ニューラルネットワークを用いた
高精度な
オブジェクトカリングシステムの提案

内村創
于承中

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

自己紹介 / 于承中

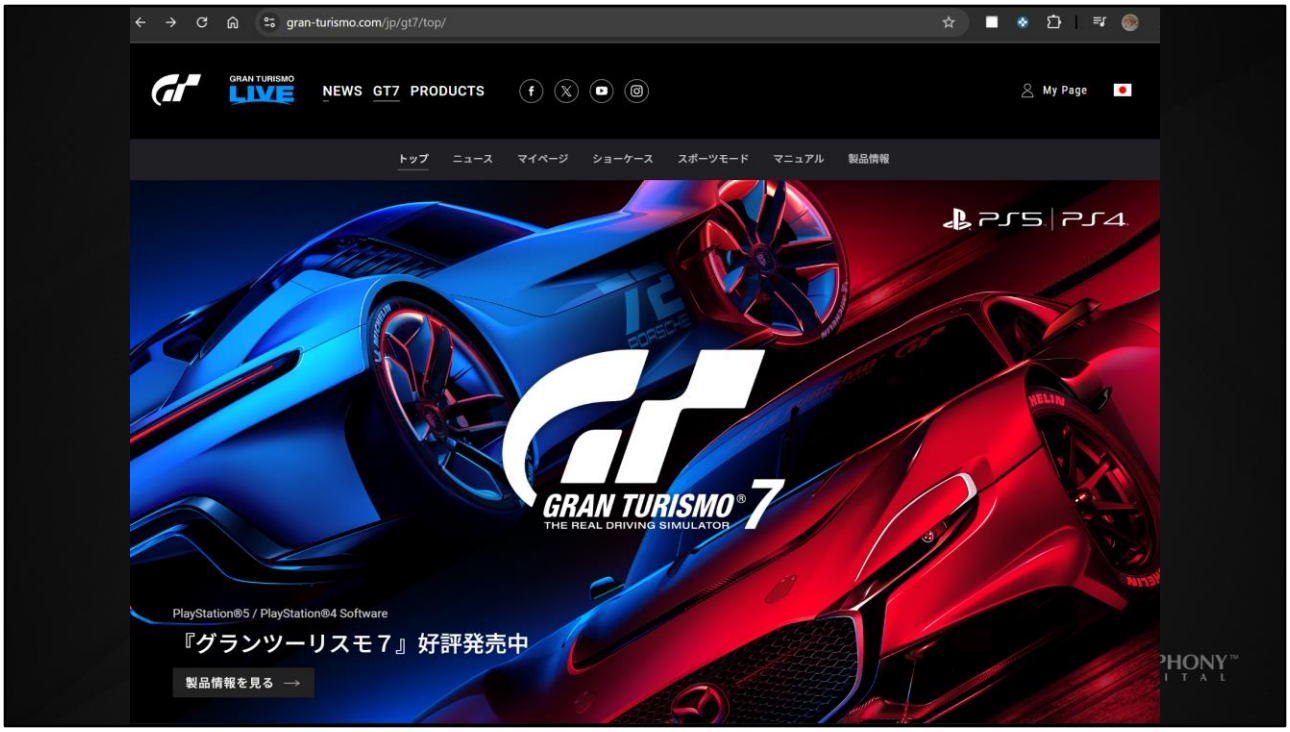
- 東京理科大学卒
- 2024年入社

- 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

得意な分野

今年入社です
趣味？



グランツーリスモは、実在車両やコースを精緻に再現することを特徴としたレースゲームで、プレイステーション4とプレイステーション5向けに製作・発売しています。
ポリフォニー・デジタルは、グランツーリスモシリーズの製作スタジオで、ソニーインタラクティブエンタテインメントの100%子会社です。

グランツーリスモは、精緻に再現された車両や背景と物理挙動を特徴としたレースゲームです。

収録されている車両はレースカーに限らず、幅広い車を楽しむことができるのもグランツーリスモの特徴です。

レースゲームのヒーローはクルマですが、実は忘れてはならない要素が一つあります。

それは背景です。

どんなにかっこいいクルマでも、背景が無くてはレースゲームとして成立しません。
グランツーリスモでは、この背景のことを、コース、と呼んでいます。

本セッションでは、コースをできる限り軽量にレンダリングするために
グランツーリスモシリーズで開発してきた、
事前計算カリングの手法と、その最新の
知見について紹介します。

Agenda

- 前半
  - 事前計算カリングとは
  - 基本的なアイデア
  - 具体的な実装
  - 手法の限界
- 後半
  - ニューラル可視化フィールド
  - 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

事前計算カリングとは

POLYPHONY™ DIGITAL

事前計算カリングとは

- オブジェクトカリングを事前計算しておく手法

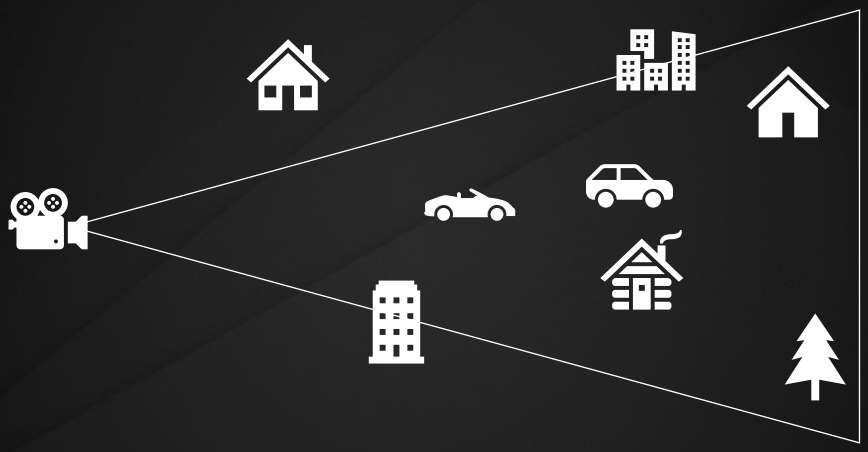
オブジェクトカリングとは

- 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？

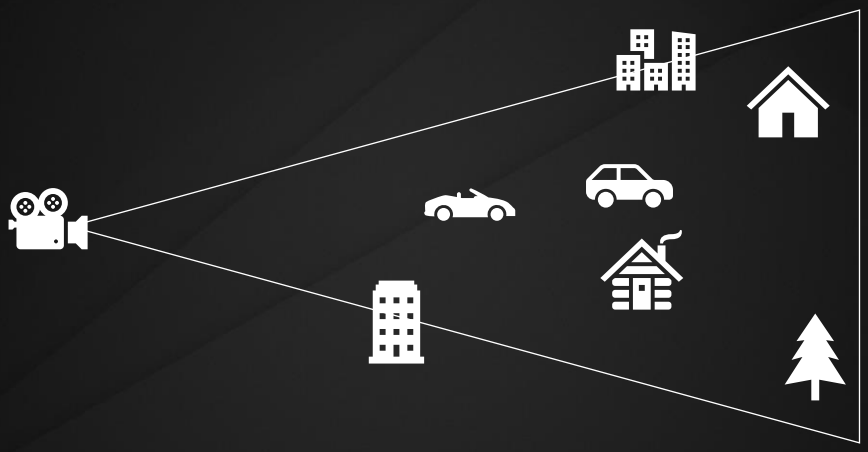
- 軽量になるから
  - 処理が減るので、CPUが軽くなる
  - 描画が減るので、GPUも軽くなる

典型的なシーンを考える

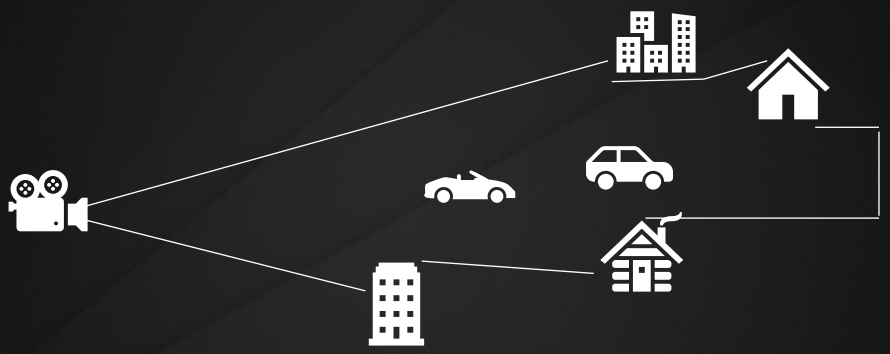


POLYPHONY™ DIGITAL

カメラの範囲外は描画しない (フラスタムカリング)

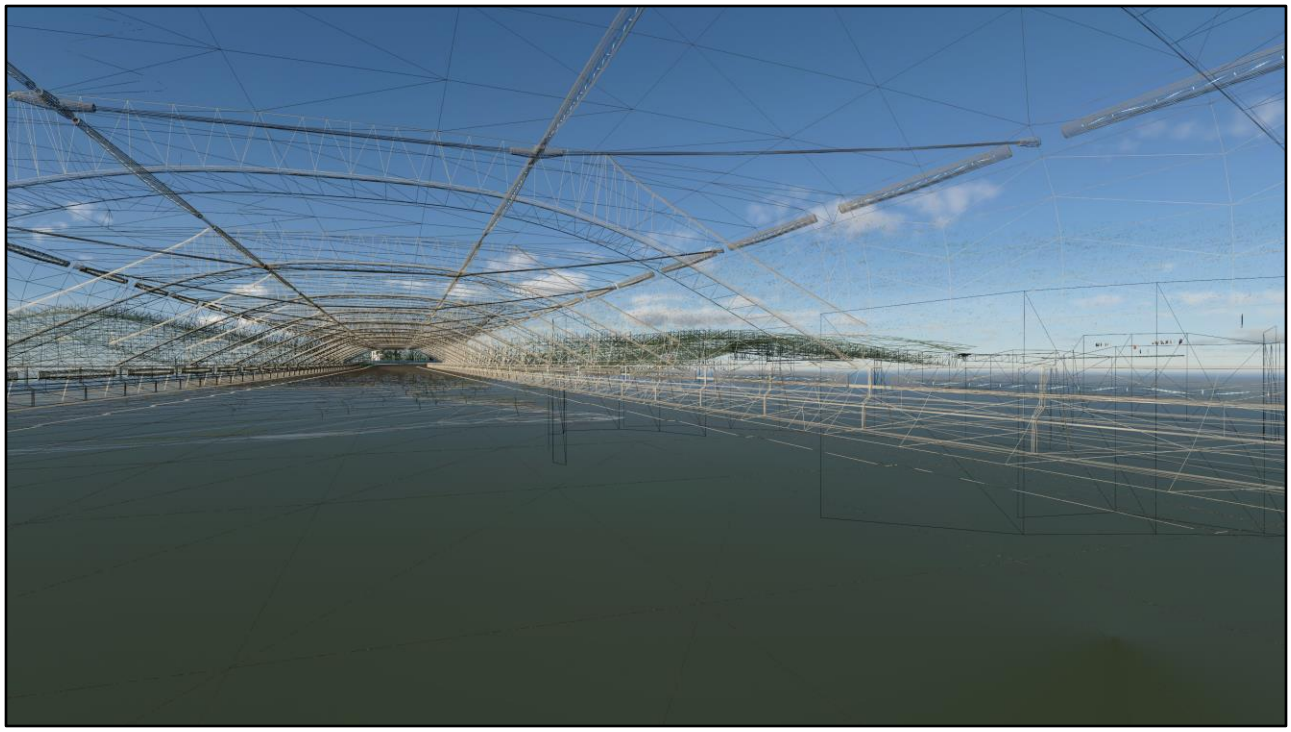


手前に隠されるものも表示しない (オクルージョンカリング)

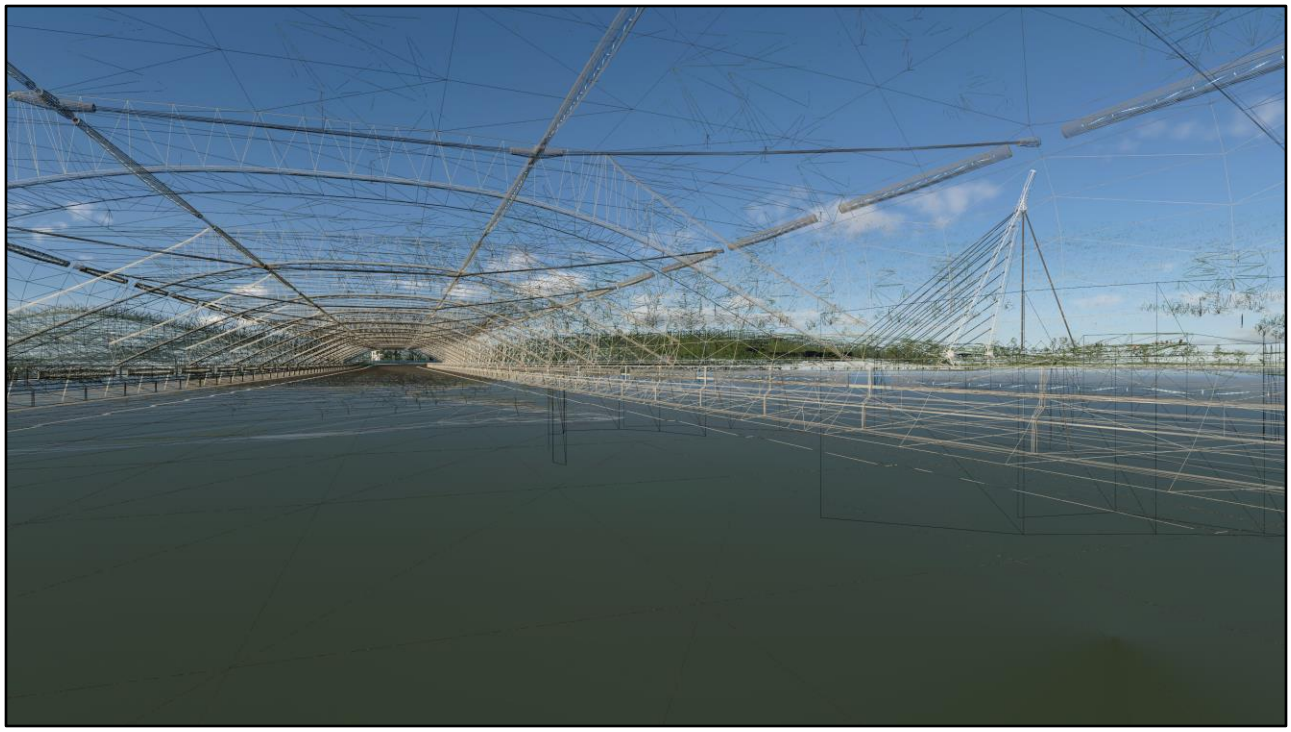


POLYPHONY DIGITAL

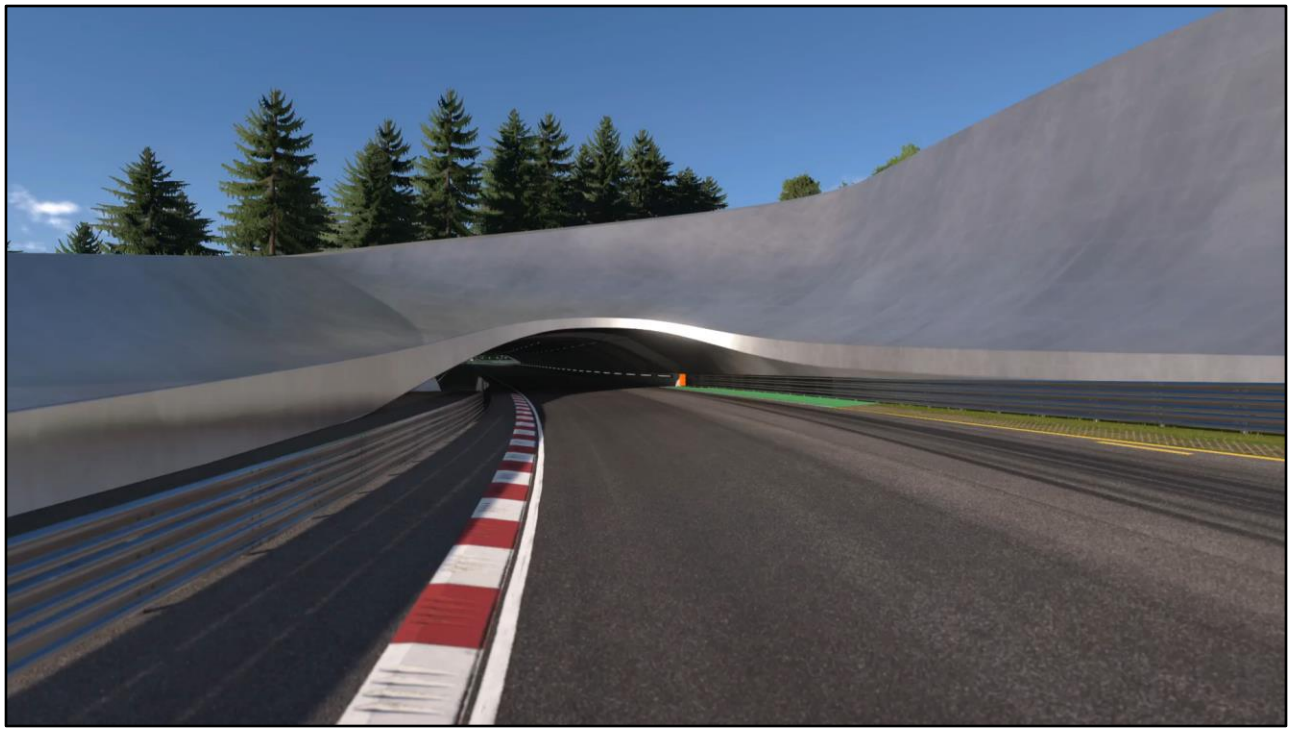
わかりやすくするために極端なシーンを
用意しました。トンネルです。



トンネルの外はほとんど何も描画する必要がないわけですが、



カリングシステムが無ければ、このように、無駄な描画をたくさんすることになってしまいます。



映像で見てみるとこのようになっています
（映像）

ほかのコースでも見てみましょう。

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 Visible Sets”

POLYPHONY™ DIGITAL

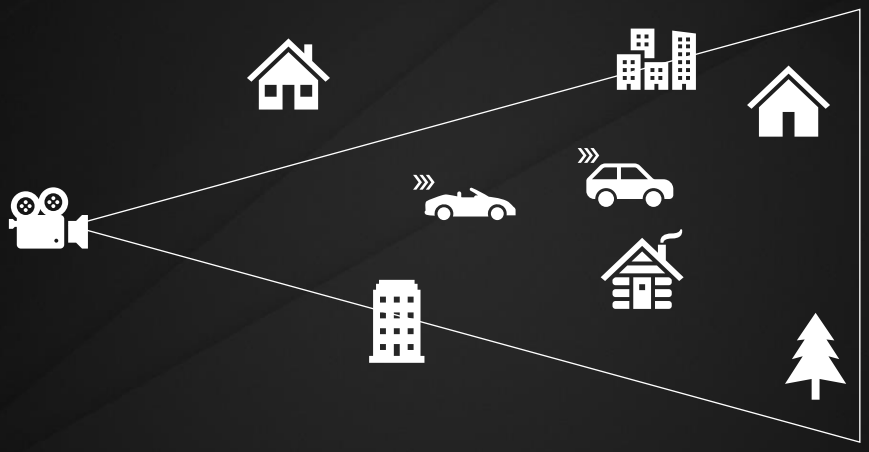
基本的なアイデア

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

POLYPHONY™ DIGITAL

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

POLYPHONY™ DIGITAL

可視判定する

POLYPHONY™ DIGITAL

可視判定結果の表

モノ	可視？
	✓
	✓
	✗
	✓
	✗

POLYPHONY DIGITAL

可視リストにする

カメラ	リスト
	

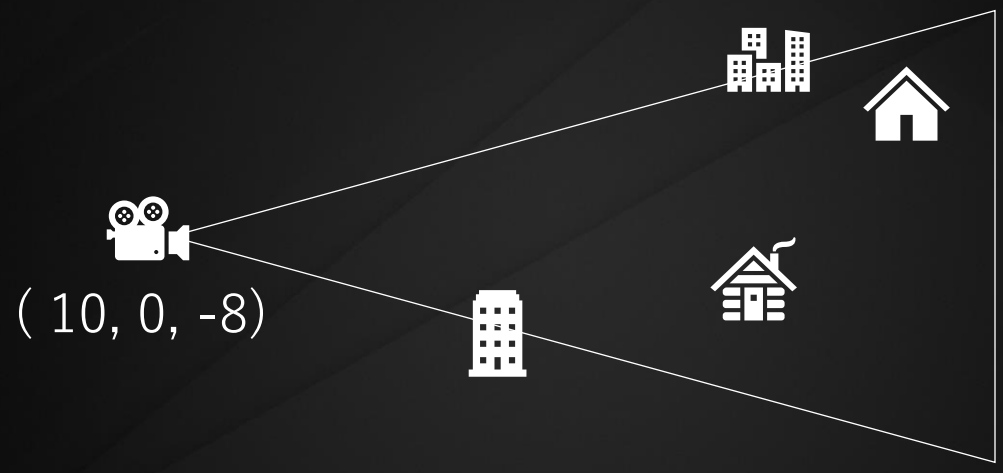
POLYPHONY™ DIGITAL

オブジェクトは
IDで管理しておく

モノ	ID
	1
	2
	3
	4
	5
	6

POLYPHONY™ DIGITAL

カメラは座標情報を持つ



可視リストの実体
(ビジョンポイント)

座標	IDリスト
(10, 0, -8)	1, 2, 5, 6

POLYPHONY™ DIGITAL

実際には可視判定は全方向に行う

POLYPHONY™ DIGITAL

カメラの移動範囲をカバーするように

POLYPHONY DIGITAL

色々な場所で可視判定を行う

POLYPHONY DIGITAL

可視リストが完成した
(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
…	…

POLYPHONY™ DIGITAL

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

POLYPHONY™ DIGITAL

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
…	…

POLYPHONY™ DIGITAL

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

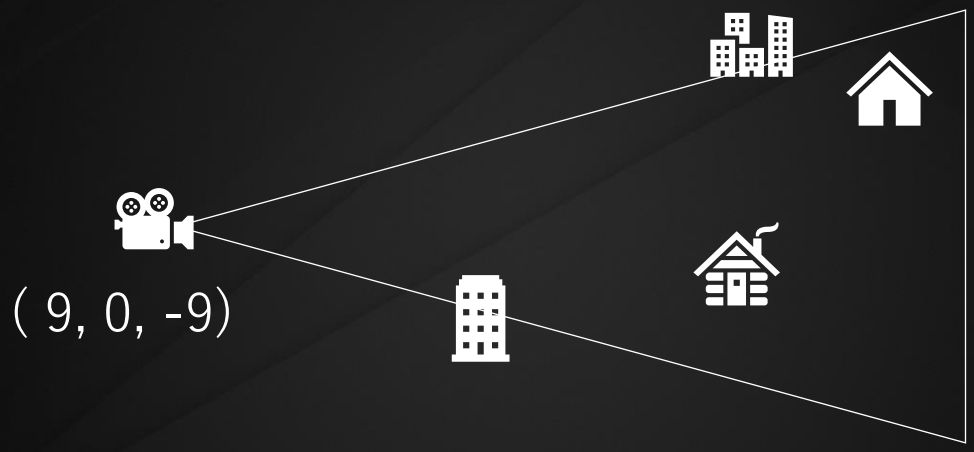
＋ →

ID	モデル
1	
2	
…	…

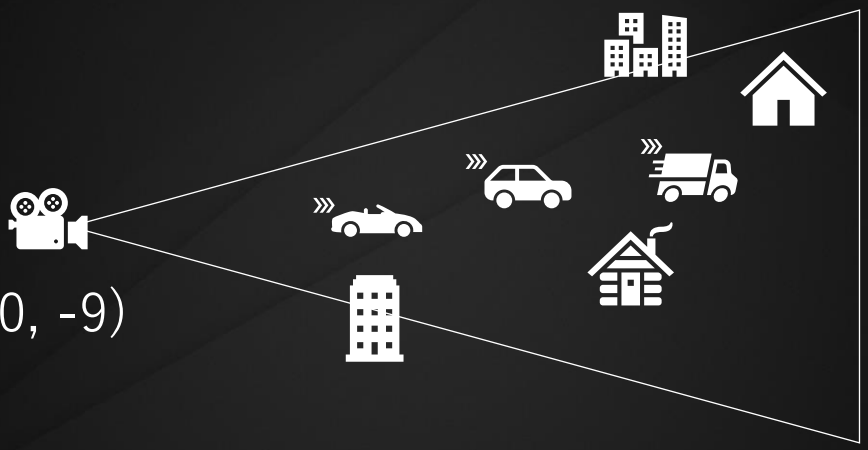
描画すべきオブジェクト


POLYPHONY™ DIGITAL

見えるオブジェクトだけ描画する



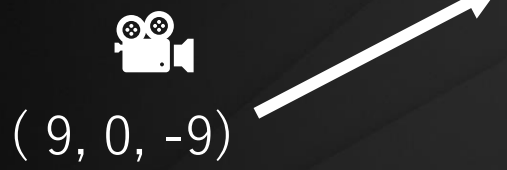
カリング対象外の
オブジェクトを追加描画する



(9, 0, -9)

POLYPHONY™ DIGITAL

あくまでも近傍なので、誤差が発生する

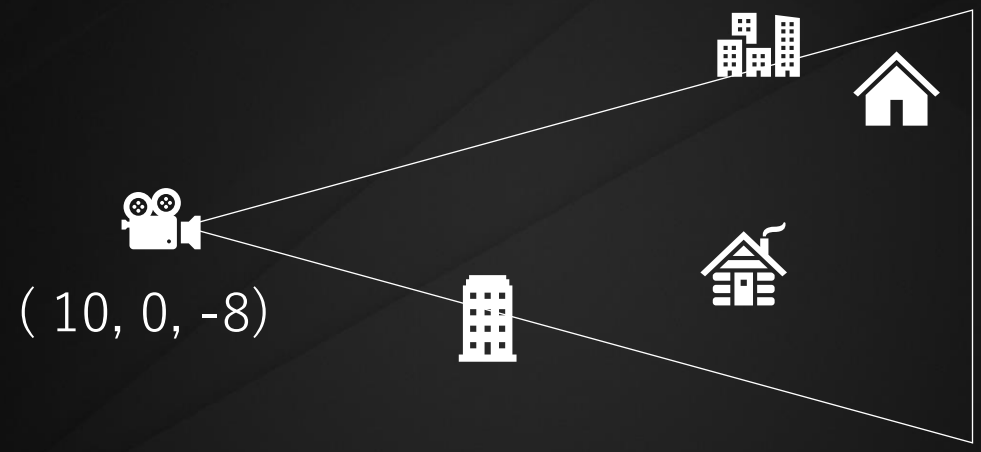


(9, 0, -9)

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
…	…

POLYPHONY™ DIGITAL

事前計算結果はこうだが



(10, 0, -8)

POLYPHONY DIGITAL

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

POLYPHONY DIGITAL

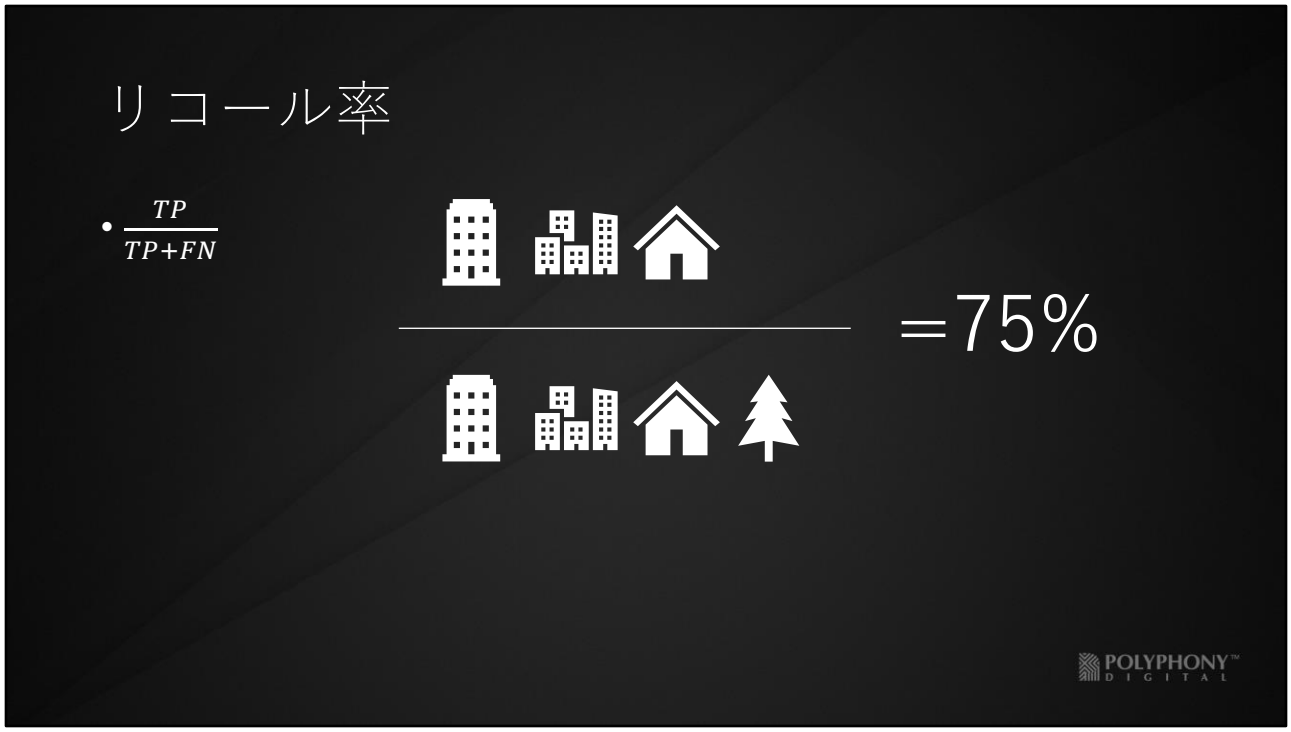
精度とリコール率

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

精度

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

精度が高いほど、余分なものが見えないということを意味します。つまり、パフォーマンスが向上する。



リコール率は逆で、高いほど、見えるべきものを見逃していない、ということになります。
つまり、リコール率が低いと、画像がバグってしまうことになります。これがゲームにおいては最も問題になります。

