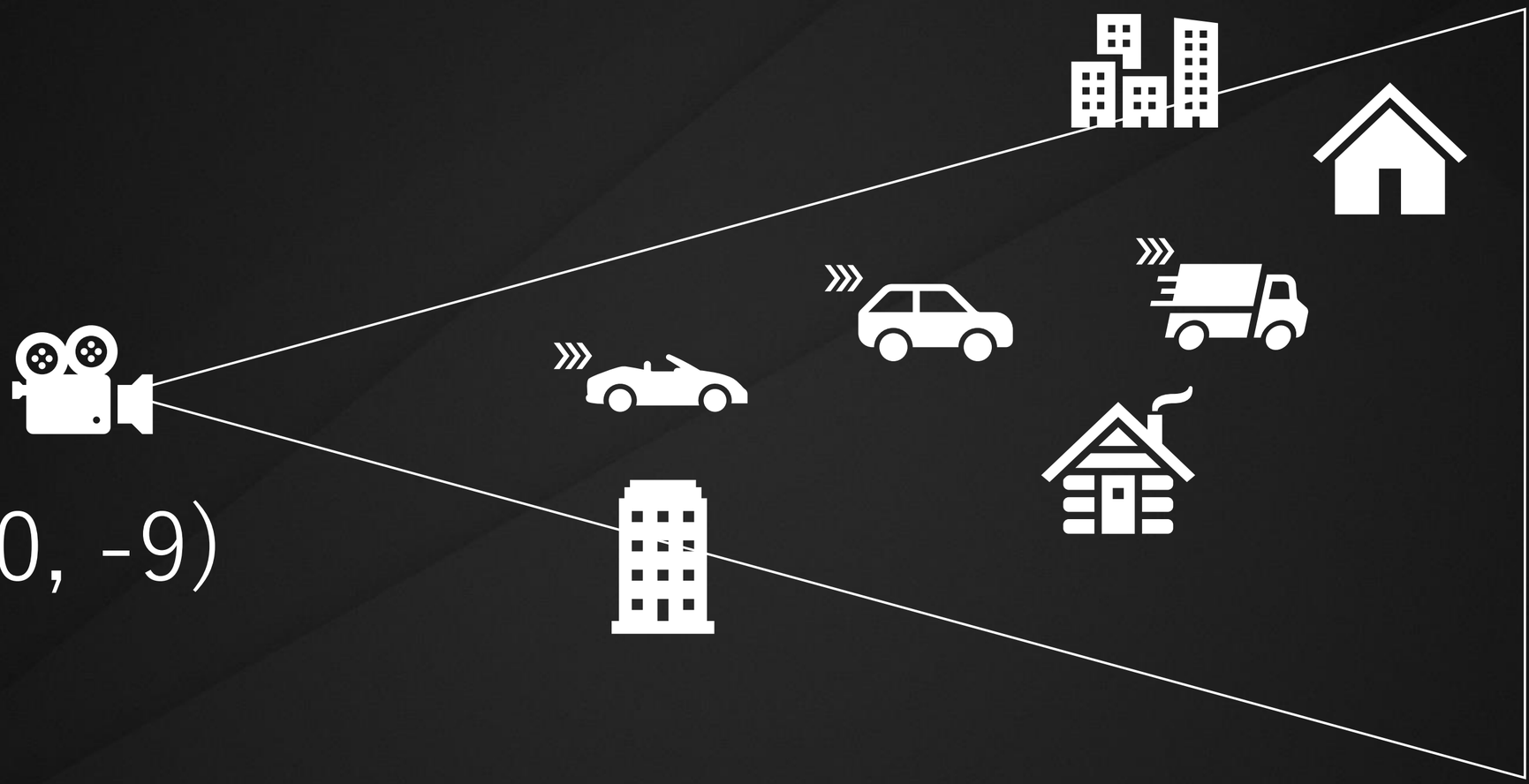
ID	モデル
1	
2	
…	…

描画すべきオブジェクト


見えるオブジェクトだけ描画する

(9, 0, -9)

カリング対象外の
オブジェクトを追加描画する



(9, 0, -9)

あくまでも近傍なので、誤差が発生する

(9, 0, -9)

座標	IDリスト
(10, 0, -8)	1, 2, 5, 6
(4, 2, 6)	2, 5, 6
(0, 0, -5)	1, 2, 3, 5, 6
…	…

事前計算結果はこうだが

(10, 0, -8)

実際のカメラとのズレがあるので

本来見えないもの
見えるべきものに誤差が出る

精度とリコール率

- 精度(Precision) $= \frac{TP}{TP+FP}$
- 再現率(Recall) $= \frac{TP}{TP+FN}$

精度

- $\frac{TP}{TP+FP}$

=75%

リコール率

- $\frac{TP}{TP+FN}$

$=75\%$

用語の説明

用語	意味
ビジョンポイント*	ある座標と、そこから見えるオブジェクトのリスト
ビジョンリスト*	ビジョンポイントのリスト
母点	カメラ座標から最も近いビジョンポイントの座標
リコール率	低いと画像がバグる
精度	高いとパフォーマンスが上がる

*：社内独自用語

具体的な実装

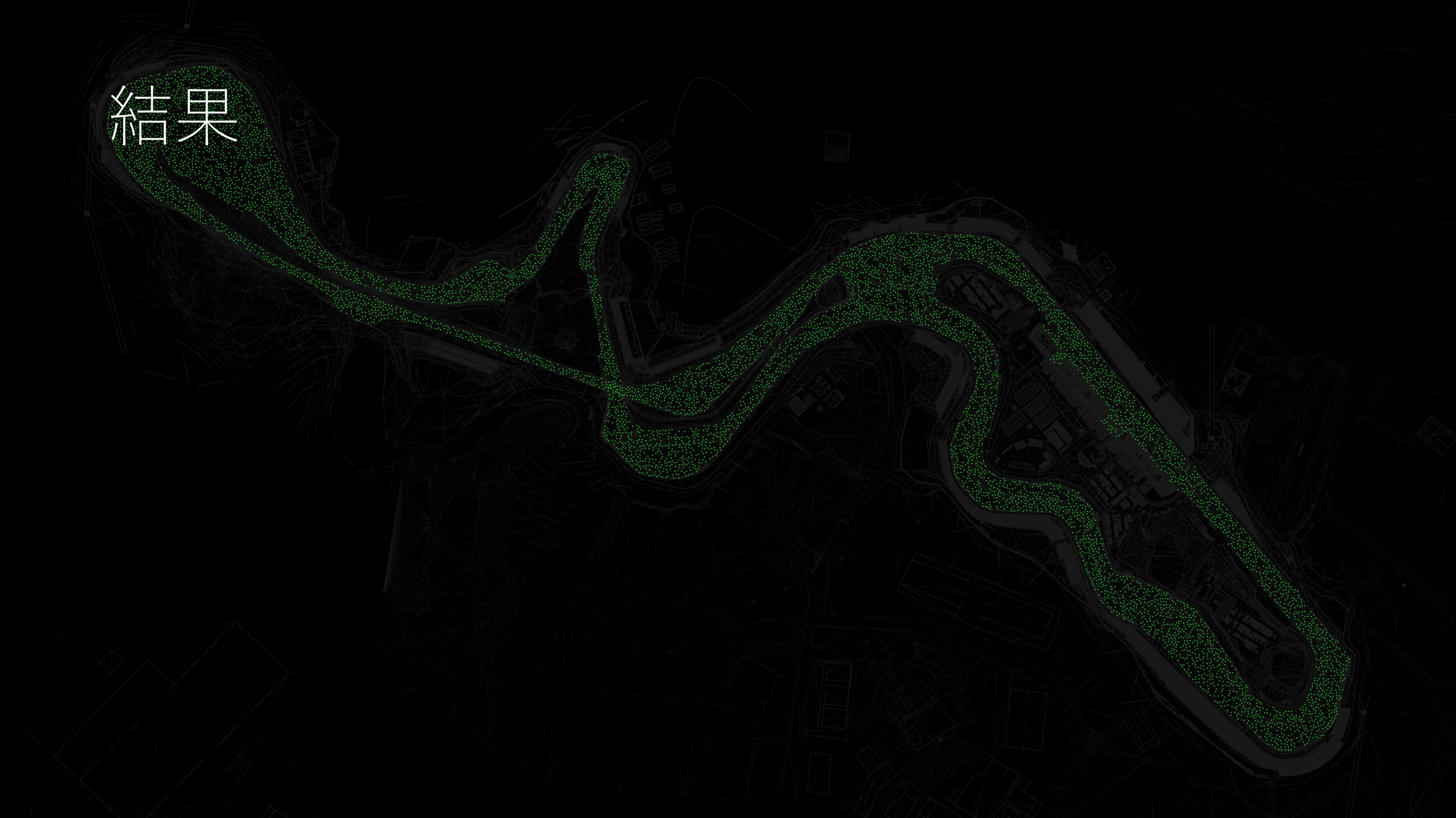
POLYPHONY™ DIGITAL

カメラの移動範囲をどう求めるか

- カメラの範囲＝走行可能な範囲
- 走行可能な範囲＝路面メッシュ

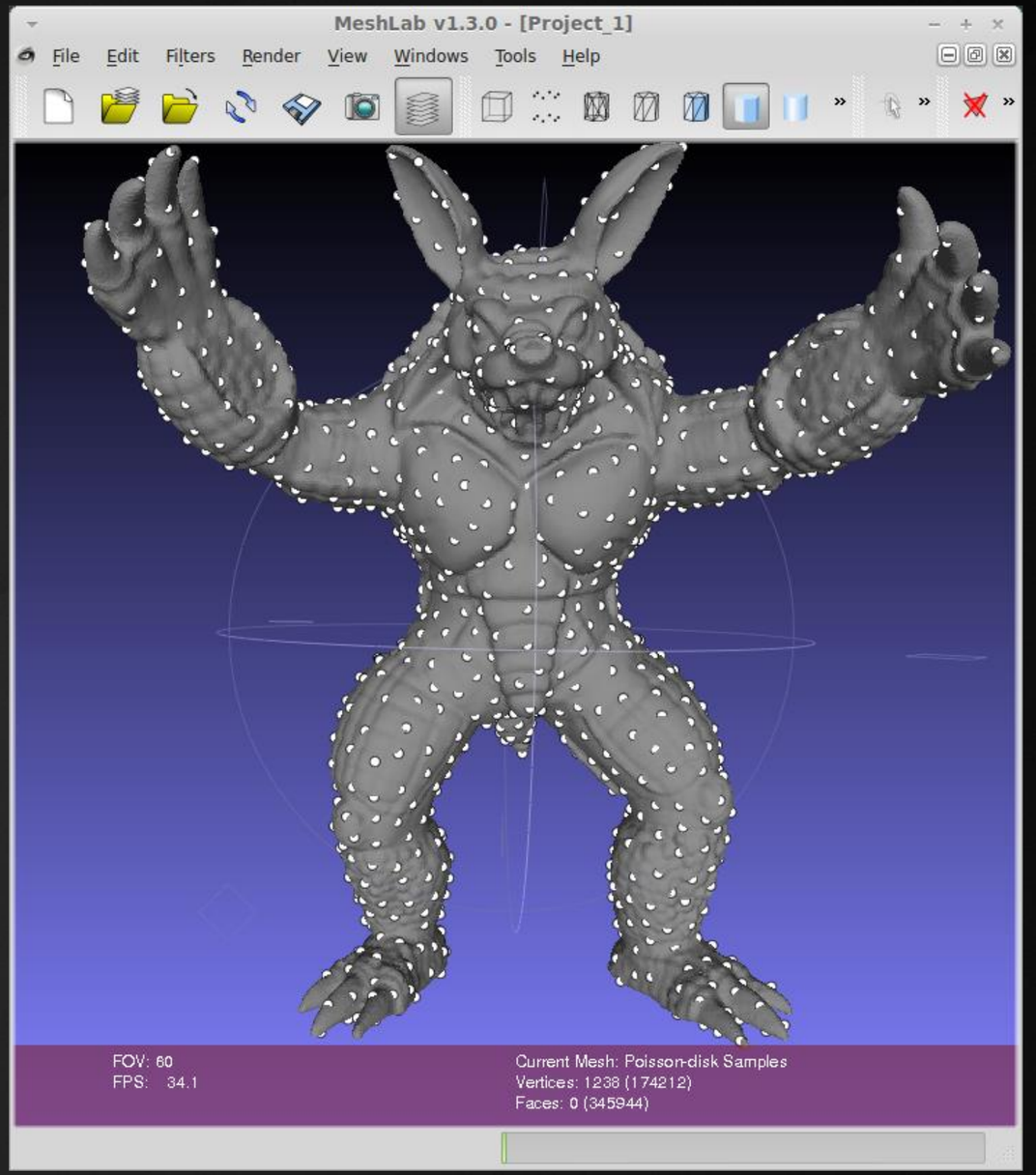
- ポワソンディスクサンプリングする

結果

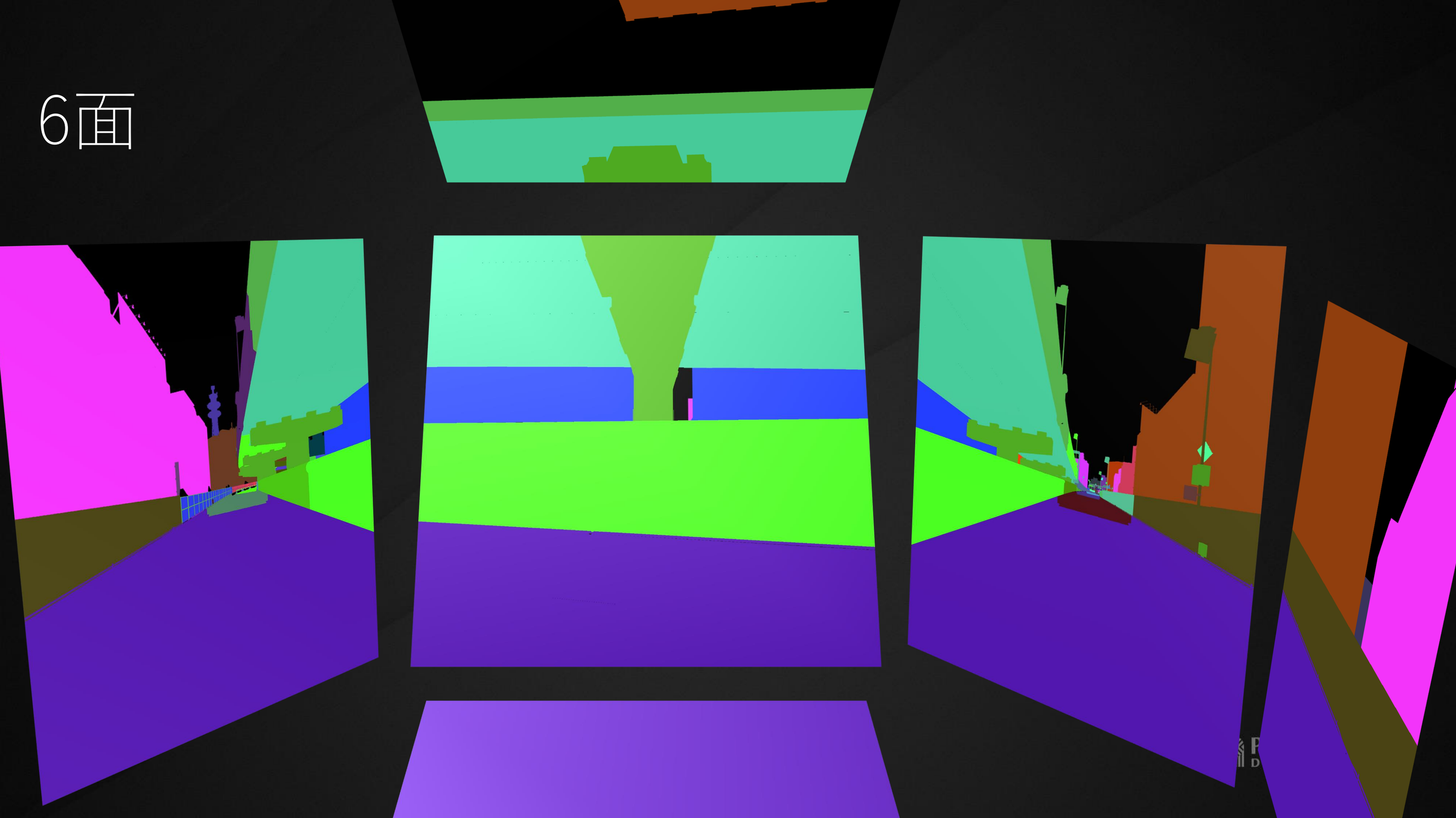


ポワソンディスクサンプリングとは

- 均一にサンプル点を生成するアルゴリズム



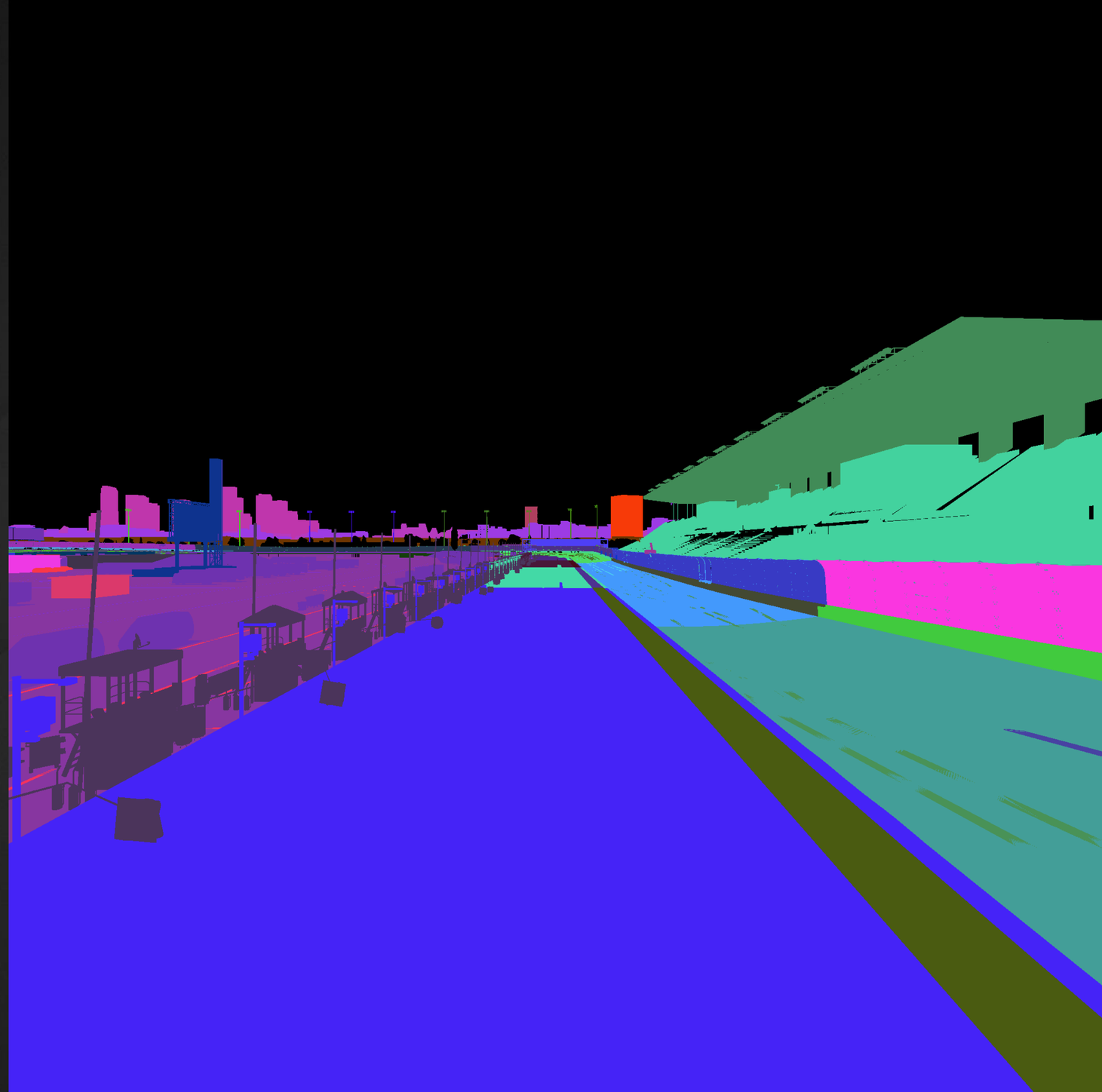
https://rodolphe-vaillant.fr/entry/37/source-code-for-poisson-disk-sampling-of-a-triangle-mesh


6面

オブジェクトIDバッファ

- OpenGLでレンダリング
- 各ピクセルはID

- ピクセルがある
  ＝オブジェクトが可視



実際の描画ピクセル数を得る

- GL_SAMPLES_PASSEDクエリ
- Begin-End区間に描画されたピクセル数が計測できる
- とても小さいオブジェクトを無視するなどに使う

Name

glBeginQuery — delimit the boundaries of a query object

C Specification

```
void glBeginQuery( GLenum target,
                   GLuint id);

void glEndQuery( GLenum target);
```

https://registry.khronos.org/OpenGL-Refpages/gl4/html/glBeginQuery.xhtml

半透明とイベント演出

- 半透明物体の扱い
  - 可視テストはされる
  - オクルーダにはならない
- イベント演出オブジェクトの扱い
  - すべての演出パターンをテストする
- 本来の描画パスの８倍くらい描画テストが必要

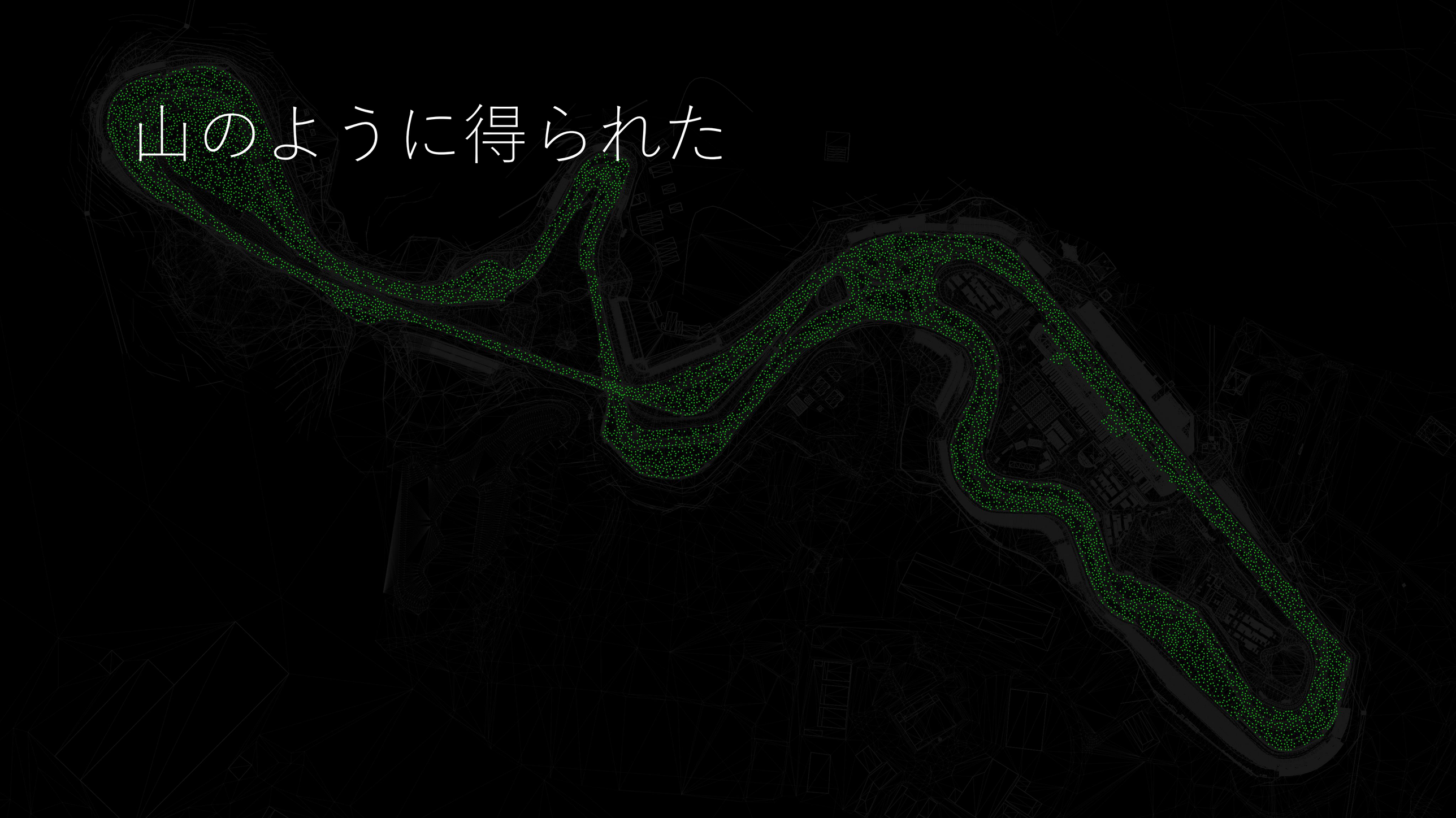
並列に生成する

- 空いているPCで
  並列に計算した
- ビジョンポイントが
  独立なので
  並列化は容易

座標とUUIDのリストが得られた


(-52.421307,15.250546,41.459705) : 313a2d4fdfca3d9a,cb30bf5ab22c92e1,ae327f72c4e73f6c,1f300eaeb15346a2,443ef7040
,fe380c13f9471c9d,9d3907d4a123f7fe,ca3b8996bbd84001,