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

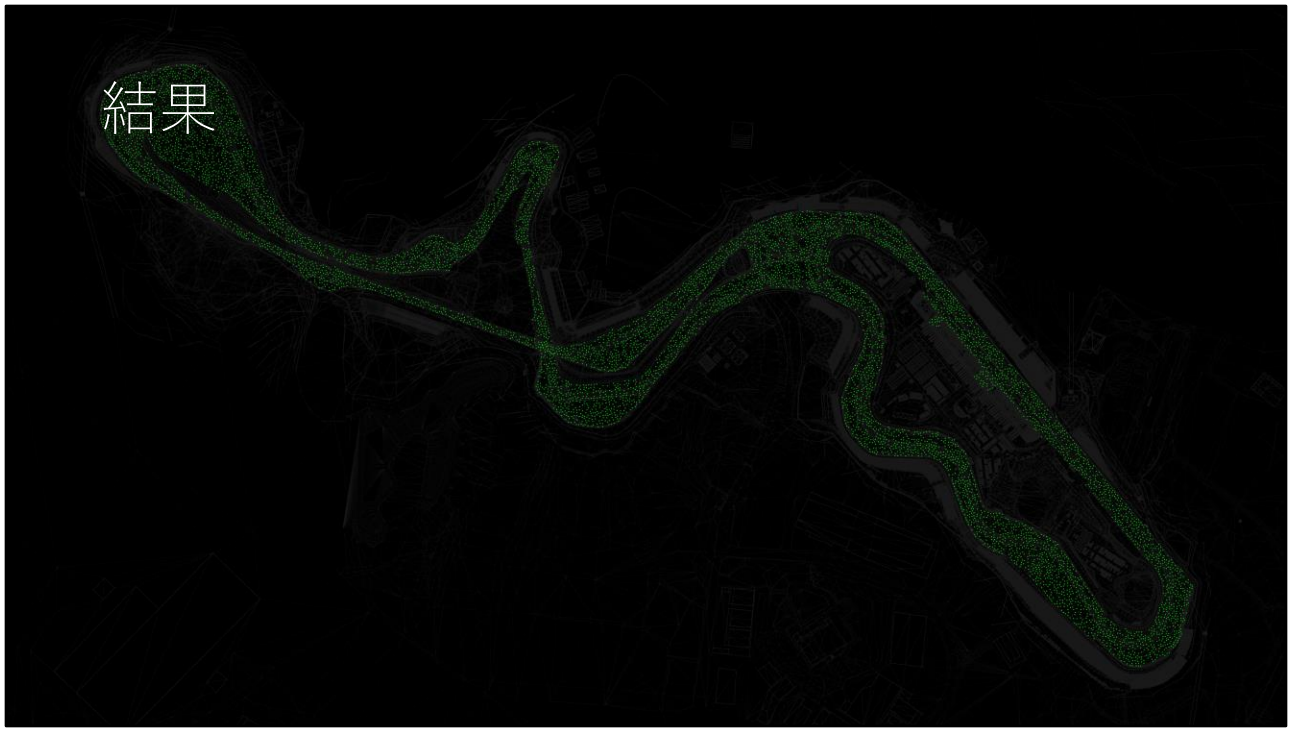
POLYPHONY™ DIGITAL

具体的な実装

カメラの移動範囲をどう求めるか

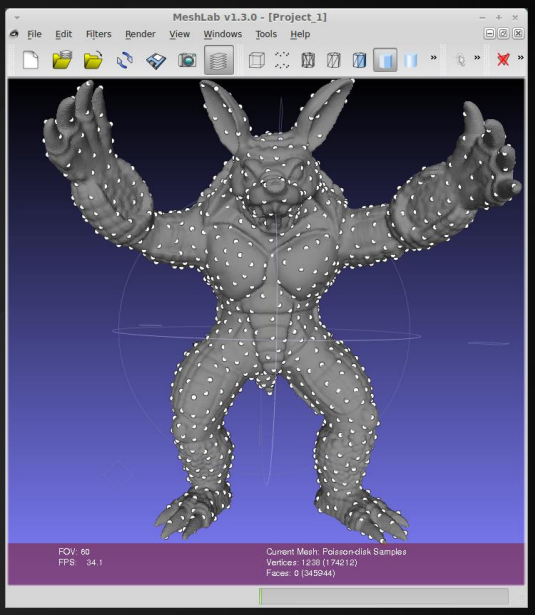
- カメラの範囲＝走行可能な範囲
- 走行可能な範囲＝路面メッシュ

- ポワソンディスクサンプリングする

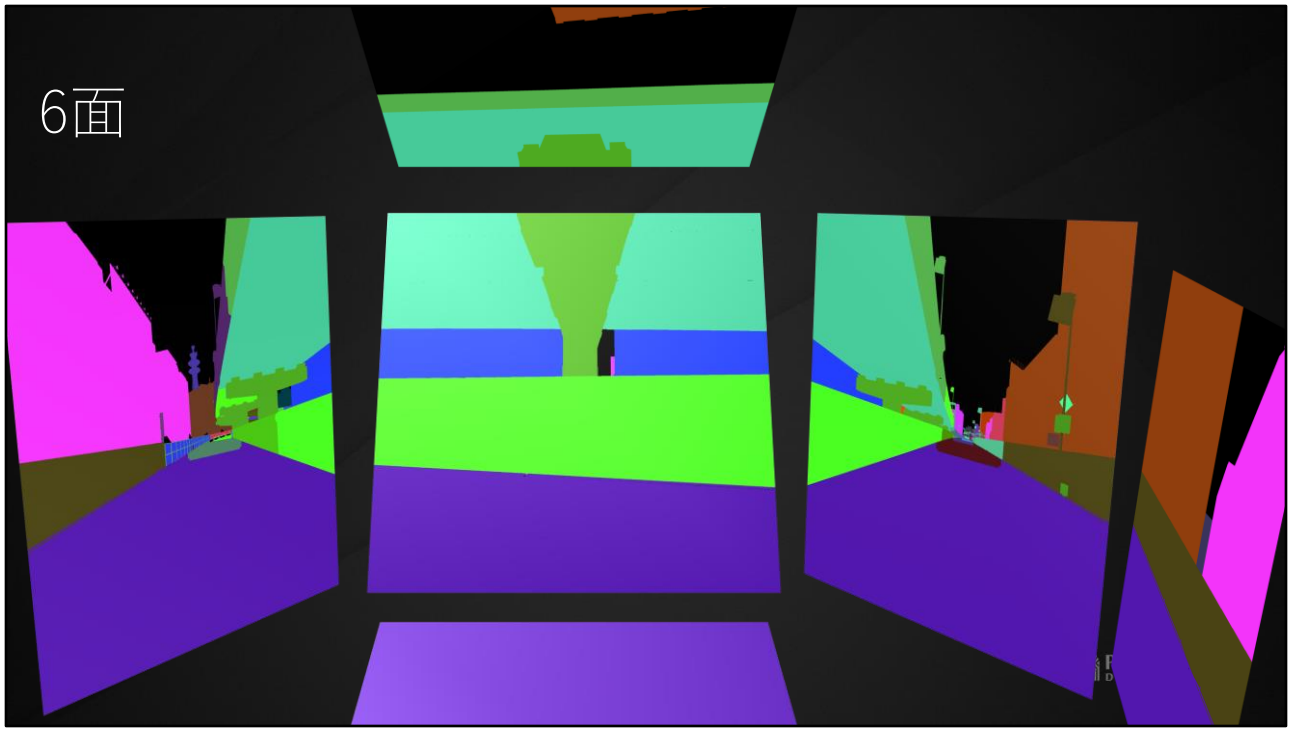

結果

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



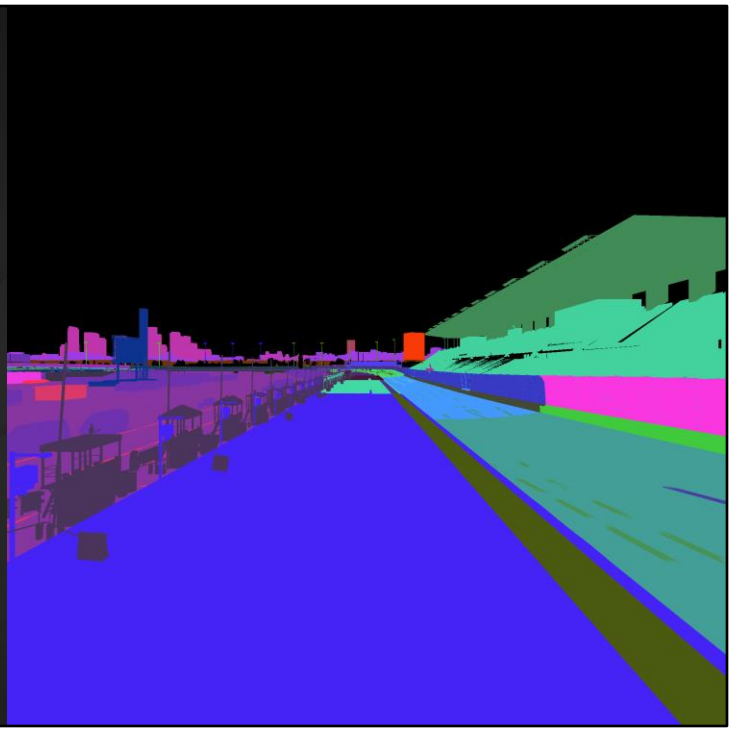
https://rodolphe-vaillant.fr/entry/37/source-code-for-poisson-disk-sampling-of-a-triangle-mesh


6面

オブジェクトID バッファ

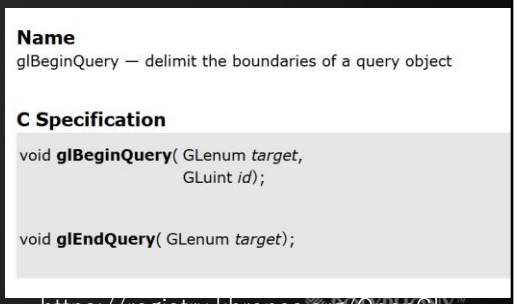
- OpenGLでレンダリング
- 各ピクセルはID

- 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クエリを用いた
- 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

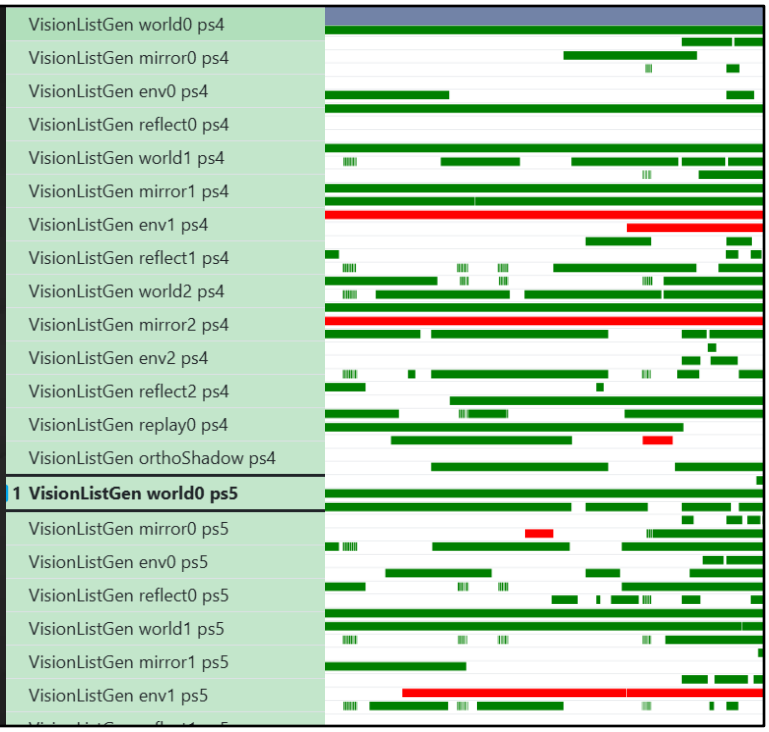
```
void glBeginQuery( GLenum target,
                   GLuint id);

void glEndQuery( GLenum target);
```

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

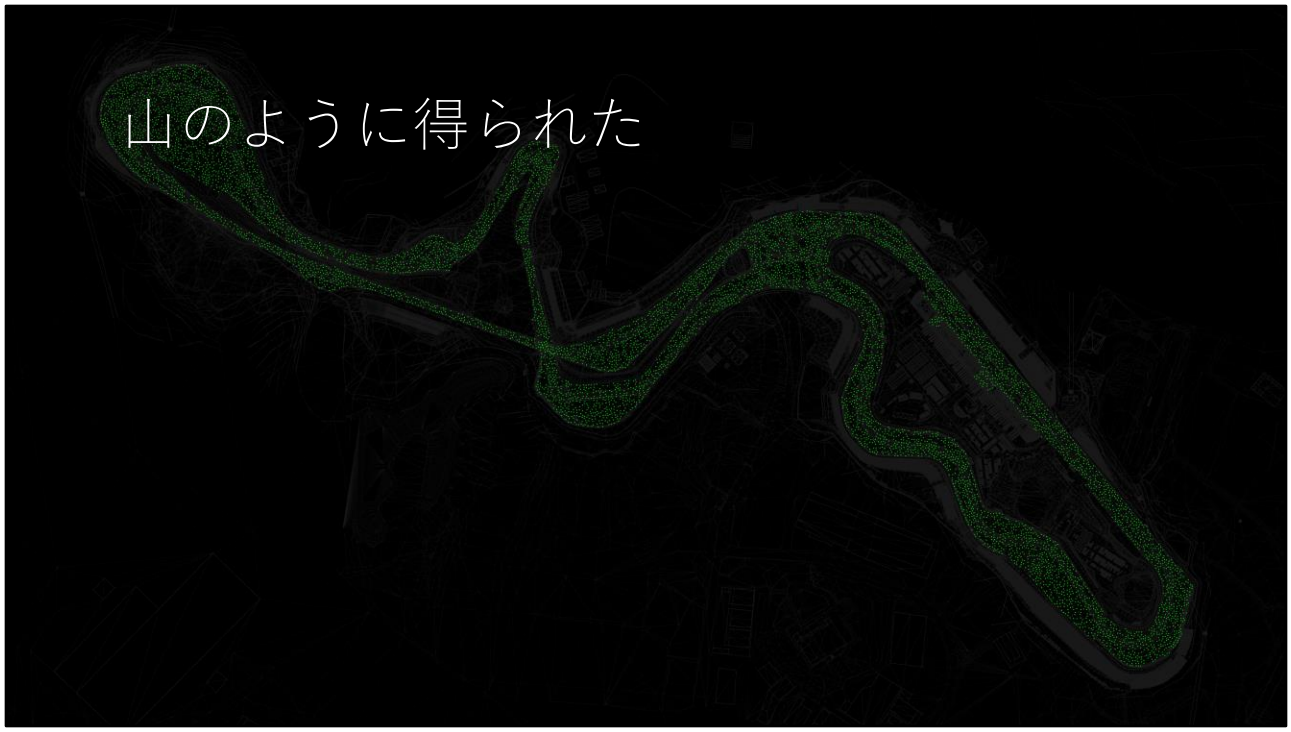
並列に生成する

- 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
i,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

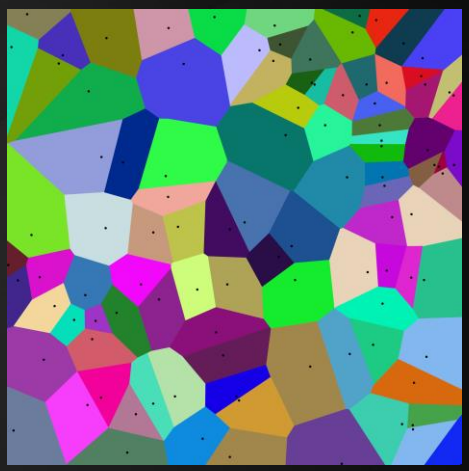
- このままでは非常に使いにくいデータなので簡略化したい

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

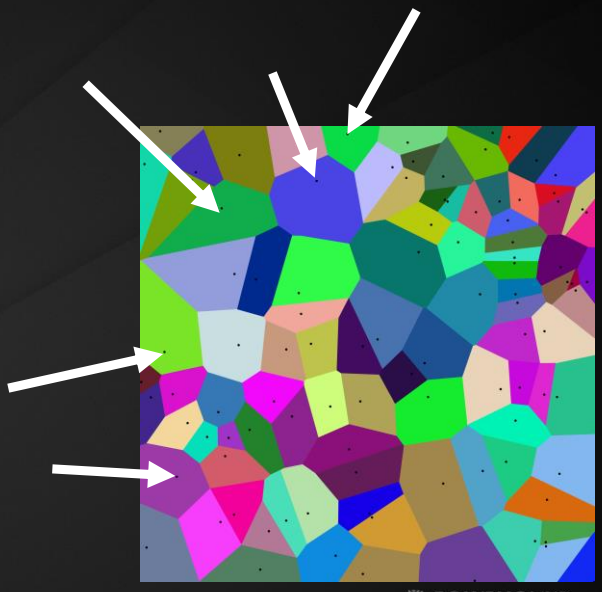
- 点の集合に対して、
  最も近い点に属するような領域分割



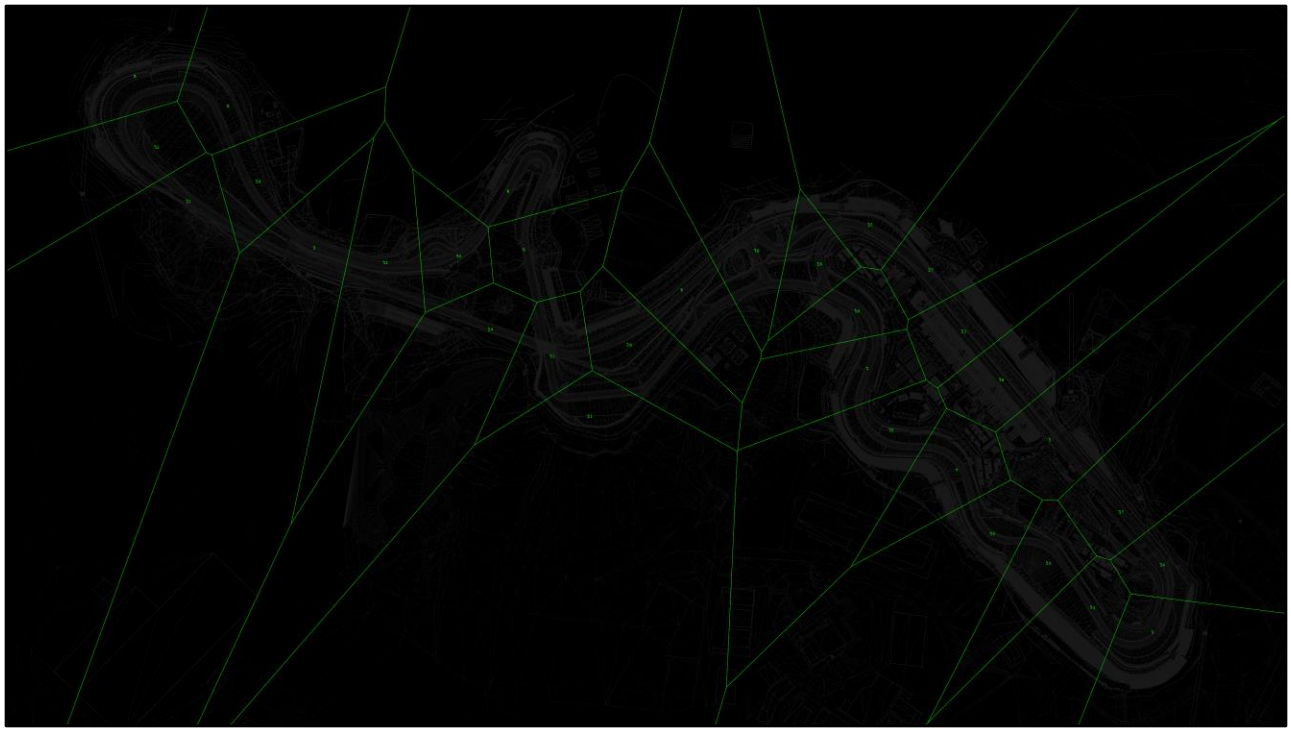
POLYPHONY™ DIGITAL

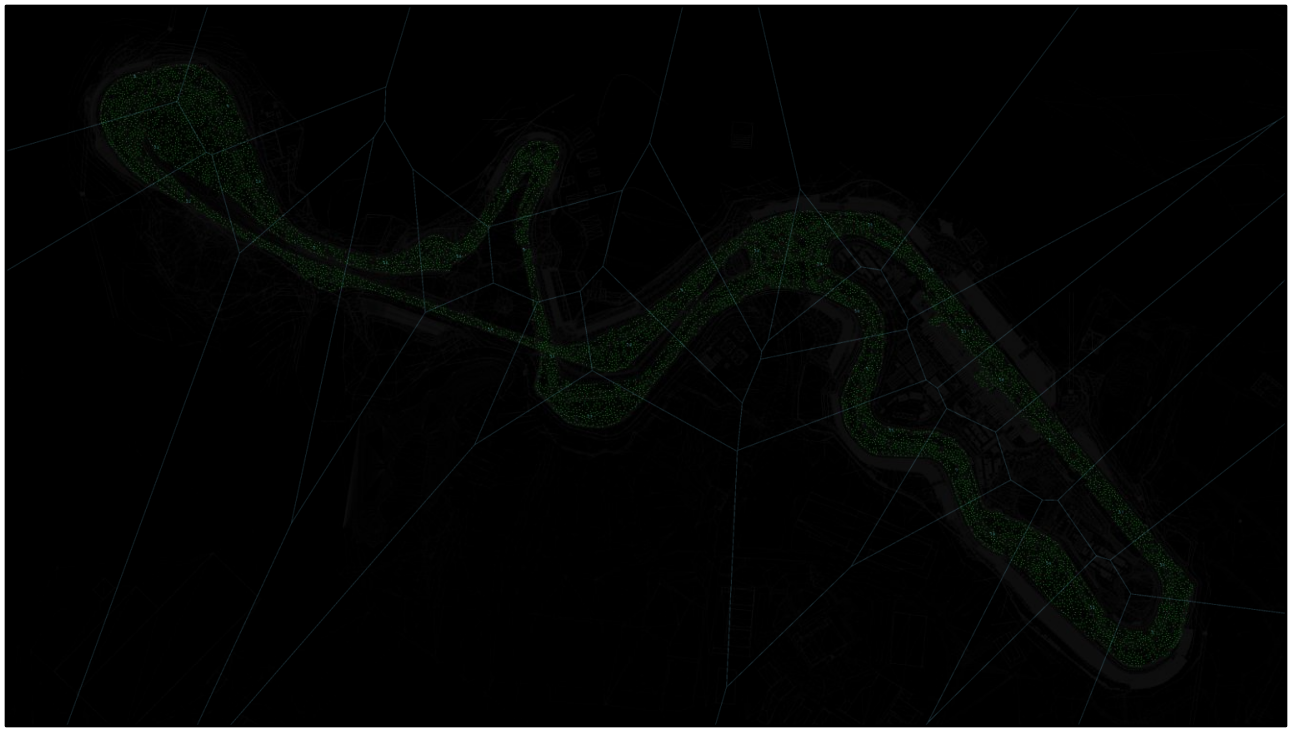
ボロノイ領域とは

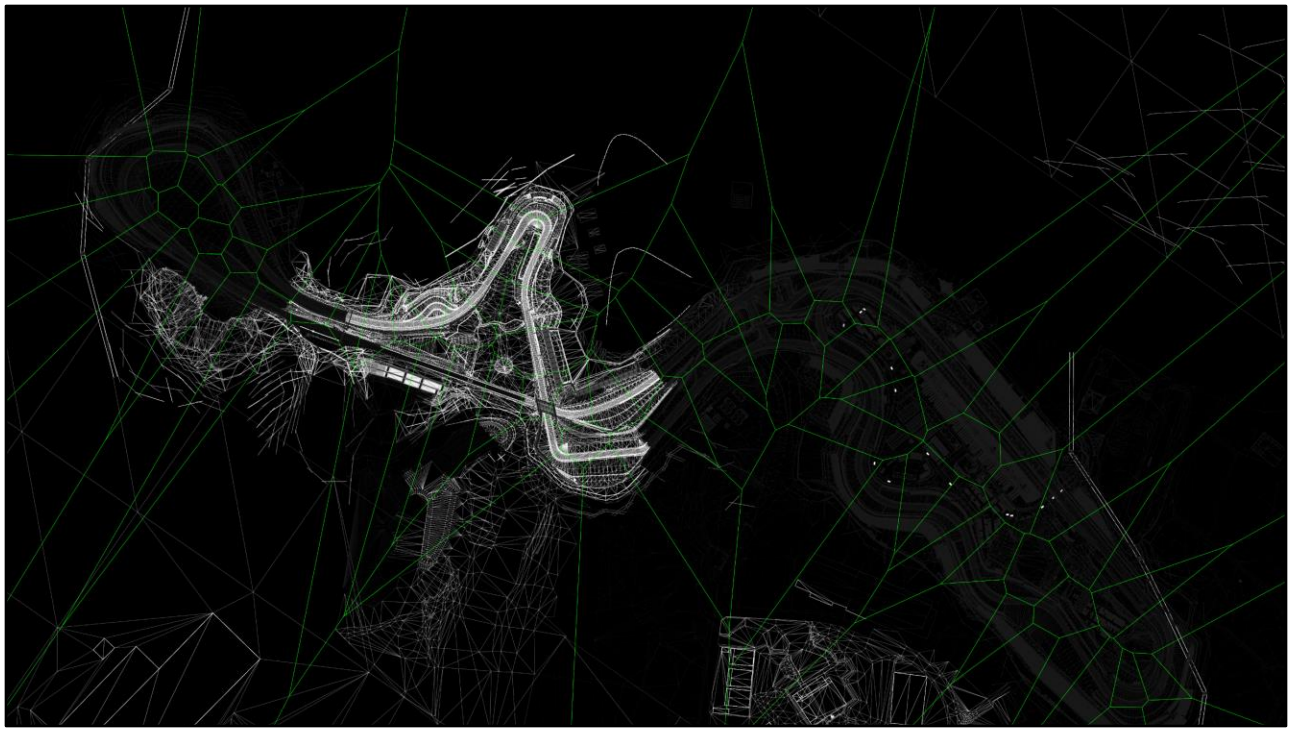
- それぞれの代表点を「母点」と呼ぶ

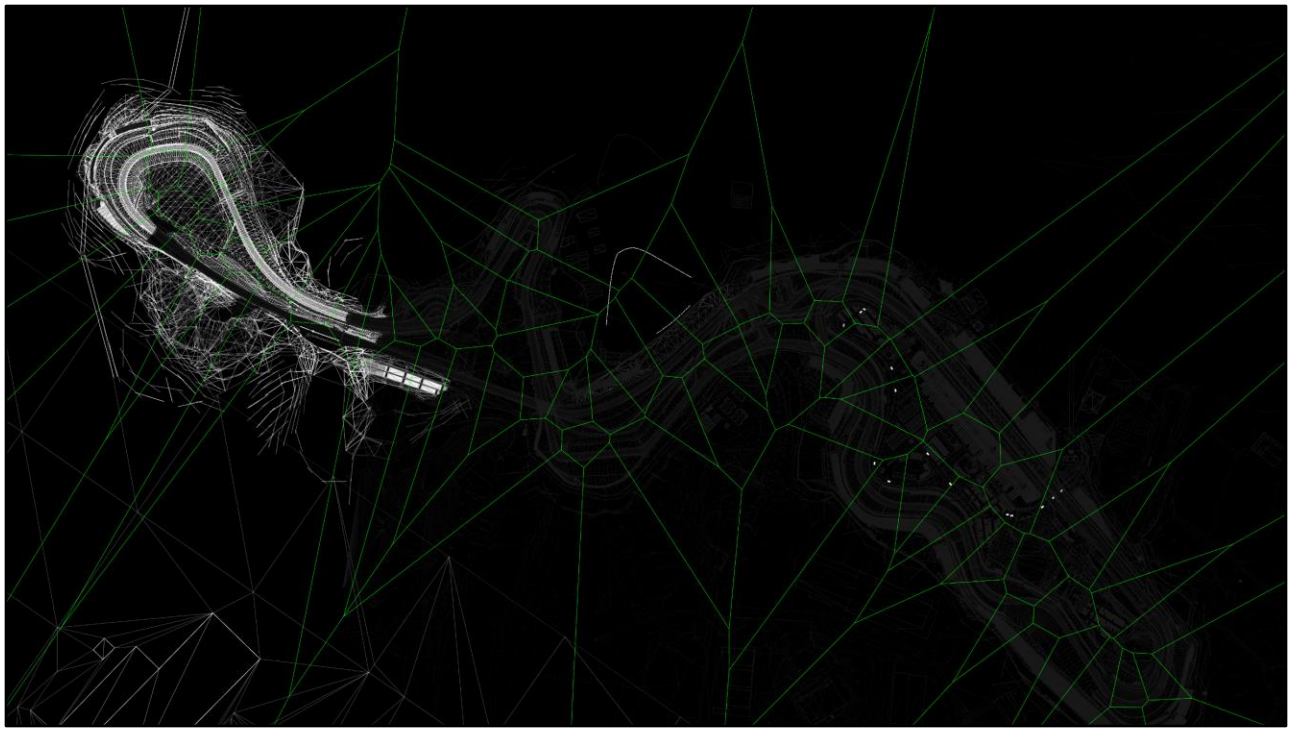


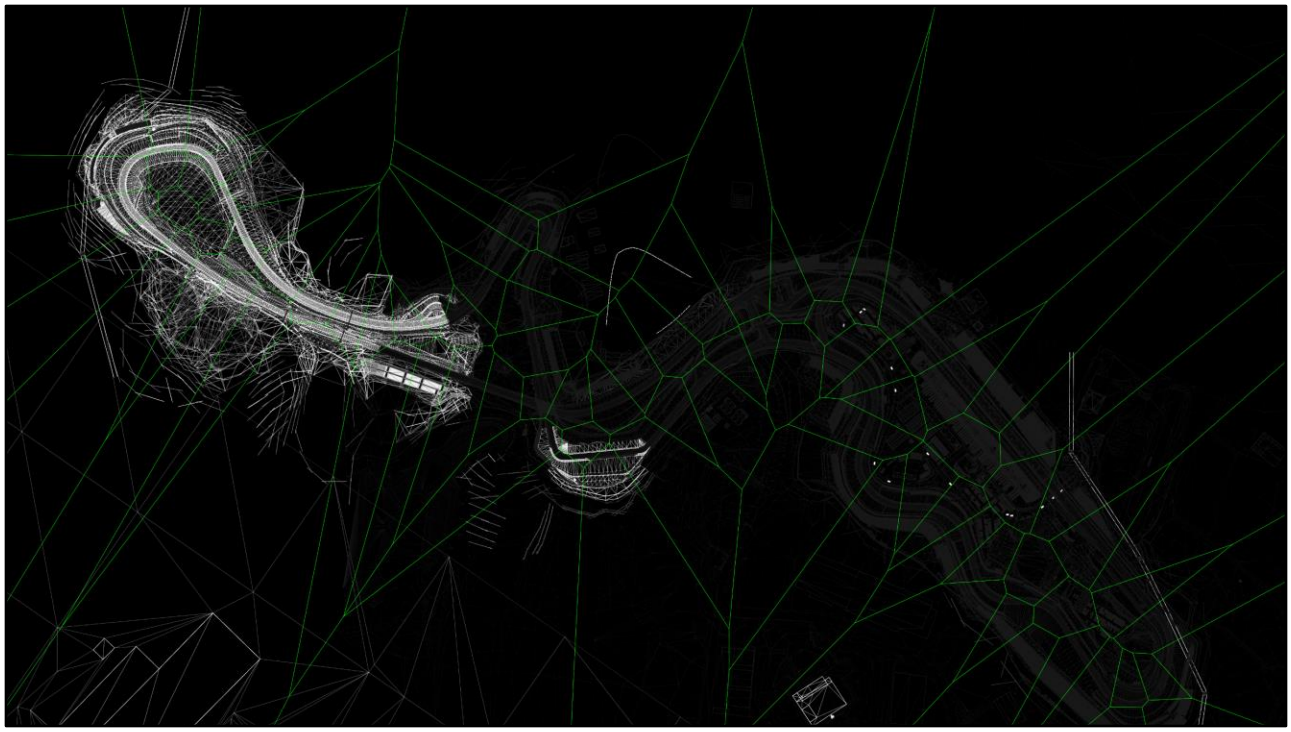
POLYPHONY™ DIGITAL









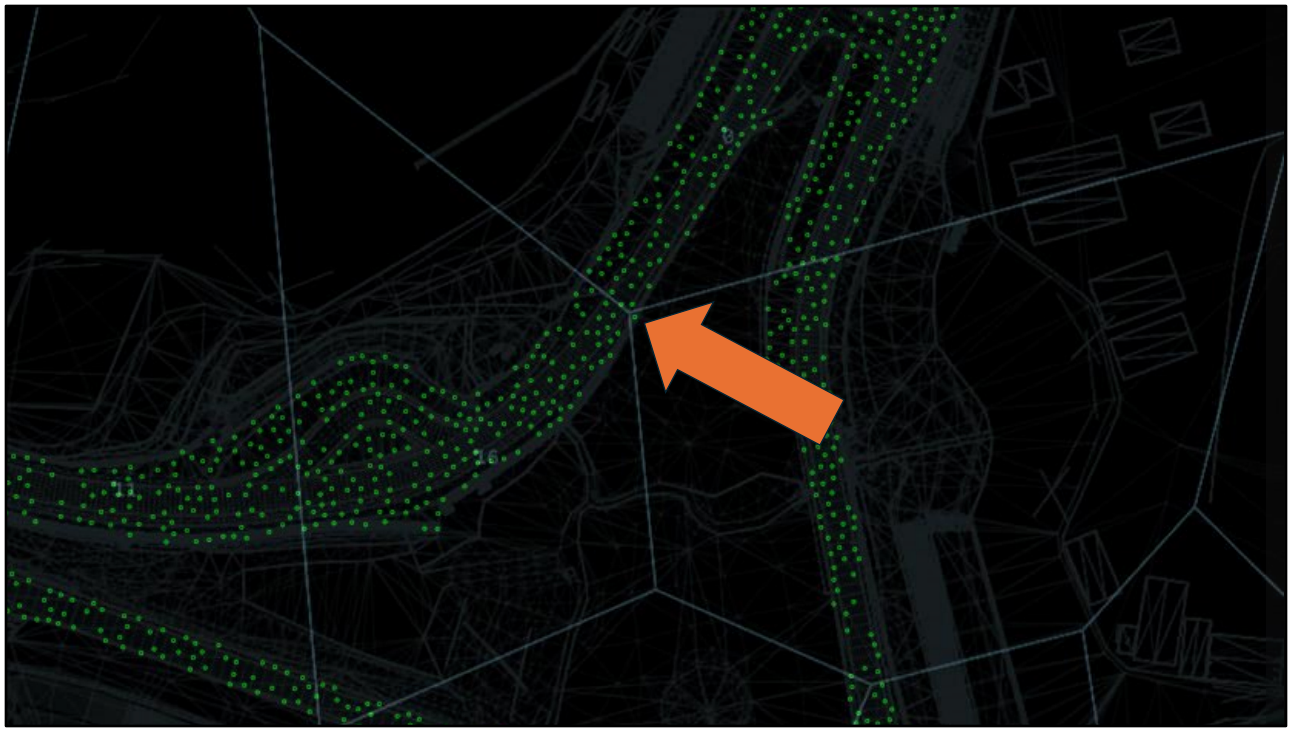


クラスタ化の目的

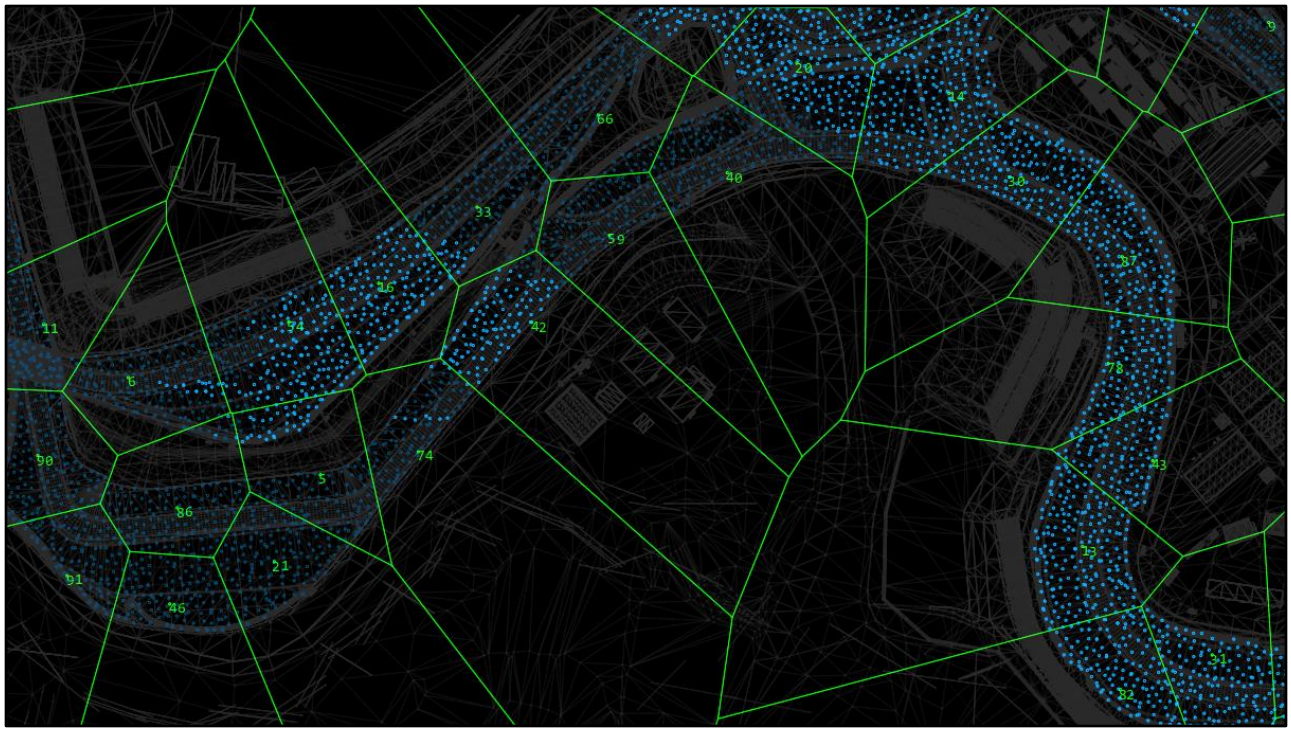
- データサイズの削減
- カメラ位置とビジョンポイントの位置の差による誤差の低減

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

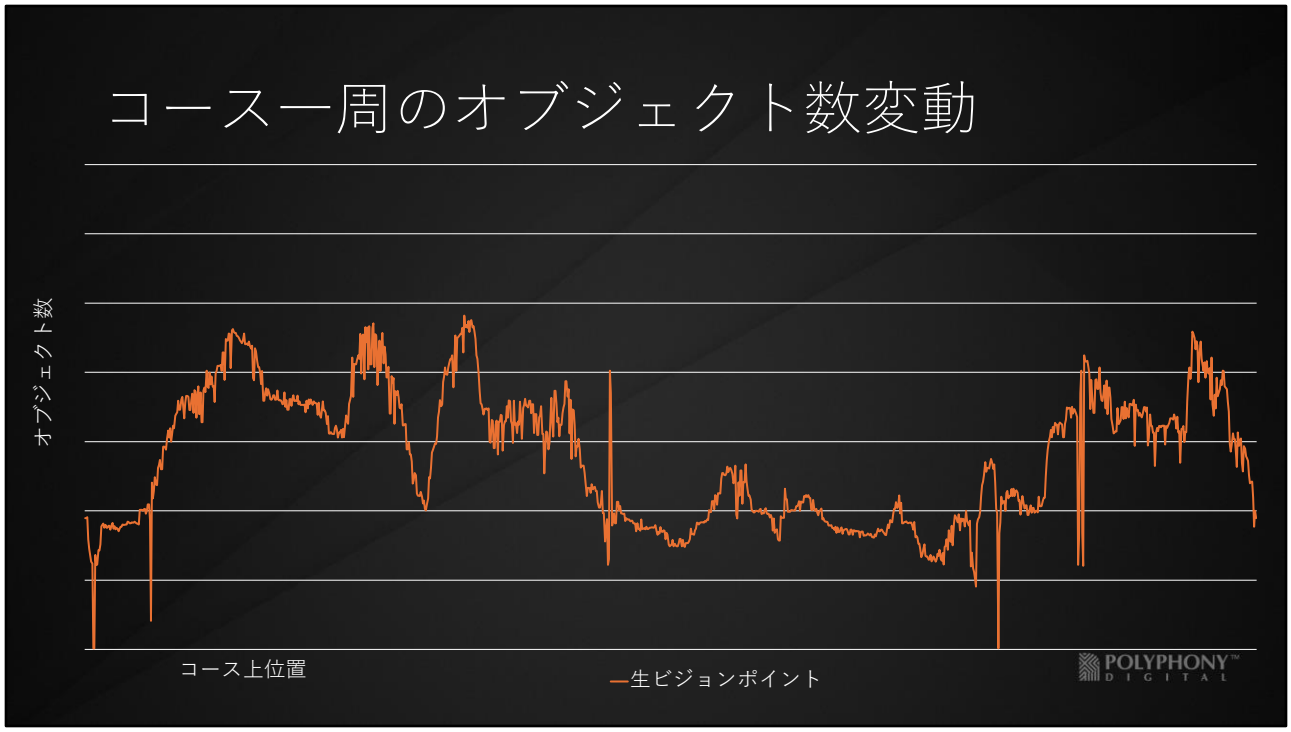


実際の可視領域は複雑な形状をしている

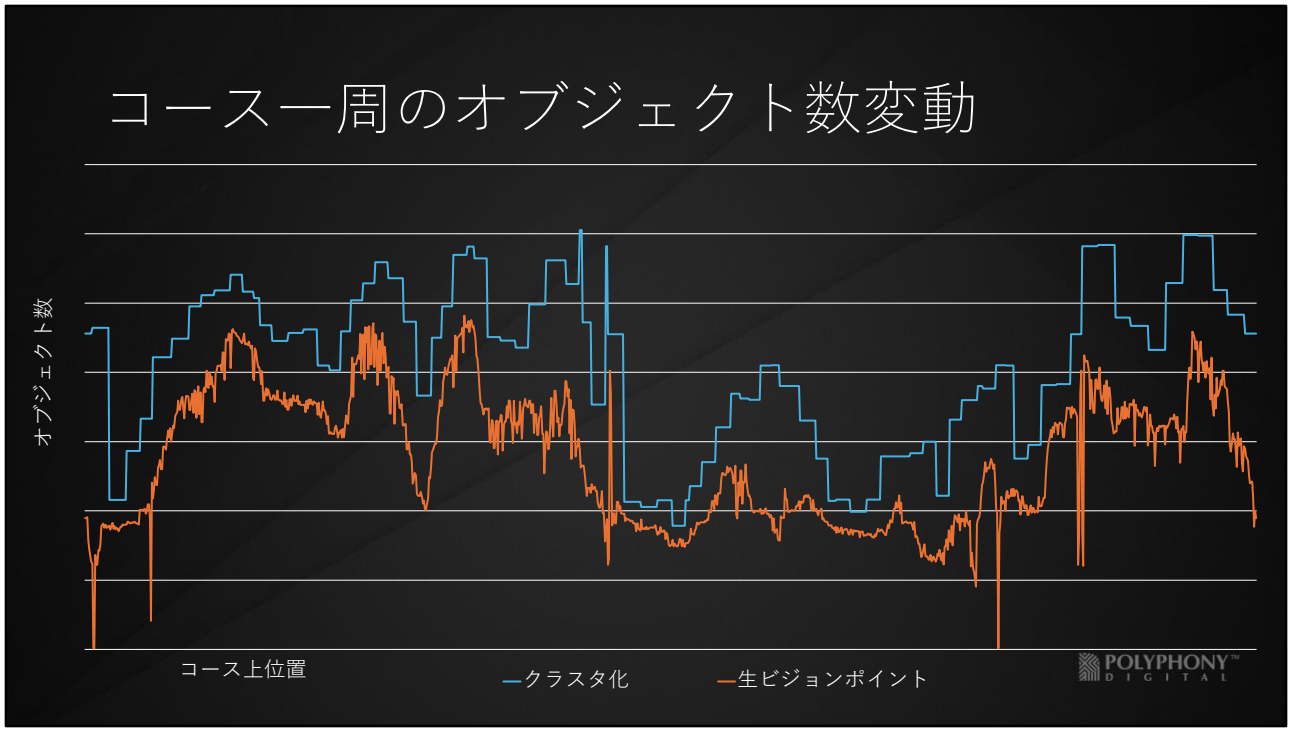


残念ながらこの可視領域の形状をもとにボロノイ領域を求めることは非常に難しく、消えすぎるよりは、出しすぎる方向、つまり安全側に倒すことで描画バグを防ぐのがクラスタ化の大きな特徴です。

この、実際に描画すべきオブジェクト数との差を埋めることがより最適で軽量な事前計算カリングシステムの実現のカギとなります。

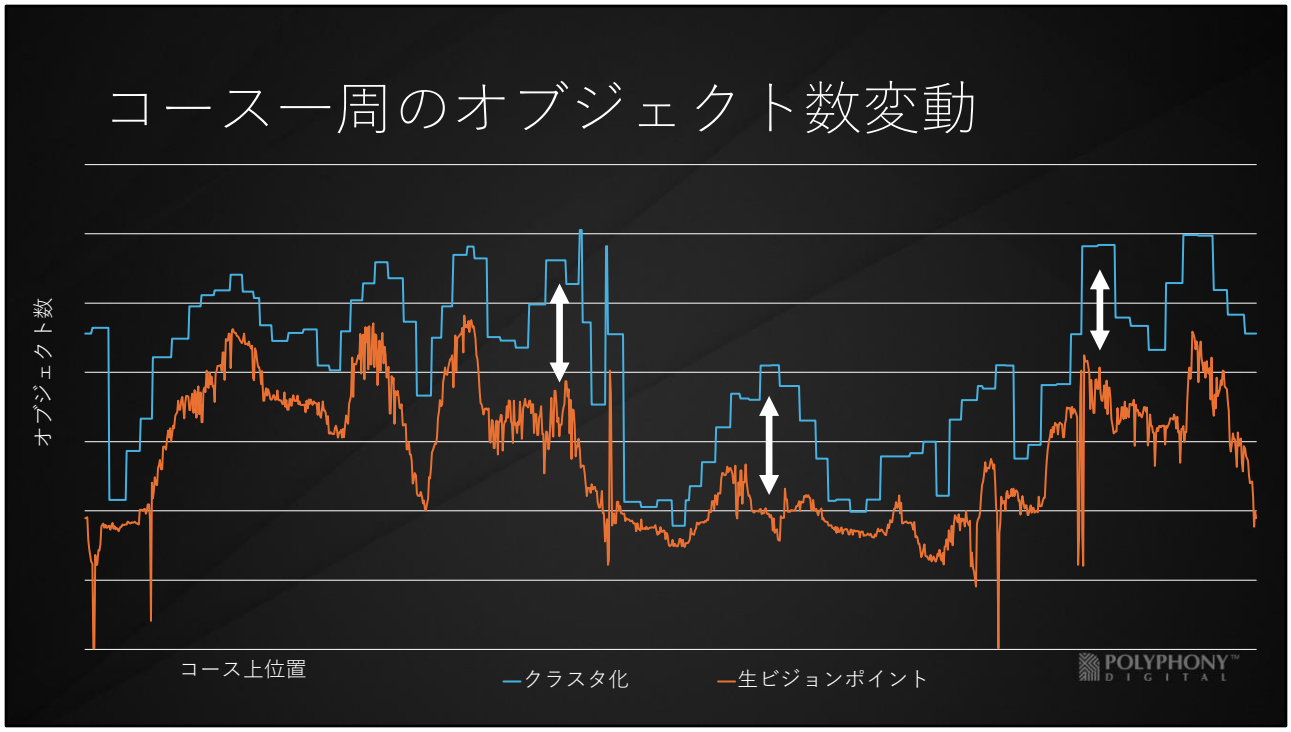


まず、計算したままクラスタリングしていないビジョンポイントを用いて、コースを一周するカメラから、見えるオブジェクトの数をプロットしてみました。
これが、理想的なカリングの結果です。
オブジェクトの見える範囲は非常に複雑な形状をしているため、パルスのように変動している部分があることがわかるかと思います。



このグラフに、クラスタ化されたビジョンリストでのオブジェクト数を追記するとこのようになります。

クラスタ化されたビジョンリストは、これを平均化することで、オブジェクトのポッピング等を防ぎつつ、
妥当な実行速度と、妥当な可視判定結果の両立を狙っています。



このふたつのグラフの間の差をより近づけることで、事前計算カリングシステムの結果をより最適にすることが可能になるはずです。
つまり、リコール率と精度を上げていく手法が必要になるわけです。

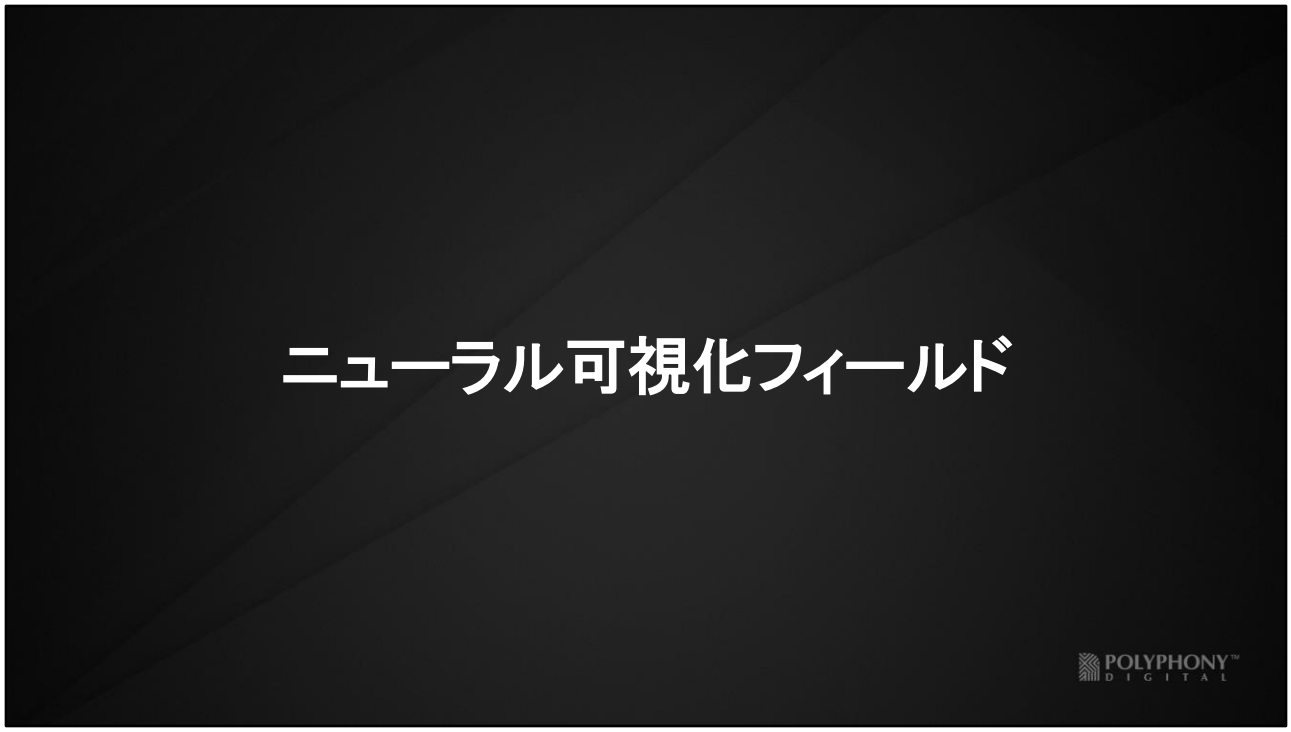
ここまでのまとめ

- 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

POLYPHONY™ DIGITAL



そこで我々はニューラル可視化フィールドを提案します。

モチベーション

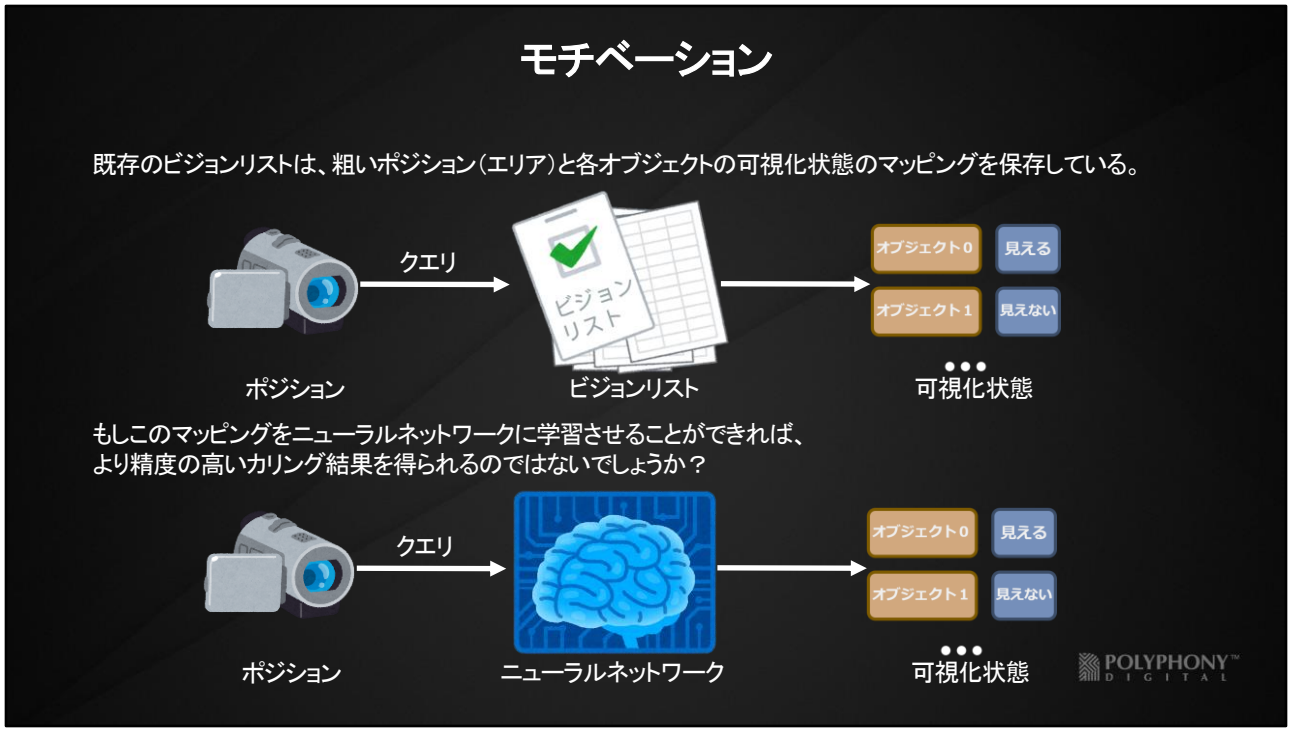
GDC Machine Learning Summit: A Robust and Fast ML-Based View of the **Frustum Culling** Method.
(堅牢で高速な機械学習ベースの視点**フラスタムカリング**手法)

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。

POLYPHONY DIGITAL

ニューラル可視化フィールドは、先ほど述べた、クラスタ化ビジョンリストの欠点を解決するための新しい方法です。

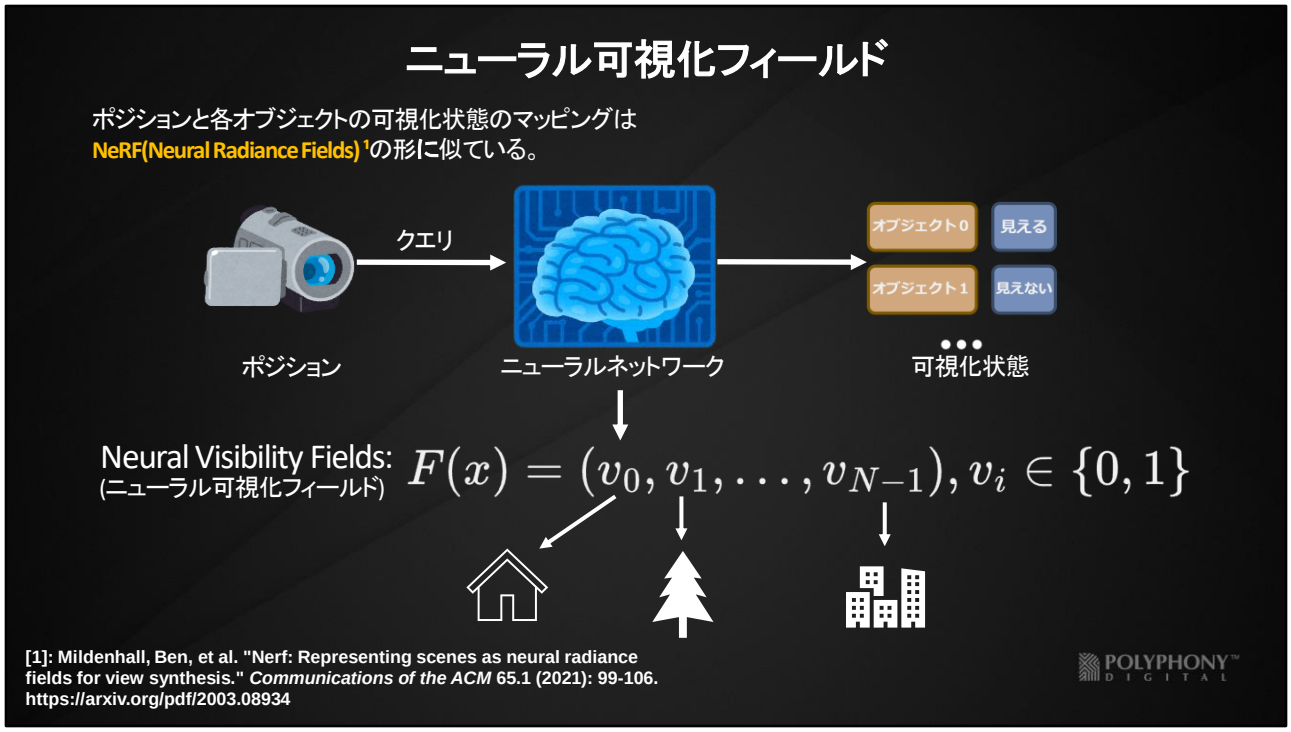
GDCでのニューラルネットワークを用いたフラスタム　カリングに関する講演（こうえん）にインスパイアされて、ニューラルネットワークはオクルージョンカリングにも応用できると感じ、開発しました。



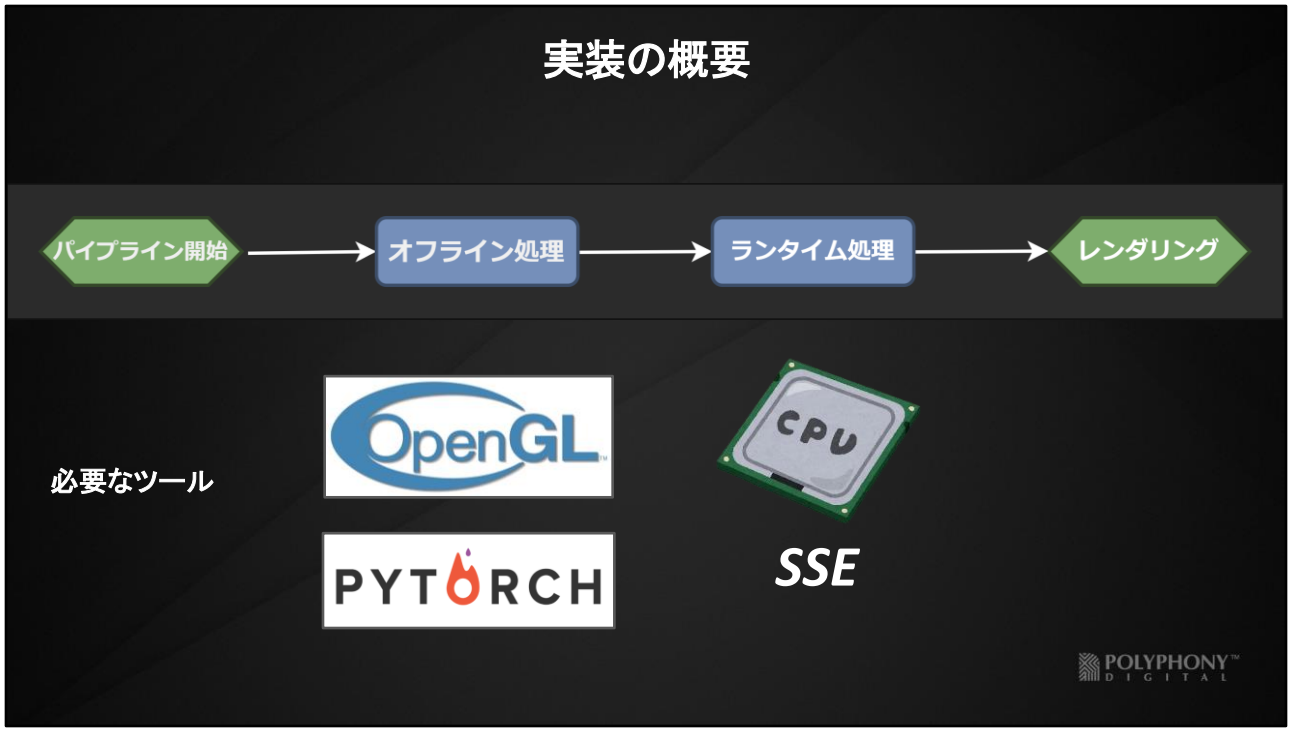
観察を通じて

既存のVision Listは、粗いポジション、つまりある程度広いエリアと、各オブジェクトの可視化状態のマッピング情報を保存しています。

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が得られるかもしれません。
では、具体的にどう進めるべきでしょうか。次に詳しく解説(かいせつ)します。