山のように得られた



点が多すぎる

- コース全体で10万以上のビジョンポイント
- データサイズ
- クエリ速度

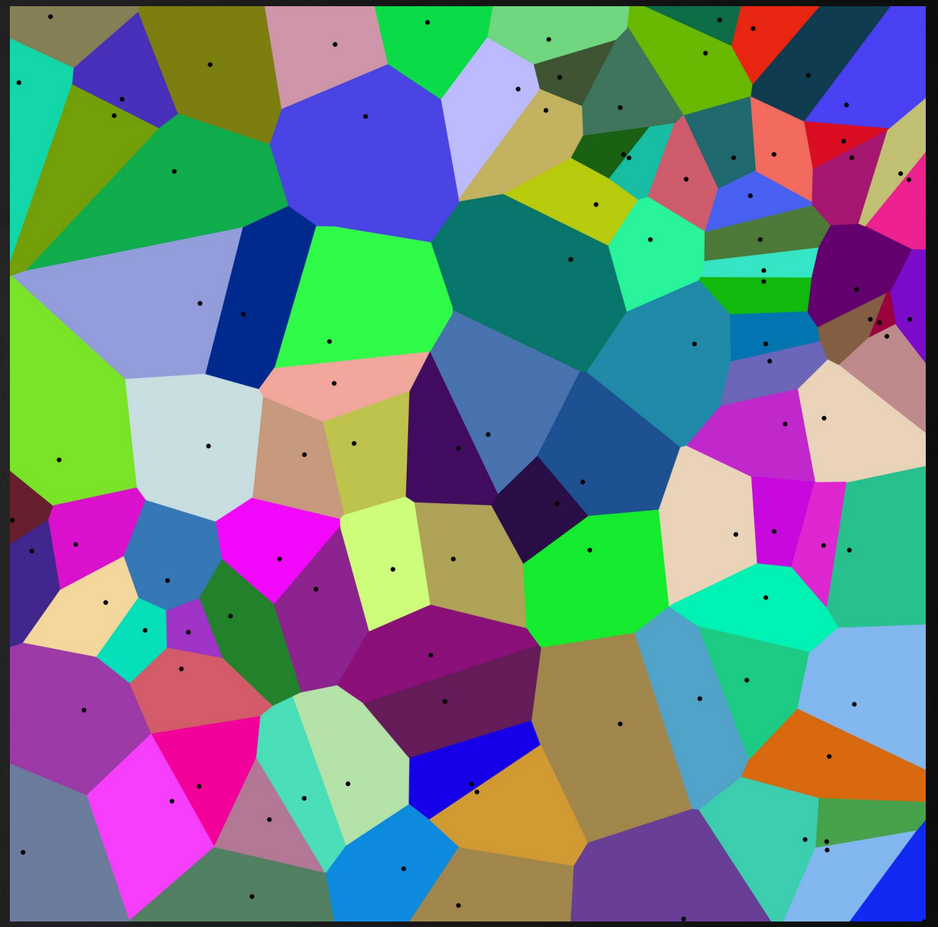
- このままでは非常に使いにくいデータなので簡略化したい

クラスタ化

1. 荒くポワソンサンプリングして代表点を作る
2. ボロノイ領域に含まれるポイントをすべてマージする

ボロノイ領域とは

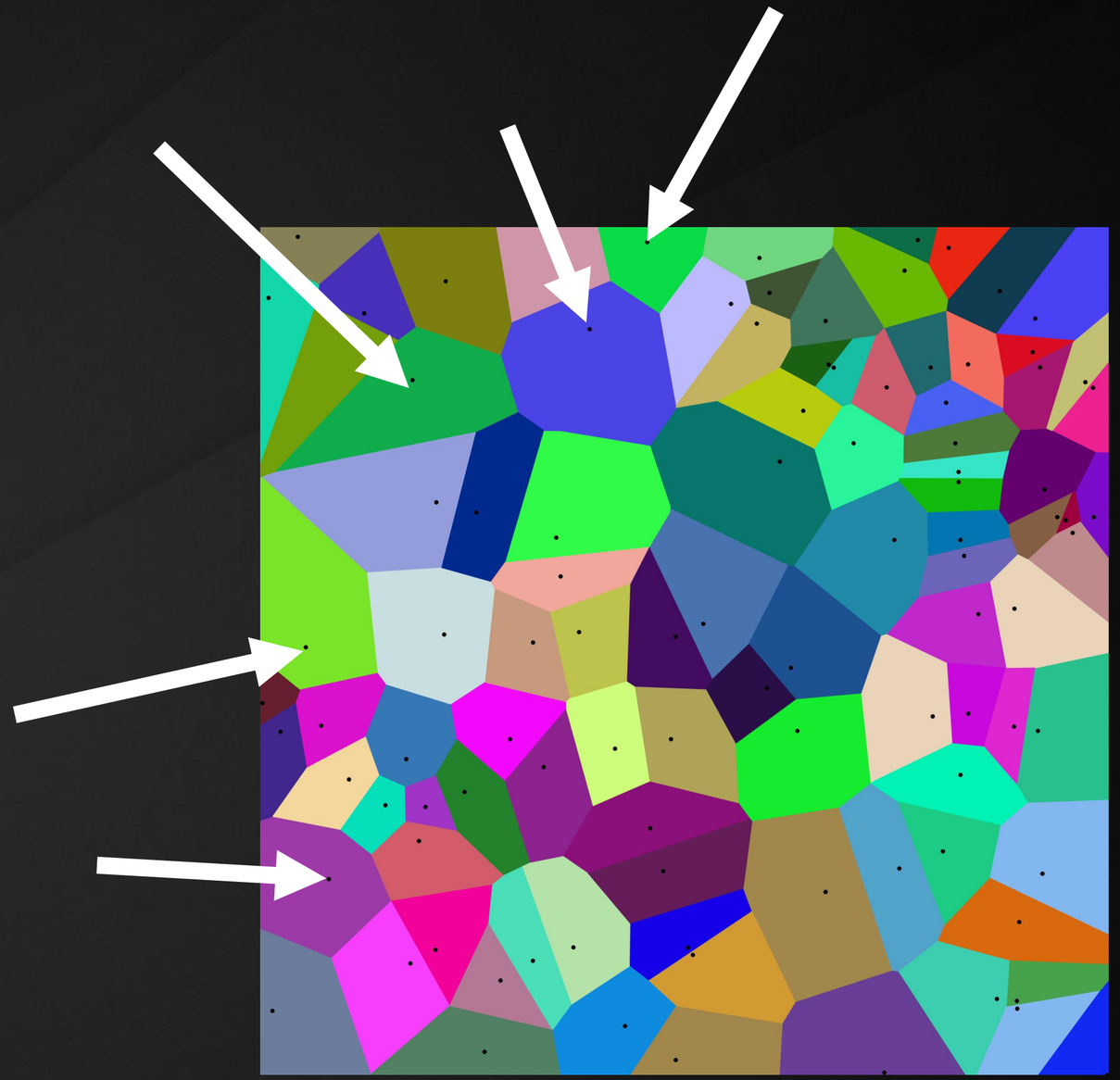
- 点の集合に対して、
  最も近い点に属するような領域分割



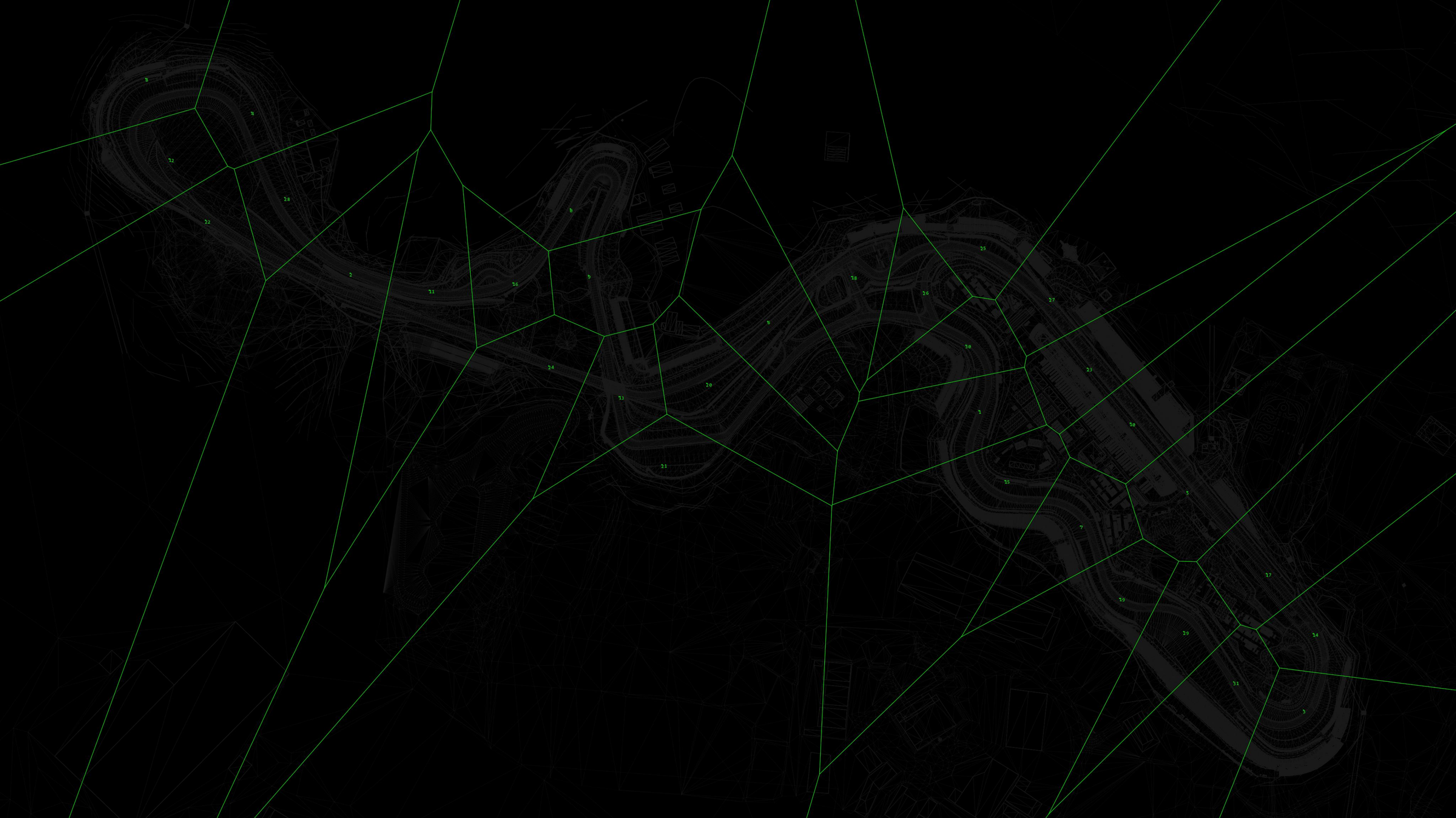
POLYPHONY™ DIGITAL

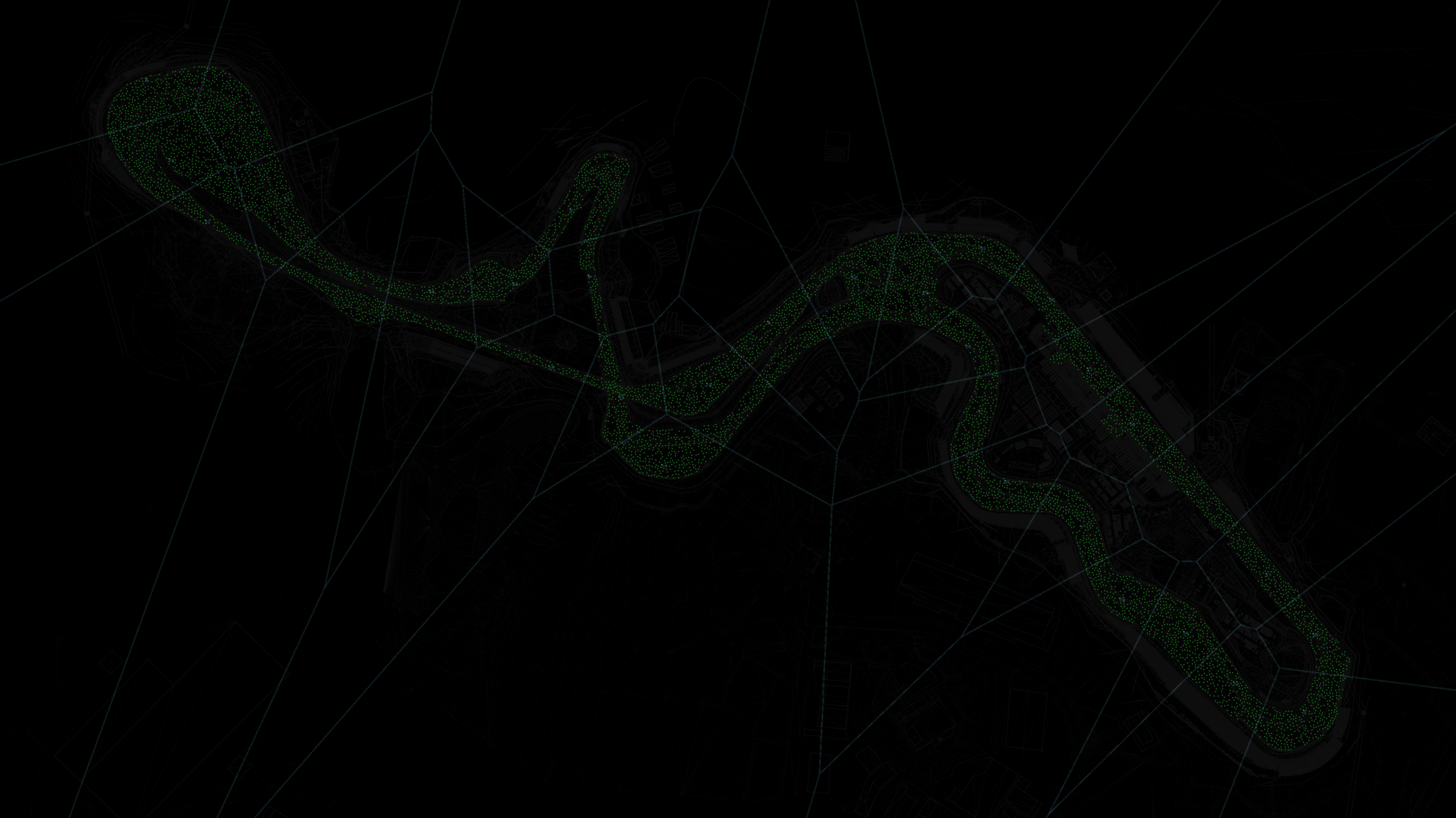
ボロノイ領域とは

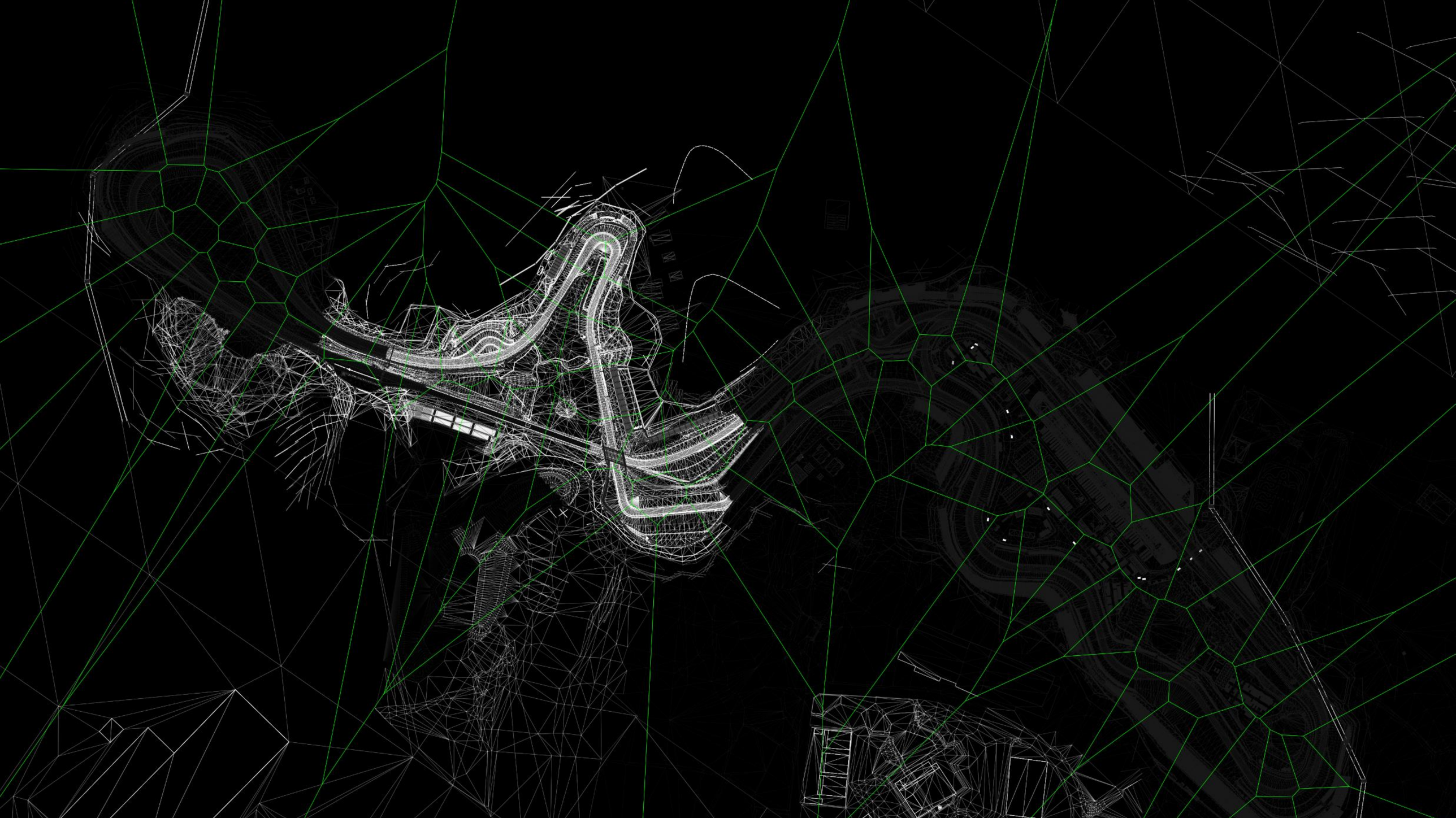
- それぞれの代表点を
  「母点」と呼ぶ

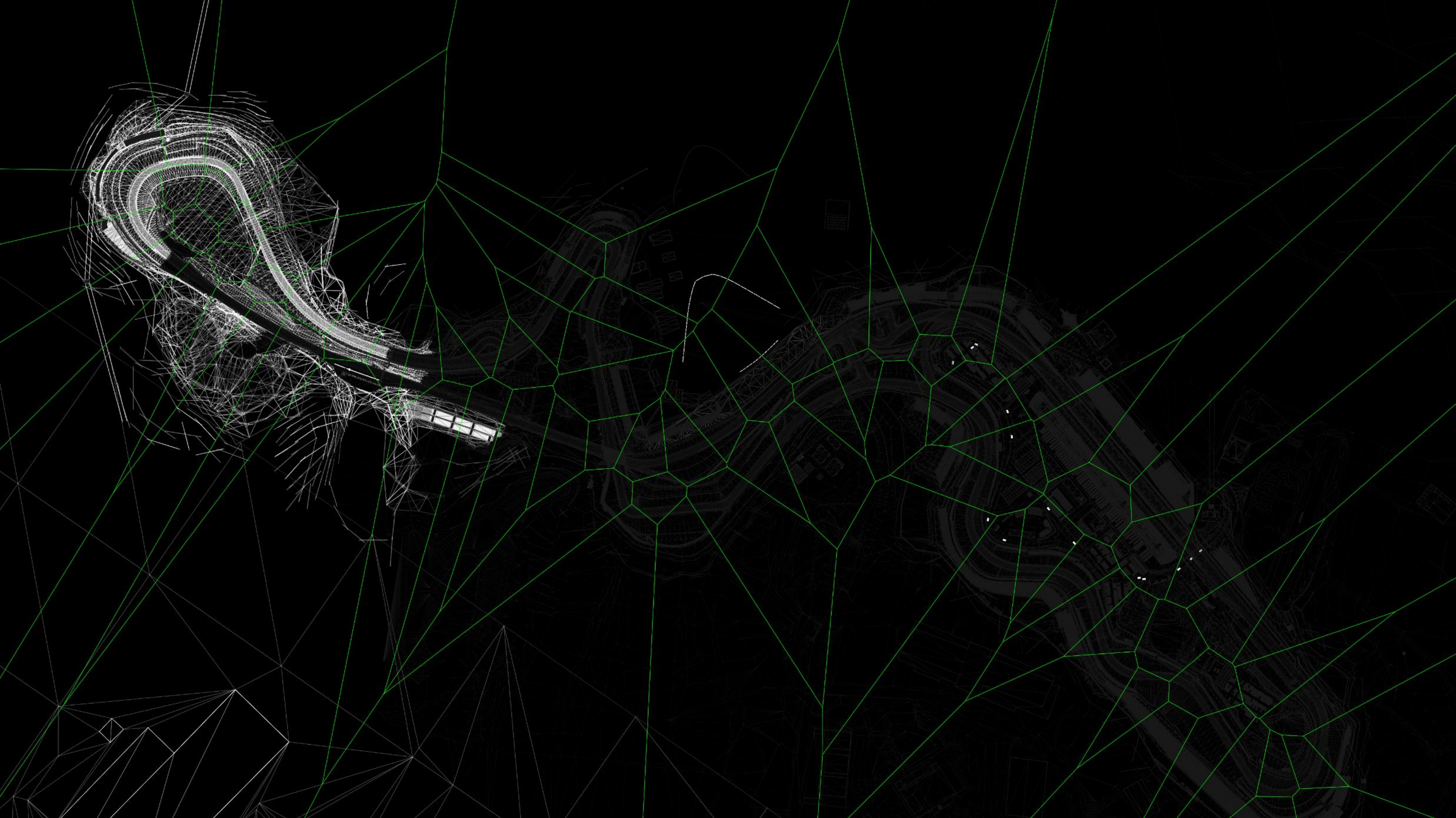


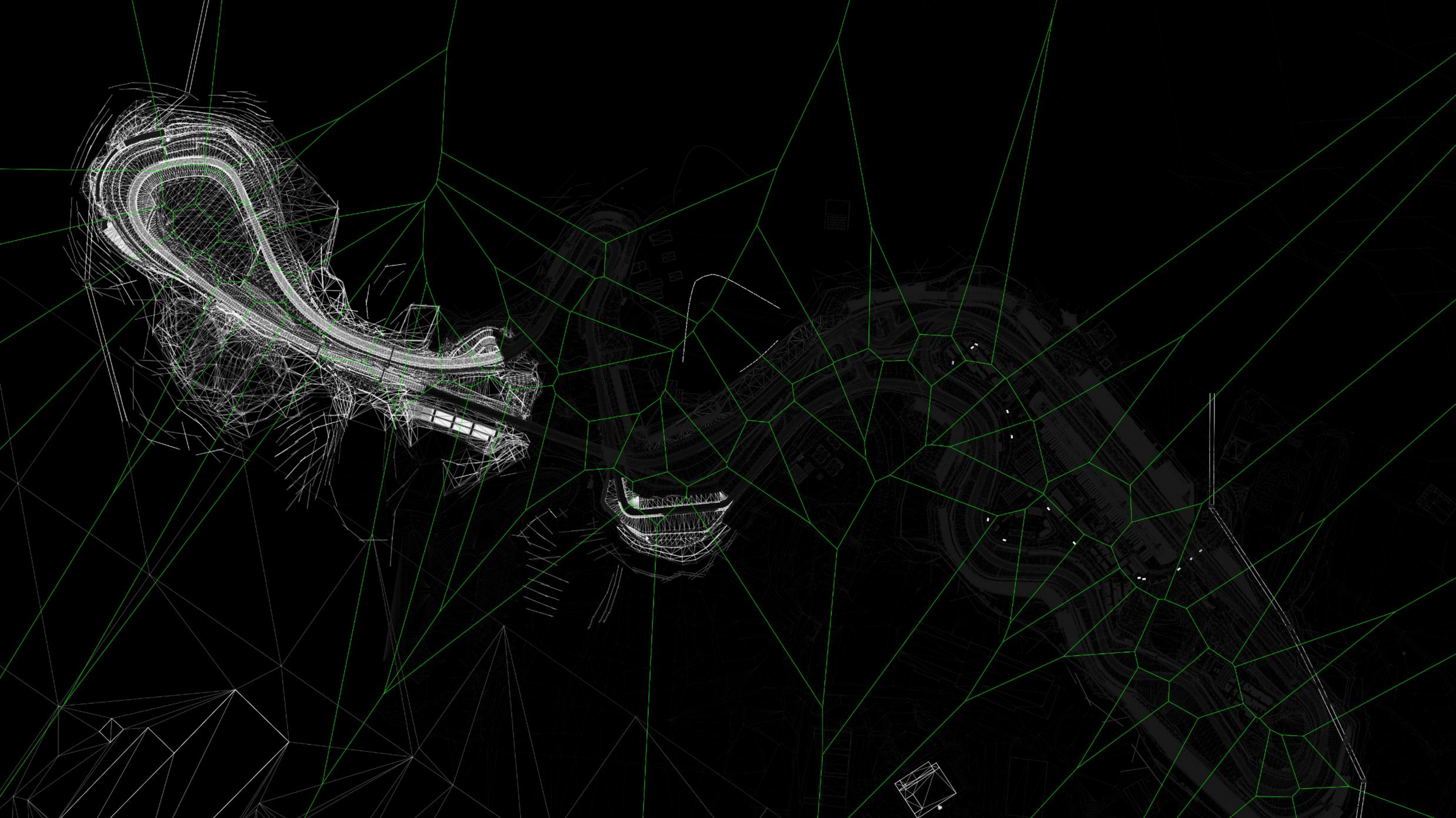
POLYPHONY DIGITAL


8
4
12
28
22
2
11
16
9
6
18
25
26
27
30
24
20
23
13
1
10
21
15
5
7
17
19
29
14
31
3