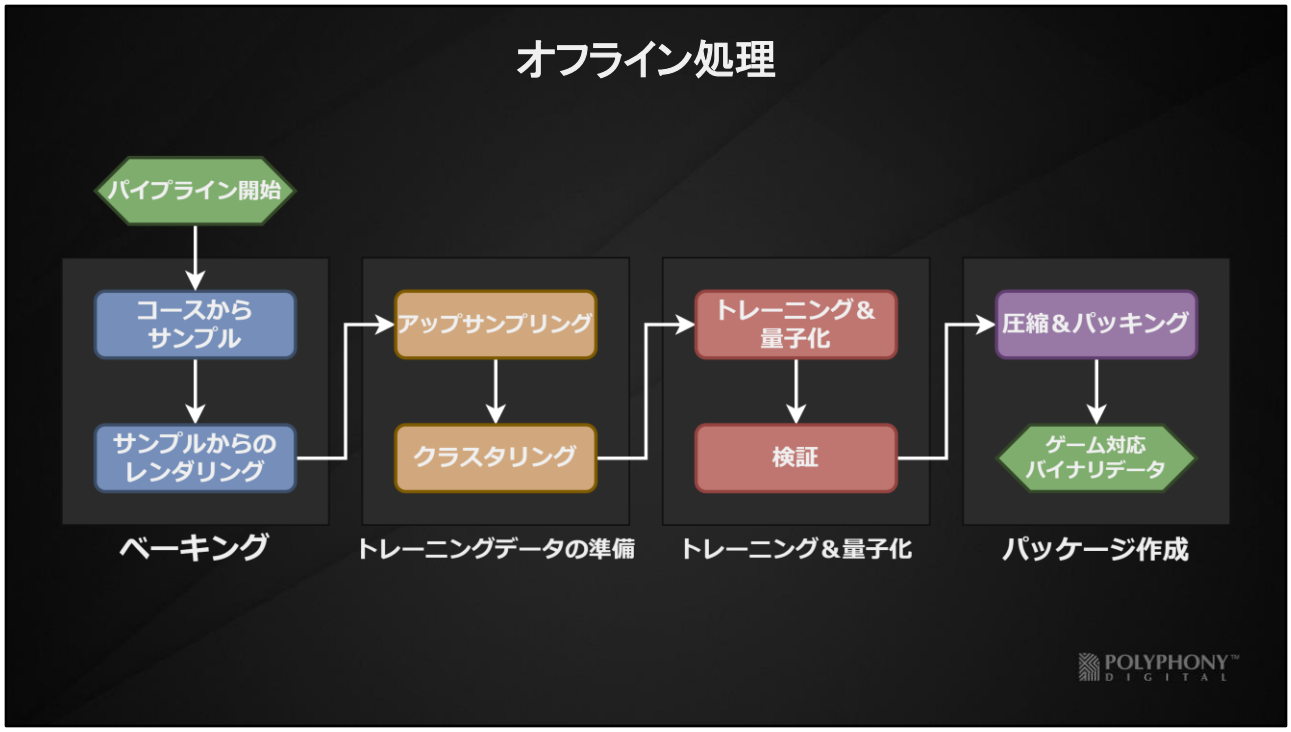


ポジションと各オブジェクトの可視化状態のマッピングは、NeRF（Neural Radiance Fields）の形に似ています。
そのため、このタスクを次の式に変換して、形式を　ニューラル可視化フィールド「Neural Visibility Field」と呼びます。

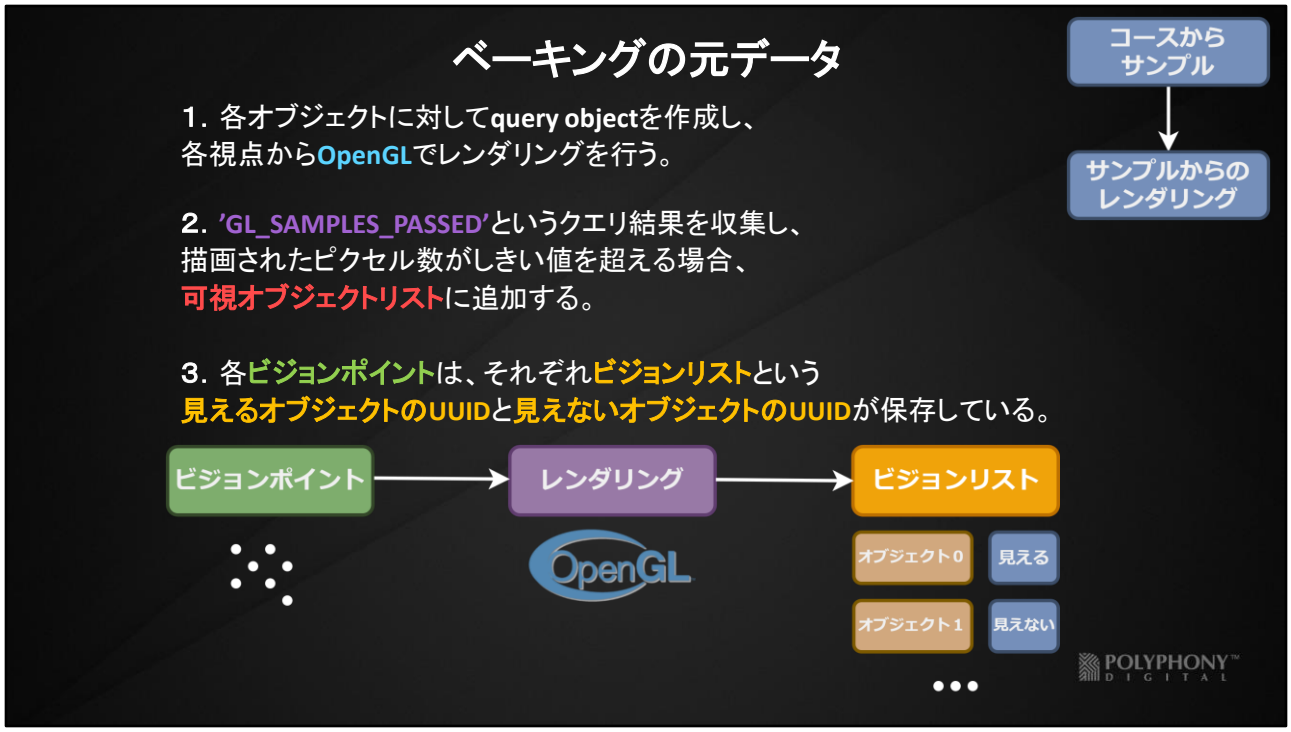


グランツーリスモ　セブンの実装は2つの部分に分(わ)かれています。

1つはOpenGLとPyTorchを使用するオフライン処理部分で、
もう1つはCPUのSSE命令セットを使用するランタイム部分です。
カリングされたデータは最終的にエンジンに渡され(わたされ)、レンダリングされます。



まず、オフライン処理の部分に入ります。
下のフローチャートからわかるように、オフライン処理は
ベイキング、トレーニングデータの準備、トレーニング＆量子化、そして最後のパッケージ化の4つに（わかれています）分かれています。
このうち、最初の3つの部分について詳しく説明します。

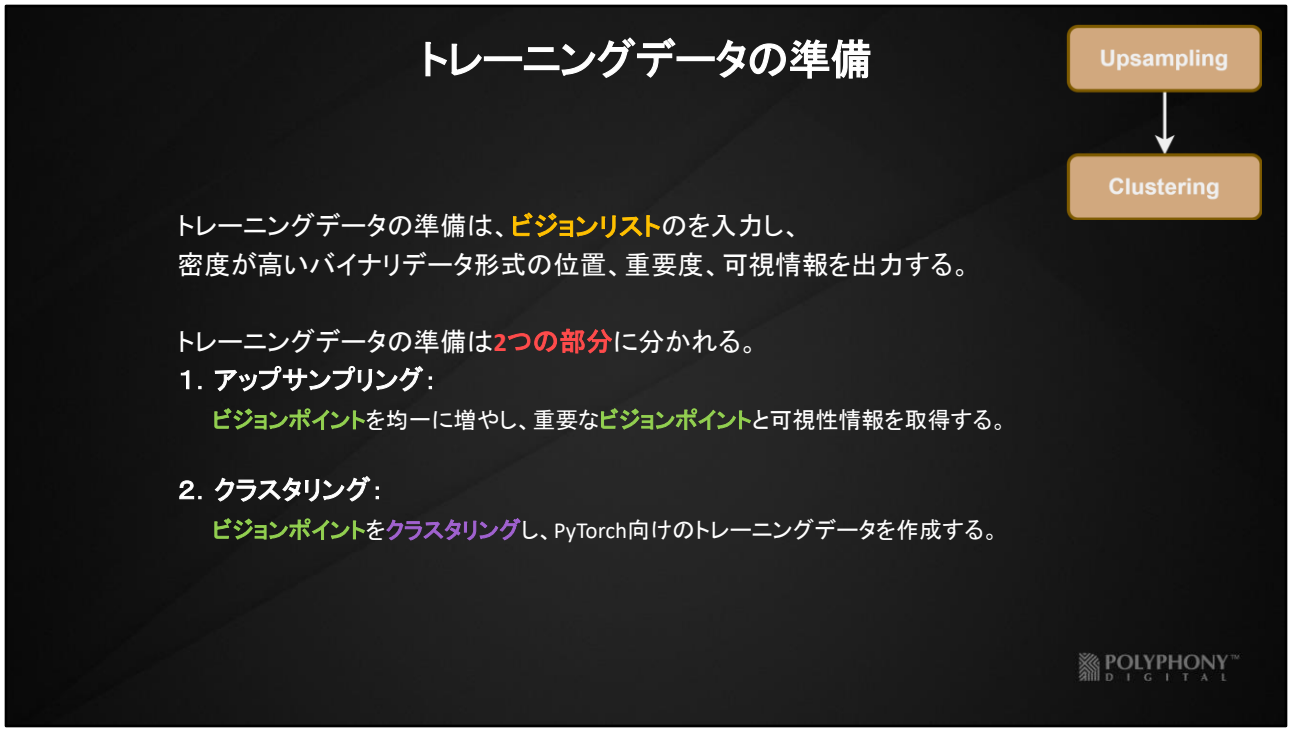


先ほど内村さんが紹介してくださったように、
私たちはOpenGLを使用してレンダリングを行い、
一連のビジョンリストを得ました。

この部分は、以前のスライドの内容と同様です。

次に、このビジョンリストを使ってトレーニング用データを作成します。

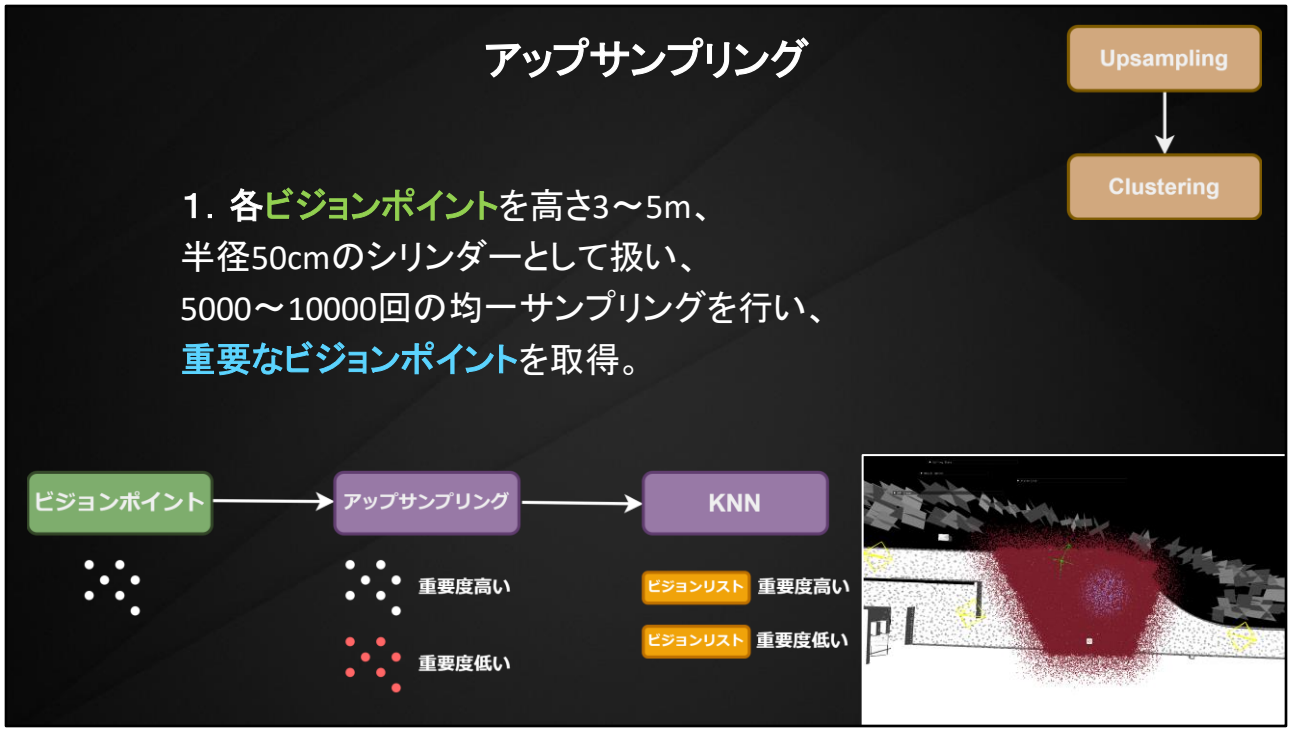
次に、トレーニング用のデータについて説明いたします。



トレーニングデータの準備では、ビジョンリストを入力として、密度が高いバイナリデータ形式で、ポジション、重要度、可視情報を出力します。
このトレーニングデータの準備は2つの段階に(わかれています)分かれています。

まず1つ目が「アップサンプリング」です。
ここでは、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抽出します。

そして2つ目が「クラスタリング」です。
ここでビジョンポイントをグループ化し、PyTorchでのトレーニングに適したデータを作成します。

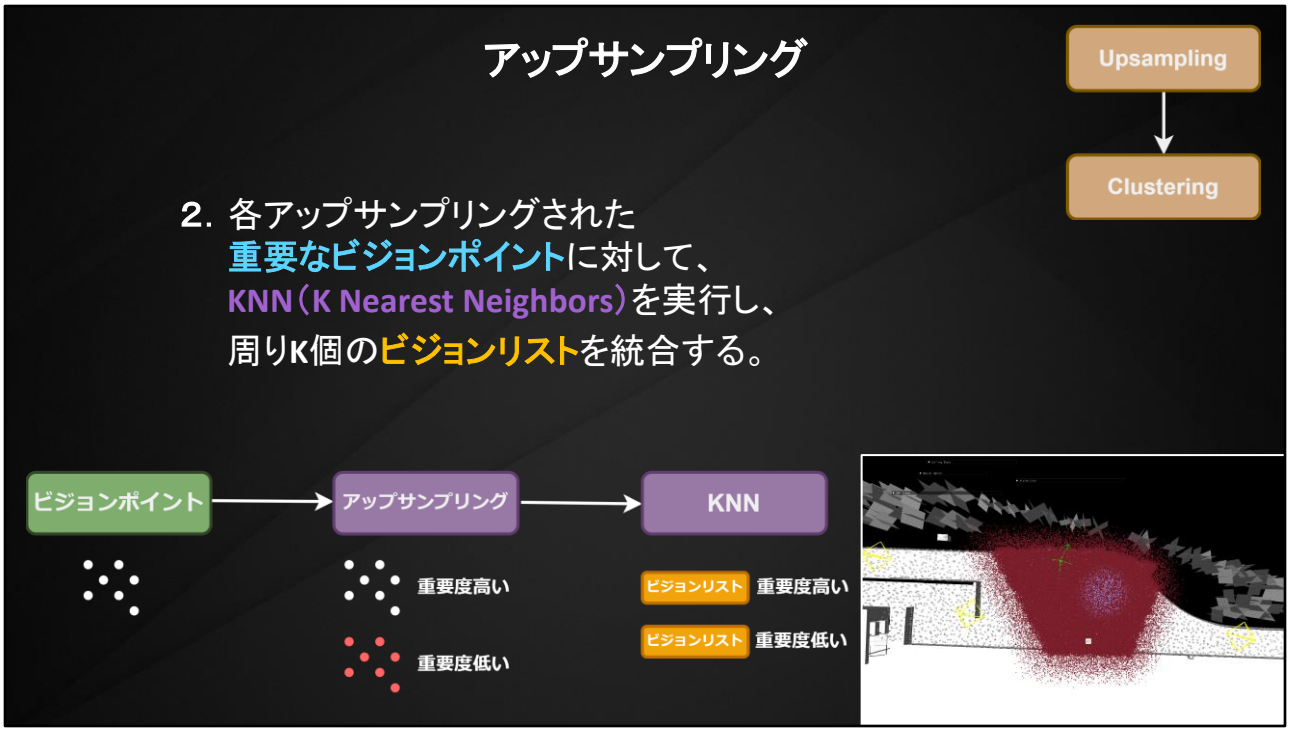


次に、アップサンプリングについて説明します。

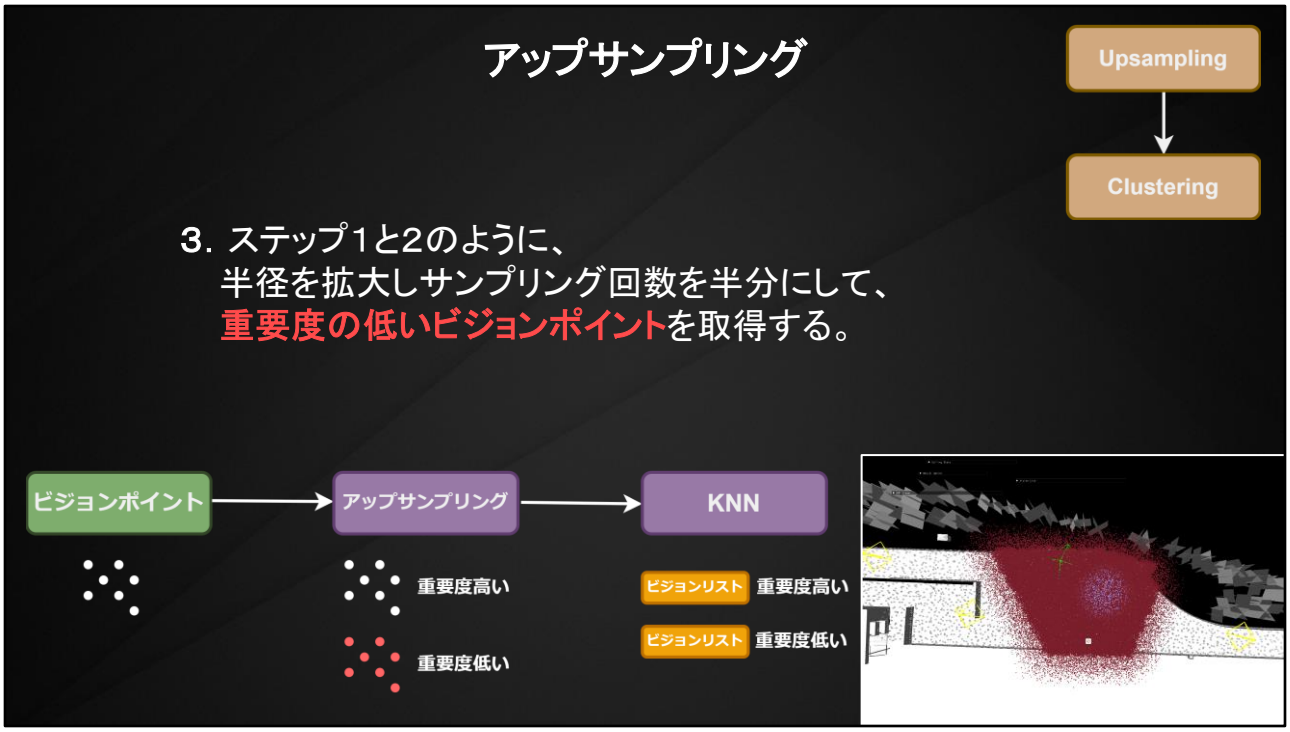
ニューラルネットワークを学習させるには、元のビジョンポイントでは量や密度が不足しているので、これをアップサンプリングして、学習データの連続性を向上させます。

まず、各ビジョンポイントを高さ3～5m（メートル）、半径50cm（センチメートル）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ます。

これにより、特に重要なビジョンポイントを抽出します。



次に、アップサンプリングされたそれぞれの重要なビジョンポイントに対して、KNN（K近傍法(きんぼうほう)）を用いて、周囲のK個のビジョンリストを統合します。



最後に、ステップ1と2と同様に、シリンダーの半径を拡大し、サンプリング回数を半分にすることで、重要度の低いビジョンポイントを抽出します。

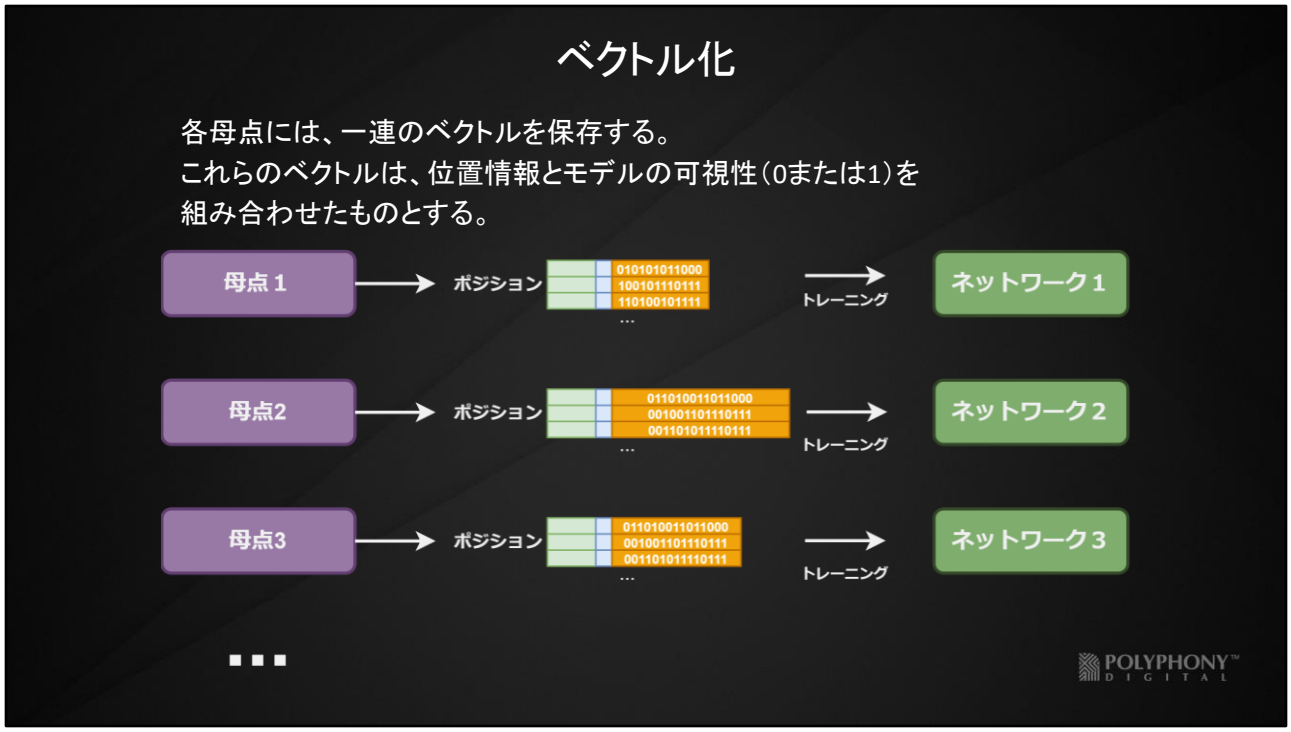
次に、得られた重要なビジョンポイントと重要度の低いビジョンポイントに対して行うクラスタリングについて説明します。

まず、**1つ目**として、重要なビジョンポイントにポアソンサンプリングを実行し、N個の母点を生成します。
それぞれの母点には個別のニューラルネットワークが割り当てられます。

次に、2つ目として、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、それぞれもう一度ポアソンサンプリングを行い、さらに均一に分布したビジョンポイントを取得します。

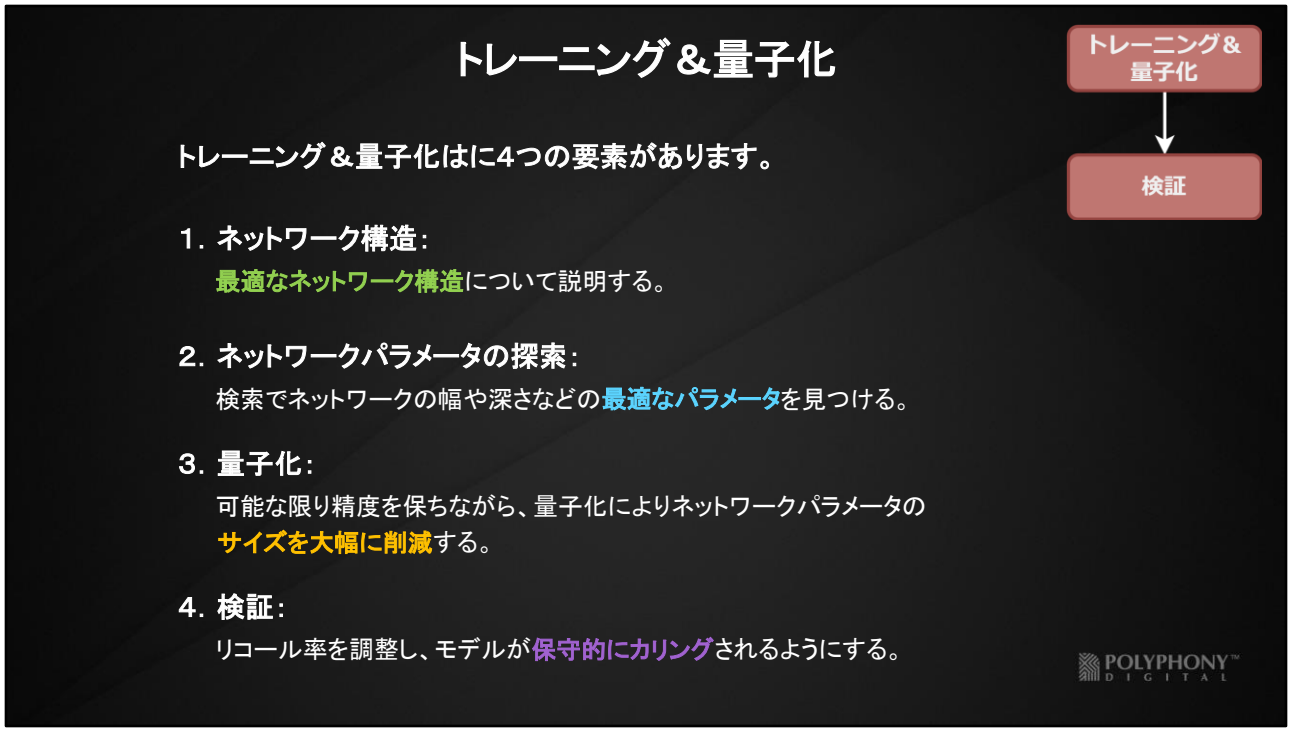
3つ目として、各ビジョンポイントについて最も近い母点を見つけ、その母点に割り当てます。
クラスタ化ビジョンリストとほとんど同様の処理になります。

最後に4つ目として、PyTorchで読み取れるように、各母点内のビジョンポイントに関して、位置、重要度、そして0と1で表現されたビジョンリストをベクトルとして出力します。
ここでは、直接**バイナリデータ**として出力することにします。



先ほどのベクトル化について、具体的に説明いたします。
まず、各母点には一連のベクトルが保存されています。これらのベクトルは、ポジション情報とモデルの可視性を示す0または1を組み合わせたものです。
このベクトル化に加えて、クラスタリングを活用することで、エリアごとの複雑さに応じて異なる複雑さのネットワークを適用することが可能になります。
また、各ネットワークは相互に独立しているため、トレーニングを分散して効率的に実行できるという利点もあります。

次に、トレーニング＆量子化の部分に入ります。



トレーニング＆量子化は、4つの要素に分かれます。

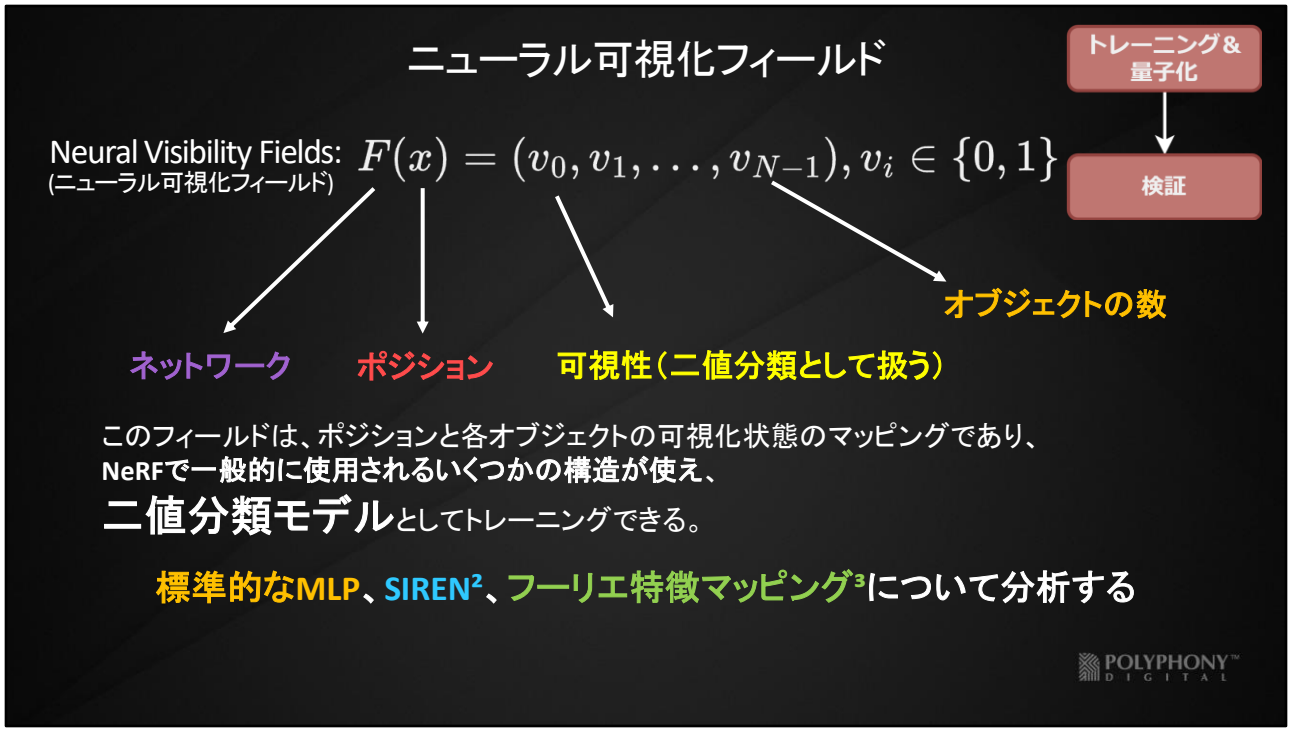
まず、1つ目は「ネットワーク構造」です。
ここでは、最適なネットワーク構造について説明いたします。

次に、2つ目は「ネットワークパラメータの探索」です。
ここでは、検索を通じてネットワークの幅や深さなど、最適なパラメータを見つけます。

3つ目は「量子化」です。できる限り精度を保ちながら、量子化によってネットワークパラメータのサイズを大幅に削減します。

そして最後の4つ目が「検証」です。ここでは、リコール率を調整し、オブジェクトが保守的にカリングされるようにします。

トレーニング

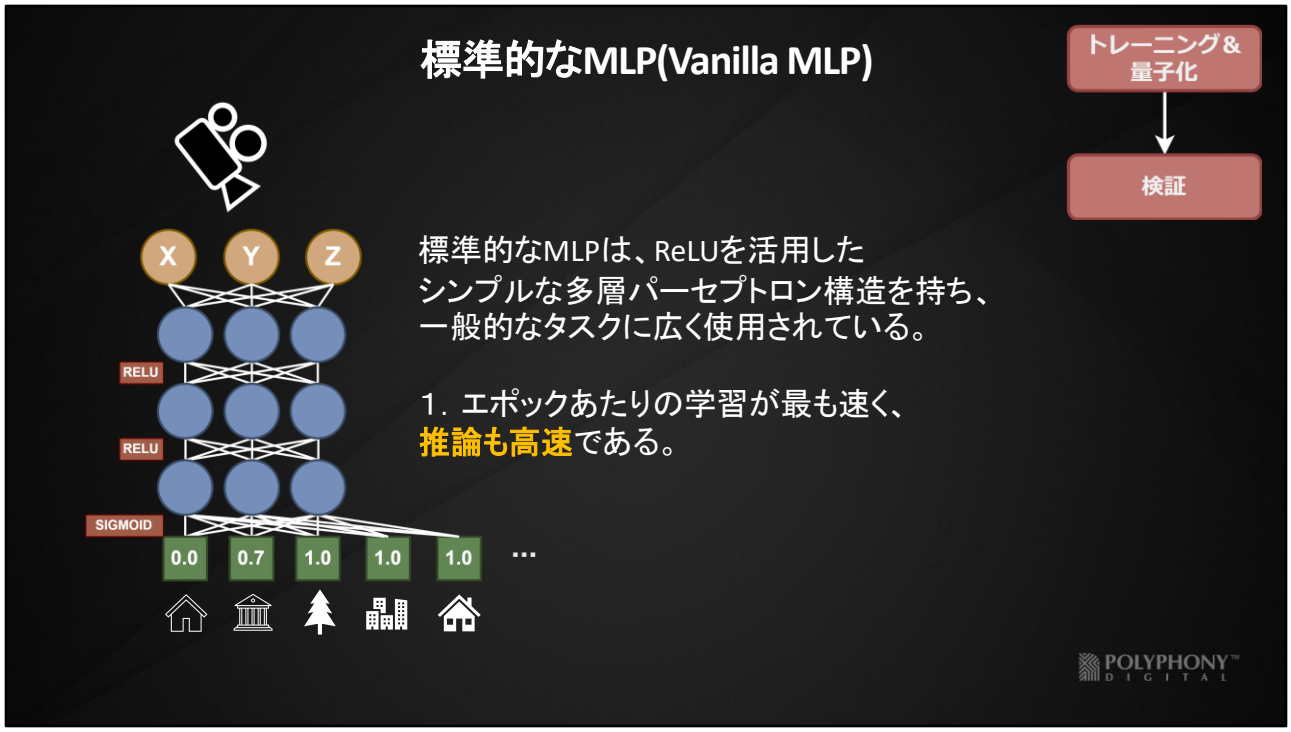


先ほど紹介されたNeural可視化フィールドは具体的に次のようになります。

入力はPositionで、
出力は0と1で構成される長さがオブジェクトの数に等しいベクトルです。

この出力された可視性情報は、二値分類として扱います。
また、NeRFで一般的に使用されるいくつかの構造を活用できます。

そのため、標準的なMLP、SIREN、フーリエ特徴マッピングなどのストラクチャーについて分析していきます。



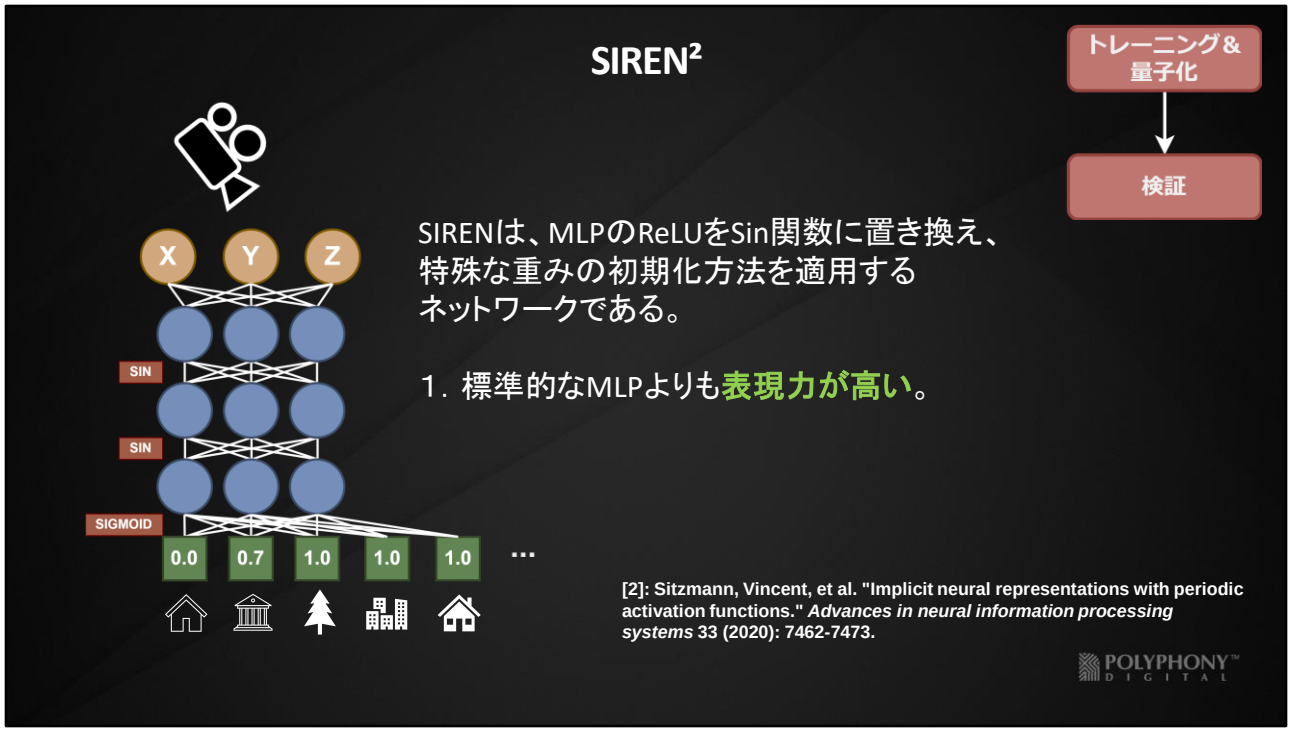
まず、標準的なMLP（Vanilla MLP）について話します。

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で、一般的なタスクで幅広く使われている手法です。
特徴は3つあります。
まず、**1つ目**として、1エポックあたりの学習が非常に速く、推論も高速に行える点です。

2つ目に、長時間トレーニングを行っても、リコール率と精度のバランスに限界があるという点です。

トレーニング収束(しゅうそく)しないとあるエリア内の多くのオブジェクトが同時に消(け)されてしまう可能性が高いです。

おすすめ度は星2つです。



次に、SIRENについて話します。

SIRENは、MLPのReLUをSin関数に置き換え、さらに特殊な重みの初期化方法を適用したネットワークです。

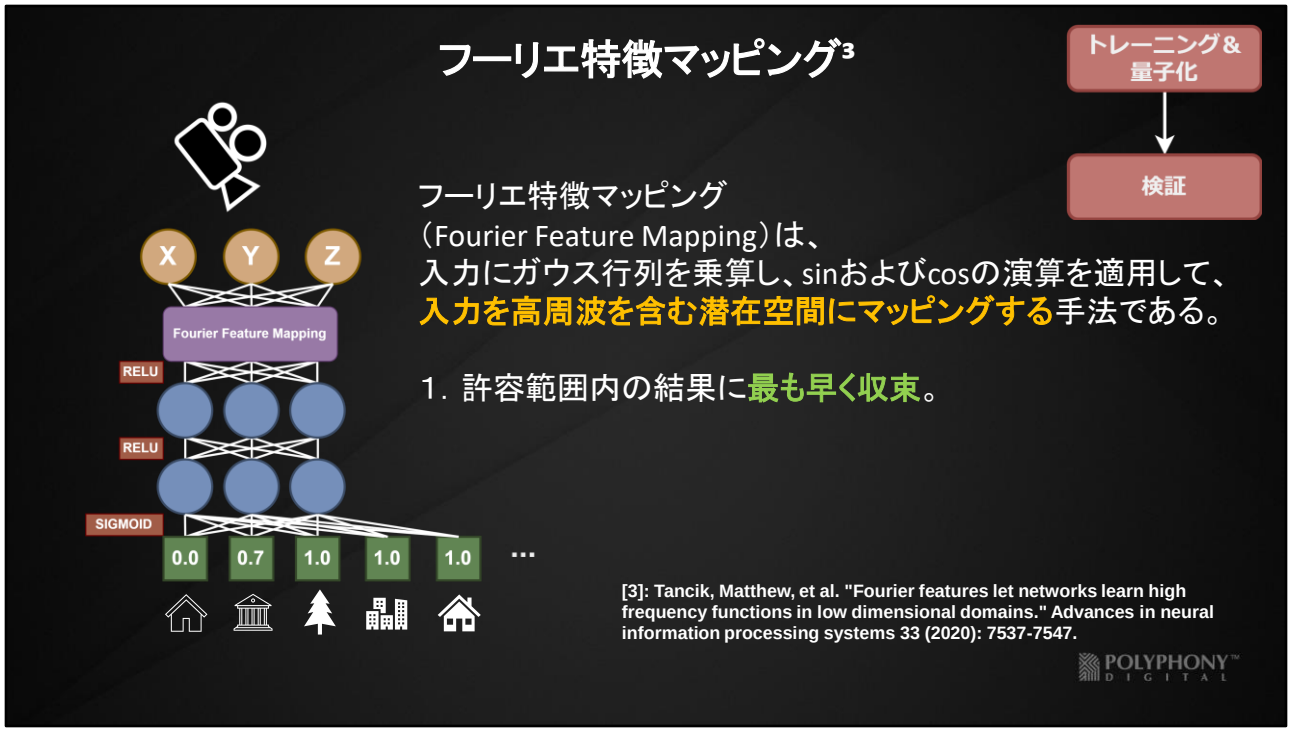
SIRENの特徴は次の3点です。
まず、1つ目に、標準的なMLPよりも表現力が高いという点があります。

2つ目として、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になることです。

そして、3つ目に、収束が難しく、同等の品質を求めると学習速度が十分ではないです。

大規模なプロジェクトに適用するには、依然として力不足と考えています。

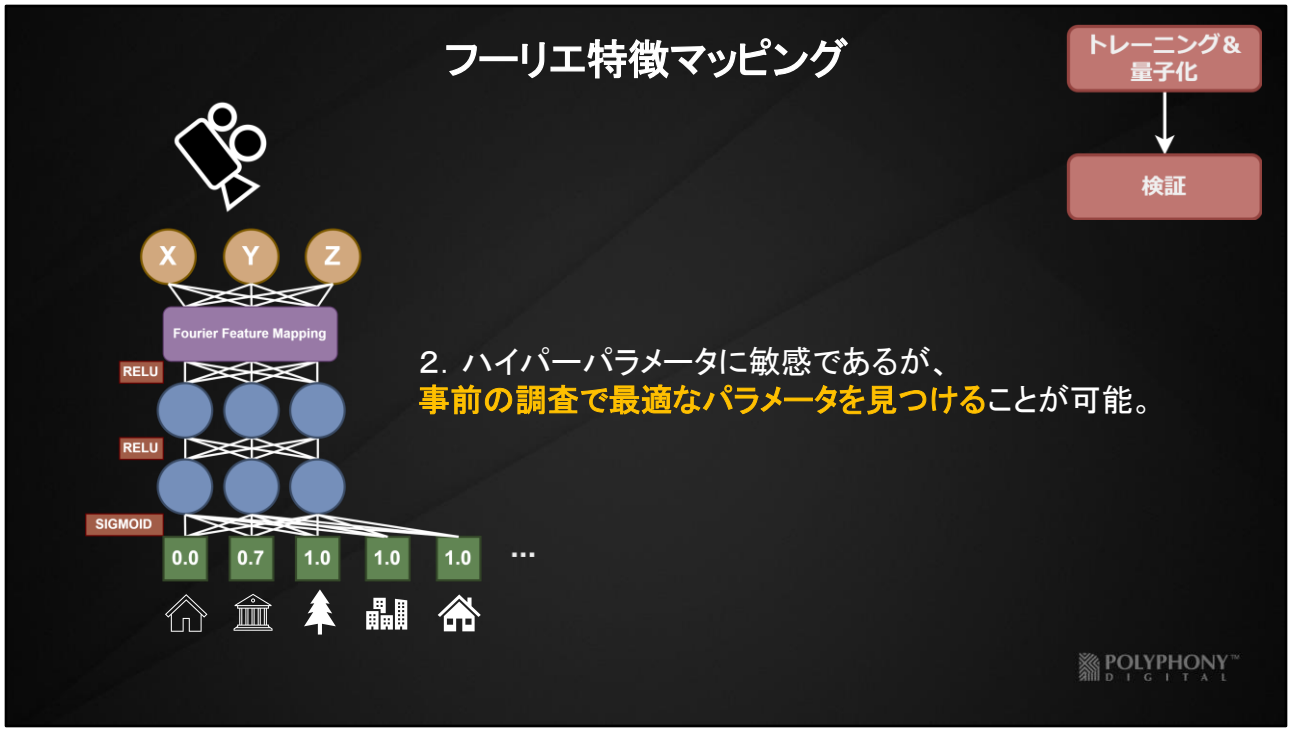
おすすめ度は星4つです。



最後に、**フーリエ特徴マッピング**について話します。

フーリエ特徴マッピング（Fourier Feature Mapping）は、入力に分散がハイパーパラメータである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（こうしゅうは）情報を含む潜在空間にマッピングする手法です。

フーリエ特徴マッピングの特徴は次の3点です。
まず、**1つ目**に、許容（きょよう）範囲内の結果に最も早く収束することができます。

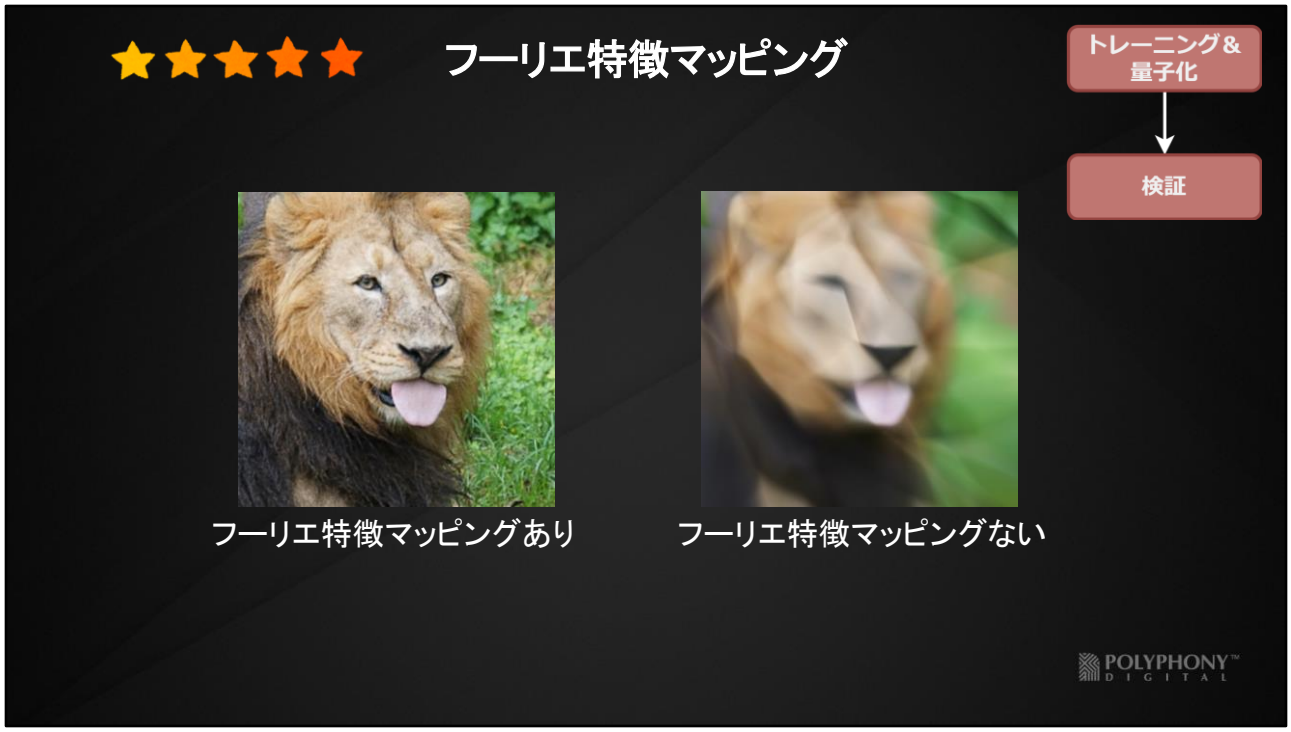


2つ目として、ハイパーパラメータに敏感ではありますが、事前に最適なパラメータを調整することで対応が可能です。

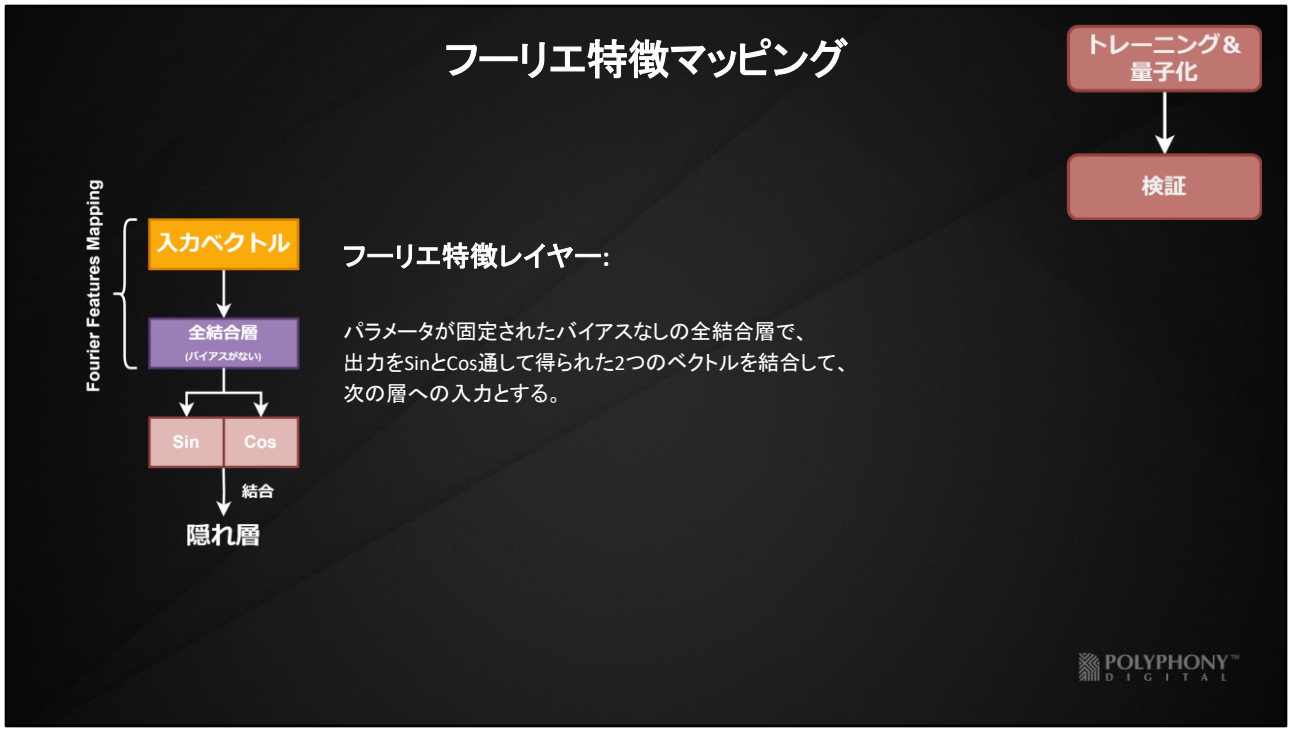
そして、**3つ目**に、同等（どうとう）のパラメータ量では、最高のリコール率と精度を持っています。

これらの理由から、最終的にこの手法を選択しました。

おすすめ度は星5つ（いつつ）です。



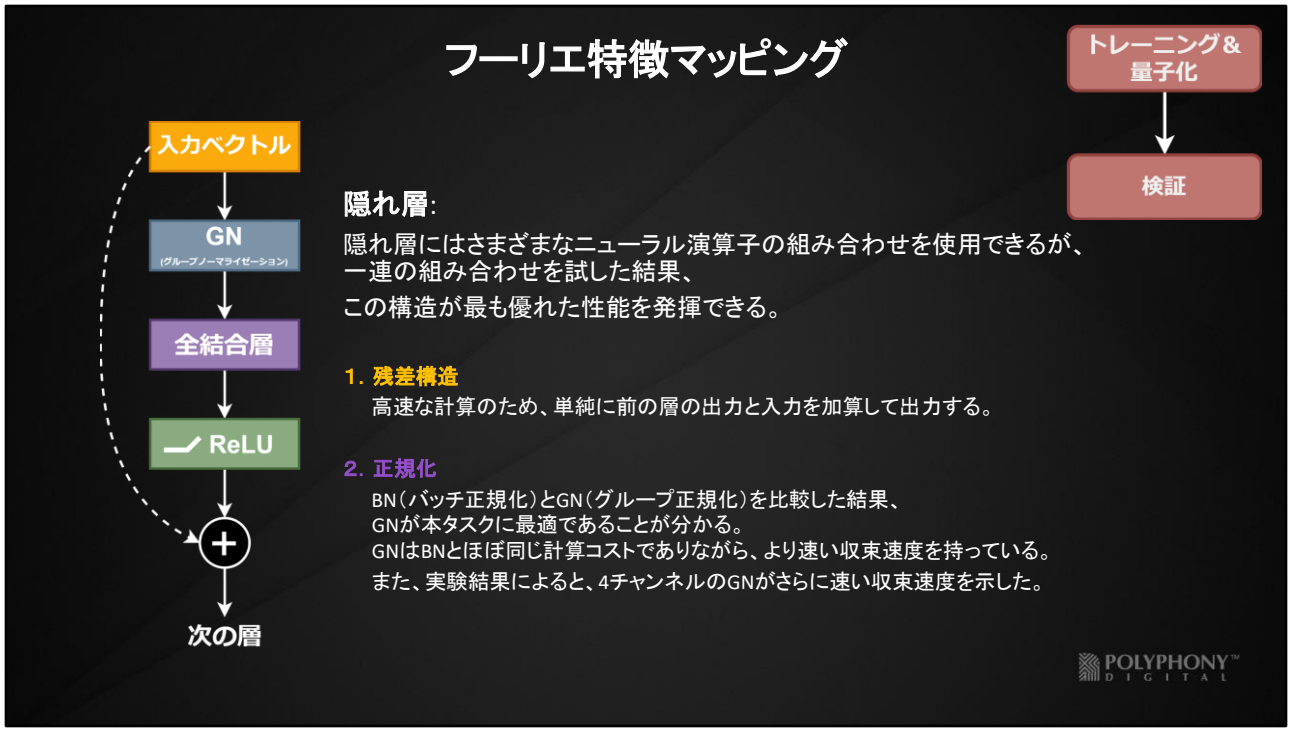
フーリエ特徴マッピングの論文にある画像復元タスクの結果は以下となります。
この図の通りに、左側がフーリエ特徴があるネットワークが学習した結果、右側がフーリエ特徴がないネットワークが学習した結果です。
画像復元タスクの結果からわかるように、フーリエ特徴を追加することで、ネットワークが高周波の特徴をより学習しやすくなります。



では、フーリエ特徴マッピングは具体的にどのように実装されているのでしょうか？

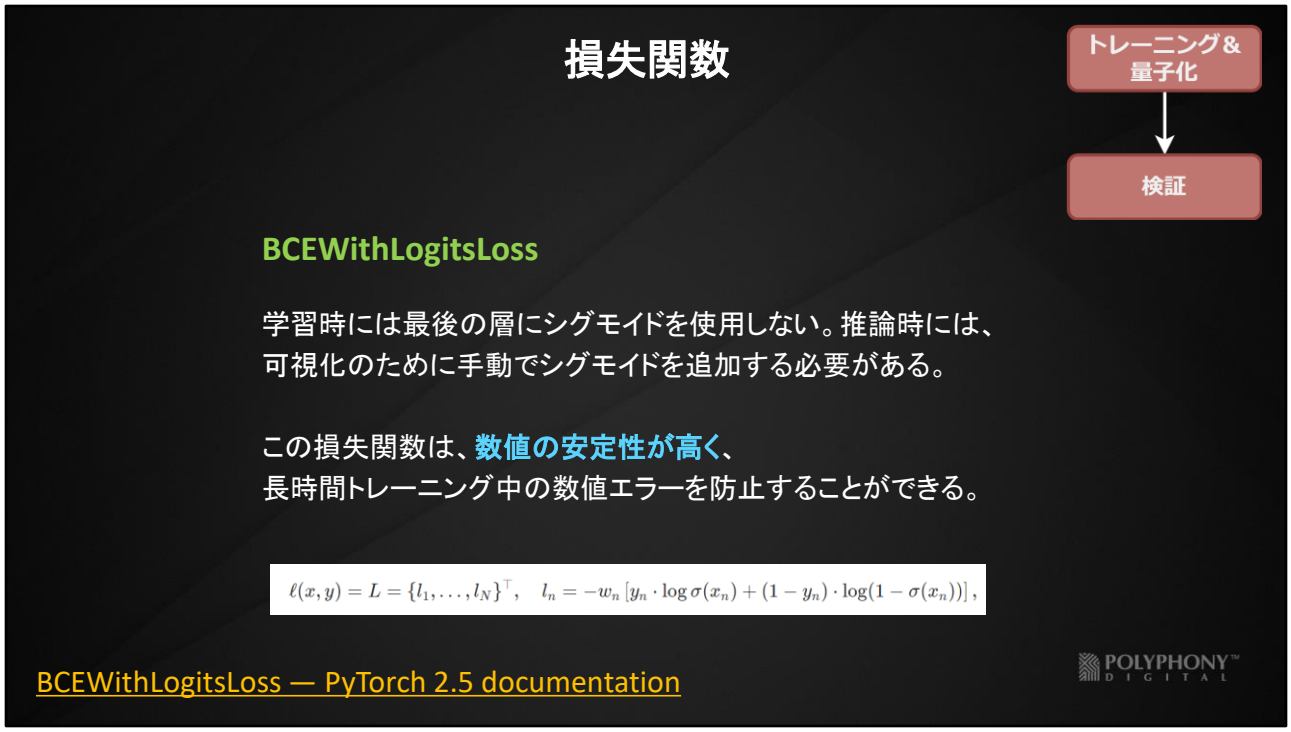
ネットワークの最初には、フーリエ特徴レイヤーがあります。

このレイヤーは、パラメータが固定され、バイアスがない全結合層です。
出力はSinとCosを通して2つのベクトルに変換され、それらを結合して次の層への入力とします。