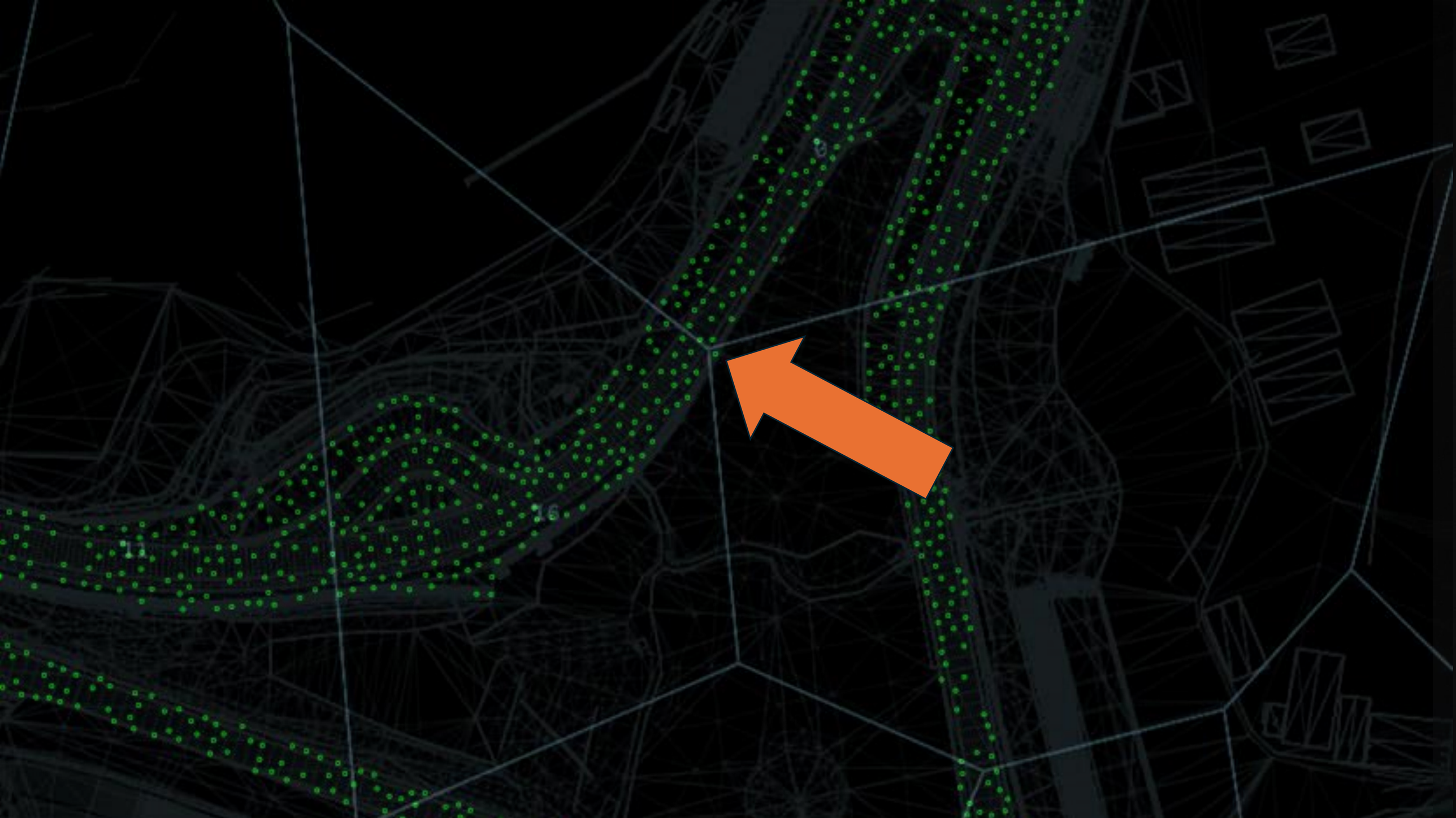


クラスタ化の目的

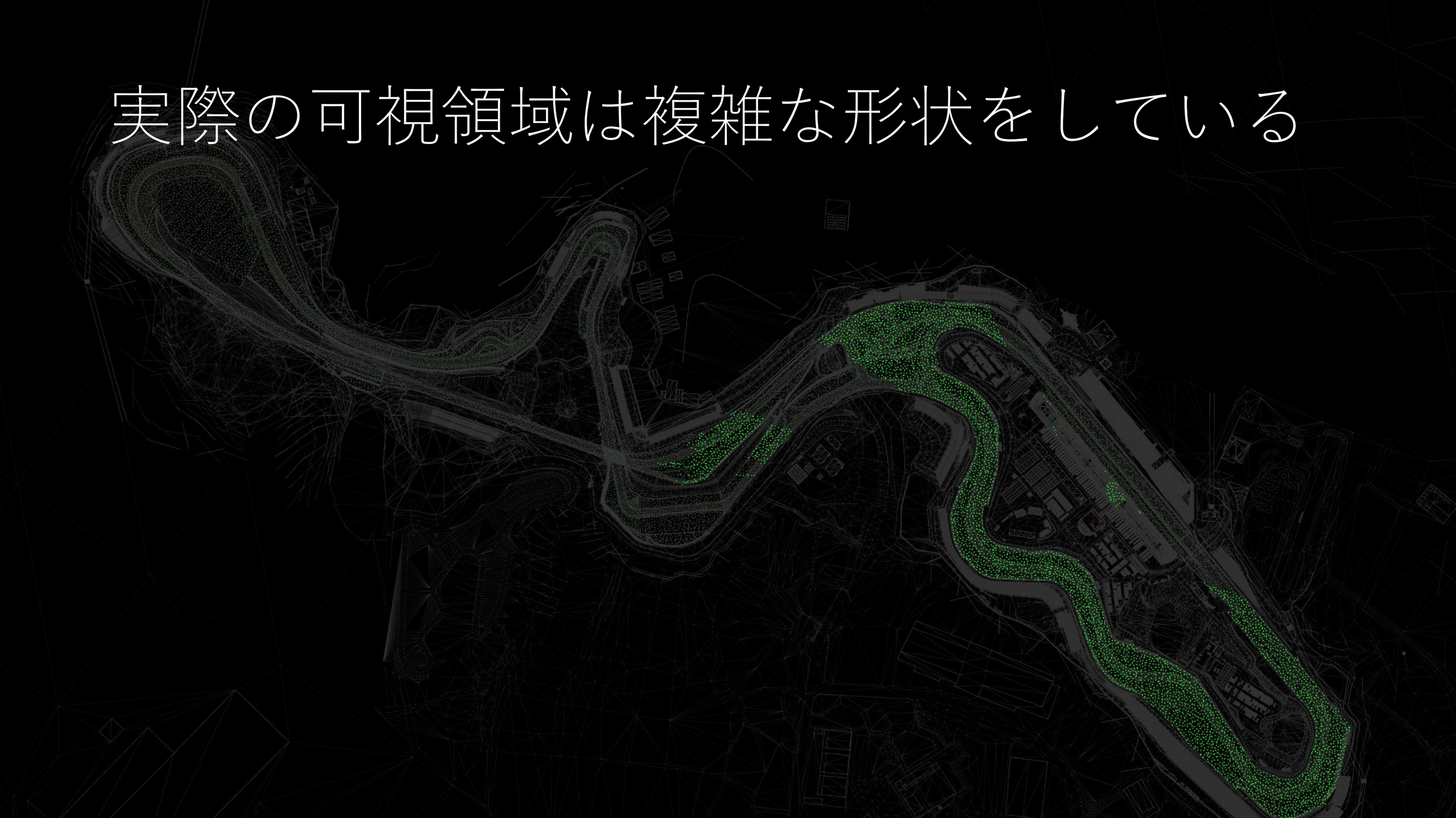
- データサイズの削減
- カメラ位置とビジョンポイントの位置の差による誤差の低減

クラスタ化のデメリット

- 原理的に不連続な変化しかできない
- ボロノイ境界がコースと一致しないことがよくある
- 代表点の数がハイパーパラメータになってしまう



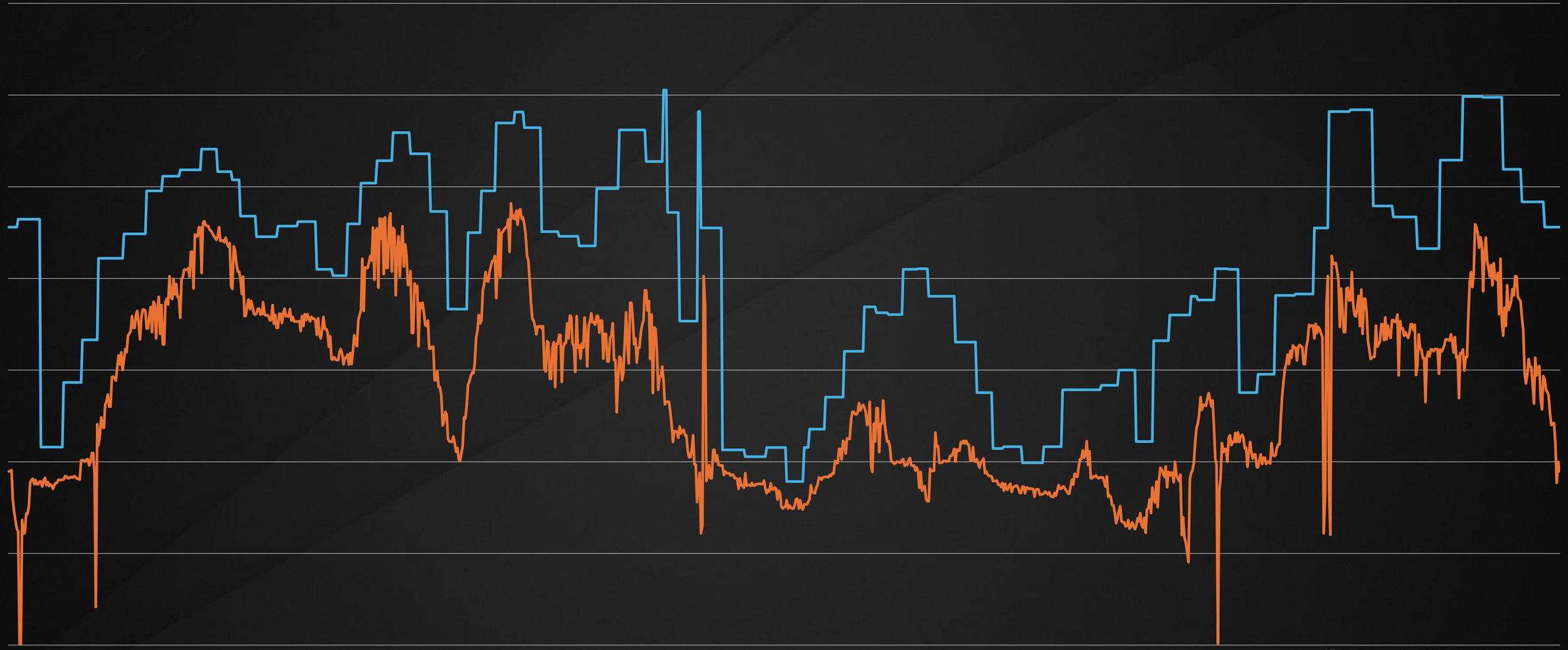
実際の可視領域は複雑な形状をしている




66
20
14
9
40
30
33
59
87
16
74
42
11
6
78
74
90
5
43
86
13
21
91
46
31
82

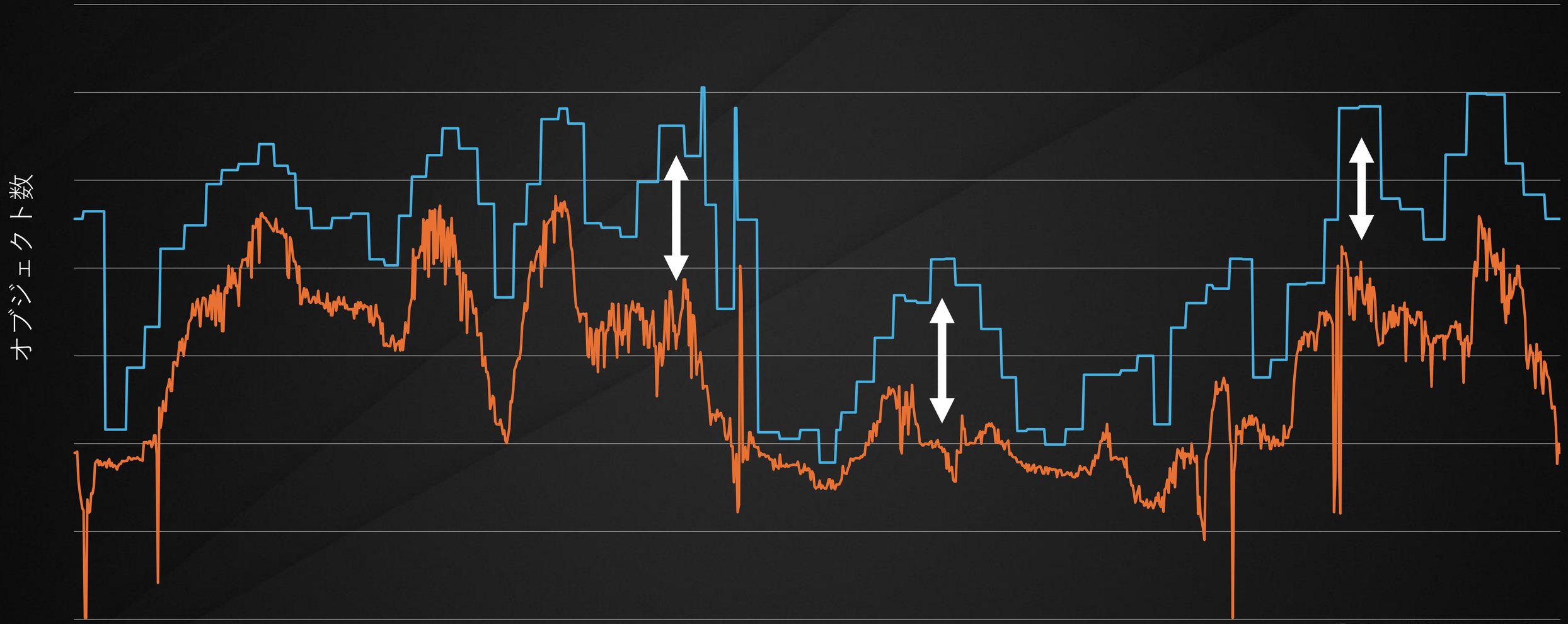
コース一周のオブジェクト数変動

コース一周のオブジェクト数変動



POLYPHONY™ DIGITAL

コース一周のオブジェクト数変動



POLYPHONY™ DIGITAL

ここまでのまとめ

- 事前計算カリングシステムは描画負荷を下げる仕組み

- グランツーリスモでは
  - 走行路面上にカメラを配置して事前にレンダリング
  - レンダリング結果から可視リストを生成
  - 結果をクラスタリングし、ポッピングなどを防いだ

- クラスタリングの欠点を解決したい。
  - 原理的に不連続な変化しかできない
  - ボロノイ境界がコースと一致しない

NeuralPVS

POLYPHONY™ DIGITAL

モチベーション

GDC Machine Learning Summit: A Robust and Fast ML-Based View of the **Frustum Culling** Method.
(堅牢で高速な機械学習ベースの視点**フラスタムカリング**手法)

ニューラルネットワークは
Occlusion Culling(**オクルージョンカリング**)
にも応用できると感じる。

モチベーション

既存のビジョンリストは、粗いポジション（エリア）と各オブジェクトの可視化状態のマッピングを保存している。

モチベーション

もしこのマッピングをニューラルネットワークに学習させることができれば、
より精度の高いカリング結果を得られるのではないでしょうか？

NeuralPVS

ポジションと各オブジェクトの可視化状態のマッピングは
NeRF(Neural Radiance Fields) [Mildenhall 21]の形に似ている。

Neural PVS: $F(x) = (v_0, v_1, \ldots, v_{N-1}), v_i \in \{0, 1\}$

Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106. https://arxiv.org/pdf/2003.08934

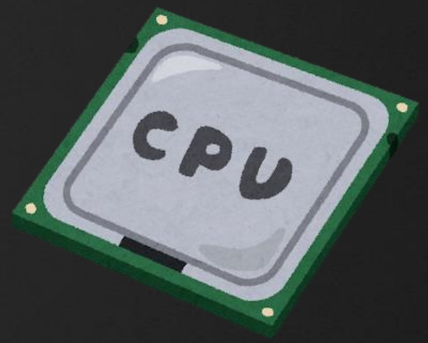
実装の概要

パイプライン開始 → オフライン処理 → ランタイム処理 → レンダリング

必要なツール

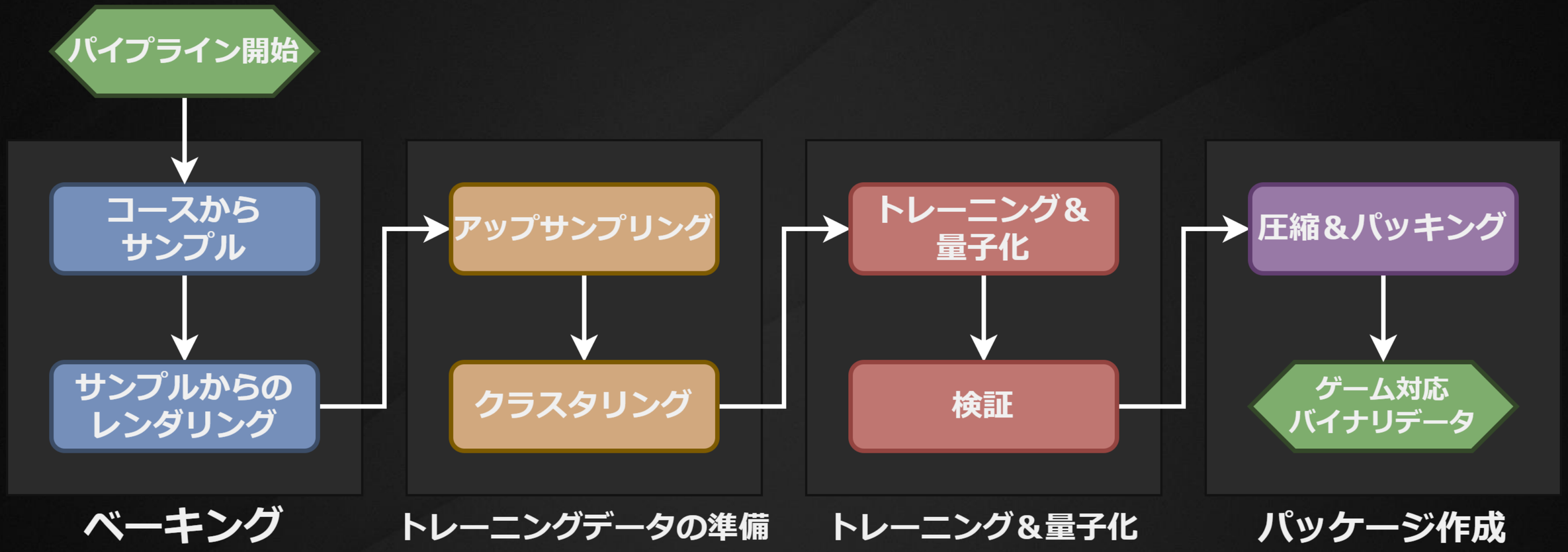
OpenGL

PYTORCH



SSE

オフライン処理

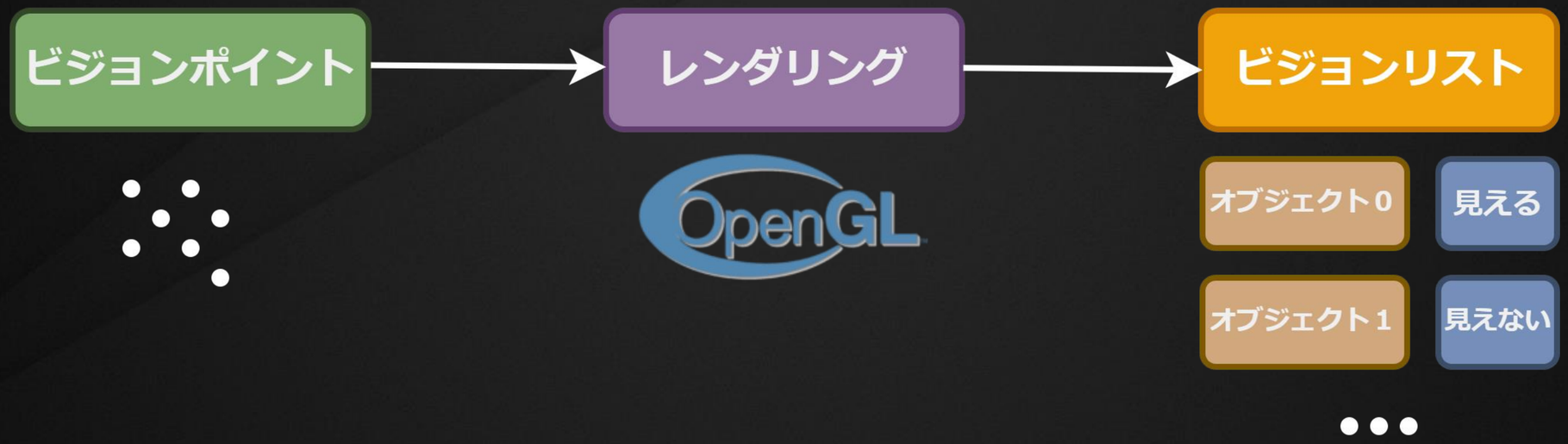


ベーキングの元データ

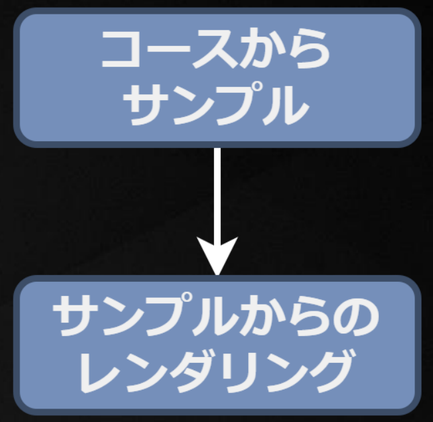
コースから
サンプル

サンプルからの
レンダリング

1. 各オブジェクトに対してquery objectを作成し、
各視点からOpenGLでレンダリングを行う。



ベーキングの元データ



2. 'GL_SAMPLES_PASSED'というクエリ結果を収集し、
描画されたピクセル数が閾値を超える場合、
可視オブジェクトリストに追加する。

POLYPHONY™ DIGITAL

ベーキングの元データ

コースから
サンプル

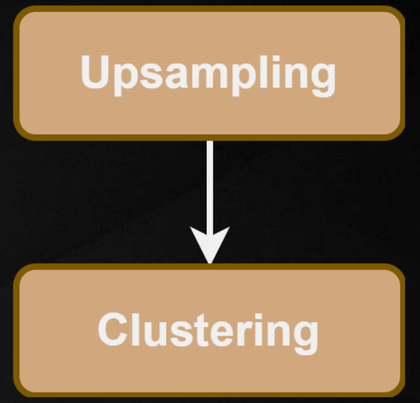
サンプルからの
レンダリング

3. 各ビジョンポイントは、それぞれビジョンリストという
見えるオブジェクトのUUIDと見えないオブジェクトのUUIDが保存している。

POLYPHONY DIGITAL

トレーニングデータの準備

トレーニングデータの準備

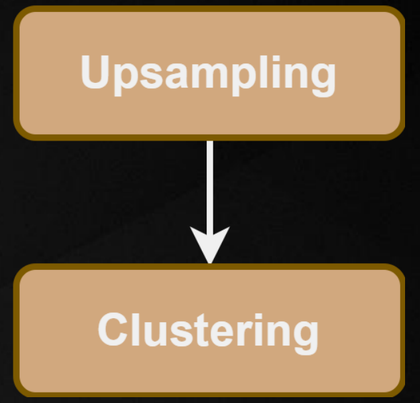


トレーニングデータの準備は、**ビジョンリスト**のを入力し、
密度が高いバイナリデータ形式の位置、重要度、可視情報を出力する。
トレーニングデータの準備は**2つの部分**に分かれる。

1. **アップサンプリング**：

   ビジョンポイントを均一に増やし、重要な**ビジョンポイント**と可視性情報を取得する。

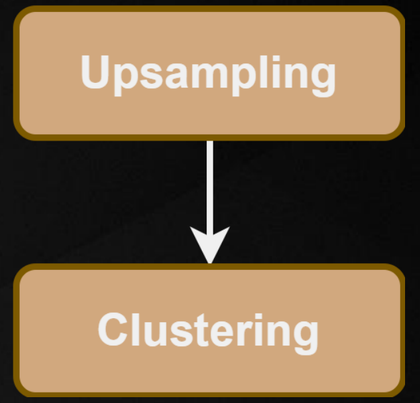
トレーニングデータの準備



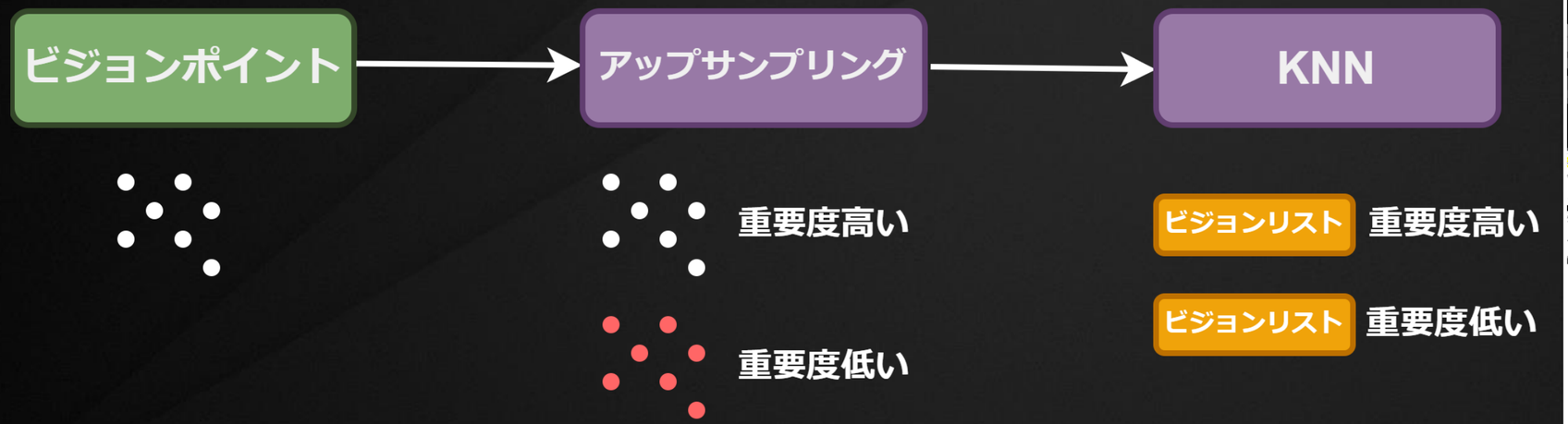
2. クラスタリング：

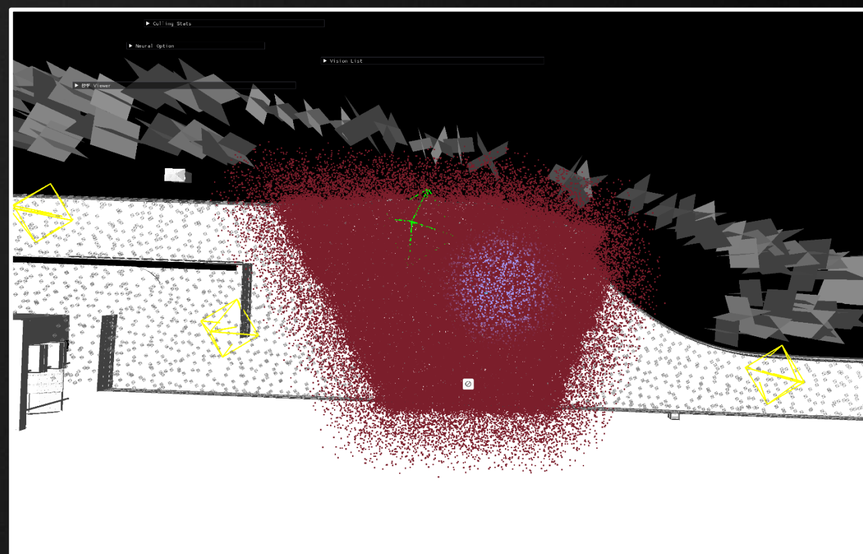
ビジョンポイントをクラスタリングし、PyTorch向けのトレーニングデータを作成する。

アップサンプリング

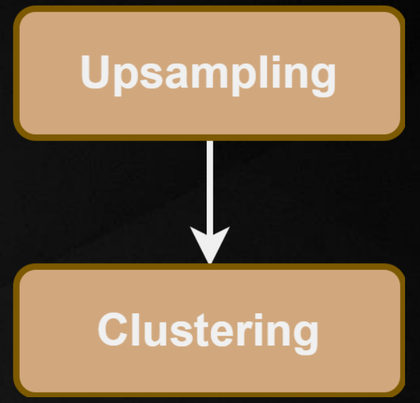


1. 各**ビジョンポイント**を高さ3～5m、
半径50cmのシリンダーとして扱い、
5000～10000回の均一サンプリングを行い、
重要なビジョンポイントを取得。