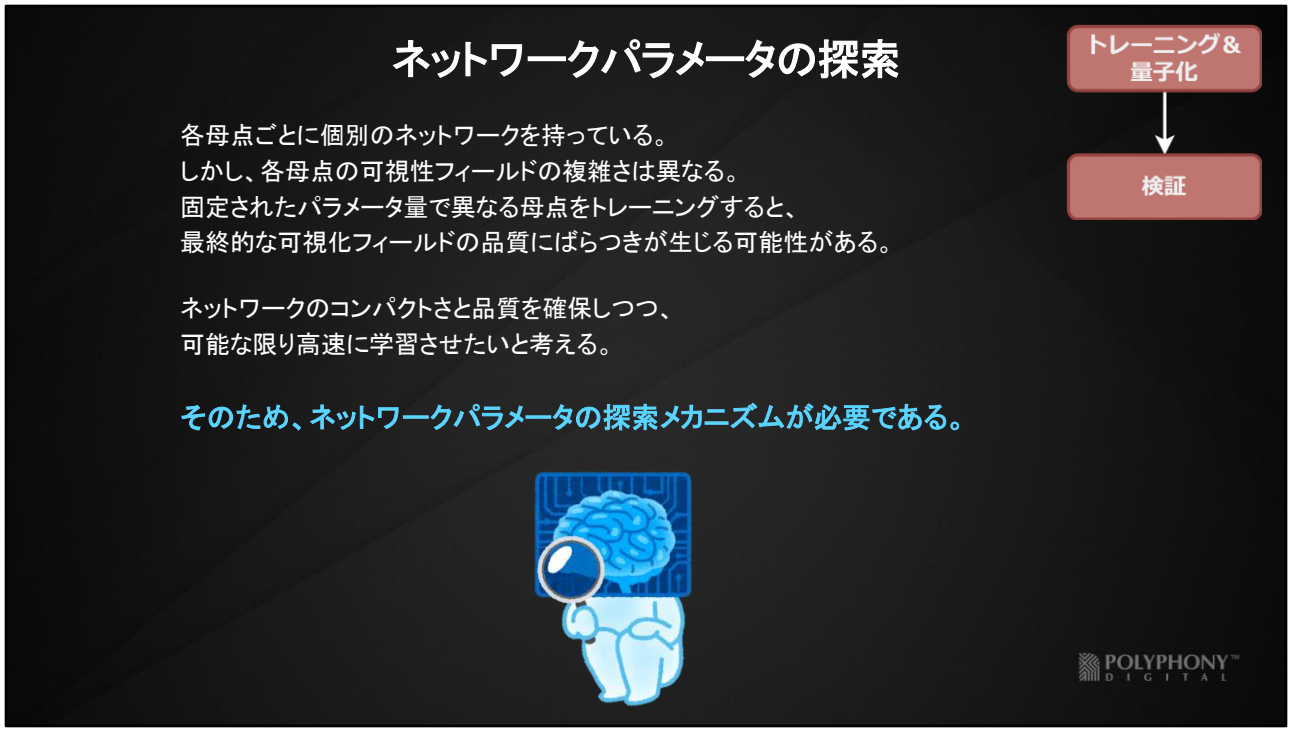
次に、ネットワークの隠れ層について説明します。

隠れ層(かくれそう) には、さまざまなニューラル演算子(えんざんし)の組み合わせが使用できますが、複数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ことがわかりました。
最適なストラクチャーは以下の2点です。
1つ目は、「残差構造(こうぞう)」です。残差構造は、層の出力と層の入力を単純に加算し、次の層の入力とします。
2つ目は「正規化」です。BN(バッチ正規化)とGN(グループ正規化)を比較した結果、GNがこのタスクに最適であることが分かりました。
GNはBNとほぼ同じ計算コストでありながら、より速い収束することができます。
また、実験結果から、4チャンネルのGNはさらに速い収束を示しました。



次に、損失関数について説明いたします。このプロジェクトは、BCEWithLogits(ロジット)Lossを使用しています。

この損失関数では、学習時には最後の層にシグモイドを使用しませんので、数値の安定性も高く、長時間レーニング中の数値エラーを防止する効果があります。

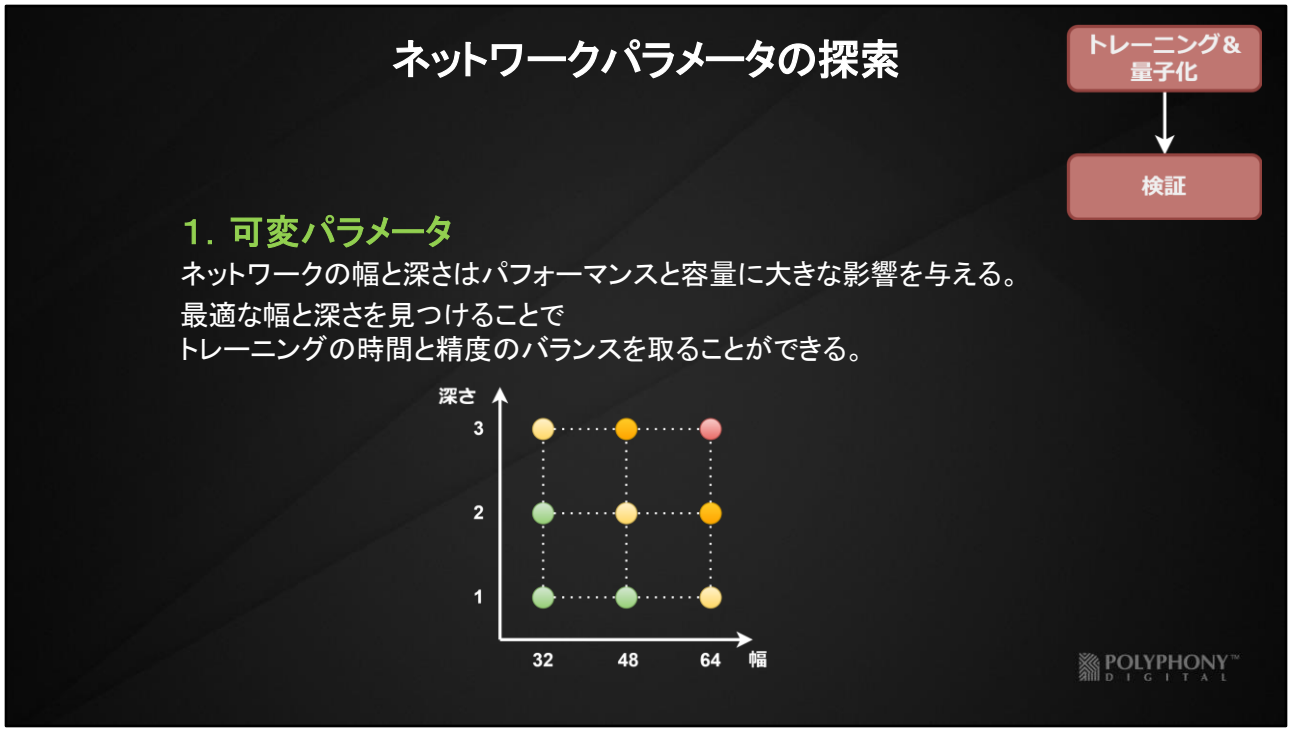


各母点ごとに個別のネットワークを持っていますが、各母点の可視性フィールドの複雑さは異なります。

そのため、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ります。

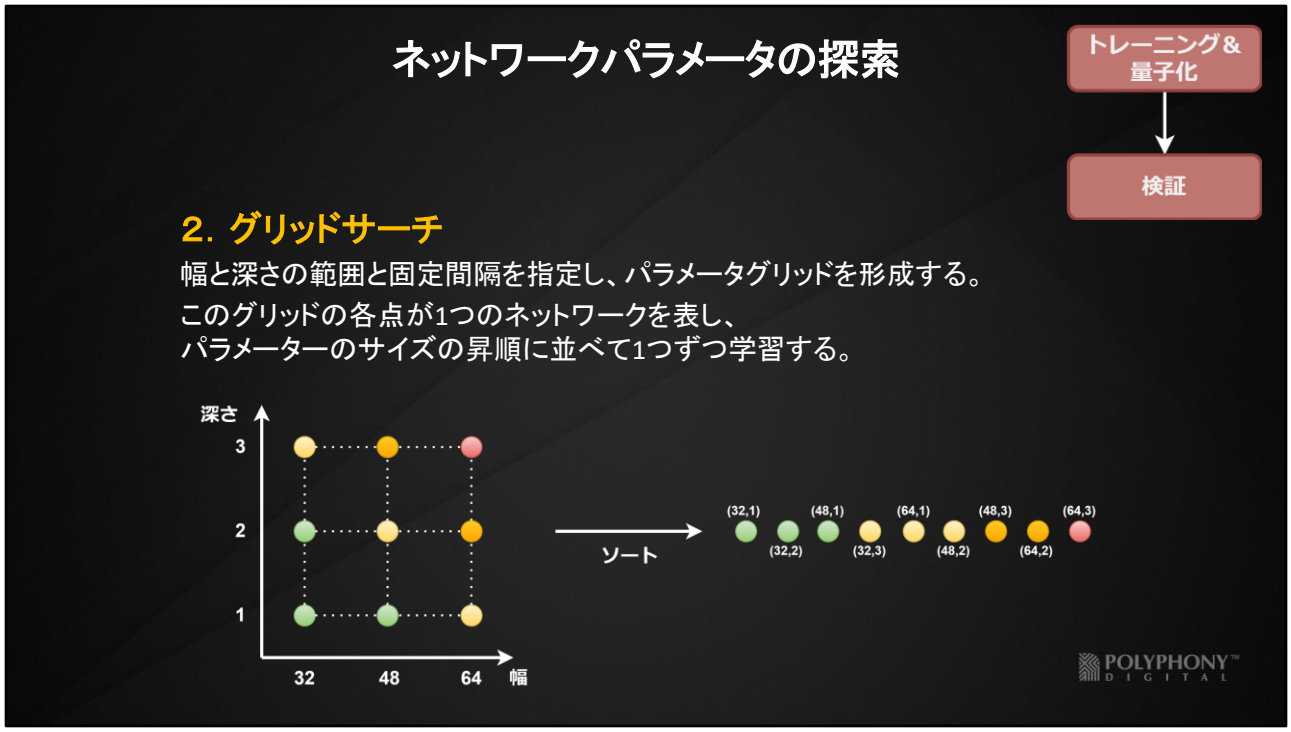
ネットワークの コンパクトさ と 品質 を確保しながら、可能な限り高速に学習させたいと考えています。

そのためには、ネットワークパラメータの探索メカニズムが必要となります。

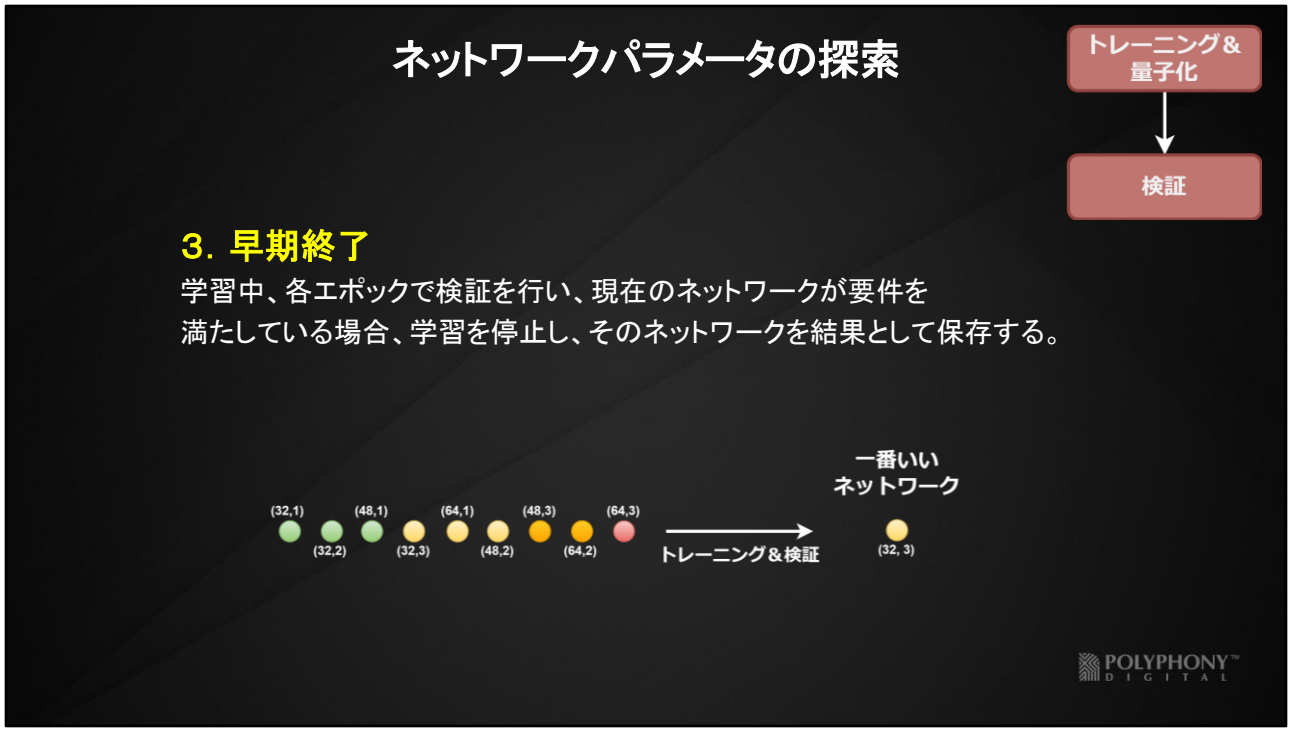


異なる母点に対して調整するパラメータは主にネットワークの深さと幅です。この2つのパラメータがネットワークのパラメータ量と最終的なフィッティングの品質を決定します。

具体的なネットワークパラメータの探索方法について説明します。
まず、**1つ目**は「可変パラメータ」です。ネットワークの幅（はば）と深さ(ふかさ)はパフォーマンスと容量(ようりょう)に大きく影響します。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可能になります。

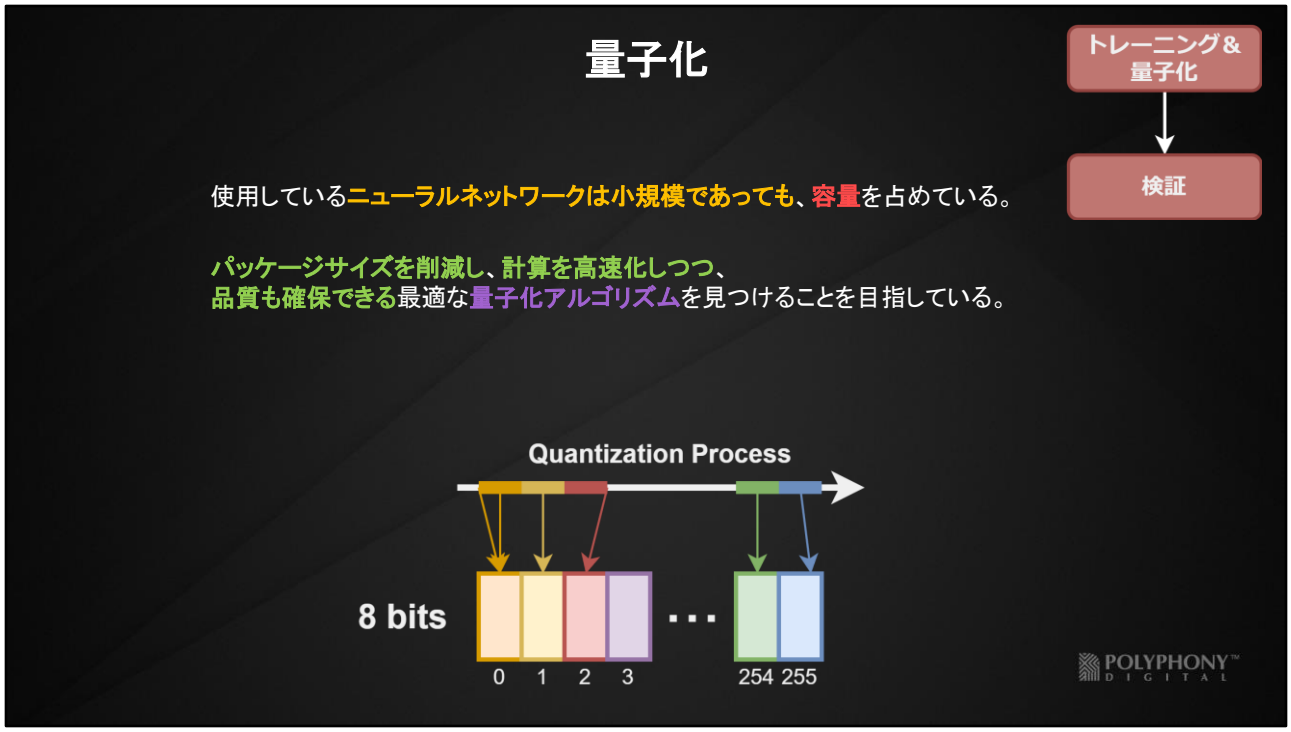


次に、2つ目は「グリッドサーチ」です。幅と深さの範囲と固定間隔を指定し、パラメータグリッドを形成します。このグリッドの各点が1つのネットワークを表しており、パラメーターのサイズ昇順（しょうじゅん）に並べて1つずつ学習を進めていきます。



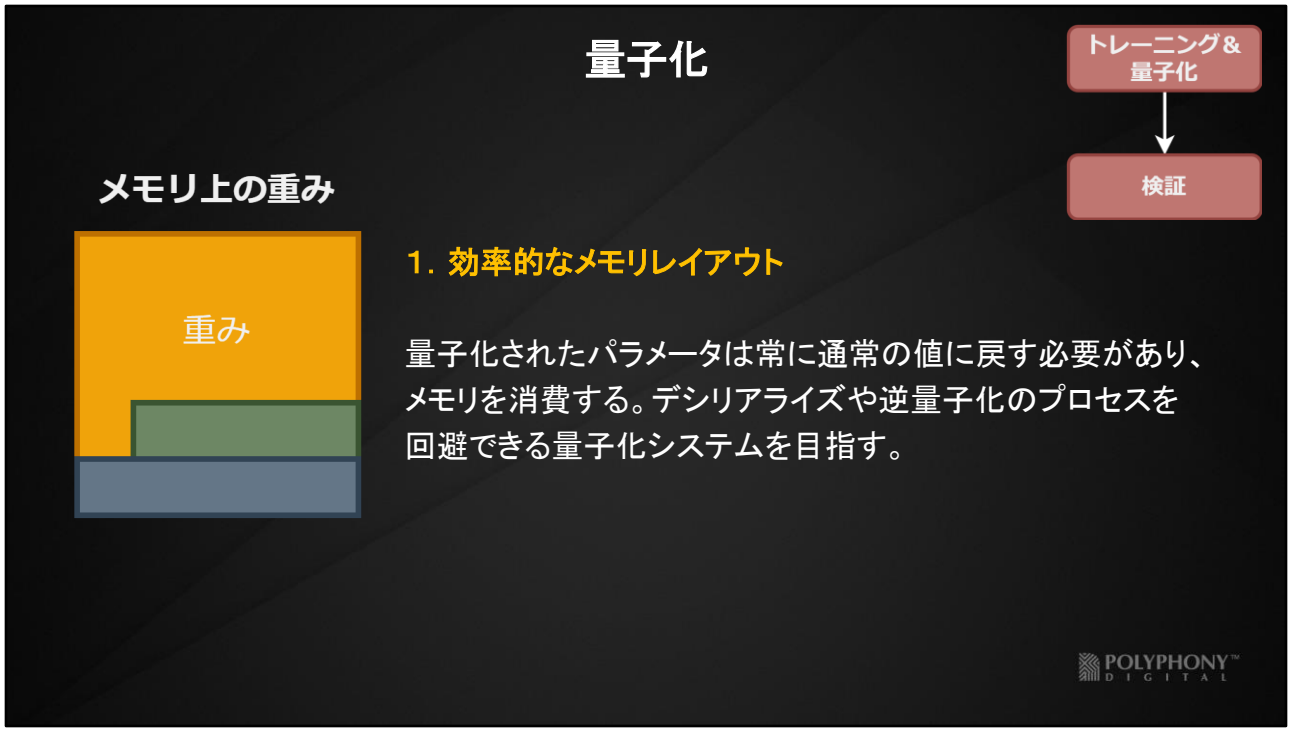
そして、**3つ目**が「早期終了（そうきしゅうりょう）」です。
学習中に各エポックで検証を行い、現在のネットワークが要件を満たしている場合は、学習を停止し、そのネットワークを結果として保存します。

量子化

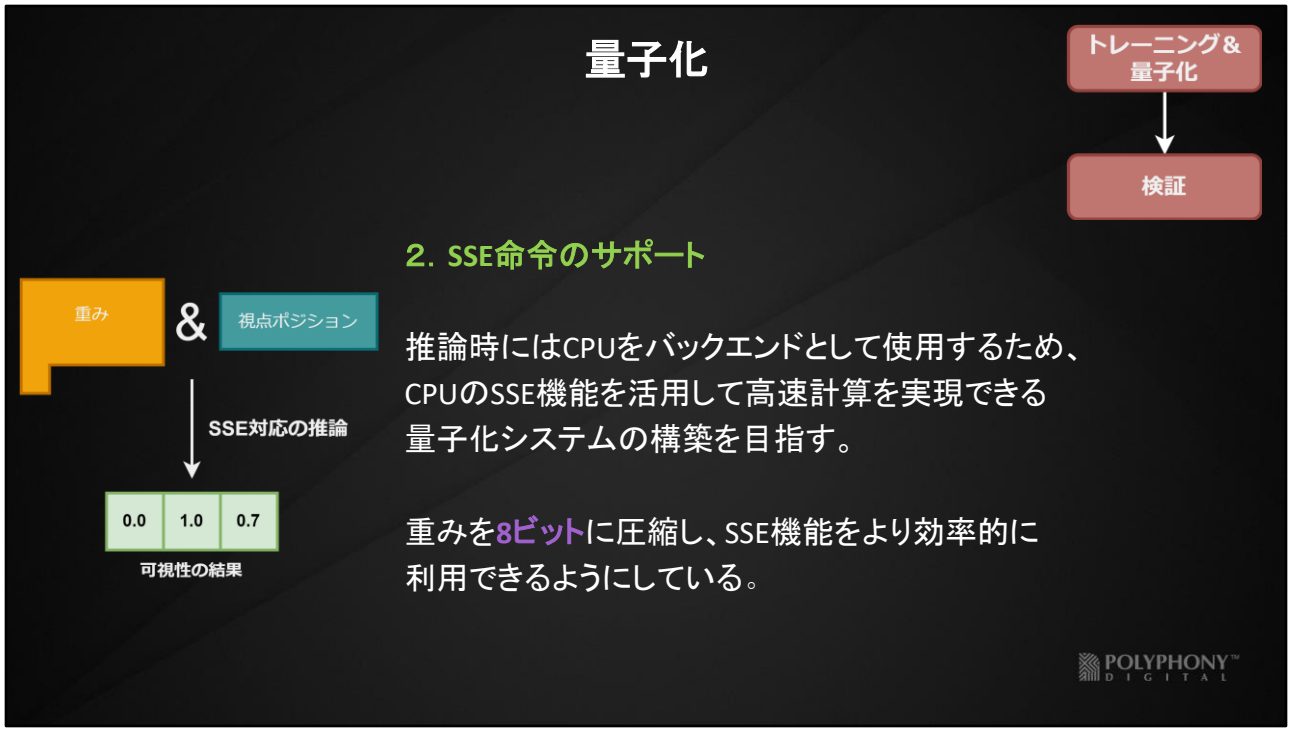


ネットワークパラメータの探索を通じて、品質が保証され、かつコンパクトなネットワークを得ることができます。
しかし、使用しているニューラルネットワークは小規模であっても、容量を占めています。

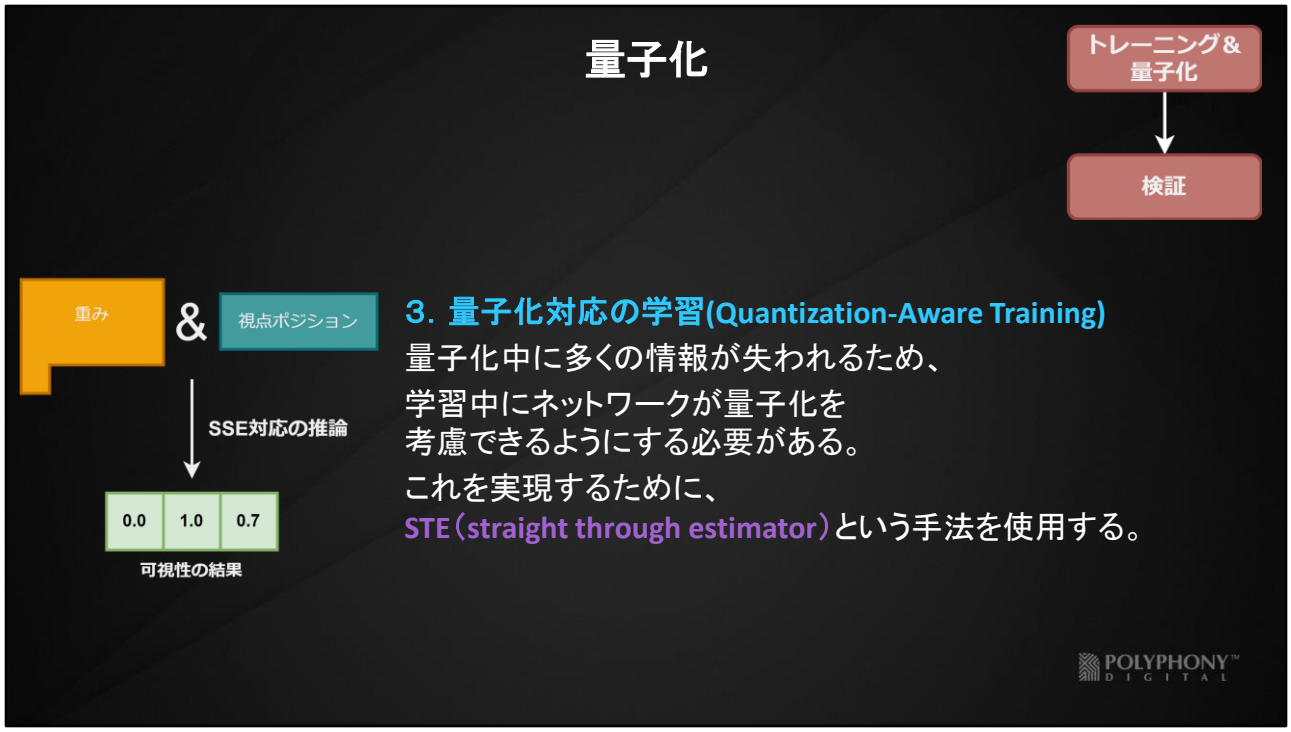
コンソールゲームではメモリ容量をできる限り削減する必要があります。
そのため、パッケージサイズを削減し、計算を高速化しながら品質も確保できる、最適な量子化アルゴリズムを見つける必要があります。



量子化アルゴリズムには、次の3つの条件を満たす必要があります。
1つ目は「効率的なメモリレイアウト」です。
量子化されたパラメータは通常の値(あたい)に戻す必要があって、解凍された結果はメモリを消費します。そのため、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しています。

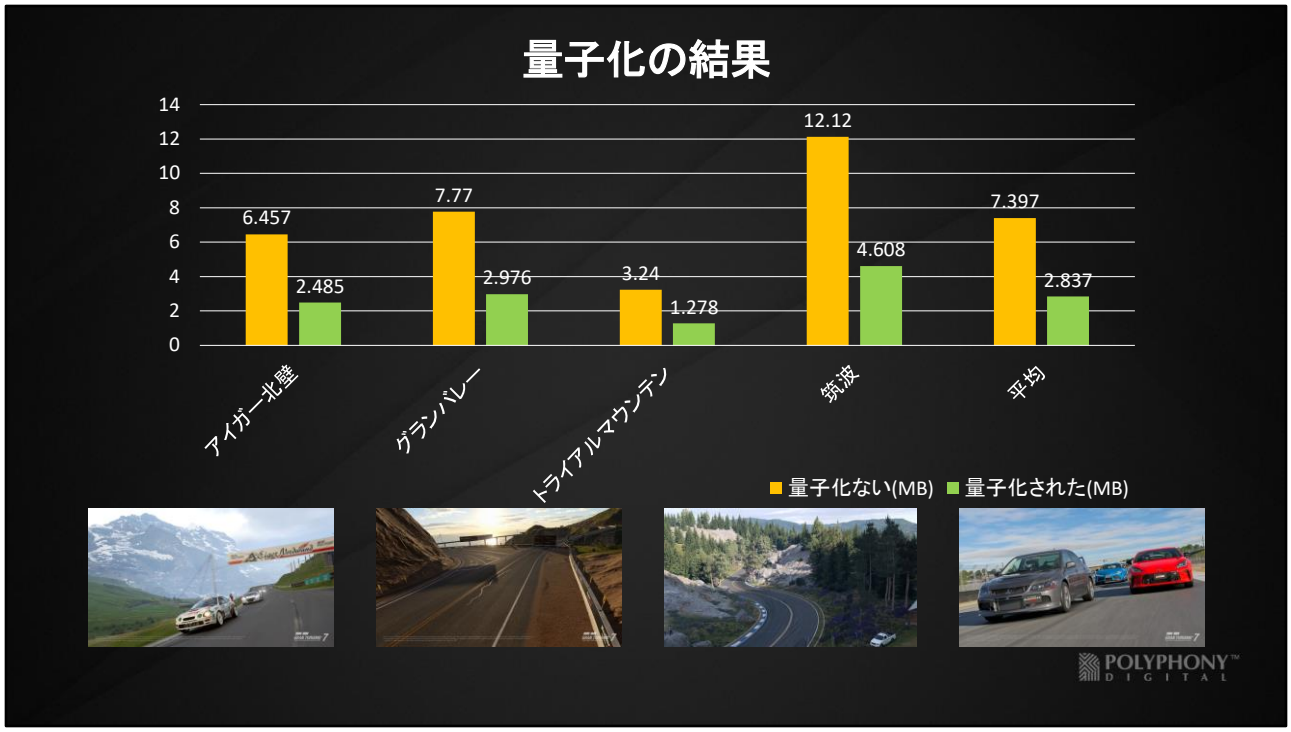


2つ目は「SSE命令のサポート」です。
推論時にはCPUをバックエンドとして使用するため、CPUのSSE機能を活用して高速な計算を実現できる量子化システムの構築を目指しています。
具体的には、重みを8ビットに圧縮し、SSE機能をより効率的に利用できるようにしています。

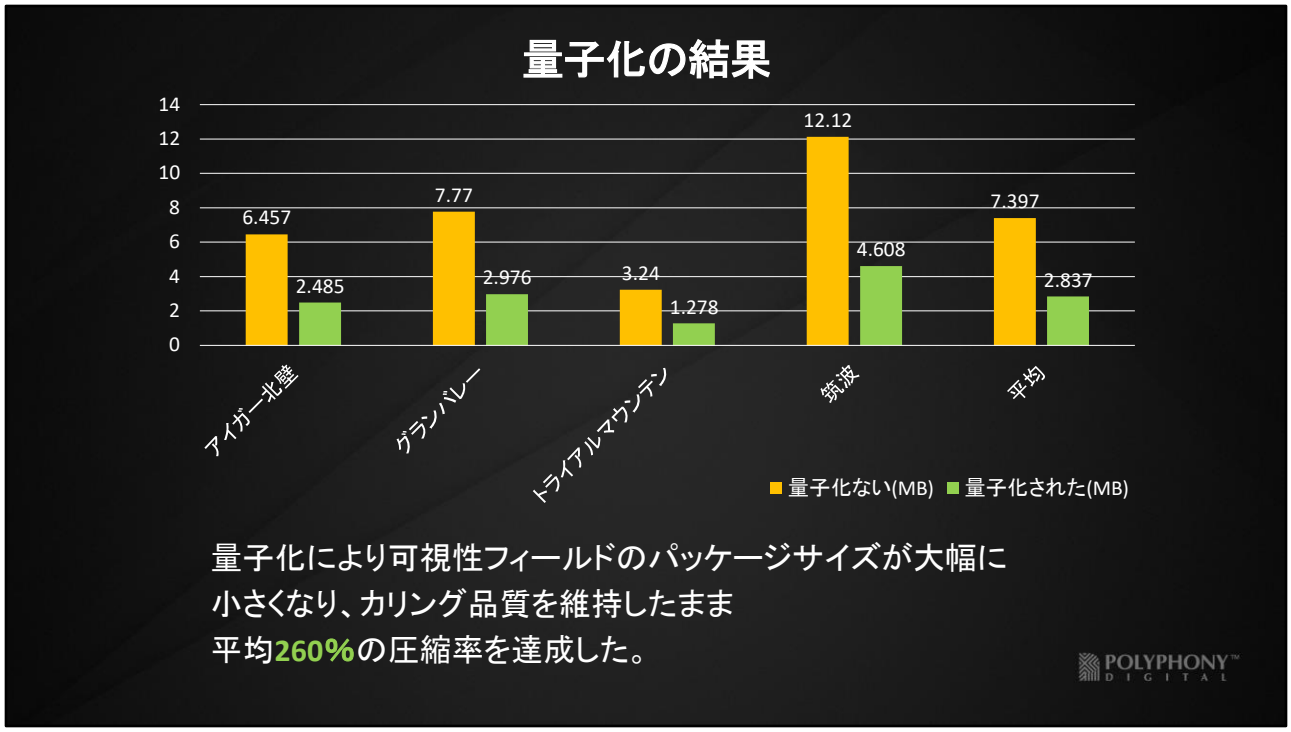


3つ目は「量子化対応の学習（Quantization-Aware Training）」です。
量子化中に多くの情報が失われるため、学習中にネットワークが量子化を考慮(こうりょ)できるようにする必要があります。これを実現するために、STE（straight through estimator）という手法を使用しています。

具体的な量子化の実装は、後ろのセクションに紹介します。



量子化の結果は以下となります。
私たちはほとんどのコースでテストを行い、有名なコースをいくつか例として取り上げました。
黄色の左側は量子化されていない場合のボリューム、緑の右側は量子化された場合のボリュームとなります。



その結果、量子化によってパッケージサイズが大幅（おおはば）に小さくなり、カリングの品質を維持（いじ）しつつ、平均260%の圧縮率を達成しました。

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。

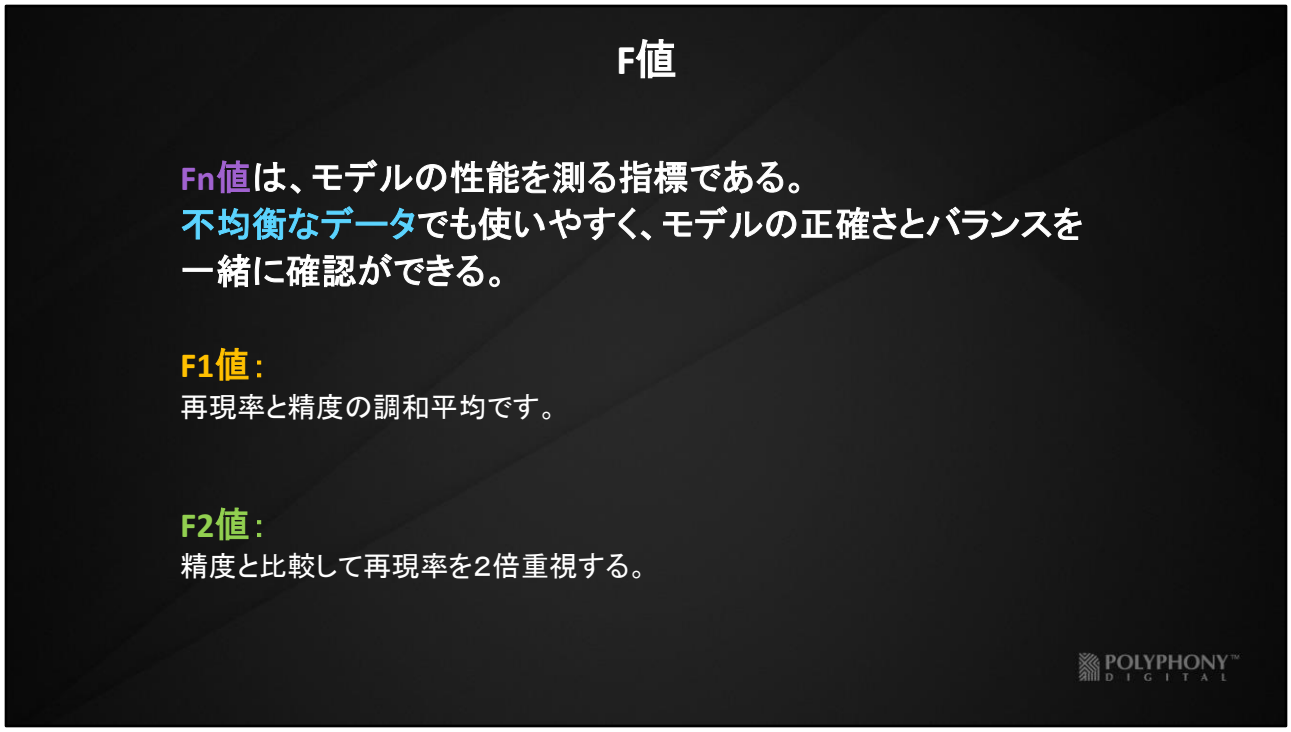
精度(適合率):
見えると判定されたオブジェクトの中に、本当に見えるものの割合である。
精度が高いほど、カリングの効果が良い。

POLYPHONY DIGITAL

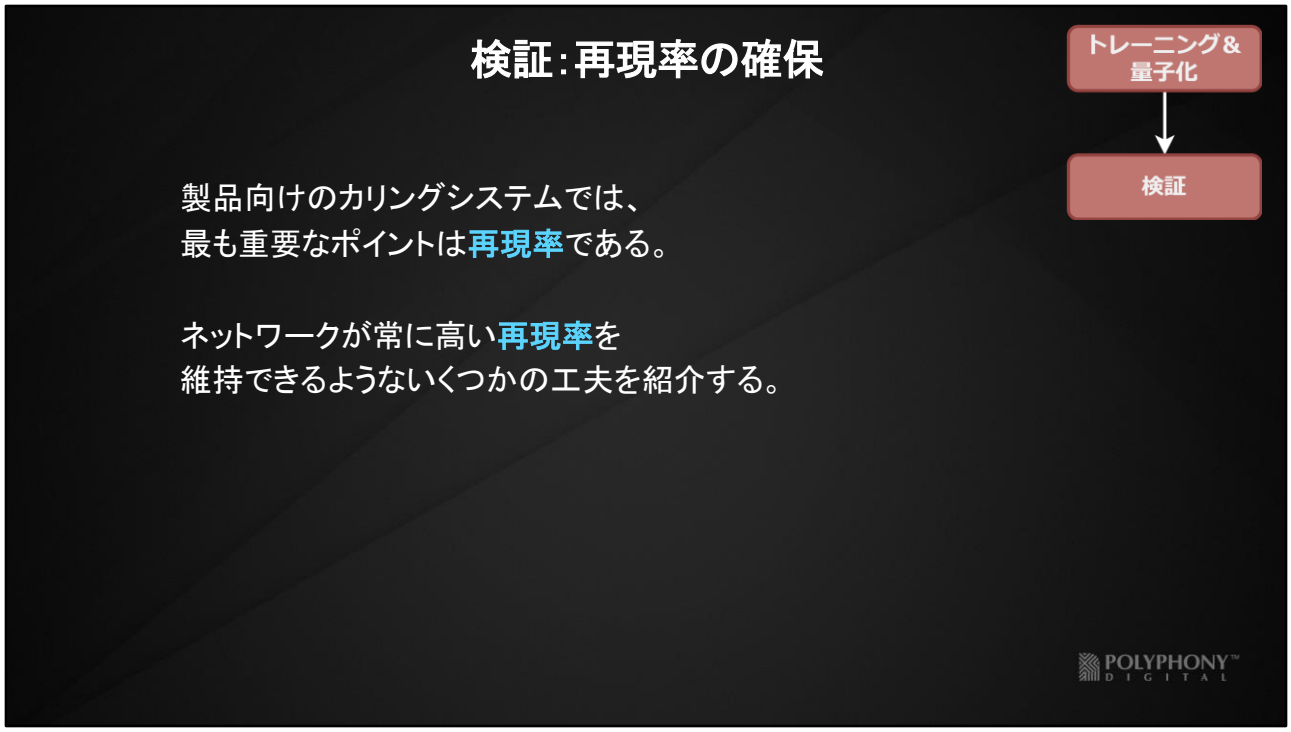
このタスクにおいて、ネットワークの品質を評価する上で最も重要な2つの指標があります。
1つ目は**再現率**です。
これは、見えるオブジェクトが確実に「見える」と判断される割合を指します。
再現率が低くなると、本来見えるはずのオブジェクトが誤ってカリングされてしまう可能性が高まります。

2つ目は精度です。
精度は「見える」と判定されたオブジェクトの中で、実際に見えるものの割合を示します。
精度が高いほど、カリングの効果が良好であることを意味します。

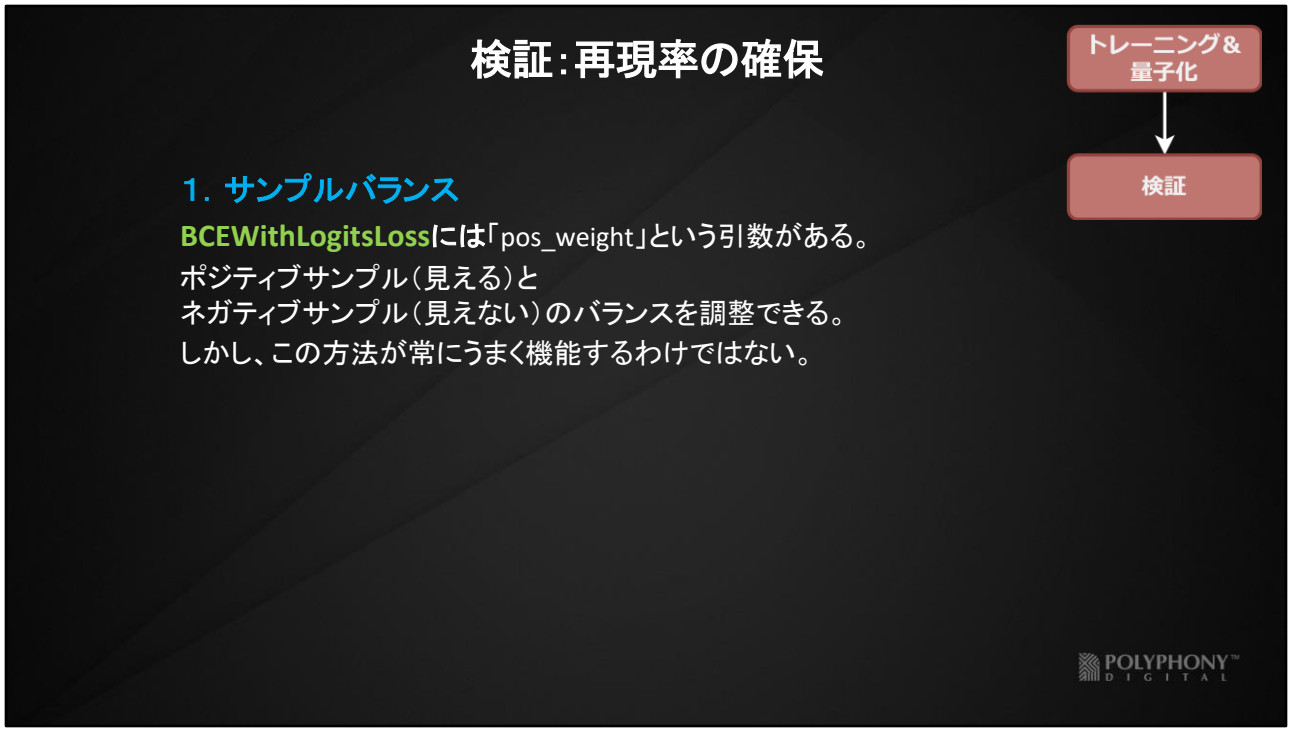
毎回のトレーニングでこれら2つの指標が確保できる手法が必要です。



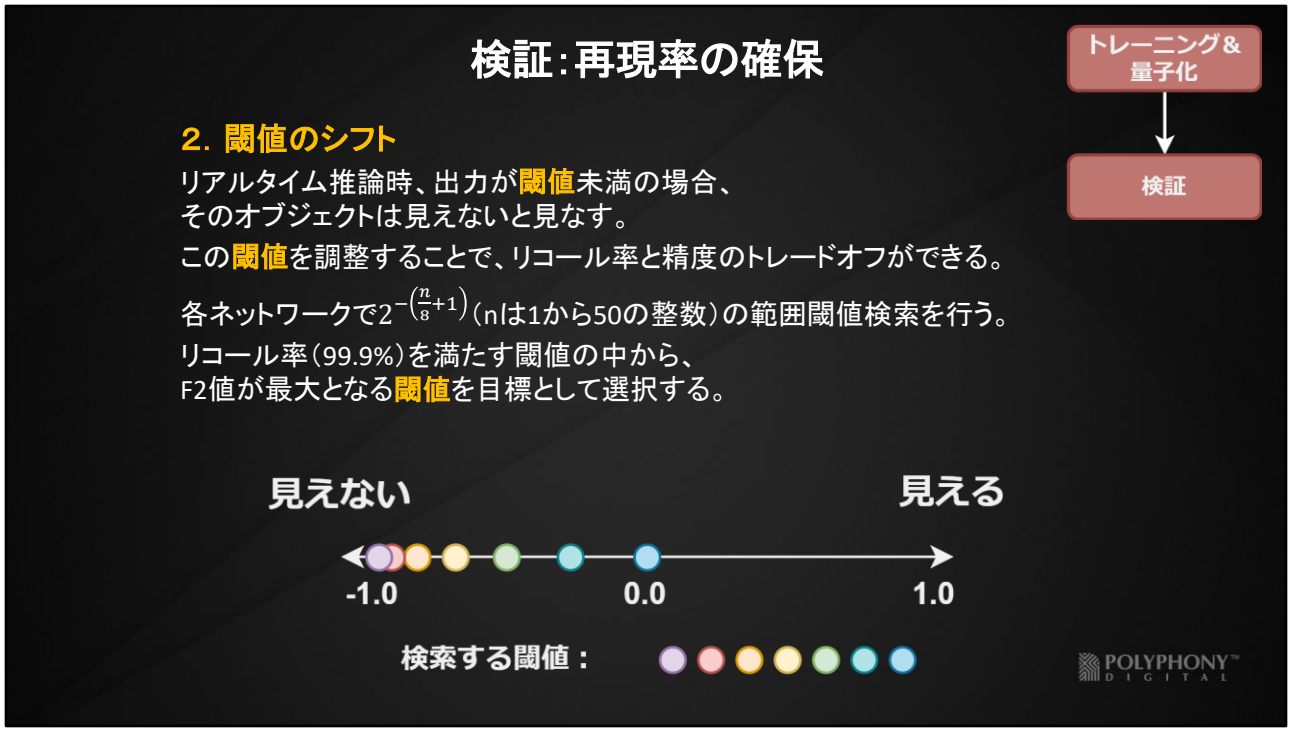
Fn値（Fnスコア）は、モデルの性能を測る指標です。不均衡なデータでも使いやすく、モデルの正しさとバランスを一緒に確認することができます。
そのなかに、F1値(エフワンち)は、再現率と精度の調和平均です。
そして、F2(エフつー)値は、加重平均（かじゅうへいきん）されて、精度と比較して再現率を2倍重視する指標です。



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す。
ここでは、ネットワークが常に高い**再現**率を維持できるような工夫について紹介いたします。



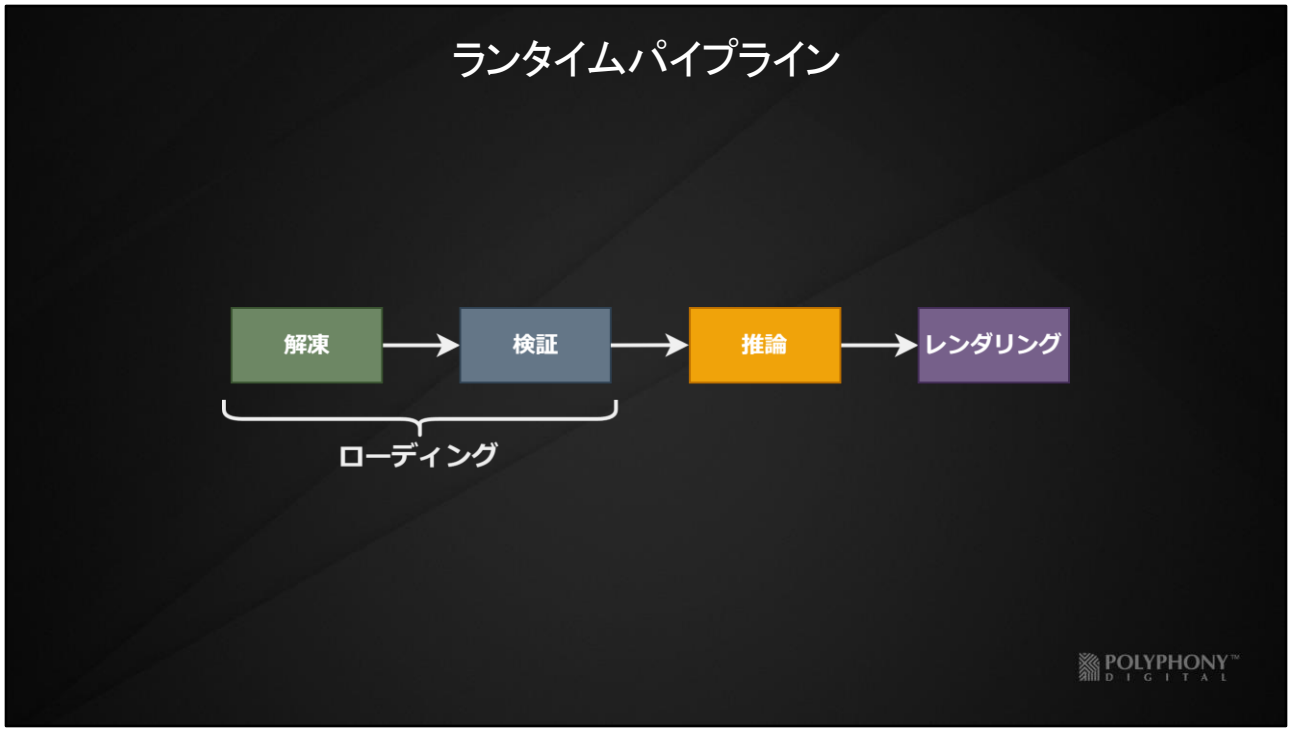
まず、**1つ目**は「サンプルバランス」です。
私たちはBCEWithLogits（ロジット）Lossを使用していますので、「pos_weight」という引数を使ってpositiveポジティブサンプル（見える）とnegativeネガティブサンプル（見えない）のバランスを調整することができます。
ただし、この方法が常にうまく機能するわけではありません。



2つ目は「閾値のシフト」です。
リアルタイム推論時、出力が閾値未満である場合、そのオブジェクトは「見えない」と見なします。この閾値を調整することで、再現率と精度のトレードオフが可能です。
各ネットワークに対して、
この式のように、ある範囲で閾値検索を行います。再現率が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します。

ランタイムパイプライン

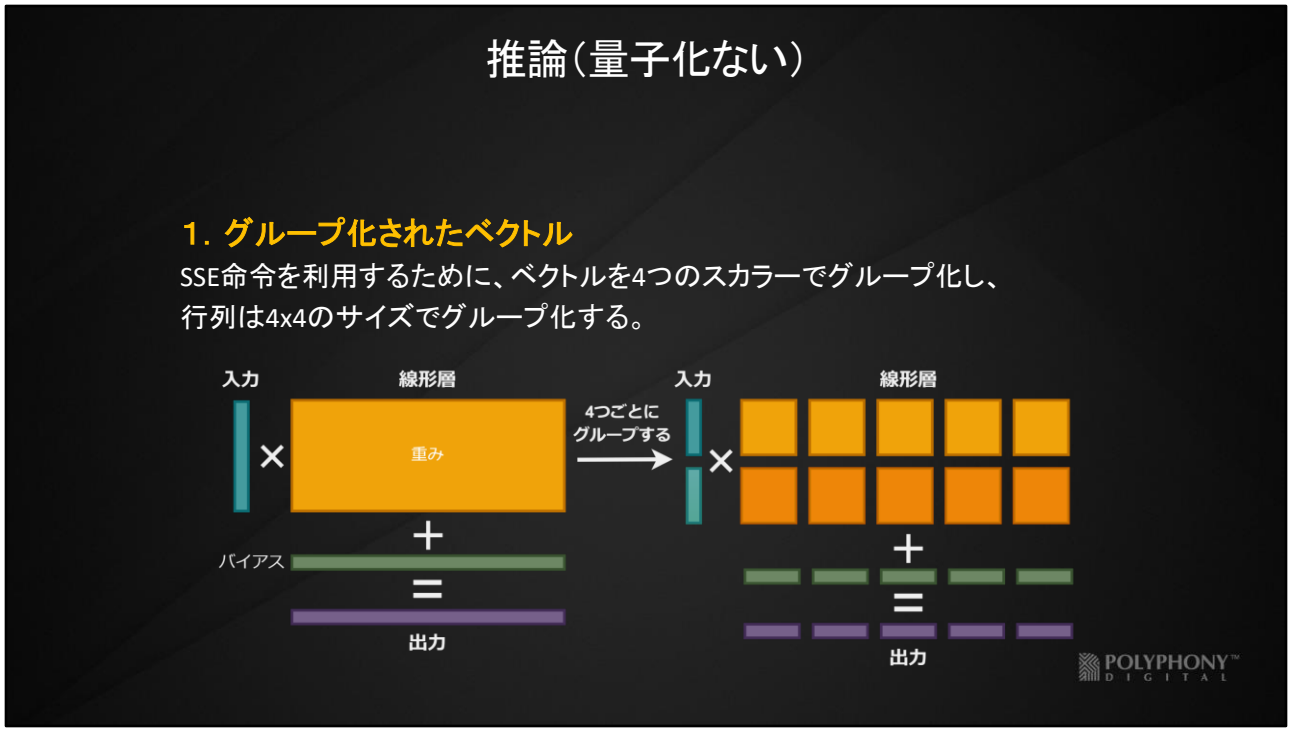
POLYPHONY™ DIGITAL



次に、ランタイムのパイプラインについて説明いたします。

ランタイムは4つの部分に分かれており、ここでは、推論の方法について具体的にご紹介します。

推論



次に、最も重要な部分である**リアルタイムの推論**について説明いたします。

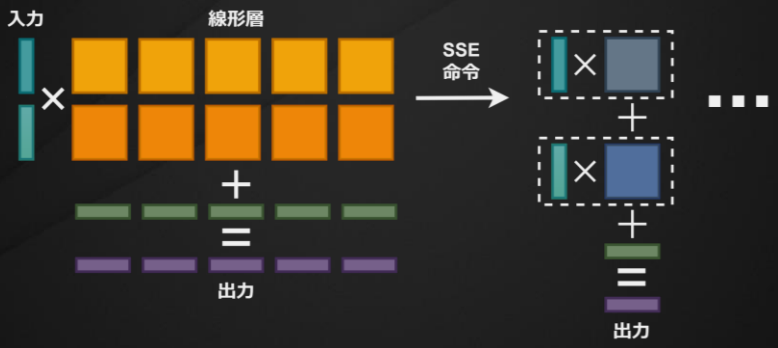
量子化がない時のリアルタイム推論には、主に以下の3つのポイントがあります。
1つ目は「グループ化されたベクトル」です。
SSE命令を活用するために、ベクトルは4つのスカラーでグループして、マトリックスは4x4のサイズでグループ化しています。

推論(量子化ない)

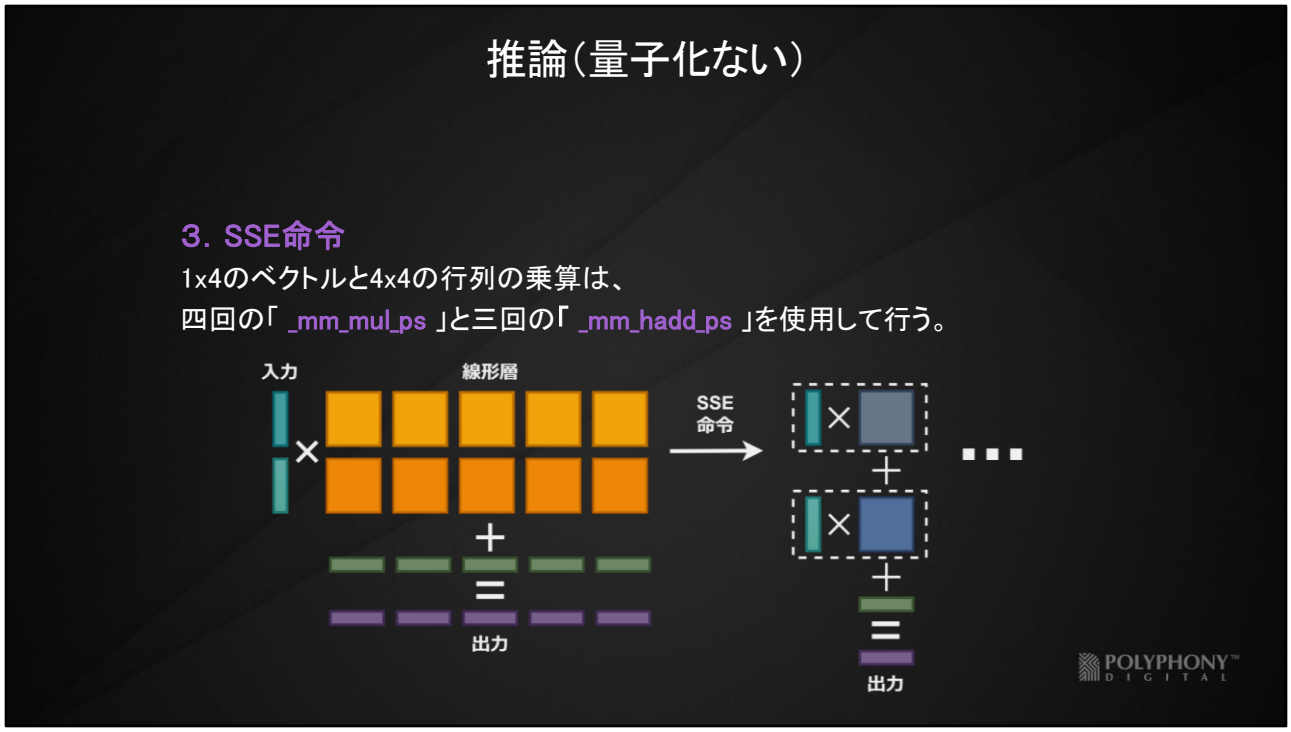
2. 数値計算バックエンド

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。



グループされた後、
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ます。
変換をされた形式は一般的なCGで使用される行列乗算と共通するものになります。
『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ます。



最後に、**3つ目は「SSE命令」です**。
1x4のベクトルと4x4のマトリックスの乗算は、
「_mm_mul_ps」（ベクトルの掛け算（かけざん））を4回、「_mm_hadd_ps」（水平和）を3回使用して行います。

水平和のことを、英語では horizon sum と呼ぶこともあります。

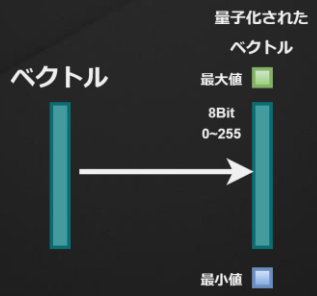
推論(量子化あり)

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。

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



POLYPHONY DIGITAL

量子化後の推論流れは大きく変化しています。

前のトレーニングの章でも述べたように、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ります。
量子化の手順は以下のとおりです。
まず、1つ目は「ベクトル」です。
元のベクトルの範囲を0～1に正規化し、最小値と最大値を32ビットのfloatとして保存します。
その後、0～1の範囲を整数の0～255にマッピングし、8ビットに量子化します。

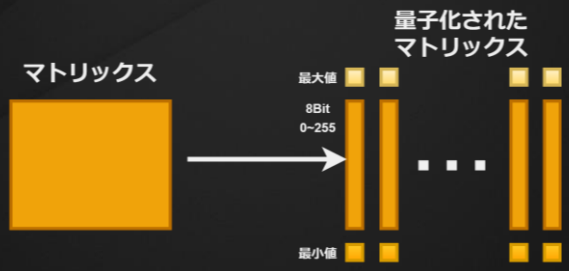
推論(量子化あり)