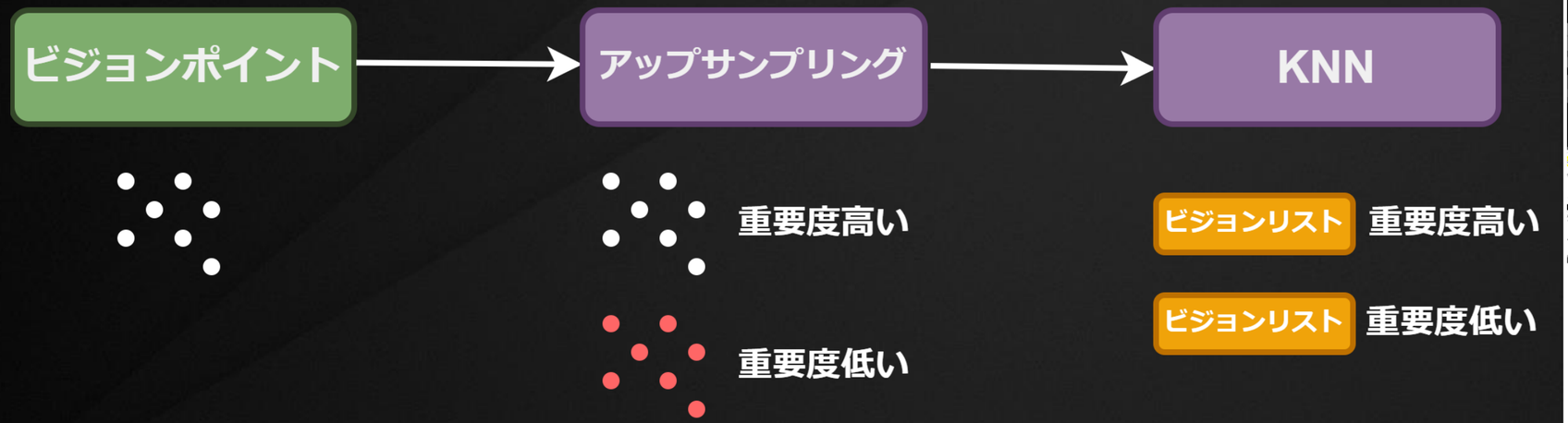


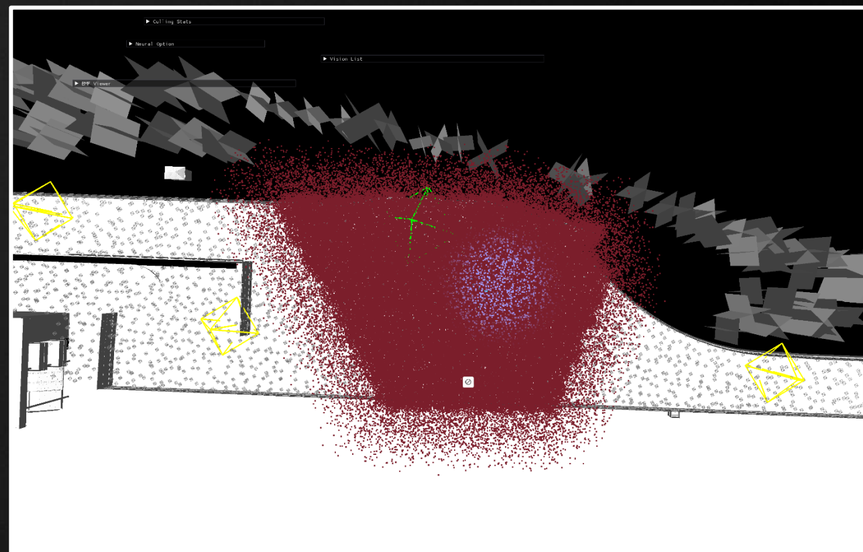


アップサンプリング

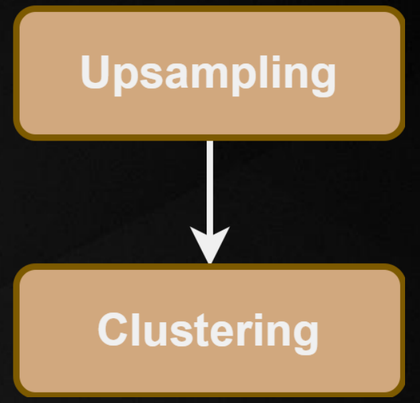


2. 各アップサンプリングされた
重要なビジョンポイントに対して、
KNN（K Nearest Neighbors）を実行し、
周りK個の**ビジョンリスト**を統合する。

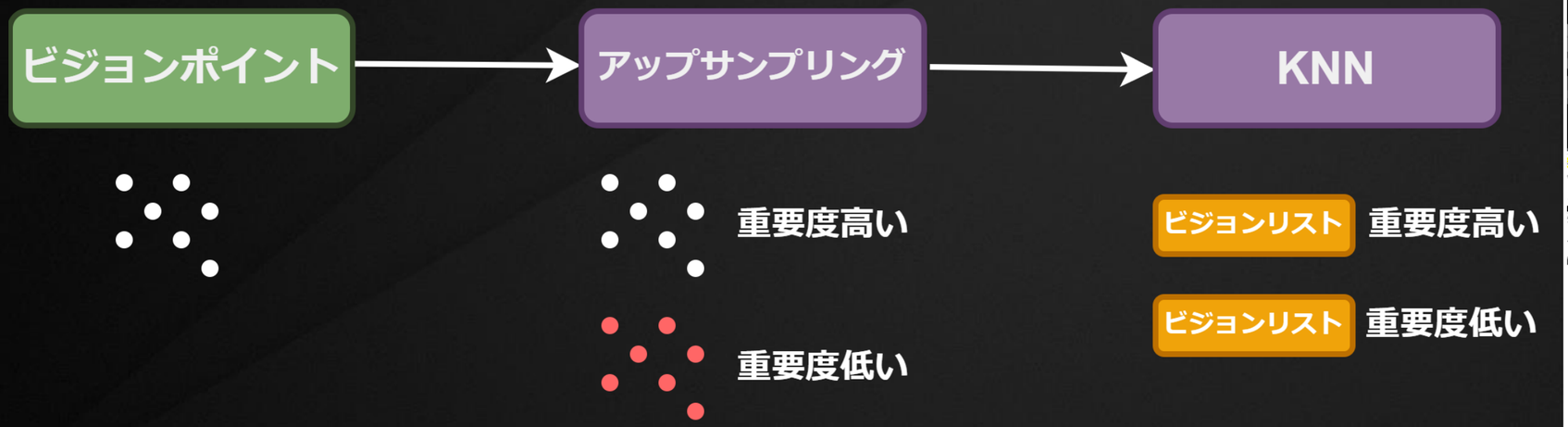


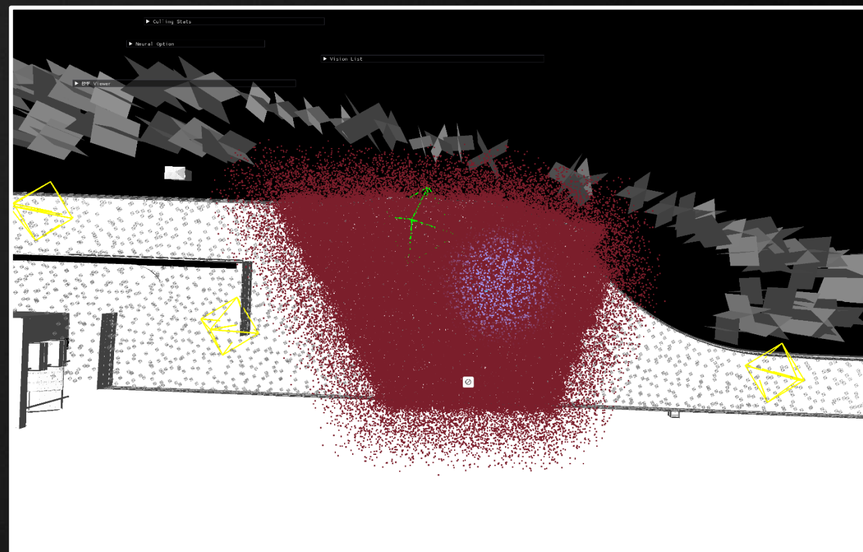


アップサンプリング

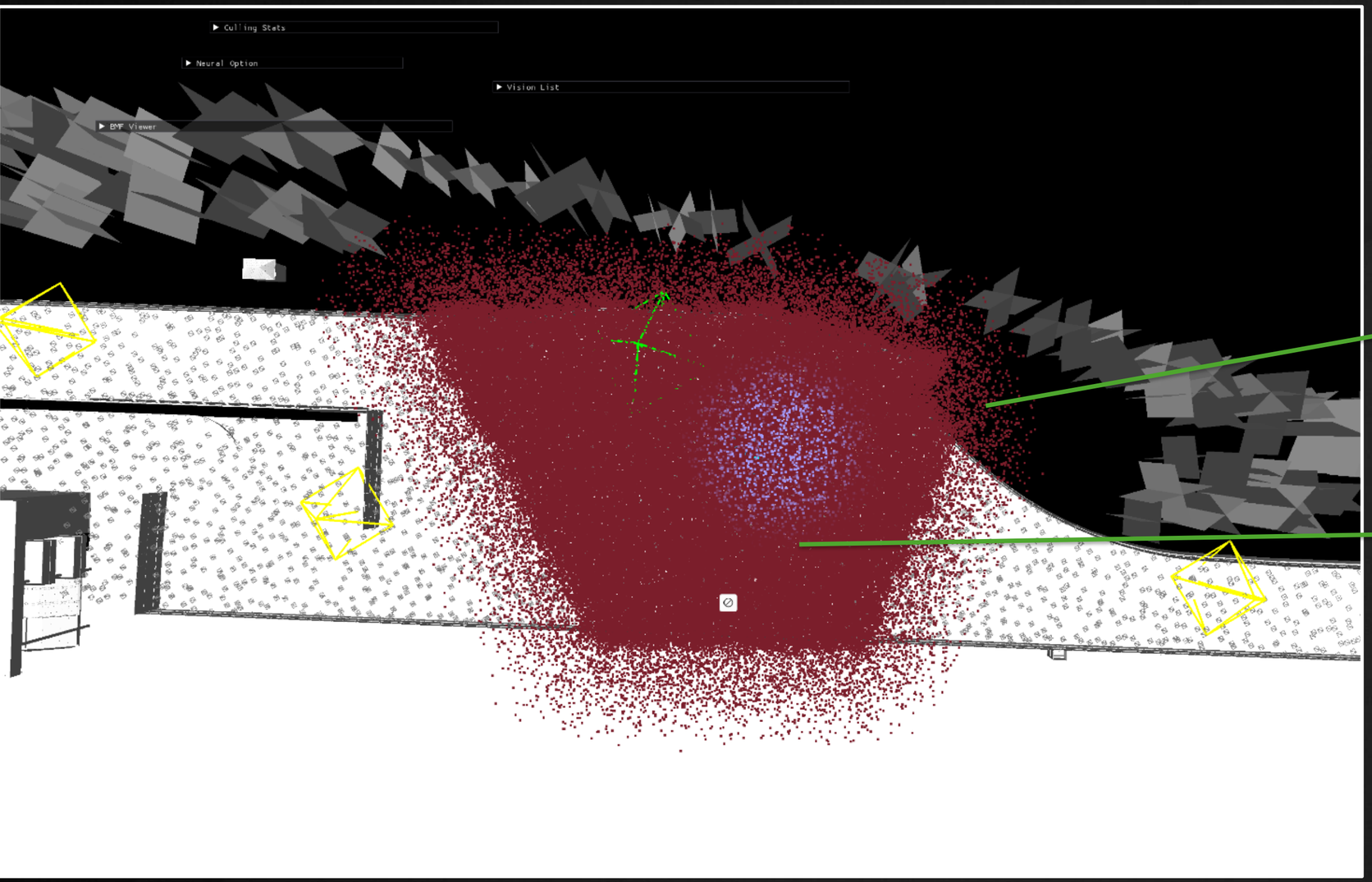


3. ステップ1と2のように、
半径を拡大しサンプリング回数を半分にして、
重要度の低いビジョンポイントを取得する。





アップサンプリング



Upsampling

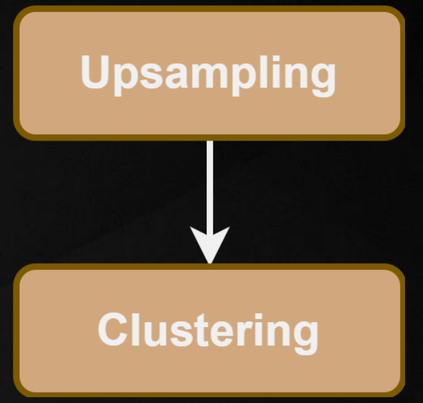
↓

Clustering

重要度の低いポイント

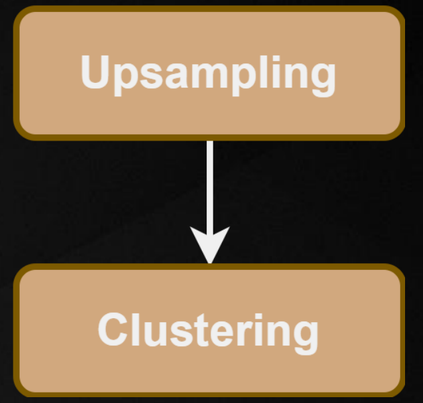
重要度の高いポイント

クラスタリング



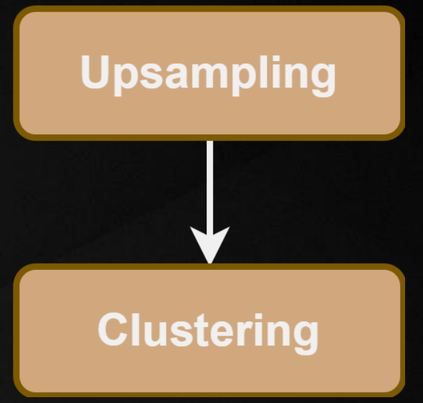
1. **重要なビジョンポイント**対して
   ポアソンサンプリングを行い、
   N個の**母点**を生成する。
   各**母点**には個別のニューラルネットワークが使用される。

クラスタリング



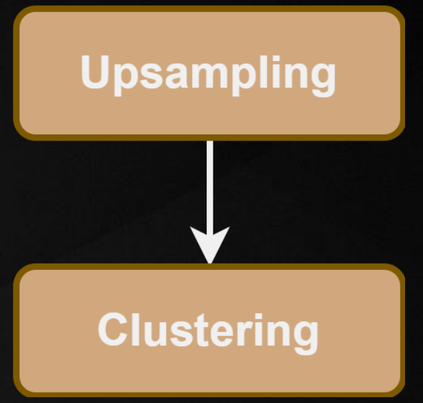
2. **重要なビジョンポイント**と
重要度の低いビジョンポイントに対して、
別々にもう一回**ポアソンサンプリング**を行い、
もっと均一な**ビジョンポイント**を取得する。

クラスタリング



3. 各**ビジョンポイント**について、
最も近い**母点**を見つけて、その母点に割り当てる。

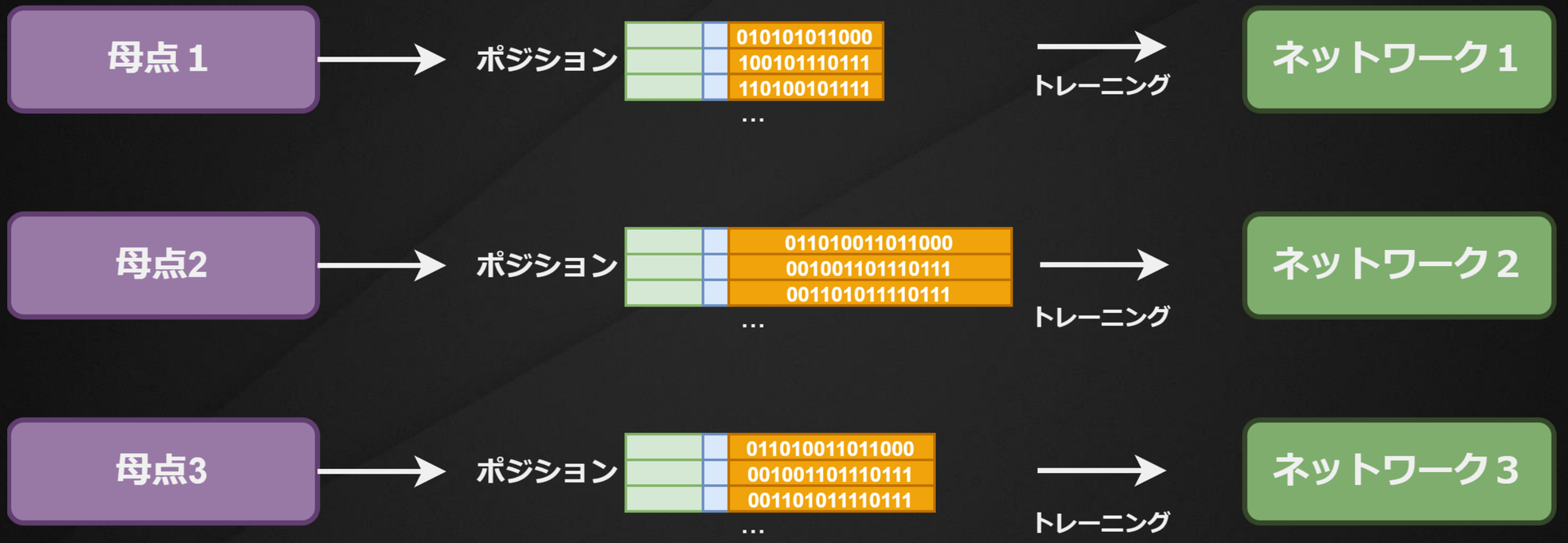
クラスタリング



4. PyTorchで読み取るために、
各**母点**内の**ビジョンポイント**について、
位置、重要度、そして0と1で表現された可視情報を
ベクトルとして出力する。

ベクトル化

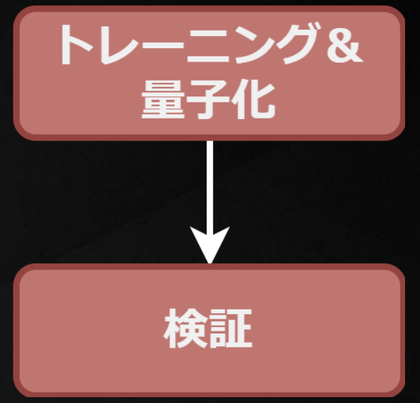
各母点には、一連のベクトルを保存する。
これらのベクトルは、位置情報とモデルの可視性（0または1）を
組み合わせたものとする。



■ ■ ■

トレーニング＆量子化

トレーニング＆量子化

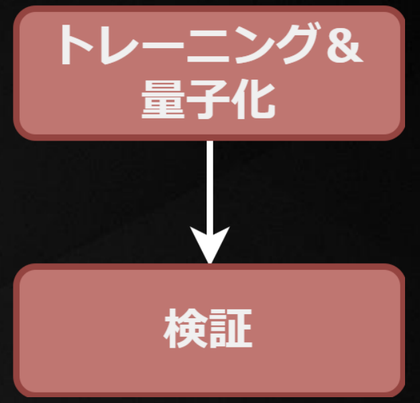


トレーニング＆量子化はに4つの要素があります。

1. ネットワーク構造：

   最適なネットワーク構造について説明する。

トレーニング＆量子化

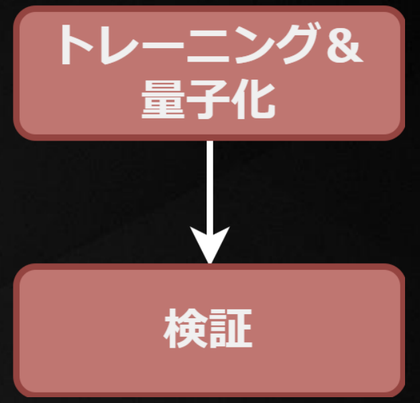


トレーニング＆量子化はに4つの要素があります。

2. ネットワークパラメータの探索：

検索でネットワークの幅や深さなどの最適なパラメータを見つける。

トレーニング＆量子化

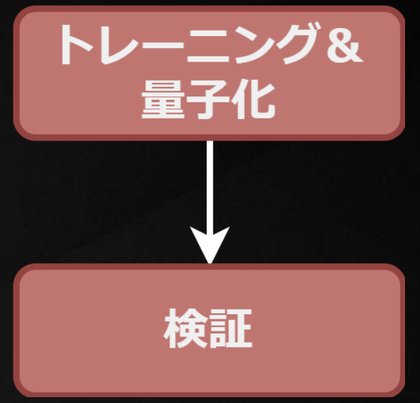


トレーニング＆量子化はに**4つ**の要素があります。

3. **量子化**:

   可能な限り精度を保ちながら、量子化によりネットワークパラメータの**サイズを大幅に削減**する。

トレーニング＆量子化



トレーニング＆量子化はに4つの要素があります。

4. **検証**:

リコール率を調整し、モデルが保守的にカリングされるようにする。

トレーニング

POLYPHONY™ DIGITAL

Neural PVS

トレーニング＆量子化

検証

Neural PVS: $F(x) = (v_0, v_1, \ldots, v_{N-1}), v_i \in \{0, 1\}$

ネットワーク

ポジション

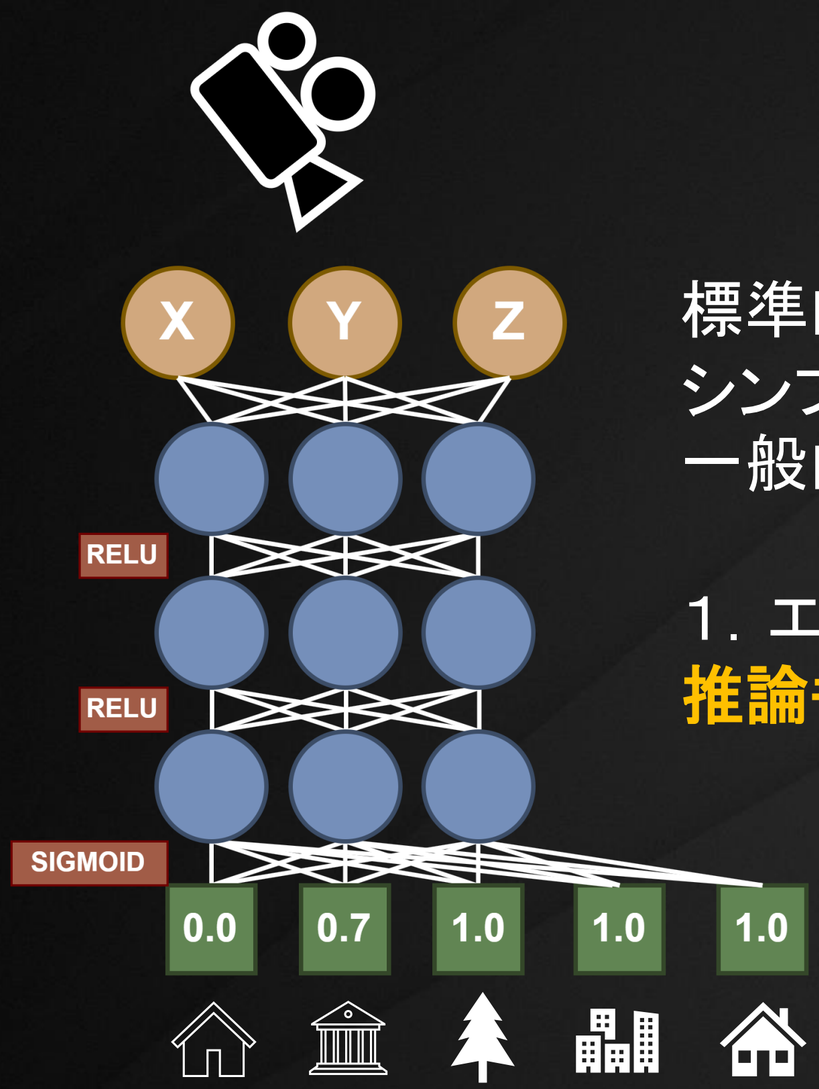
可視性（二値分類として扱う）

オブジェクトの数

このフィールドは、ポジションと各オブジェクトの可視化状態のマッピングであり、
NeRFで一般的に使用されるいくつかの構造が使え、
二値分類モデルとしてトレーニングできる。

標準的なMLP、**SIREN**、**フーリエ特徴マッピング**について分析する

標準的なMLP(Vanilla MLP)



標準的なMLPは、ReLUを活用した
シンプルな多層パーセプトロン構造を持ち、
一般的なタスクに広く使用されている。

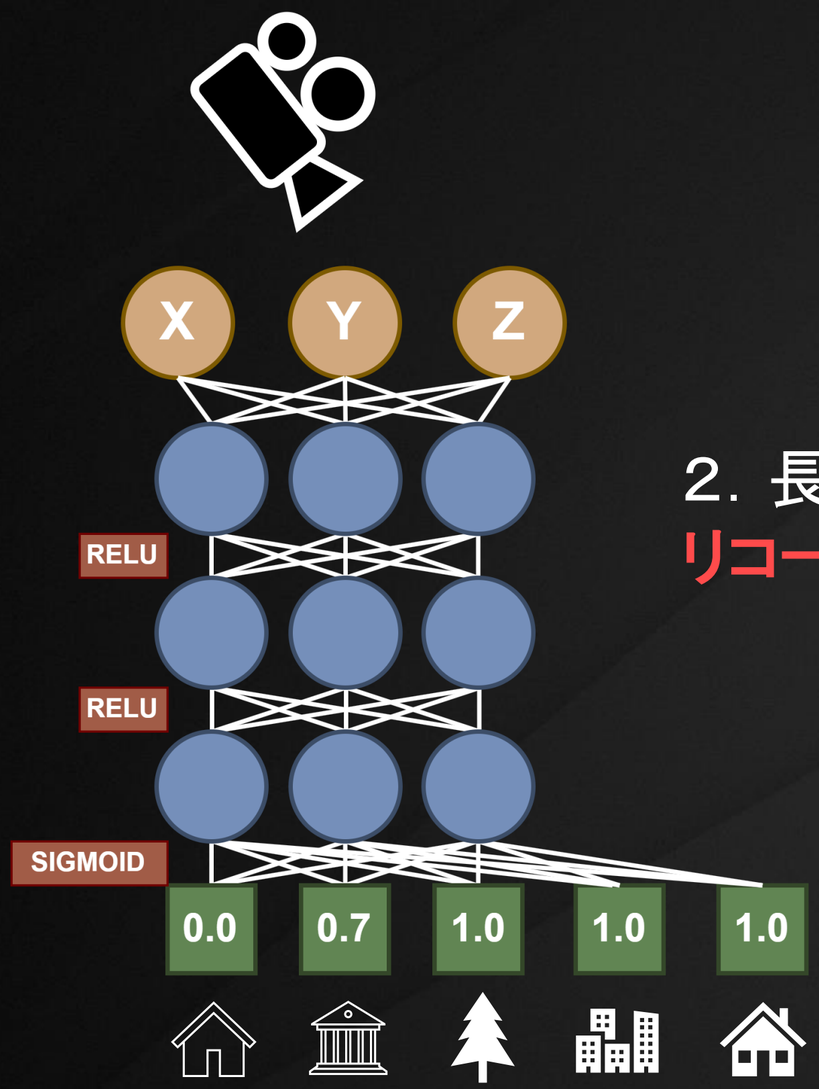
1. エポックあたりの学習が最も速く、
推論も高速である。

トレーニング＆
量子化

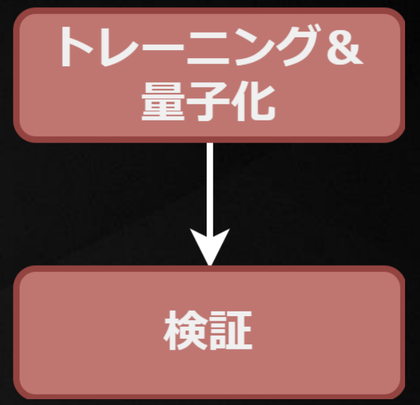
↓

検証

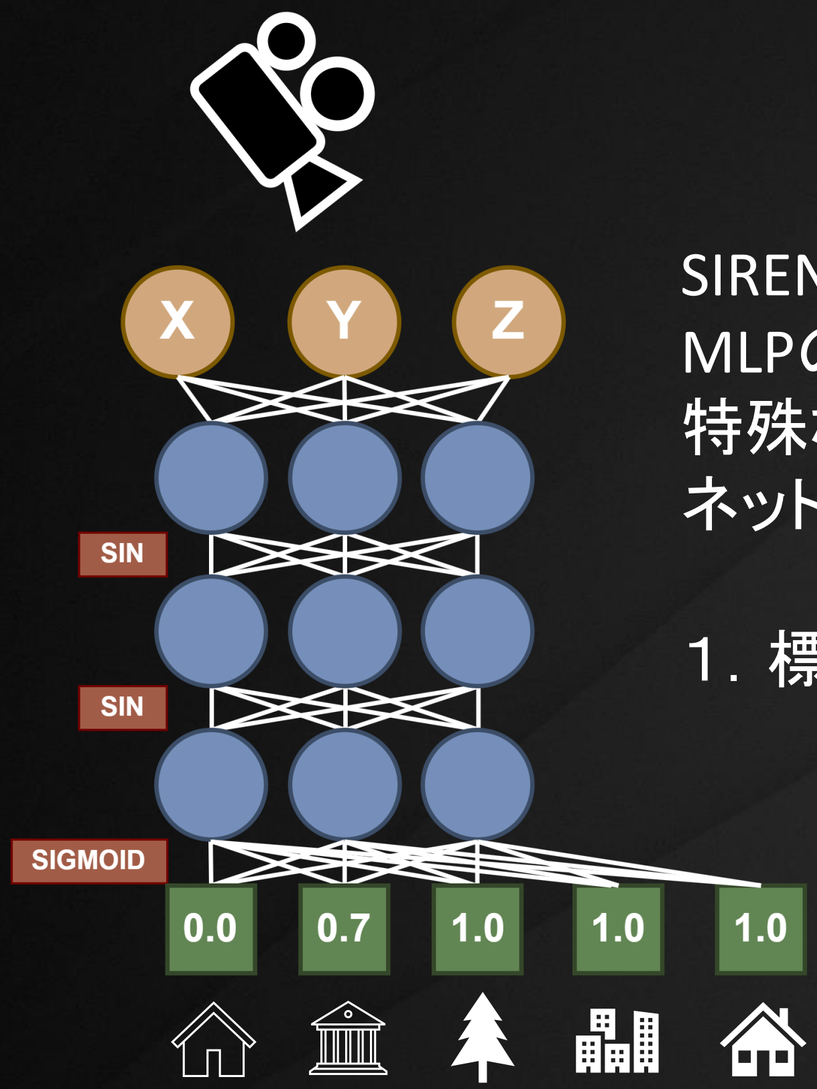
標準的なMLP(Vanilla MLP)



2．長時間トレーニングさせても
リコール率と精度のバランスに限界がある。

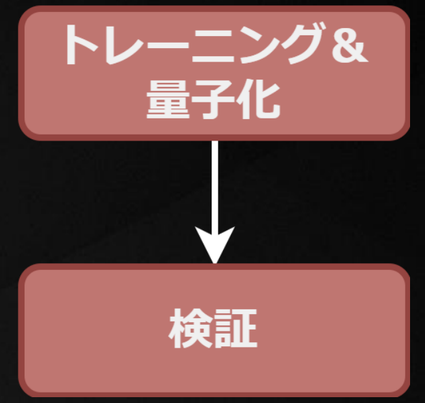


SIREN



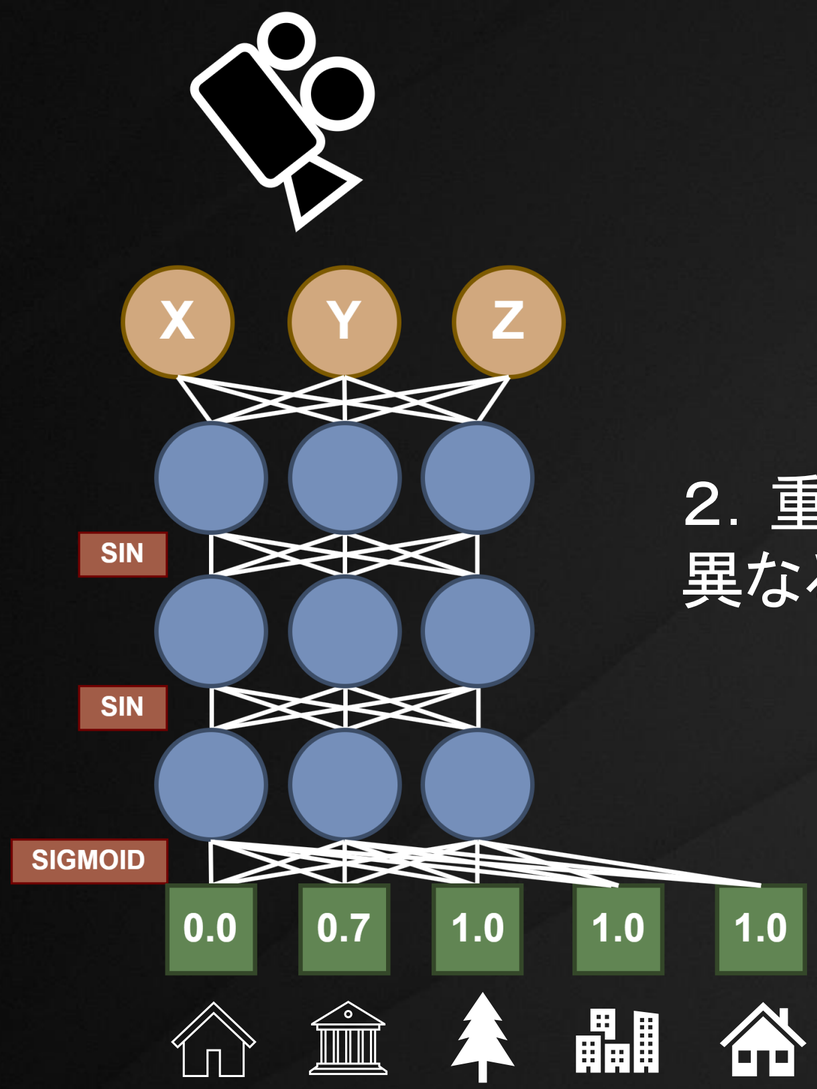
SIREN [Sitzmann20]は、
MLPのReLUをSin関数に置き換え、
特殊な重みの初期化方法を適用する
ネットワークである。

1．標準的なMLPよりも**表現力が高い**。



Sitzmann, Vincent, et al. "Implicit neural representations with periodic activation functions." *Advances in neural information processing systems* 33 (2020): 7462-7473.

SIREN



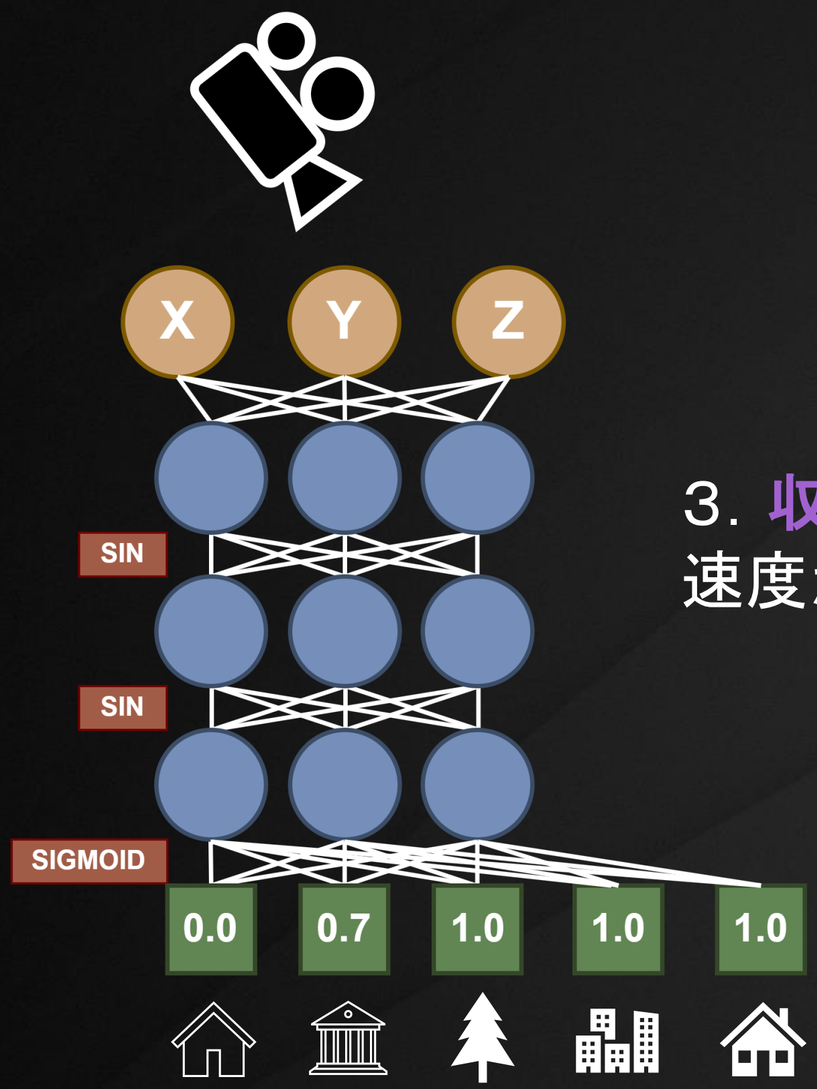
2. 重みの**初期値に敏感**で、
異なる母点に対しては学習が**不安定**。

トレーニング&
量子化

↓

検証

SIREN



3. **収束が難しく**、同等の品質を求めると速度が十分ではない。

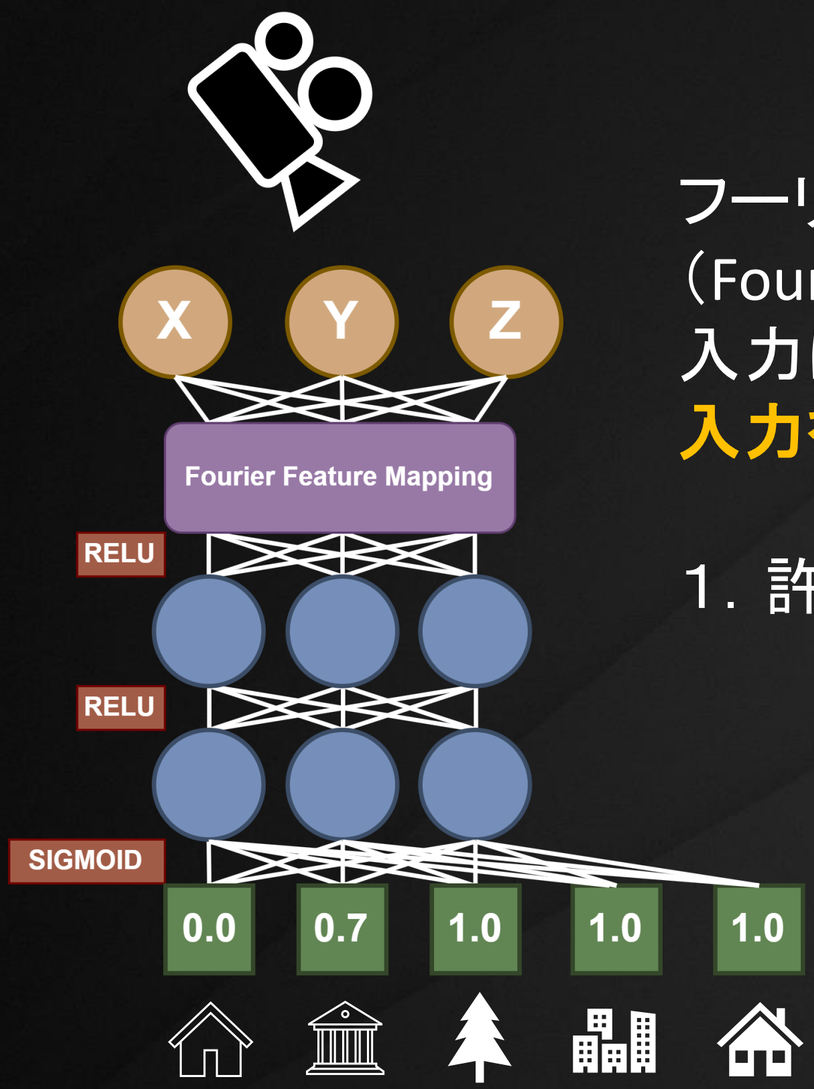
トレーニング＆量子化

↓

検証

POLYPHONY™ DIGITAL

フーリエ特徴マッピング



フーリエ特徴マッピング
（Fourier Feature Mapping）[Tancik20]は、
入力にガウス行列を乗算し、sinおよびcosの演算を適用して、
入力を高周波を含む潜在空間にマッピングする手法である。

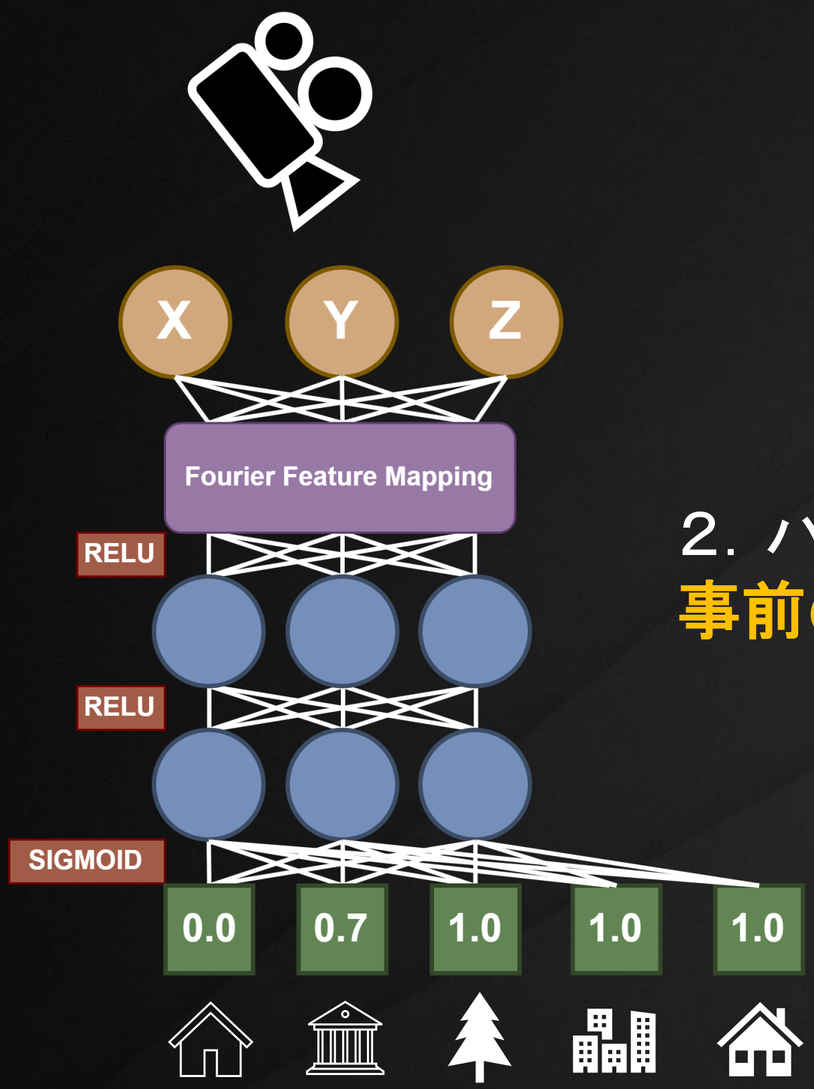
1．許容範囲内の結果に**最も早く収束**。

Tancik, Matthew, et al. "Fourier features let networks learn high frequency functions in low dimensional domains." Advances in neural information processing systems 33 (2020): 7537-7547.

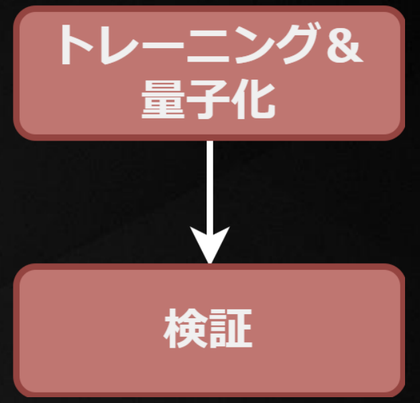
トレーニング＆
量子化

検証

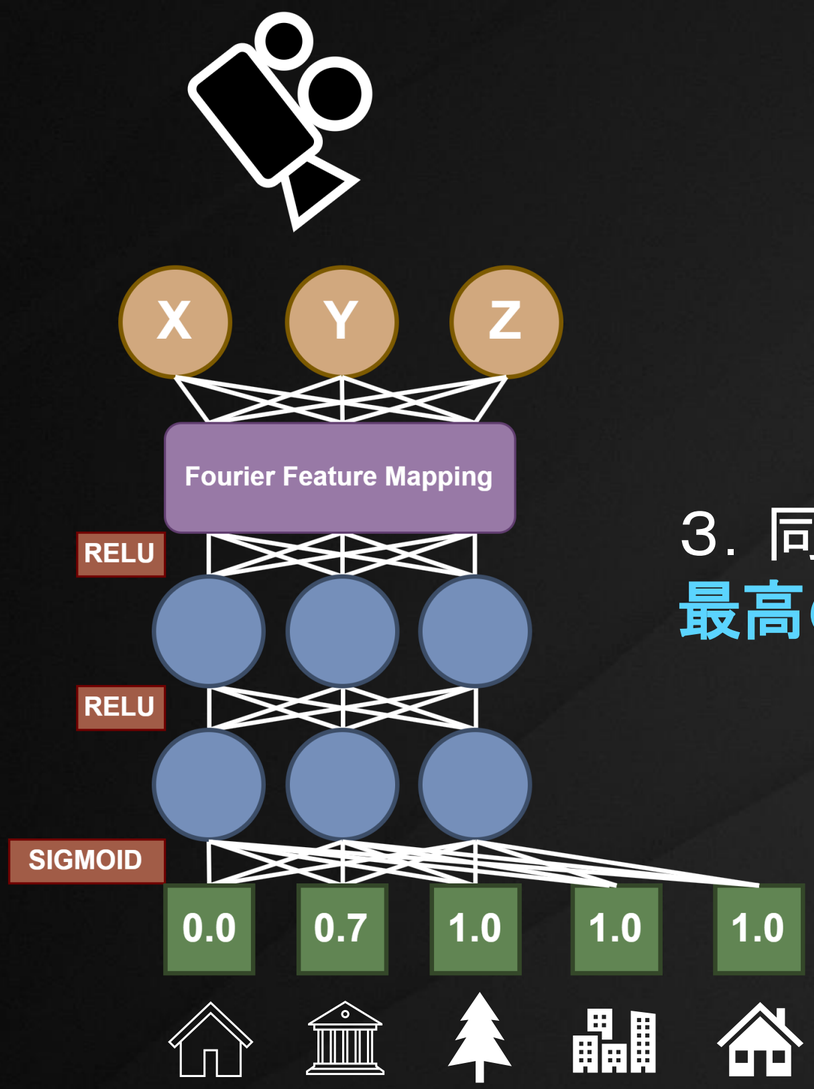
フーリエ特徴マッピング



2. ハイパーパラメータに敏感であるが、
事前の調査で最適なパラメータを見つけることが可能。



フーリエ特徴マッピング



3．同等のパラメータ量では、
最高のリコール率と精度を持っている。

トレーニング＆
量子化

↓

検証

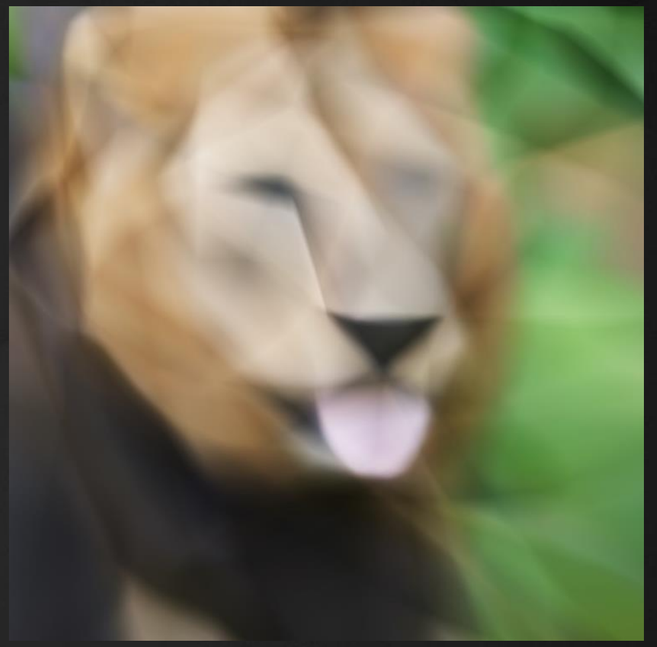
フーリエ特徴マッピング

トレーニング＆
量子化

↓

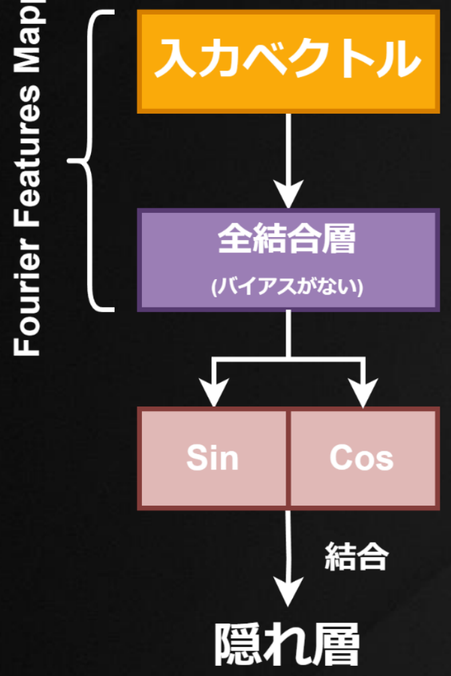
検証

フーリエ特徴マッピングあり



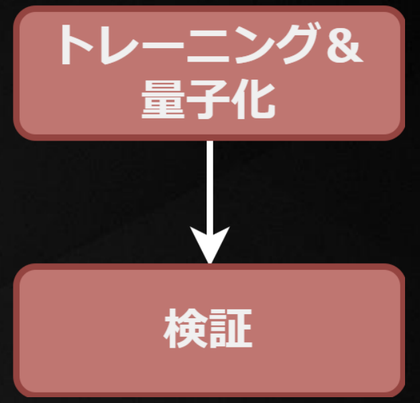
フーリエ特徴マッピングない

フーリエ特徴マッピング



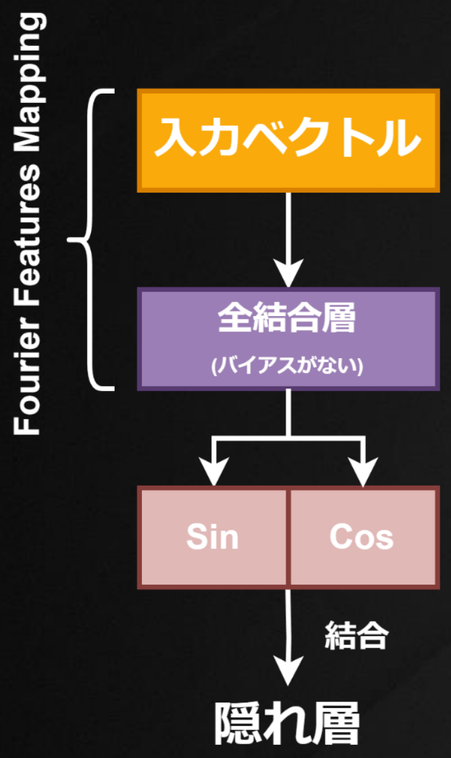
フーリエ特徴レイヤー:

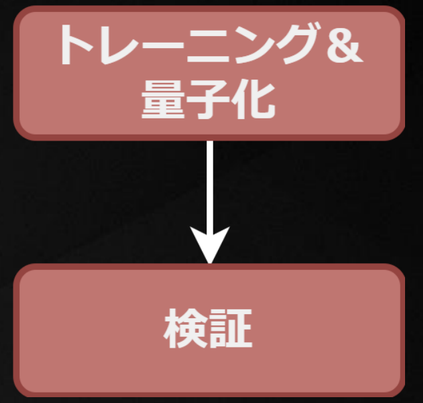
パラメータが固定されたバイアスなしの全結合層で、
出力をSinとCos通して得られた2つのベクトルを結合して、
次の層への入力とする。



POLYPHONY™ DIGITAL

フーリエ特徴マッピング





初期化:

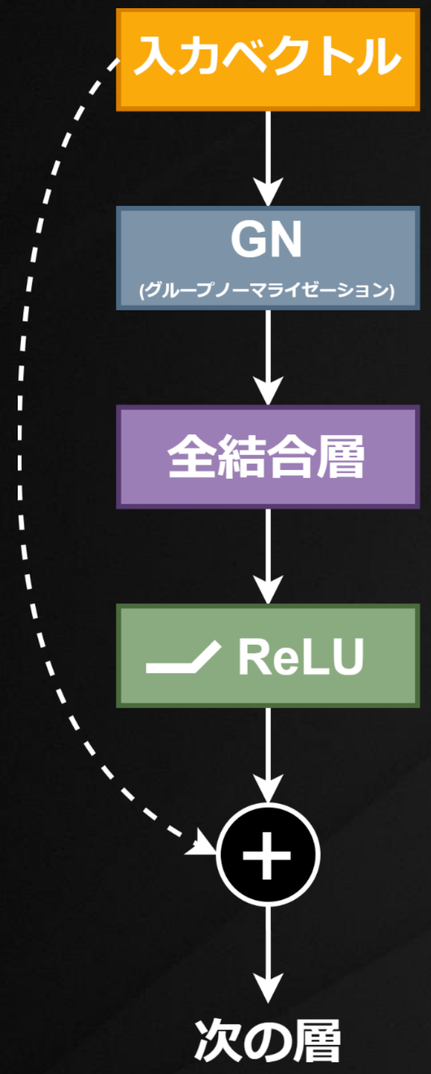
```
self.sigma_g = sigma_g
self.fft_mapping = nn.Linear(self.input_dim, self.half_pe_dim, bias = False)
nn.init.normal_(self.fft_mapping.weight.data, 0.0, self.sigma_g) # << create after default initalization avoid error
self.fft_mapping.weight.requires_grad = False # freeze this layer! we need the high frequency embedding!
#
```

Forward:

```
x = self.fft_mapping(network_input[:,0:3]) # [B, i = 3] -> [B, D//2]
x =  torch.cat([torch.sin(x), torch.cos(x)], dim = -1)
```

POLYPHONY™ DIGITAL

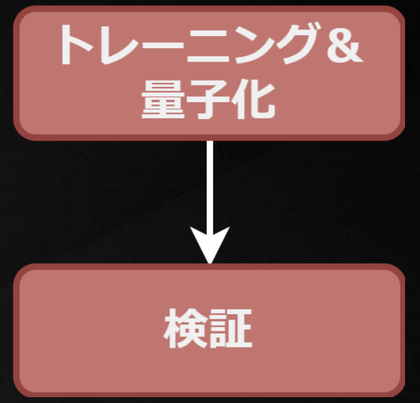
隠れ層



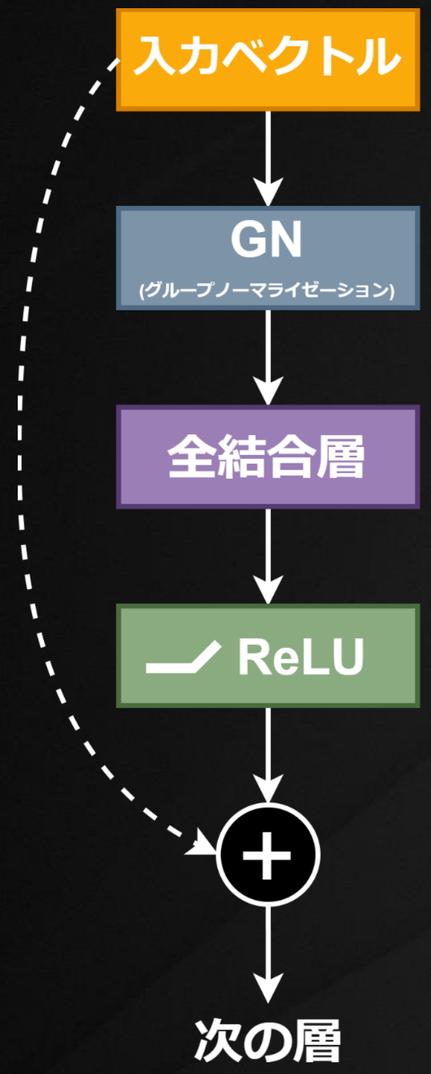
隠れ層:

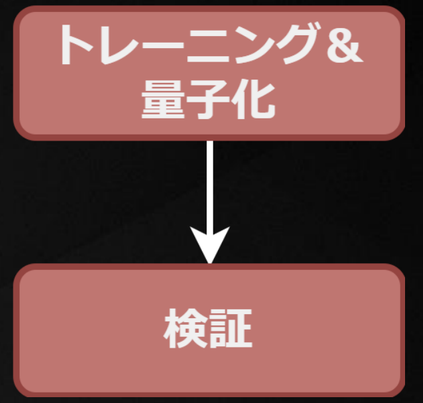
隠れ層にはさまざまなニューラル演算子の組み合わせを使用できるが、一連の組み合わせを試した結果、
この構造が最も優れた性能を発揮できる。

1. **残差構造**
   高速な計算のため、単純に前の層の出力と入力を加算して出力する。



隠れ層

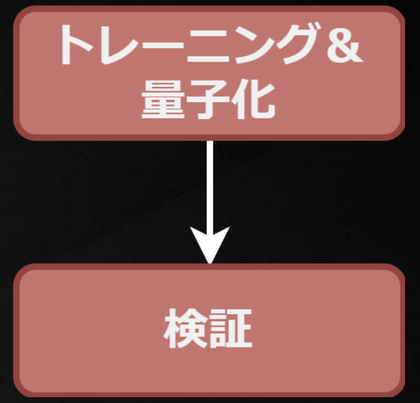




2. 正規化

BN（バッチ正規化）とGN（グループ正規化）を比較した結果、
GNが本タスクに最適であることが分かる。
GNはBNとほぼ同じ計算コストでありながら、より速い収束速度を持っている。
また、実験結果によると、4チャンネルのGNがさらに速い収束速度を示した。

損失関数



BCEWithLogitsLoss

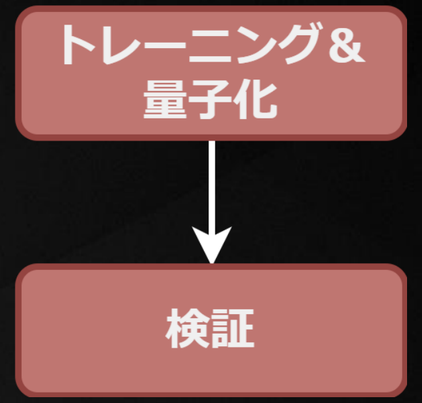
学習時には最後の層にシグモイドを使用しない。推論時には、
可視化のために手動でシグモイドを追加する必要がある。

この損失関数は、**数値の安定性が高く**、
長時間トレーニング中の数値エラーを防止することができる。

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right],$$

BCEWithLogitsLoss — PyTorch 2.5 documentation

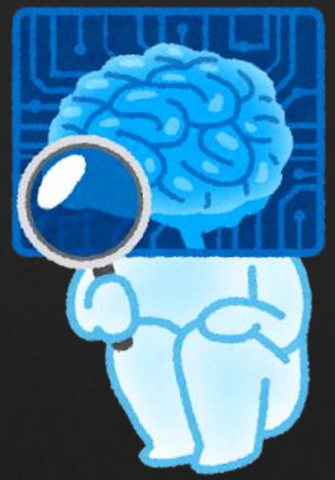
ネットワークパラメータの探索



各母点ごとに個別のネットワークを持っている。
しかし、各母点の複雑さは異なる。
固定されたパラメータ量で異なる母点をトレーニングすると、
最終的な可視化フィールドの品質にばらつきが生じる可能性がある。

ネットワークのコンパクトさと品質を確保しつつ、
可能な限り高速に学習させたいと考える。

そのため、ネットワークパラメータの探索メカニズムが必要である。



ネットワークパラメータの探索

トレーニング＆
量子化

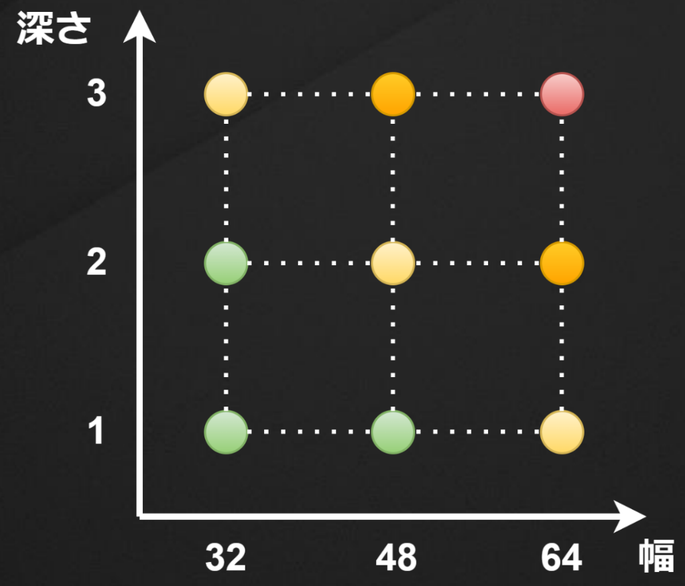
↓

検証

1．可変パラメータ

ネットワークの幅と深さはパフォーマンスと容量に大きな影響を与える。

最適な幅と深さを見つけることで
トレーニングの時間と精度のバランスを取ることができる。



POLYPHONY™ DIGITAL

ネットワークパラメータの探索

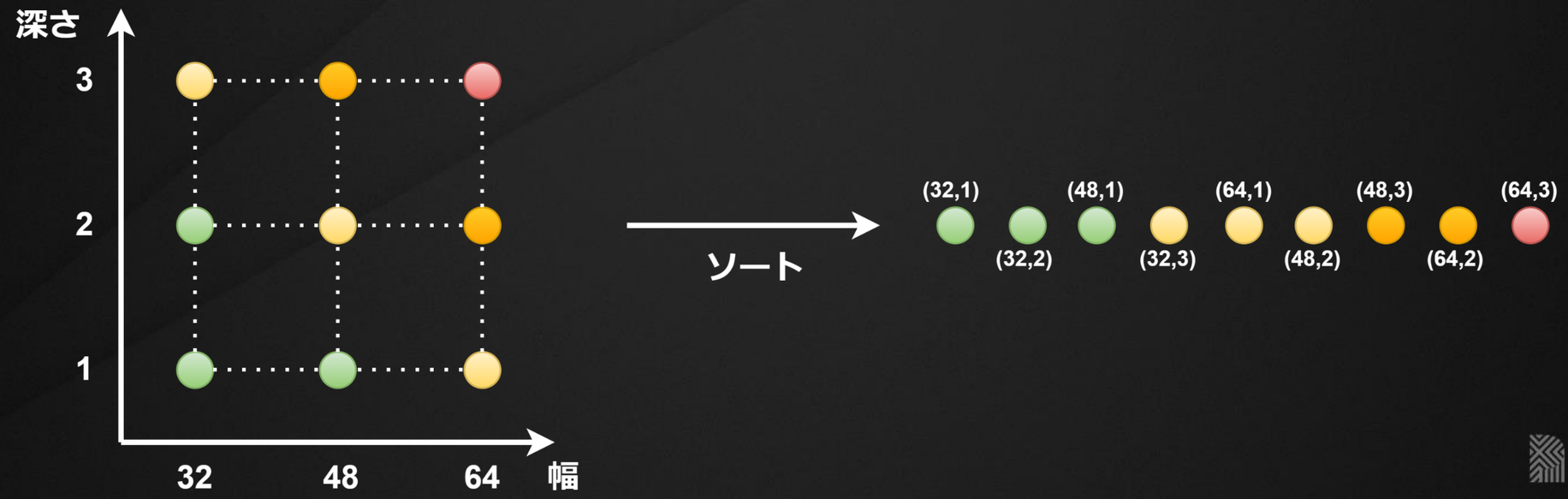
トレーニング＆
量子化

↓

検証

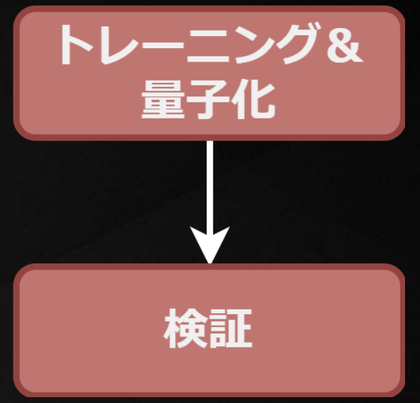
2. グリッドサーチ

幅と深さの範囲と固定間隔を指定し、パラメータグリッドを形成する。
このグリッドの各点が1つのネットワークを表し、
パラメーターのサイズの昇順に並べて1つずつ学習する。



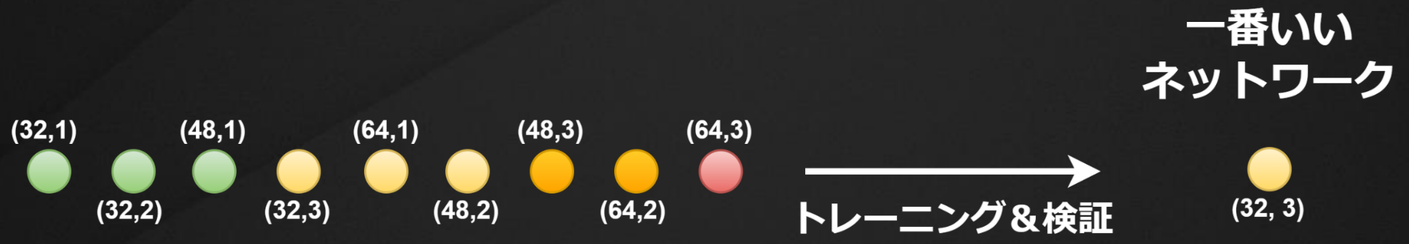
POLYPHONY™ DIGITAL

ネットワークパラメータの探索

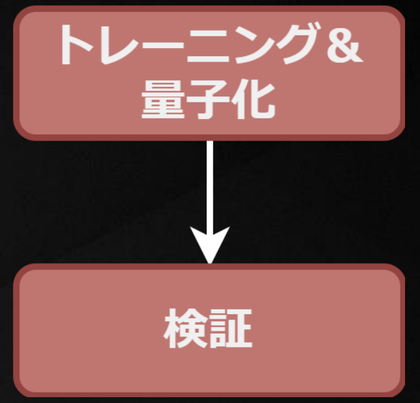


3．Early stop

学習中、各エポックで検証を行い、現在のネットワークが要件を
満たしている場合、学習を停止し、そのネットワークを結果として保存する。



トレーニング環境

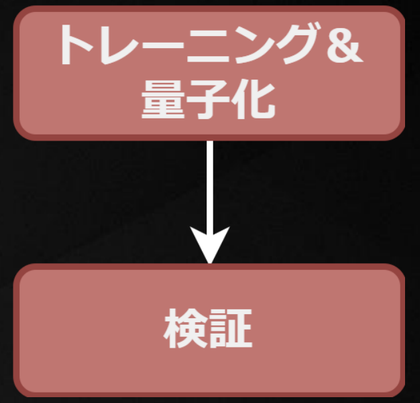


パラメータ数をとても少なく保っているため、
CPUでのトレーニングのほうが高速であった。

クラスタ単位で分散並列化している。

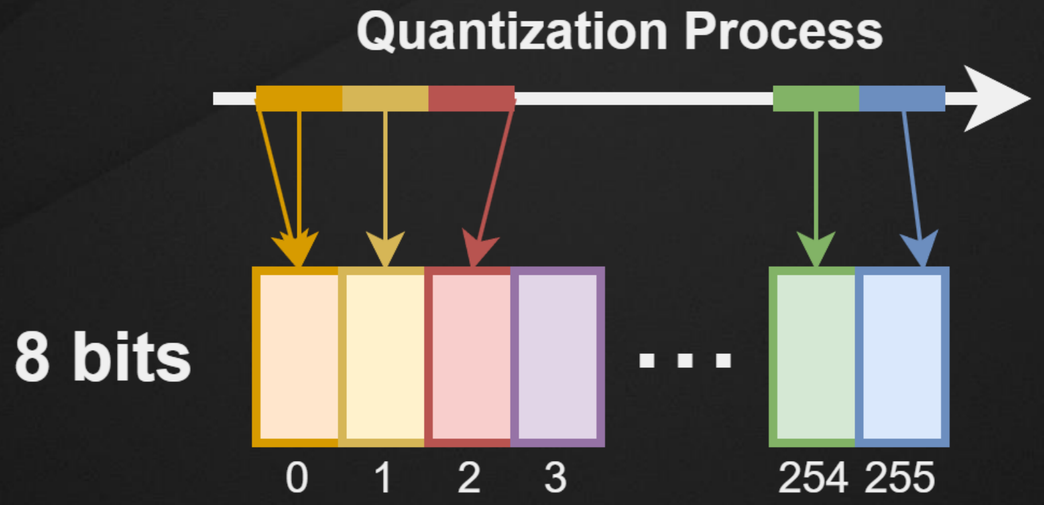
量子化

量子化



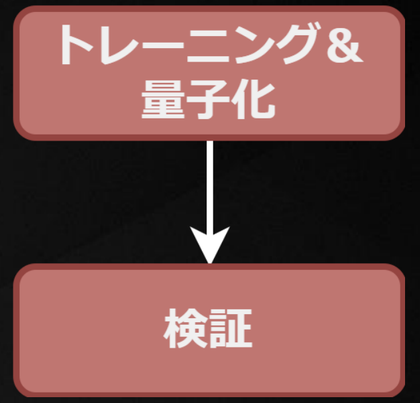
使用している**ニューラルネットワークは小規模であっても**、**容量**を占めている。

パッケージサイズを削減し、**計算を高速化しつつ**、
品質も確保できる最適な**量子化アルゴリズム**を見つけることを目指している。



POLYPHONY™ DIGITAL

量子化

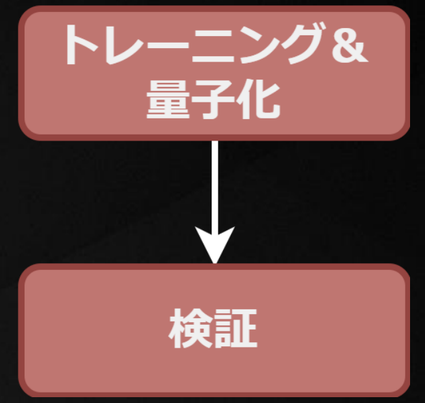


メモリ上の重み

1. 効率的なメモリレイアウト

量子化されたパラメータは常に通常の値に戻す必要があり、メモリを消費する。デシリアライズや逆量子化のプロセスを回避できる量子化システムを目指す。

量子化

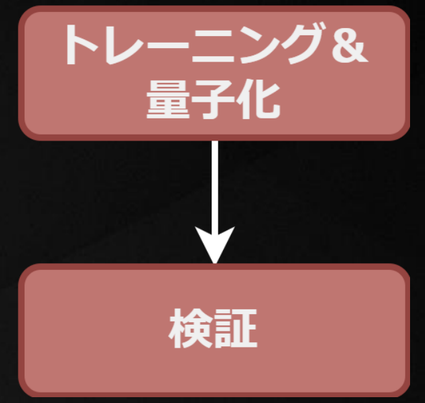


2. SSE命令のサポート

推論時にはCPUをバックエンドとして使用するため、
CPUのSSE機能を活用して高速計算を実現できる
量子化システムの構築を目指す。

重みを**8ビット**に圧縮し、SSE機能をより効率的に
利用できるようにしている。

量子化



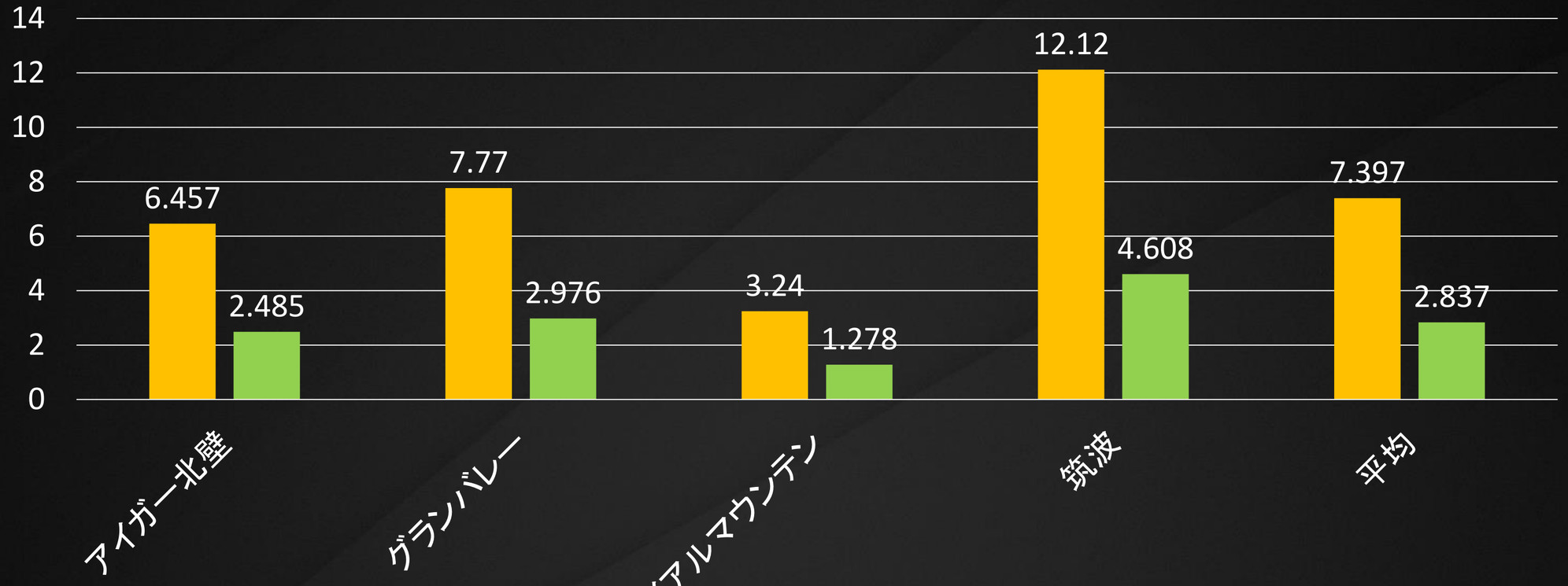
3. 量子化対応の学習(Quantization-Aware Training)

量子化中に多くの情報が失われるため、
学習中にネットワークが量子化を
考慮できるようにする必要がある。

これを実現するために、
STE（straight through estimator）という手法を使用する。

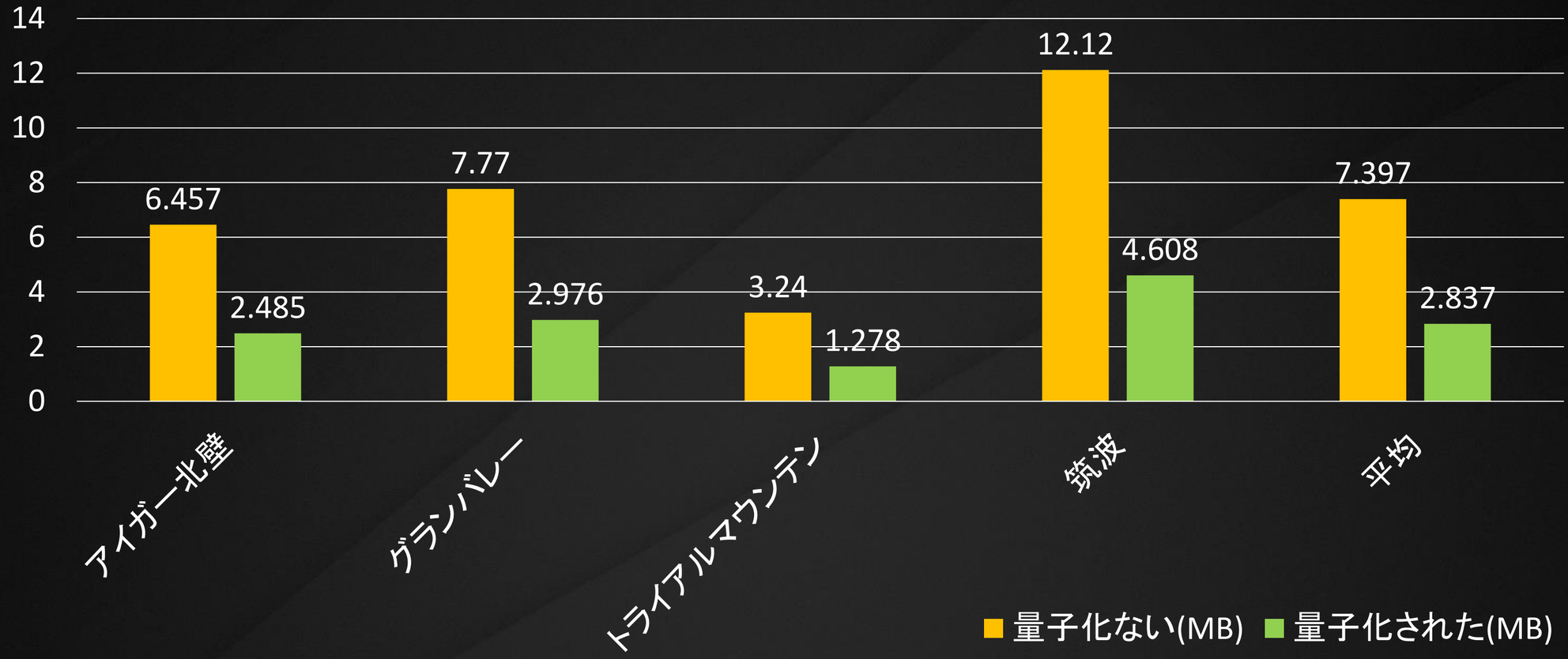
POLYPHONY™ DIGITAL

量子化の結果



POLYPHONY DIGITAL

量子化の結果



量子化により可視性フィールドのパッケージサイズが大幅に
小さくなり、カリング品質を維持したまま
平均**260%**の圧縮率を達成した。

POLYPHONY™ DIGITAL

再現率と精度

再現率(リコール率):

見えるオブジェクトが間違えなく見えると判断される比率である。

再現率が低くなると、見えるオブジェクトが間違えてカリングされた場合が多い。

精度(適合率):

見えると判定されたオブジェクトの中に、本当に見えるものの割合である。

精度が高いほど、カリングの効果が良い。

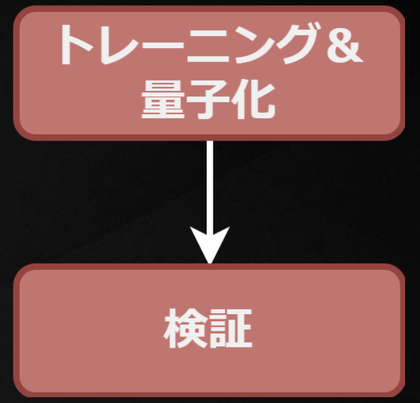
F値

Fn値は、モデルの性能を測る指標である。
不均衡なデータでも使いやすく、モデルの正確さとバランスを
一緒に確認ができる。

F1値:
再現率と精度の調和平均です。

F2値:
精度と比較して再現率を2倍重視する。

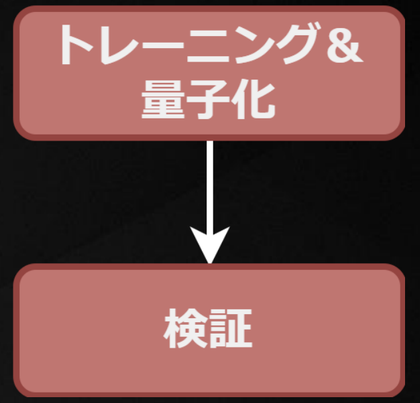
検証：再現率の確保



製品向けのカリングシステムでは、
最も重要なポイントは**再現率**である。

ネットワークが常に高い**再現率**を
維持できるようないくつかの工夫を紹介する。

検証：再現率の確保



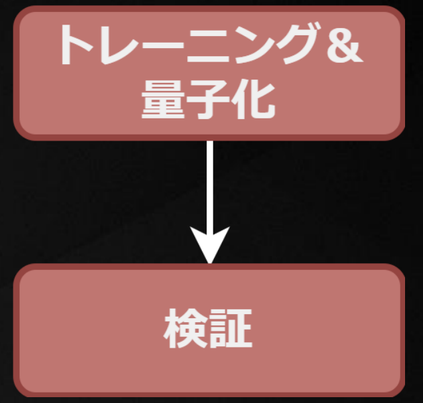
1．サンプルバランス

BCEWithLogitsLossには「pos_weight」という引数がある。

ポジティブサンプル（見える）と
ネガティブサンプル（見えない）のバランスを調整できる。

しかし、この方法が常にうまく機能するわけではない。

検証：再現率の確保



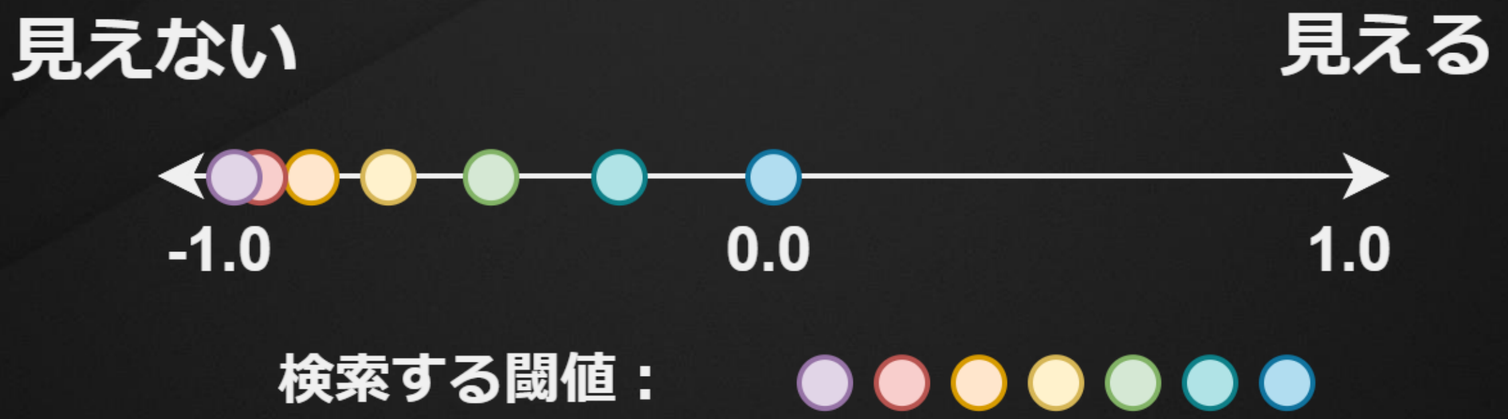
2. 閾値のシフト

リアルタイム推論時、出力が**閾値**未満の場合、
そのオブジェクトは見えないと見なす。

この**閾値**を調整することで、リコール率と精度のトレードオフができる。

各ネットワークで$2^{-\left(\frac{n}{8}+1\right)}$（nは1から50の整数）の範囲閾値検索を行う。

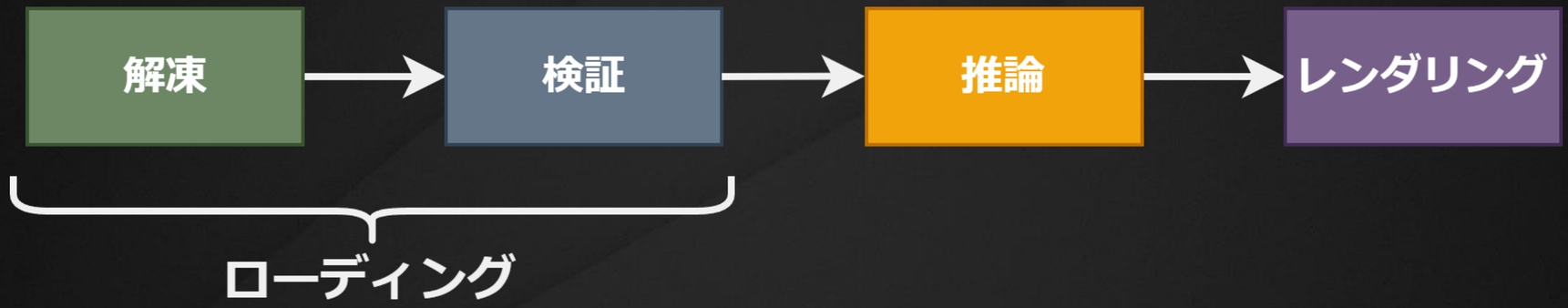
リコール率（99.9%）を満たす閾値の中から、
F2値が最大となる**閾値**を目標として選択する。



ランタイムパイプライン

POLYPHONY™ DIGITAL

ランタイムパイプライン

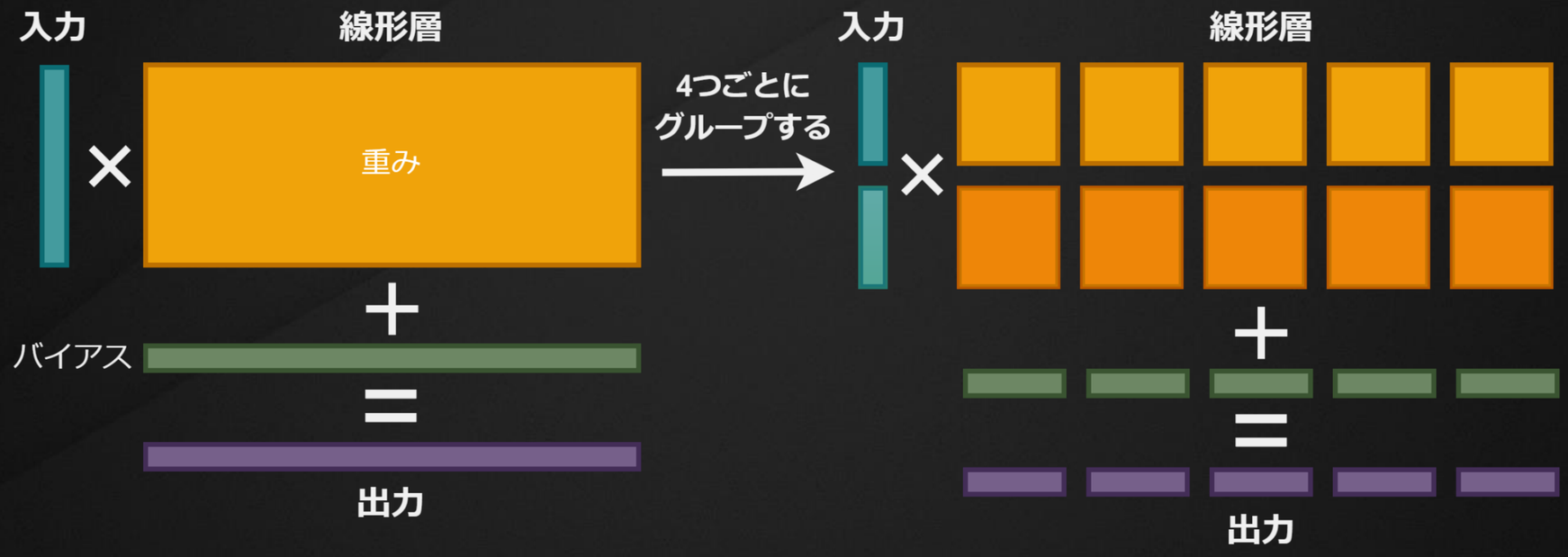


推論

推論(量子化ない)

1. グループ化されたベクトル

SSE命令を利用するために、ベクトルを4つのスカラーでグループ化し、
行列は4x4のサイズでグループ化する。

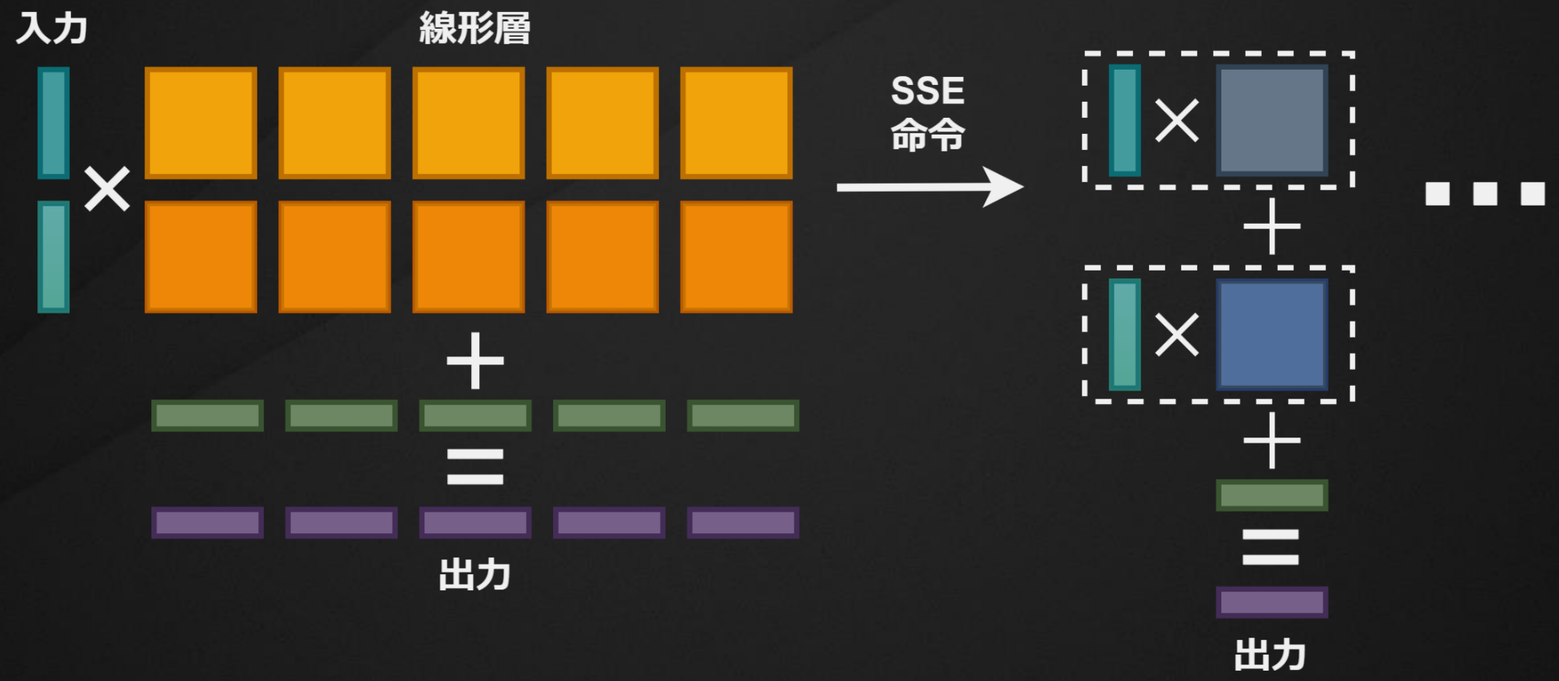


推論(量子化ない)

2. 数値計算バックエンド

グループ化後、線形層を4つのベクトルと
4x4のマトリックスの積で構成される演算子に簡単に変換できる。

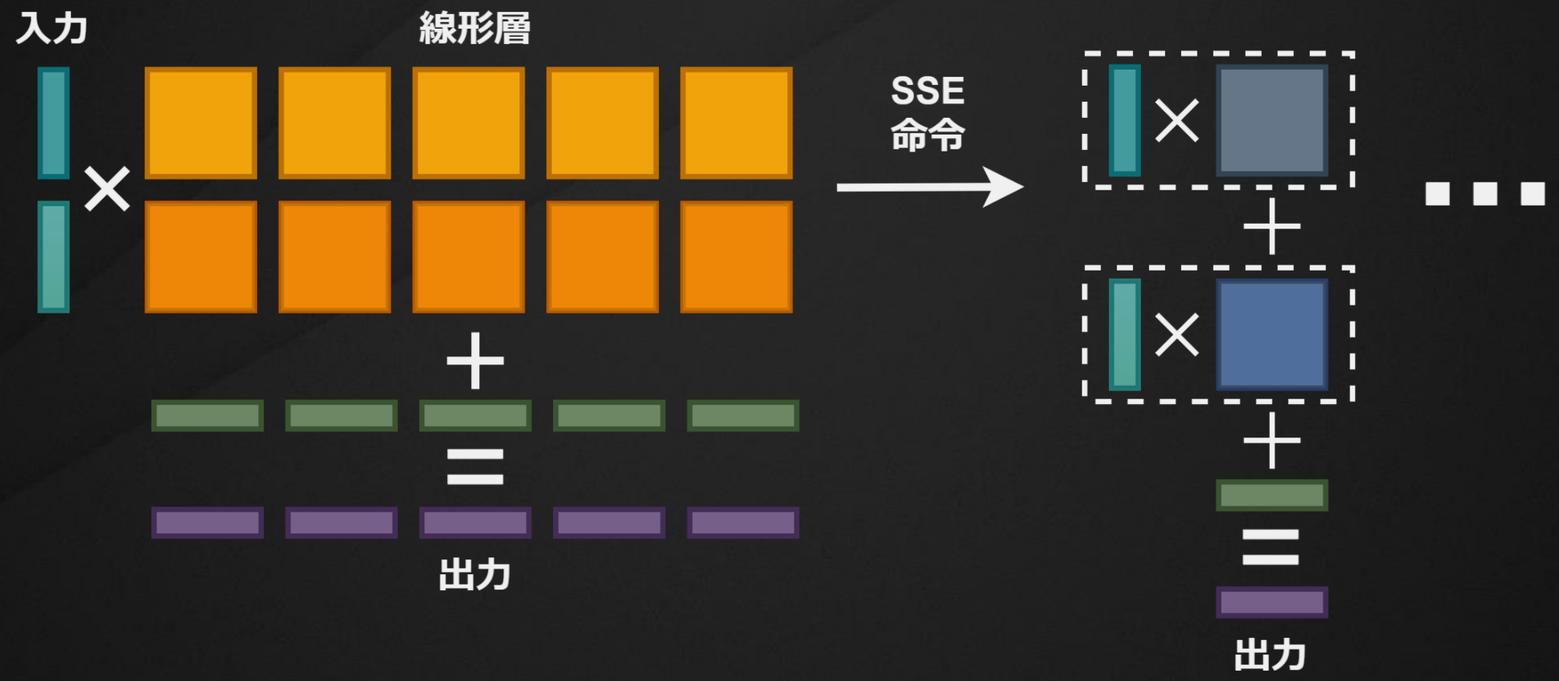
『グランツーリスモ7』では、SSE対応のカスタム数学ライブラリを使用している。



推論(量子化ない)

3. SSE命令

1x4のベクトルと4x4の行列の乗算は、
四回の「 _mm_mul_ps 」と三回の「 _mm_hadd_ps 」を使用して行う。



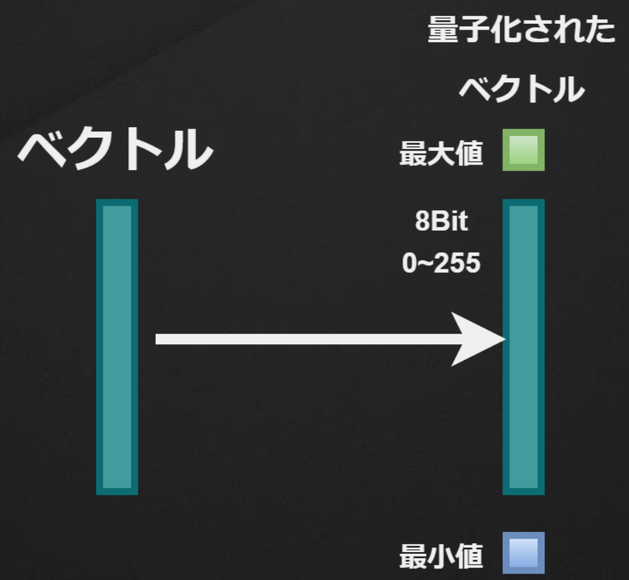
推論(量子化あり)

前のトレーニングの章で述べたように、SSE命令をより効果的に利用するために、
重みを8ビットに量子化する必要がある。

1. ベクトル

元のベクトルを範囲を0～1に正規化し、最小値と最大値を32Bitのfloatで保存してから、
0～1の範囲を整数の0～255にマッピングして8ビットに量子化する。

$$v = (\hat{v}/255) \times (v_{max} - v_{min}) + v_{min} = (\hat{v}/255) \times v_{range} + v_{min}$$



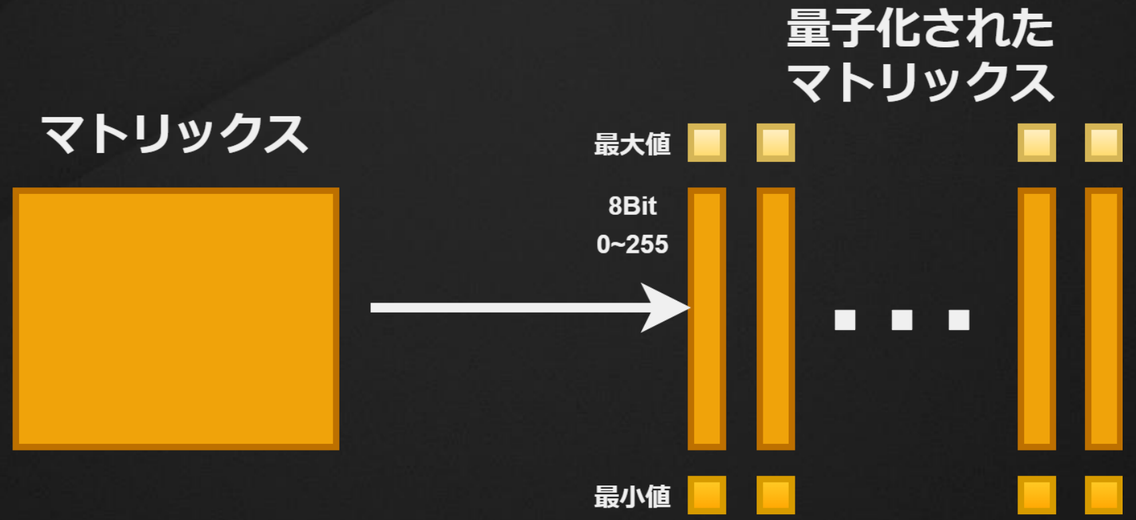
推論(量子化あり)

2. 行列

行列についても各列の範囲を0〜1に正規化し、最小値と最大値を保存した後、
8ビットに量子化する。

$$M = [m_1, m_2, \ldots, m_n]$$

$$m_i = (\hat{m}_i / 255) \times m_{i\,range} + m_{i\,min}$$



推論(量子化あり)

追加のコピーや逆量子化を回避するために、ベクトルと行列の乗算の処理と
SSE命令の利用方法を分析する必要がある。

1. 乗算

ベクトルと行列の乗算を、いくつかの**ベクトル間のドット積**として捉えることができる。

$$vM = [v \cdot m_1, v \cdot m_2, \ldots, v \cdot m_n]$$

推論(量子化あり)

2. ドット積

各ベクトルが量子化されているため、ドット積をさらにいくつかの項に分解することができる。
以下の式のように、2つの量子化されたベクトルのドット積は、最終的に
2つの整数ベクトルのドット積、2つの整数ベクトルの総和、およびいくつかの定数に分解できる。

2つの整数ベクトルのドット積以外の項は、ロードする際に計算できるので、さらに計算が軽くなる。

$$v \cdot m_i = ((\hat{v}/255) \times v_{range} + v_{min}) \cdot ((\hat{m}_i/255) \times m_{irange} + m_{imin})$$

$$v \cdot m_i = \sum_j ((\hat{v}_j/255) \times v_{range} + v_{min}) \cdot ((\hat{m}_{ij}/255) \times m_{irange} + m_{imin})$$

$$v \cdot m_i = \sum_j (\hat{v}_j \hat{m}_{ij}/255/255) v_{range} m_{irange} + v_{min}(\hat{m}_{ij}/255) + m_{imin}(\hat{v}_j/255) + v_{min} m_{imin}$$

$$v \cdot m_i = \underbrace{(\hat{v} \cdot \hat{m}_i) v_{range} m_{irange}/255/255}_{\text{整数ベクトルの点乗積}} + \underbrace{v_{min}(\sum_j \hat{m}_{ij}/255) + m_{imin}(\sum_j \hat{v}_j/255) + \sum_j v_{min} m_{imin}}_{\text{ロードする際に計算可能}}$$

POLYPHONY™ DIGITAL

整数ベクトルのドット積

1．_mm_maddubs_epi16

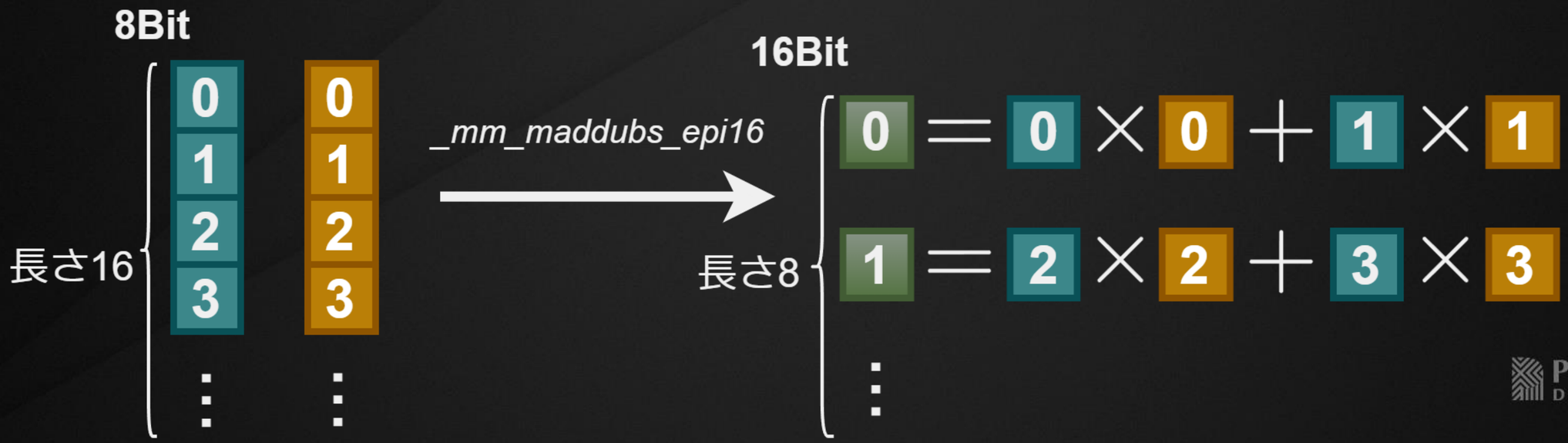
前述のように、8ビットデータのドット積に使用できるSSE命令は_

_mm_maddubs_epi16

のみである。

この命令は、2つのベクトルを8ビット単位で処理し、
隣接する2つの値のドット積を16ビットで出力する。

そのため、オーバーフローを防ぐために、
整数ベクトルには実質的に7ビットのデータしか保存できない。



整数ベクトルのドット積

2. _mm_madd_epi16

結果をさらに和を求め、オーバーフローを防ぐために、_mm_madd_epi16を使用して
先ほどの結果と1のドット積を取り、隣接する2つの値の和を32ビットで出力する。

整数ベクトルのドット積

hadd類の命令の使用をできるだけ減らすため、
前のステップで得られた4つのベクトルを1組として扱う。

3回の_mm_hadd_epi32を用いることで、この4つのベクトルをそれぞれ求和し、
最終的に1つの整数ベクトルに配置できる。

この方法により、命令の数を大幅に削減し、結果も追加の変換を必要としない。

推論の結果

コース	一般的な推論↓	量子化の推論↓	推論時間差↑
アイガー北壁	35.0μs	27.7μs	-7.3μs
グランバレー	61.3μs	50.0μs	-11.3μs
トライアルマウンテン	34.5μs	28.7μs	-5.8μs
筑波	34.0μs	26.4μs	-7.6μs
平均	41.2μs	33.2μs	-8.0μs

量子化に最適化された推論方法を使用することで、
推論にかかる時間を**さらに削減できた**。

POLYPHONY™ DIGITAL

結果:アイガー北壁 (PS5®)

POLYPHONY™ DIGITAL


GRAN TURISMO

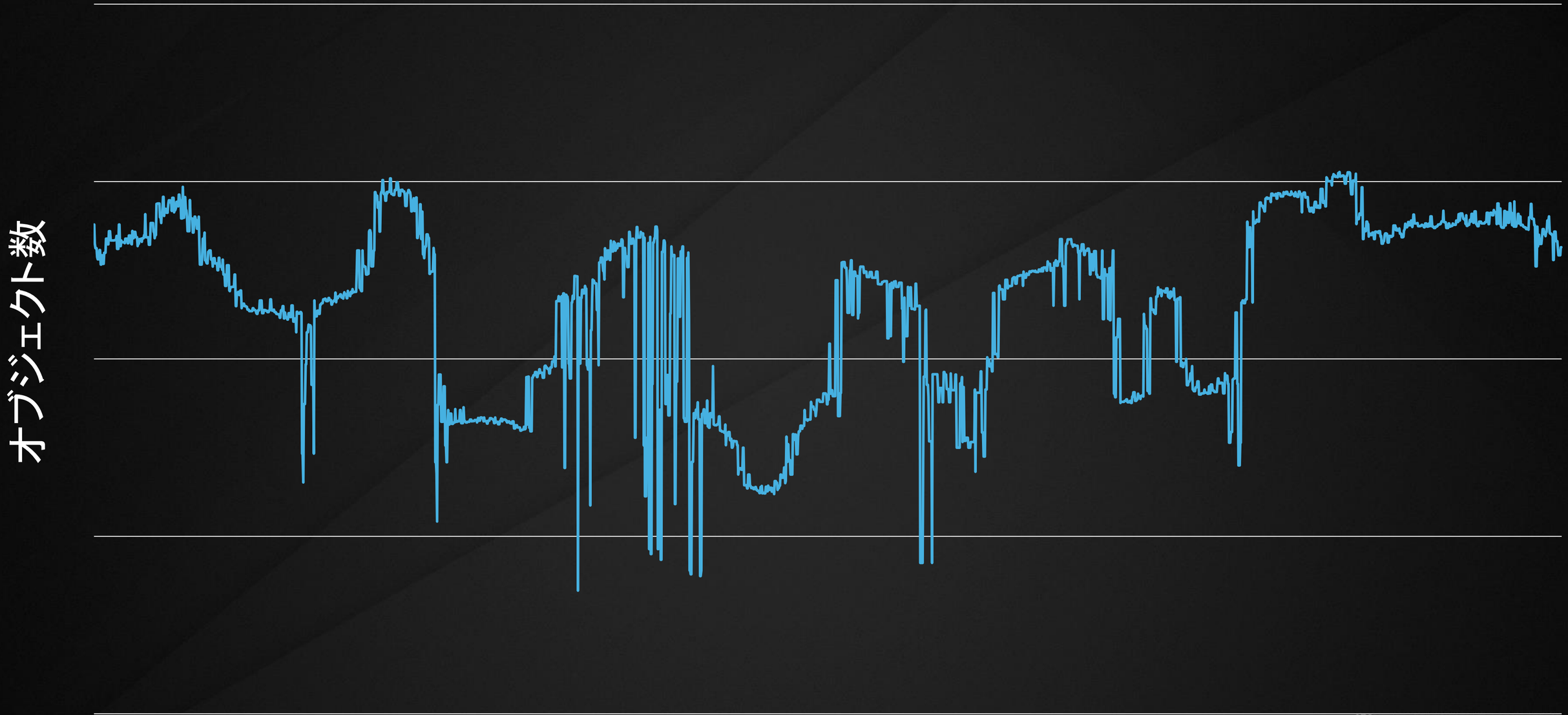
結果:アイガー北壁 (PS5®)

	w/o Neural	**w/ Neural**	Improvement
GPU avg.	6.638ms	6.612ms	**-0.026ms**
GPU min.	5.384ms	5.329ms	**-0.055ms**
GPU max.	7.973ms	7.968ms	**-0.005ms**
CPU avg.	3.944ms	3.758ms	**-0.186ms**
CPU min.	3.025ms	2.902ms	**-0.123ms**
CPU max.	5.013ms	4.897ms	**-0.116ms**

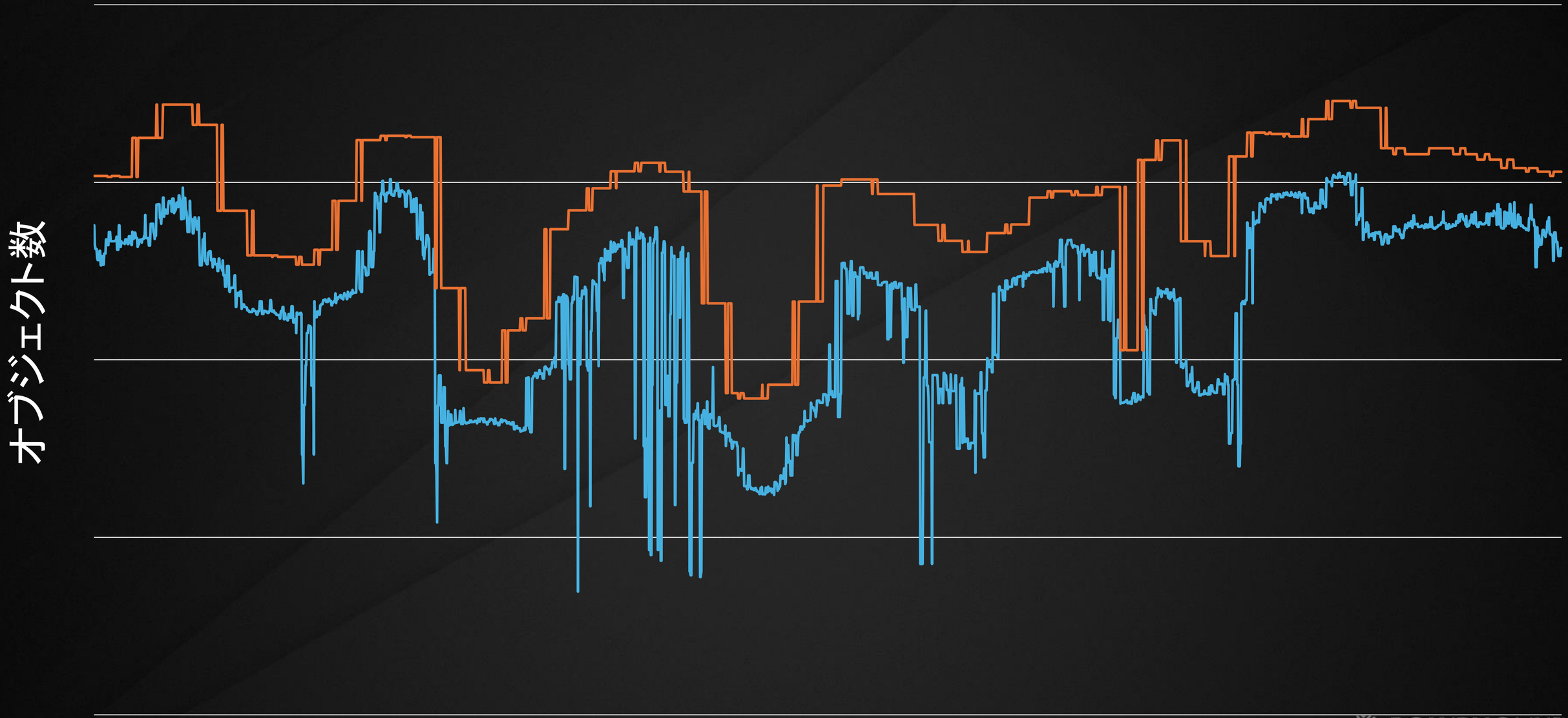
アイガーは、**遮蔽物が少ない広大なコース**である。
ニューラル可視性フィールドを使用することで、コース内のランプや丘陵などの
潜在的な遮蔽物を活用し、レンダリングの負荷をさらに軽減できる.

POLYPHONY™ DIGITAL

結果:アイガー北壁 (PS5®)

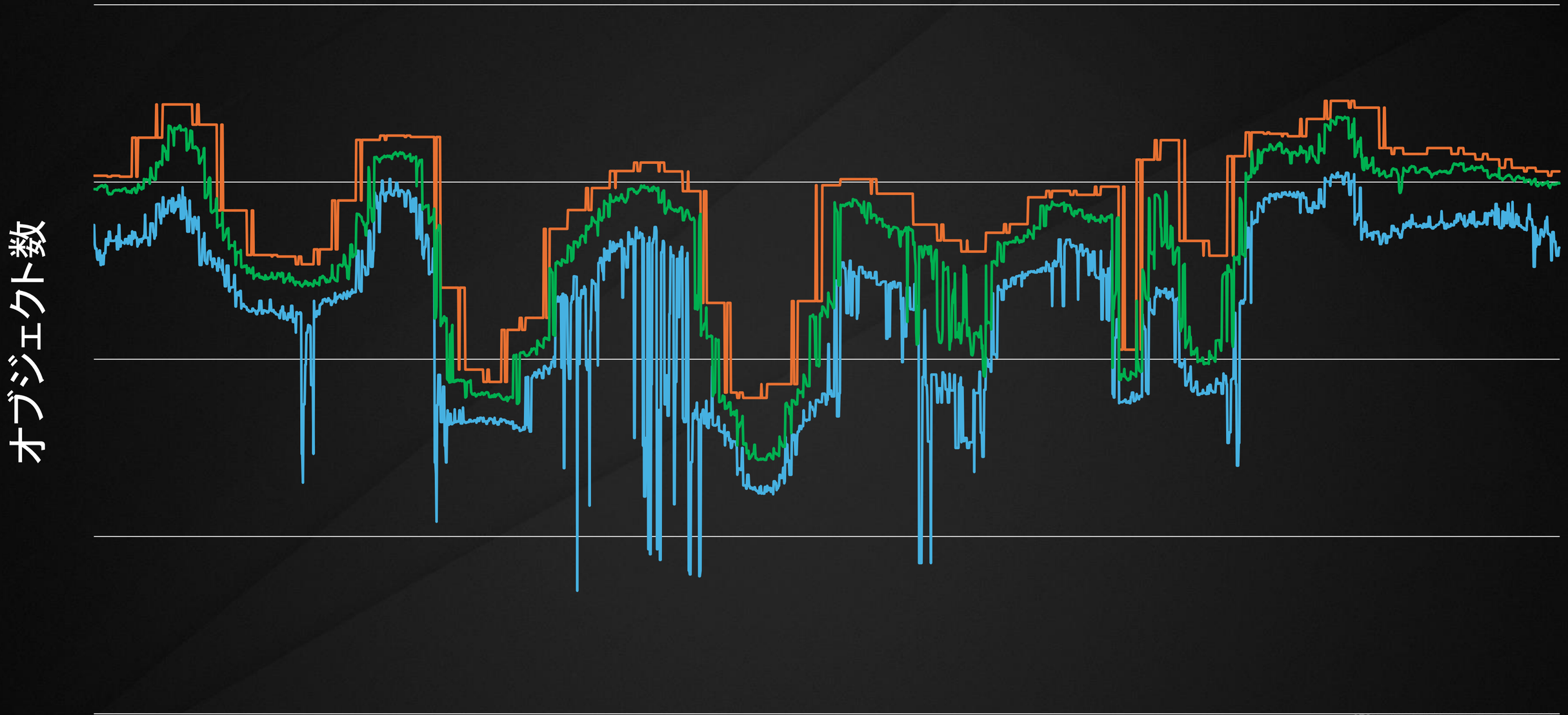


結果:アイガー北壁 (PS5®)



POLYPHONY™ DIGITAL

結果:アイガー北壁 (PS5®)



POLYPHONY™ DIGITAL

結果:グランバレー(PS5®)

POLYPHONY™ DIGITAL


HELLA
HELLA
WeatherTech
Auto. Home. Pet. Find Your Fit.
MOTHERS
Polishes•Waxes•Cleaners

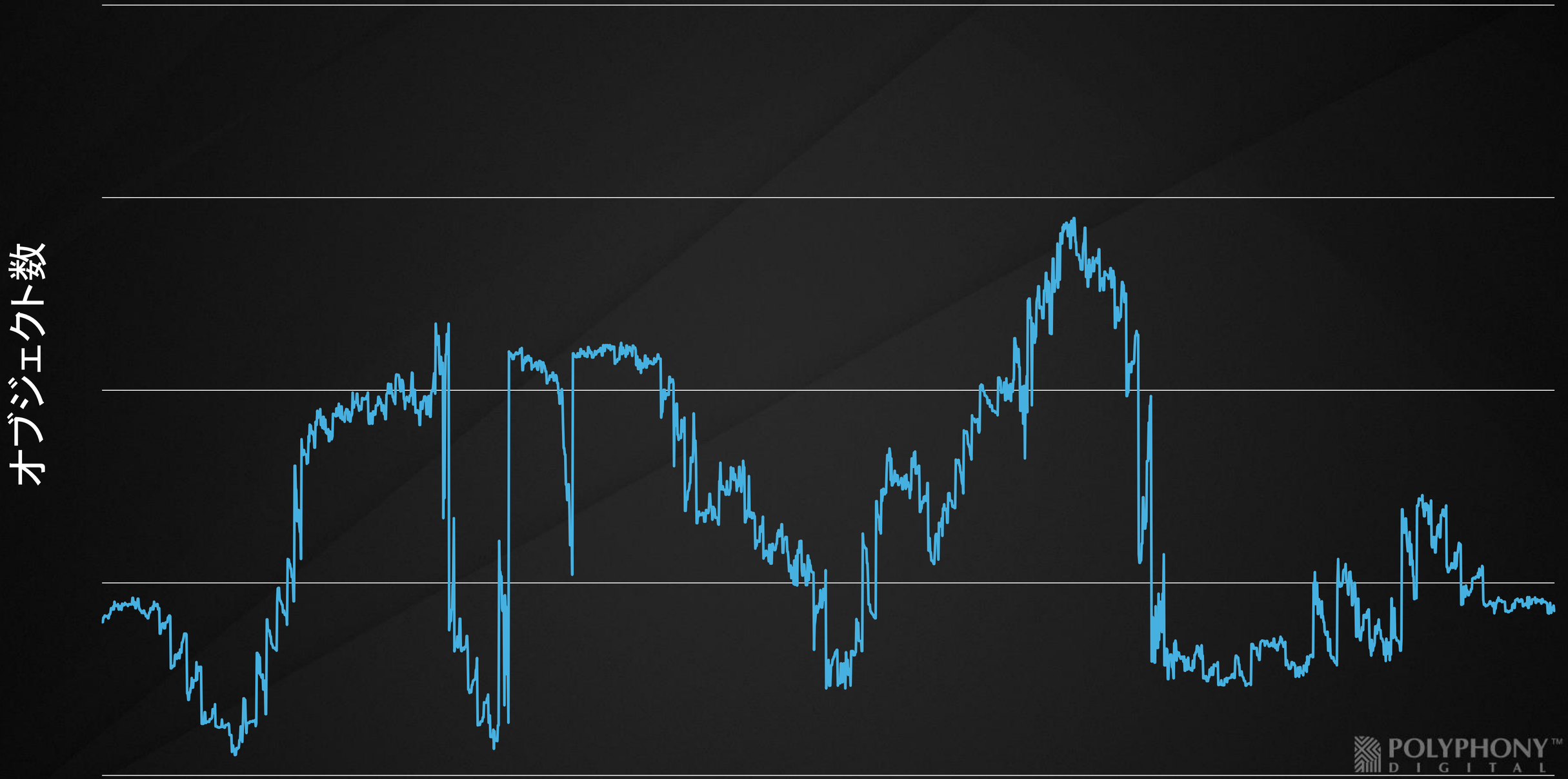
結果:グランバレー(PS5®)

	w/o Neural	**w/ Neural**	Improvement
GPU avg.	7.180ms	7.064ms	**-0.116ms**
GPU min	5.797ms	5.777ms	**-0.020ms**
GPU max	8.885ms	8.787ms	**-0.098ms**
CPU avg	4.552ms	4.256ms	**-0.296ms**
CPU min	3.229ms	2.931ms	**-0.298ms**
CPU max	6.378ms	5.849ms	**-0.529ms**

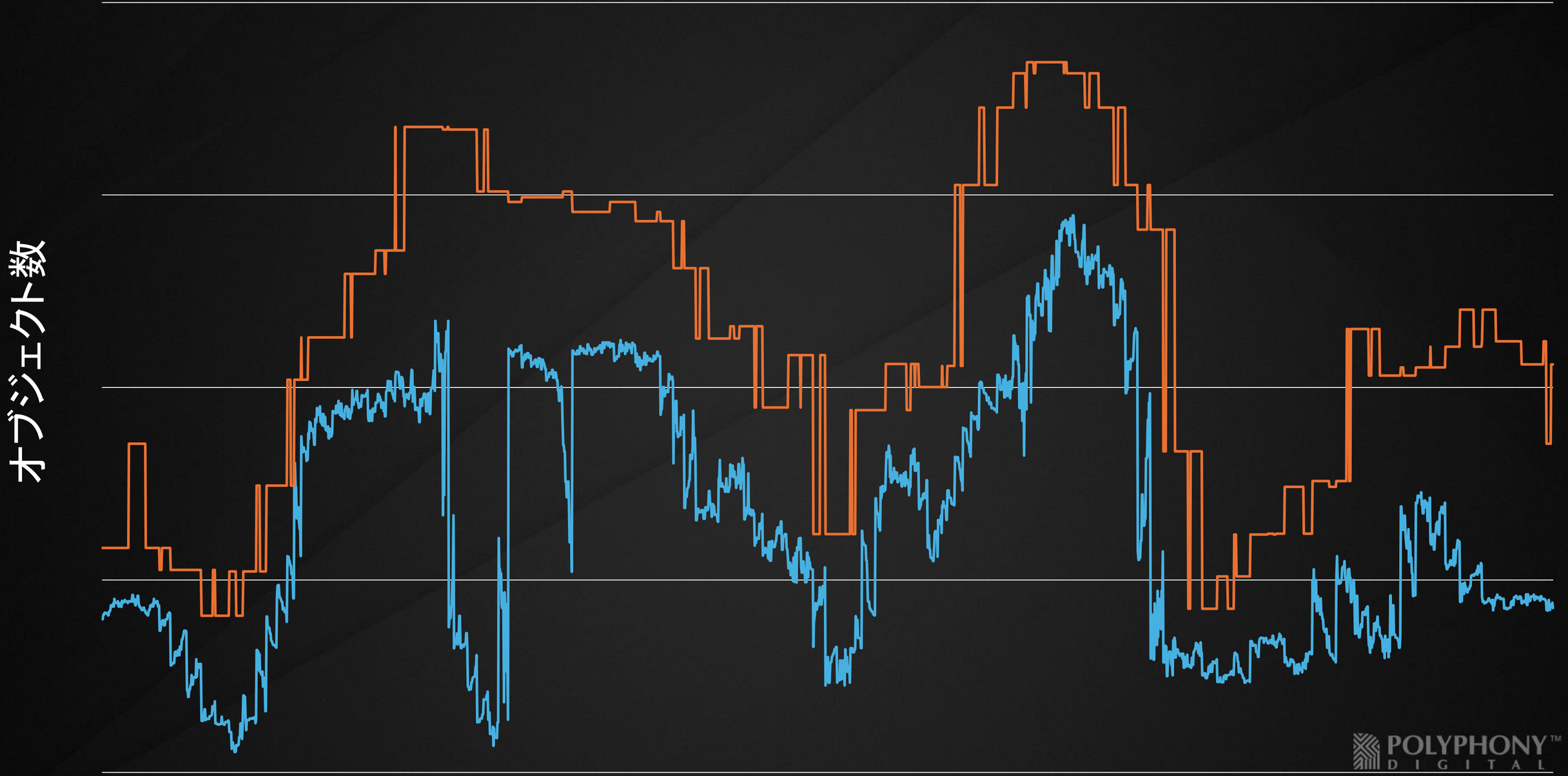
グランバレーは、**広大なスケープを持つ大規模なコースである**。
ニューラル化されたオクルージョンカリングにより、コース全体でGPUとCPUの
負荷が安定し、多数のオブジェクトが存在する一部のビューでの
最大レンダリング負荷を軽減することができる。

POLYPHONY™ DIGITAL

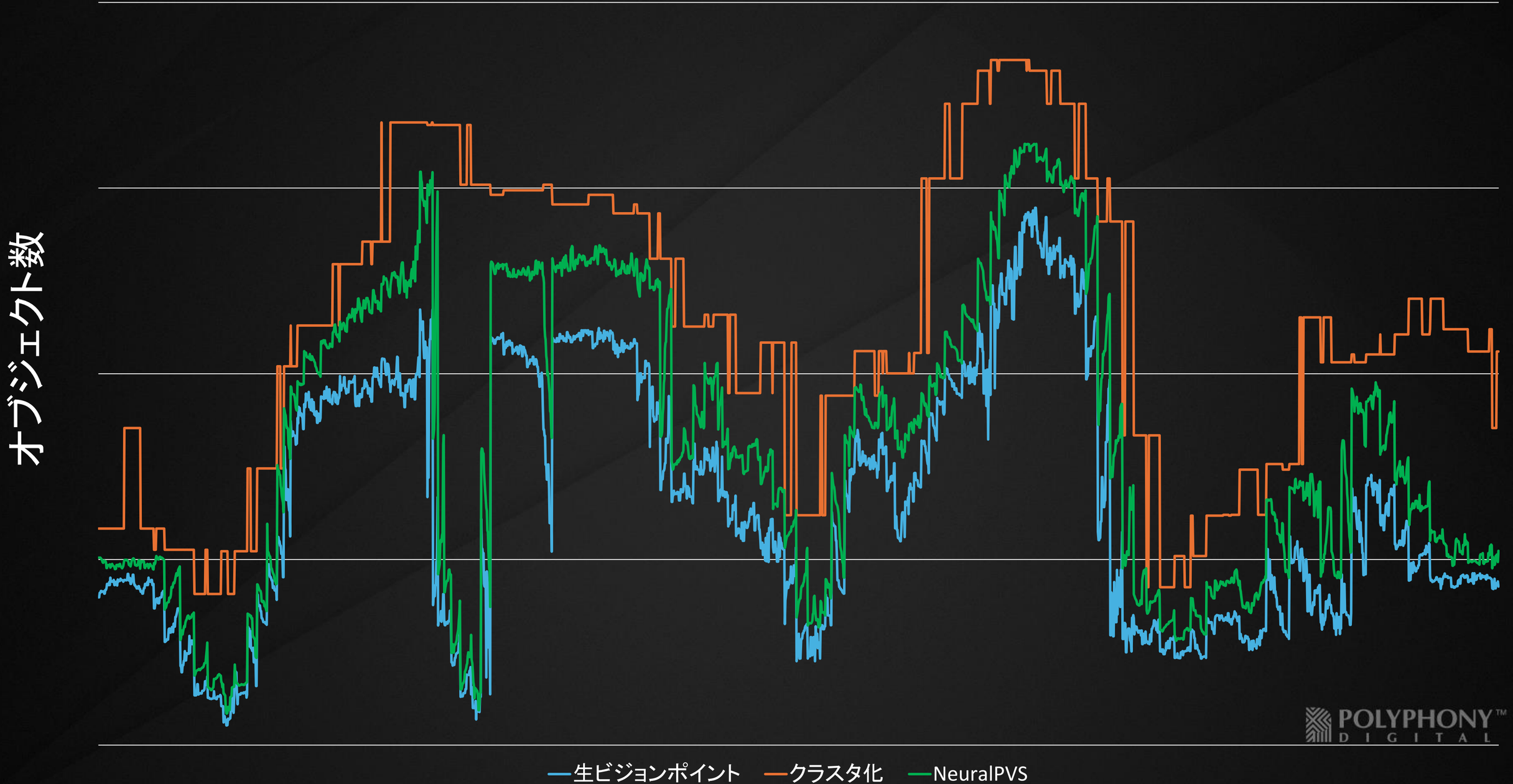
結果:グランバレー(PS5®)



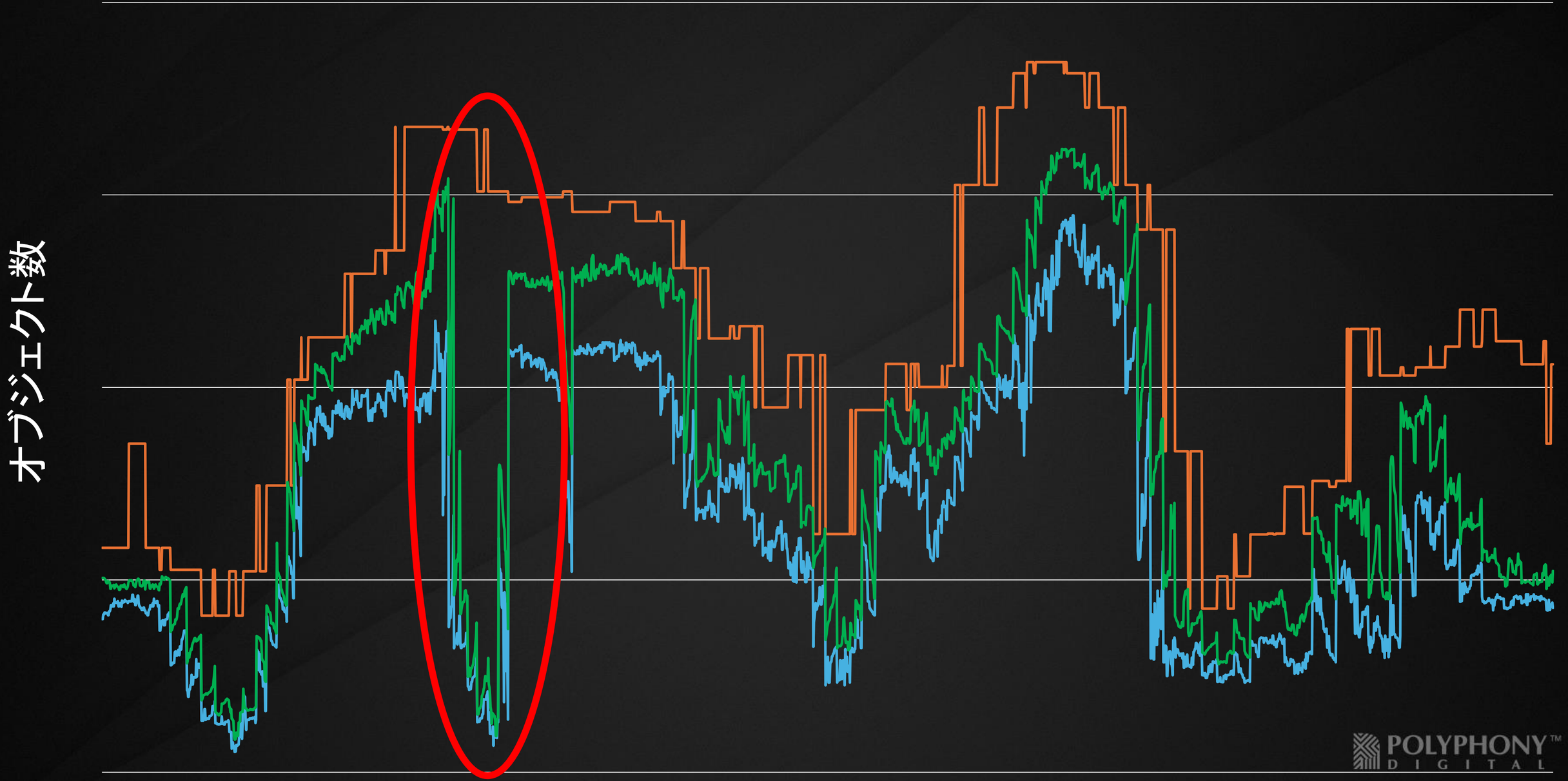
結果:グランバレー(PS5®)



結果:グランバレー(PS5®)



結果:グランバレー(PS5®)




GoPro
GoPro
GoPro
GoPro
HIGHWAY 1
MOTHERS
Polishes•Waxes•Cleaners


GoPro
GoPro
GoPro
GoPro
HIGHWAY-1
MOTHERS
Polishes•Waxes•Cleaners

まとめ

1. 従来のカリングシステムについて、その利点と欠点を紹介した。

2. ニューラルネットワークを活用することで、既存の方法の不足を解決しようと試みた。

3. ニューラルネットワークの手法の優位性を示した。

まとめ

最近のコースの高精細かつ高負荷なモデルに対しても、
ほぼ全自動でより高速に動作するオブジェクトカリングシステムを
実現しつつある。

今後、これを製品に導入することを検討しています。

Thanks and References

ご清聴ありがとうございました。
ご質問がございましたら、どうぞお気軽に。

[Nirstein02] Exact from-region visibility culling, https://dl.acm.org/doi/10.5555/581896.581921

[Shopf08] March of the Froblins: Simulation and Rendering massive crowds of intelligent and detailed creatures on GPU, https://dl.acm.org/doi/10.1145/3256724

[Mildenhall 21] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106. https://arxiv.org/pdf/2003.08934

[Sitzmann20]: Sitzmann, Vincent, et al. "Implicit neural representations with periodic activation functions." *Advances in neural information processing systems* 33 (2020): 7462-7473.

[Tancik20] Tancik, Matthew, et al. "Fourier features let networks learn high frequency functions in low dimensional domains." Advances in neural information processing systems 33 (2020): 7537-7547.

POLYPHONY™ DIGITAL