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。

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

$$m_i = (\hat{m}_i / 255) \times m_{irange} + m_{imin}$$



POLYPHONY DIGITAL

次に、2つ目は「行列」です。
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ビットに量子化します。

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

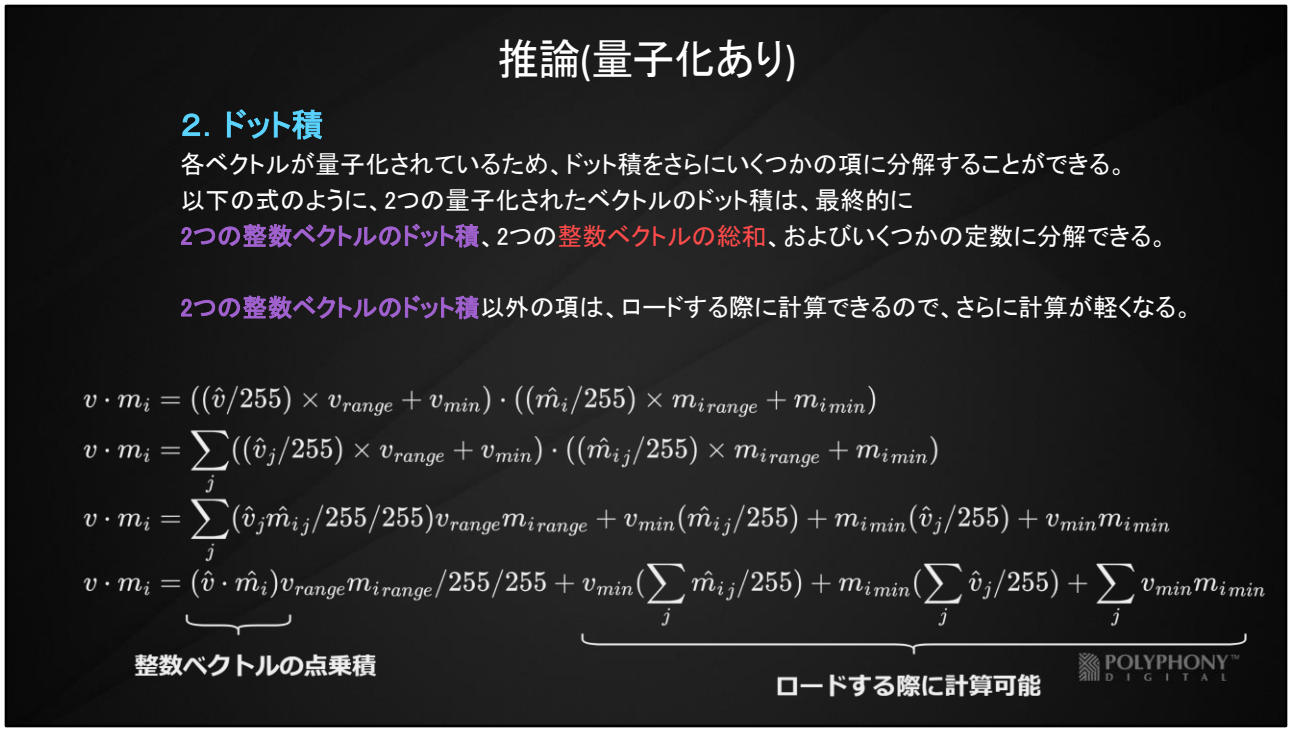
ベクトルと行列の乗算を、いくつかの**ベクトル間のドット積**として捉えることができる。

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

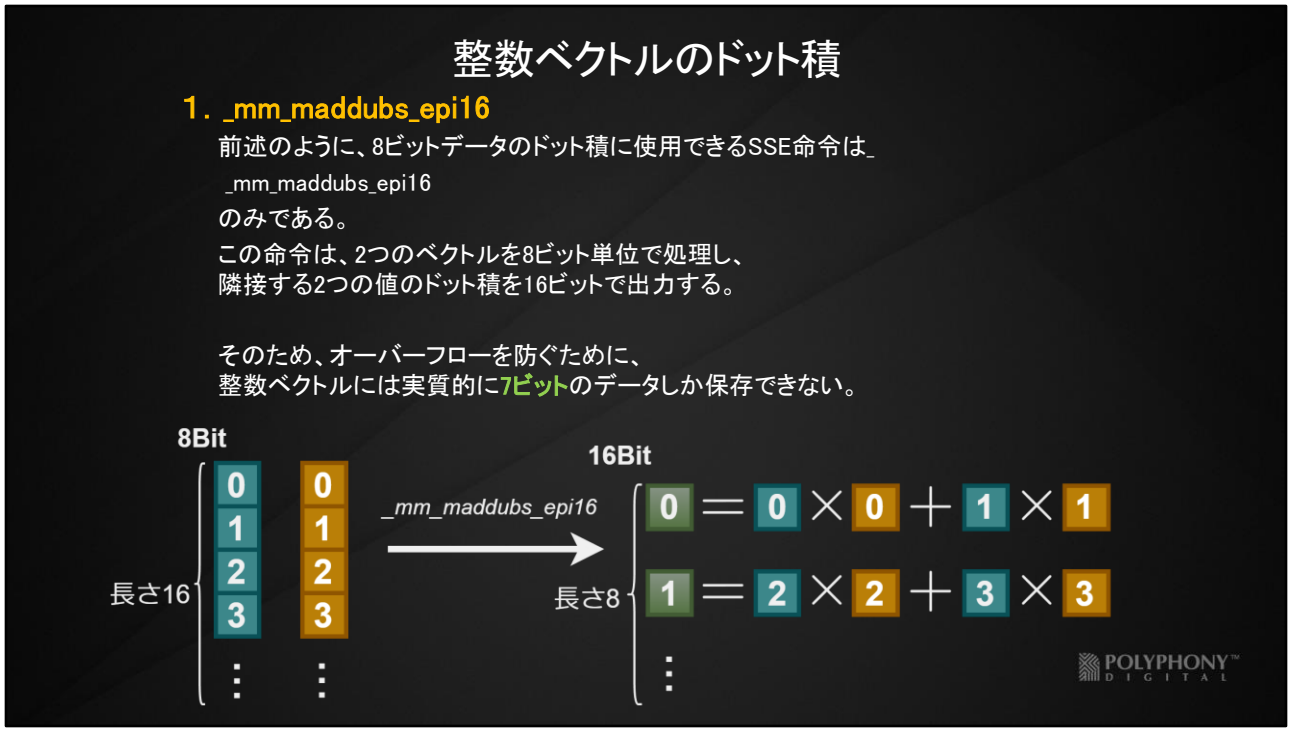
POLYPHONY DIGITAL

追加のコピーや逆量子化を回避するために、
SSE命令を利用して、直接に量子化されたベクトルとマトリックスの乗算処理、
具体的には、次の2つのステップがあります。

まず、**1つ目は「乗算」です。**
ベクトルとマトリックスの乗算を、いくつかのベクトル間のドット積として捉えることが可能です。



次に、2つ目は「ドット積」です。
各ベクトルが量子化されているため、ドット積をさらに複数の項に分解することができます。
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(そうわ)、そしていくつかの定数に分解可能です。
2つの整数ベクトルのドット積以外の項は、ロード時に計算できるため、計算がさらに軽量化されます。



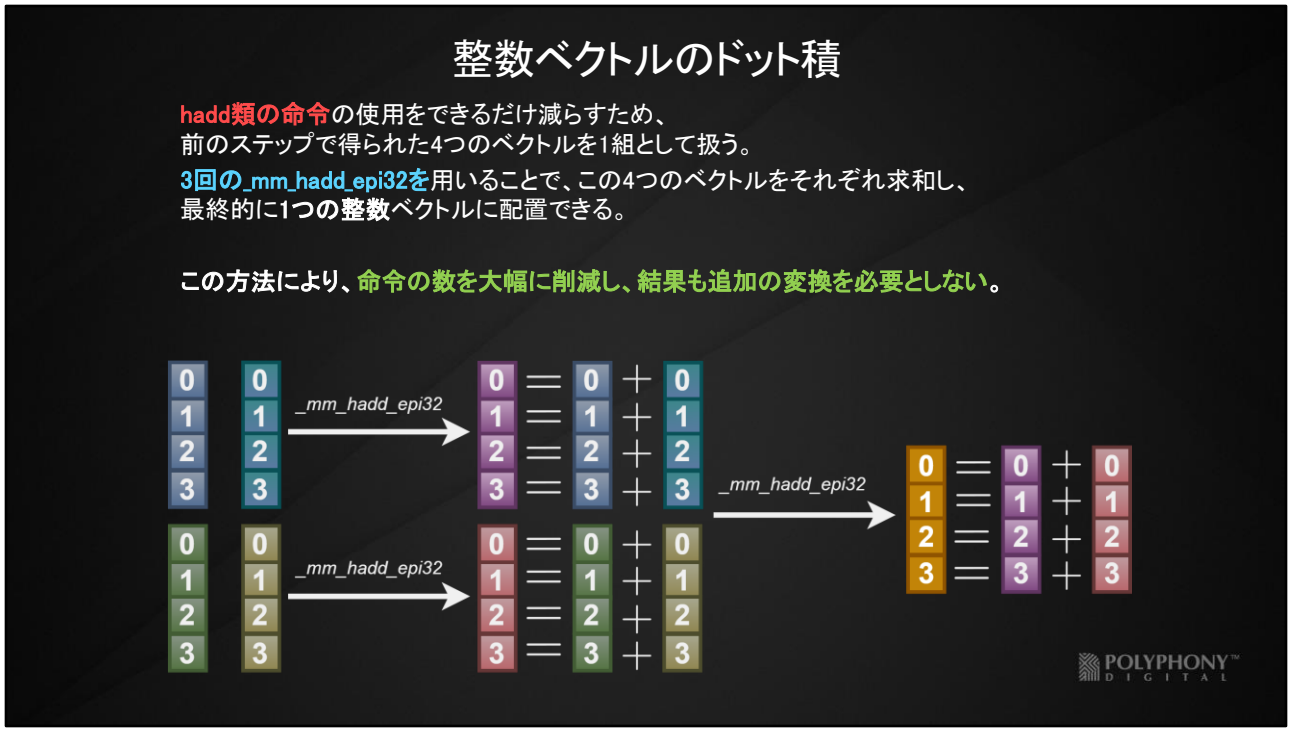
そこで、まだ解決していないのは、整数ベクトルのドット積をいかにして高速に求めることです。

このためには、次の2つのSSE命令を使用します。

まず、**1つ目は「_mm_maddubs_epi16」（この命令）です。**
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、
この_mm_maddubs_epi16のみです。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ビットで出力します。
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（じっしつ）的に7ビットのデータしか保存できません。

次に、**2つ目は「_mm_madd_epi16」です。**
ドット積の結果をさらに和として求め、
オーバーフローを防ぐために、この
_mm_madd_epi16を使用します。
これにより、先ほどの結果と1のドット積を
取り、隣接する2つの値(あたい)の和を32
ビットで出力することができます。

最後に、残る4つの32ビット整数の和を求
める必要があります。命令の使用をさら
に削減するために、より効率的な方法が
必要です。



hadd類の命令の使用をできるだけ減らすために、前のステップで得られた4つのベクトルを1組（グループ）として扱います。

ここで、
3回の_mm_hadd_epi32（すいへいわ）を使用することで、
この4つのベクトルをそれぞれ和を求めて、最終的に1つの整数ベクトルにまとめることが可能です。

この方法により、命令の数を大幅に削減でき、さらに結果に追加の変換を必要としません。

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

上記のtableから、紹介された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ることがわかります。

結果:アイガー北壁 (PS5®)

POLYPHONY™ DIGITAL

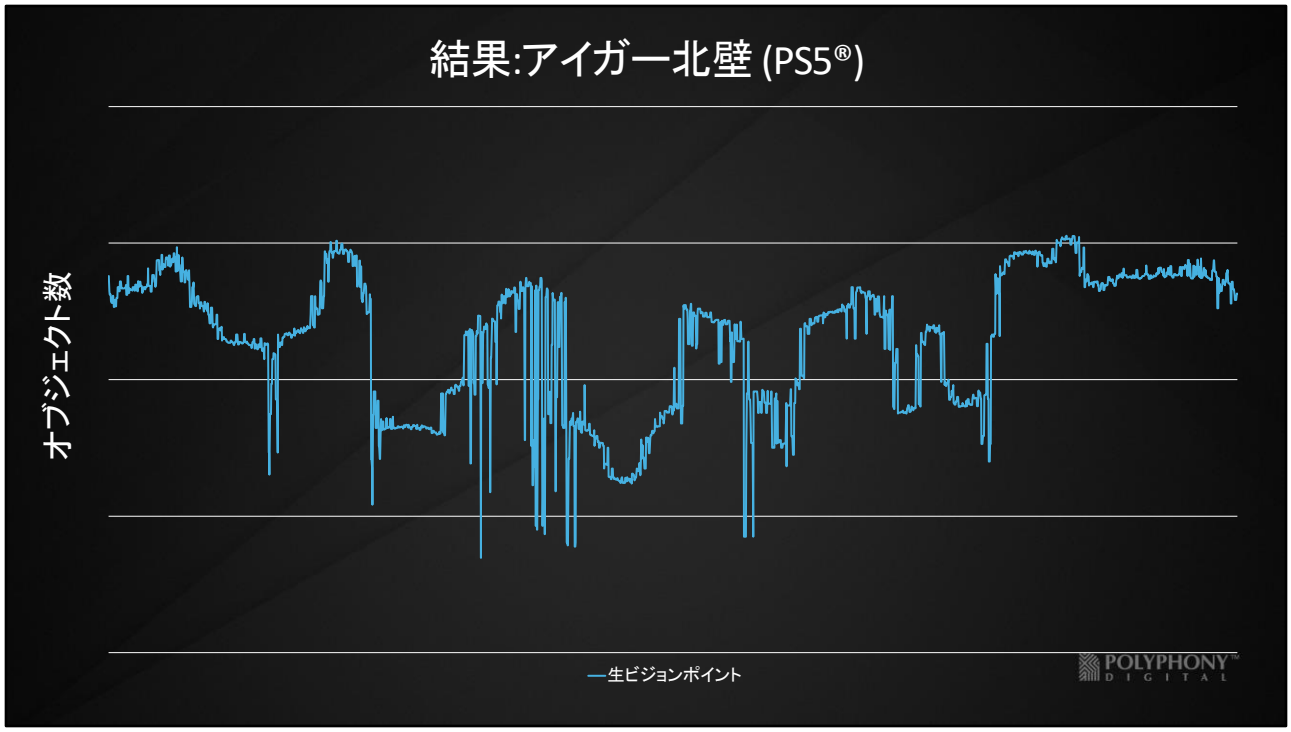
結果:アイガー北壁 (PS5®)

	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**

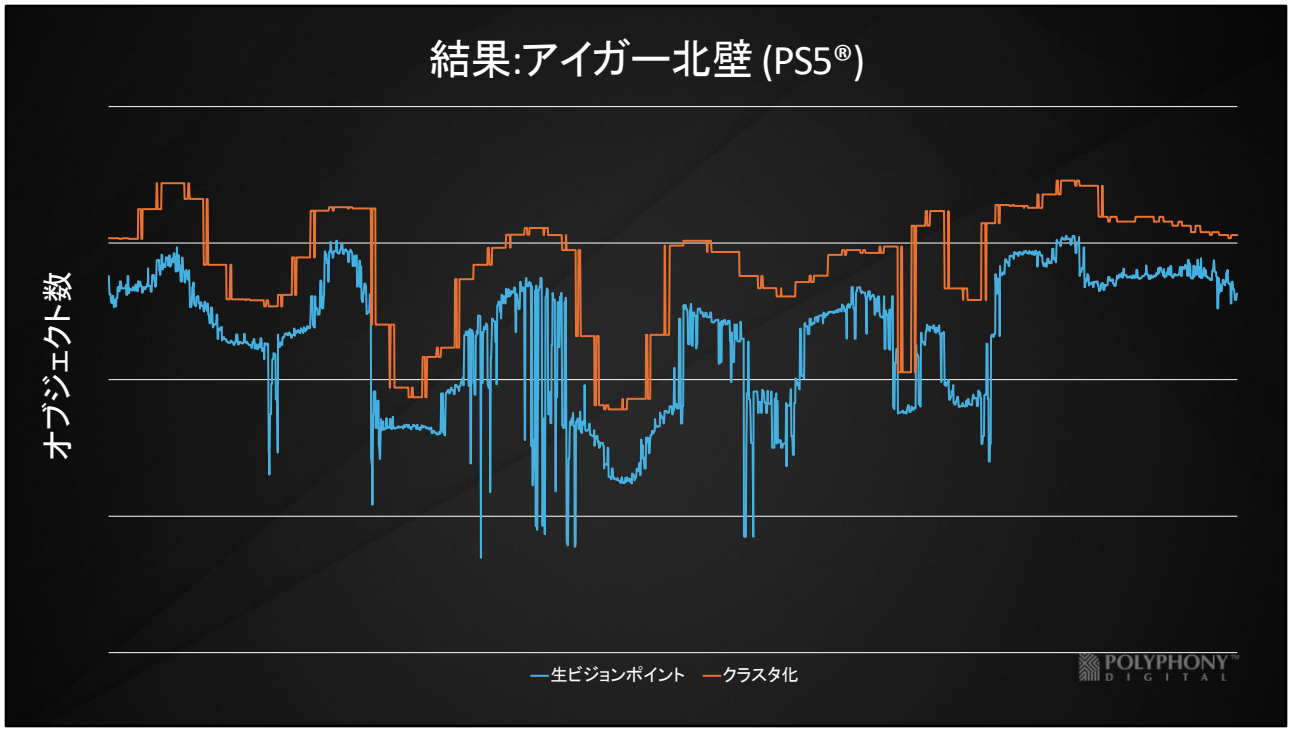
アイガーは、**遮蔽物が少ない広大なコース**である。
ニューラル可視性フィールドを使用することで、コース内のランプや丘陵などの
潜在的な遮蔽物を活用し、レンダリングの負荷をさらに軽減できる.

POLYPHONY™ DIGITAL

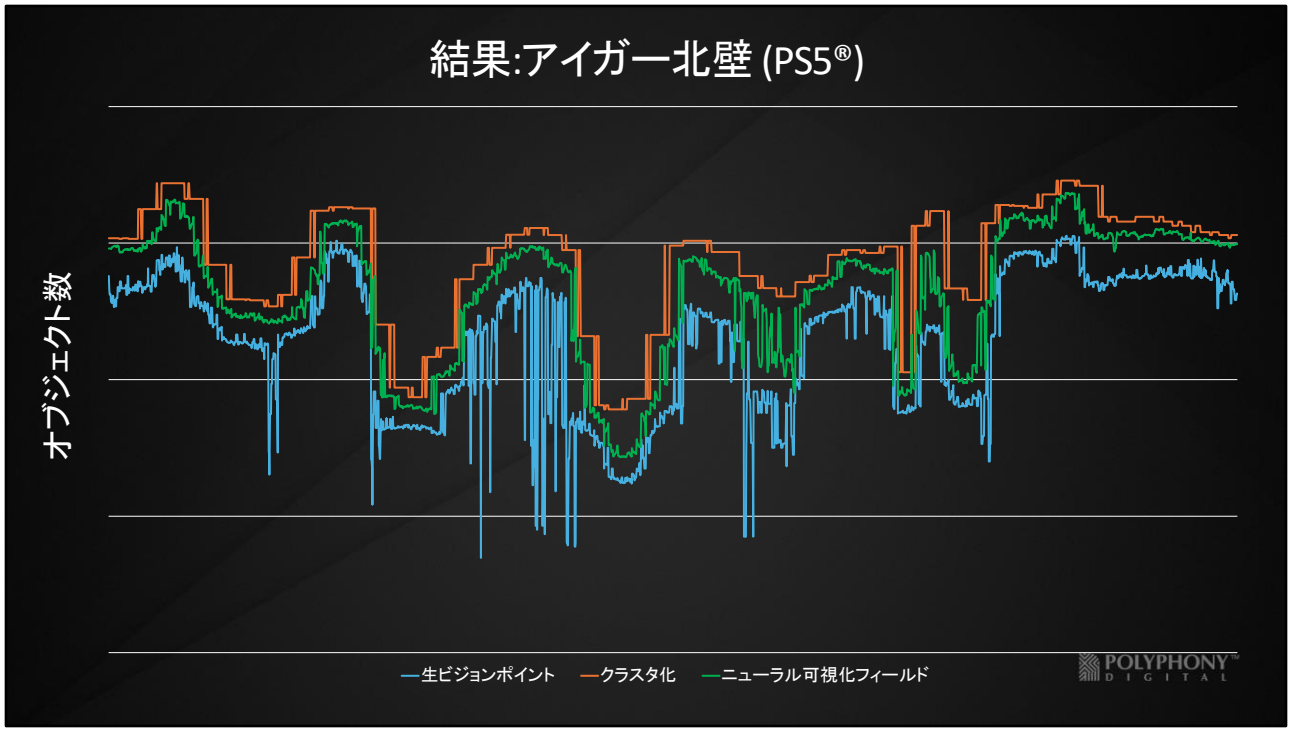
アイガーは、遮蔽物が少ない広大(こうだい)なコースです。ニューラルビジビリティフィールドを使用することで、コース内のランプや丘陵（きゅうりょう）などの潜在的な遮蔽物（しゃへいぶつ）を活用し、レンダリングの負荷（ふか）をさらに軽減（けいげん）できます。
このコースでの全体的な改善は大きいです。



アイガーのコース一周のオブジェクト数変動となります。
青い線は生のビジョンポイントの結果です。



アイガーのコース一周のオブジェクト数変動となります。
オレンジ色の線 はクラスタ化された結果です。



緑色の線はニューラル可視化フィールドの結果となります。

緑の線とオレンジの線の間のギャップが、ニューラルネットワークによって向上した性能を示しています。

また、クラスタ化ビジョンリストの欠陥(けっかん)だった、非連続な変化が、ニューラルネットワークによって、解消できたこともわかります。

結果:グランバレー(PS5®)

次にこの手法の結果について説明します。


HELLA
MOTHERS
WeatherTech
MOTHERS

結果:グランバレー(PS5®)

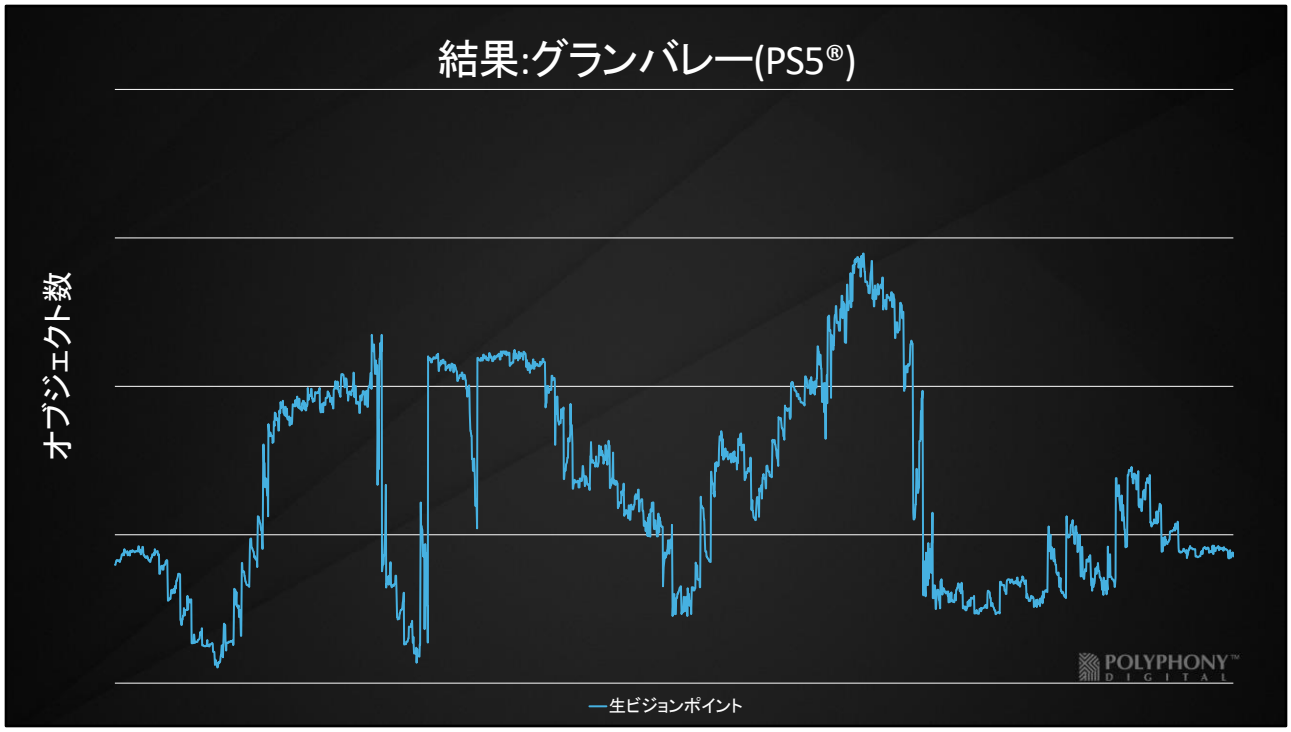
	w/o Neural	w/ Neural	Improvement
GPU avg.	7.180ms	7.064ms	-0.116ms
GPU min	5.797ms	5.777ms	-0.020ms
GPU max	8.885ms	8.787ms	-0.098ms
CPU avg	4.552ms	4.256ms	-0.296ms
CPU min	3.229ms	2.931ms	-0.298ms
CPU max	6.378ms	5.849ms	-0.529ms

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可視化フィールドにより、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。

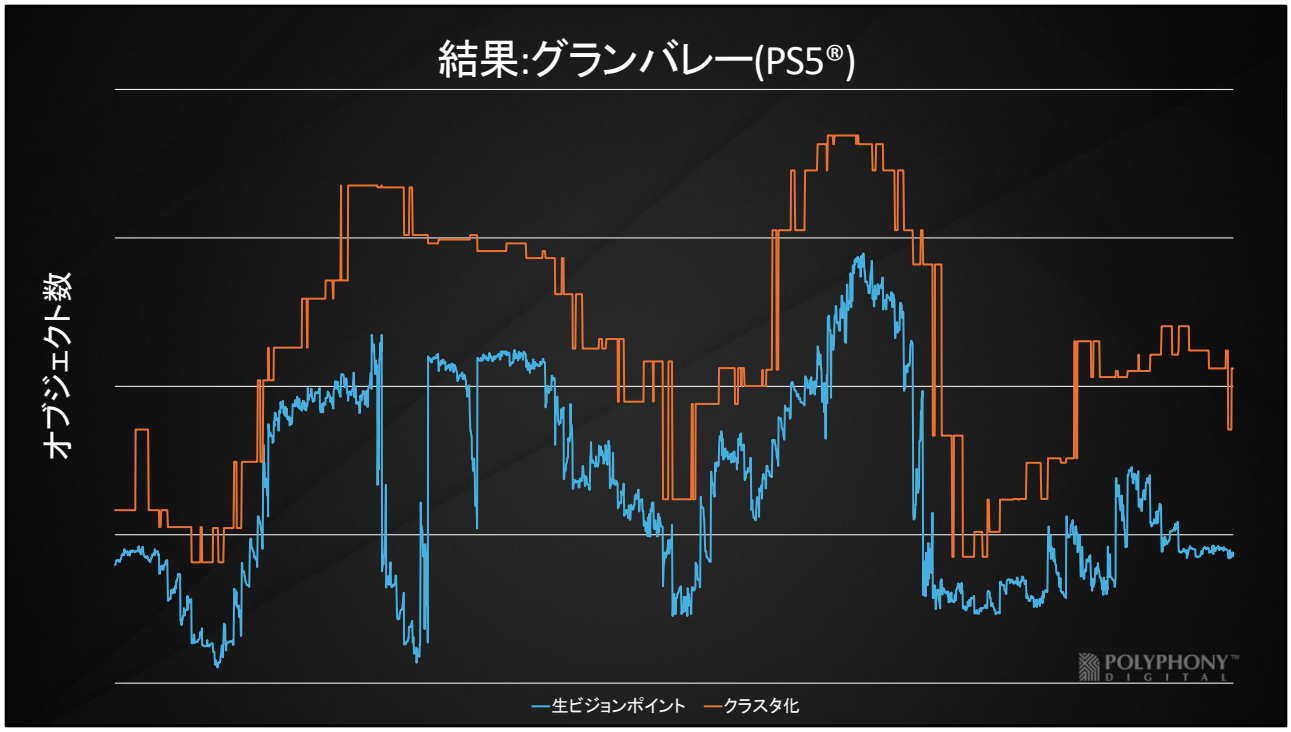
POLYPHONY DIGITAL

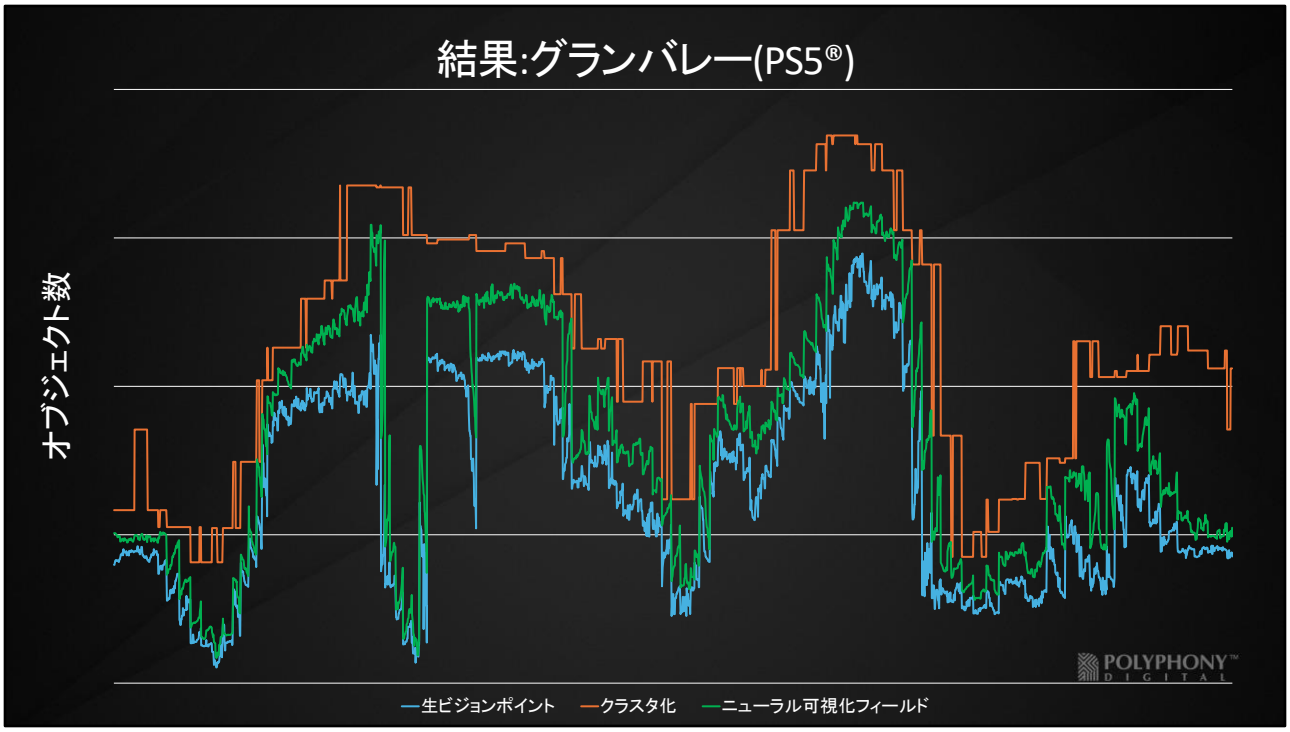
グランドバレーは、
広大(こうだい)なスケープを持つ
大規模なコースです。

ニューラル可視化フィールドにより、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レンダリング負荷を軽減することができます。

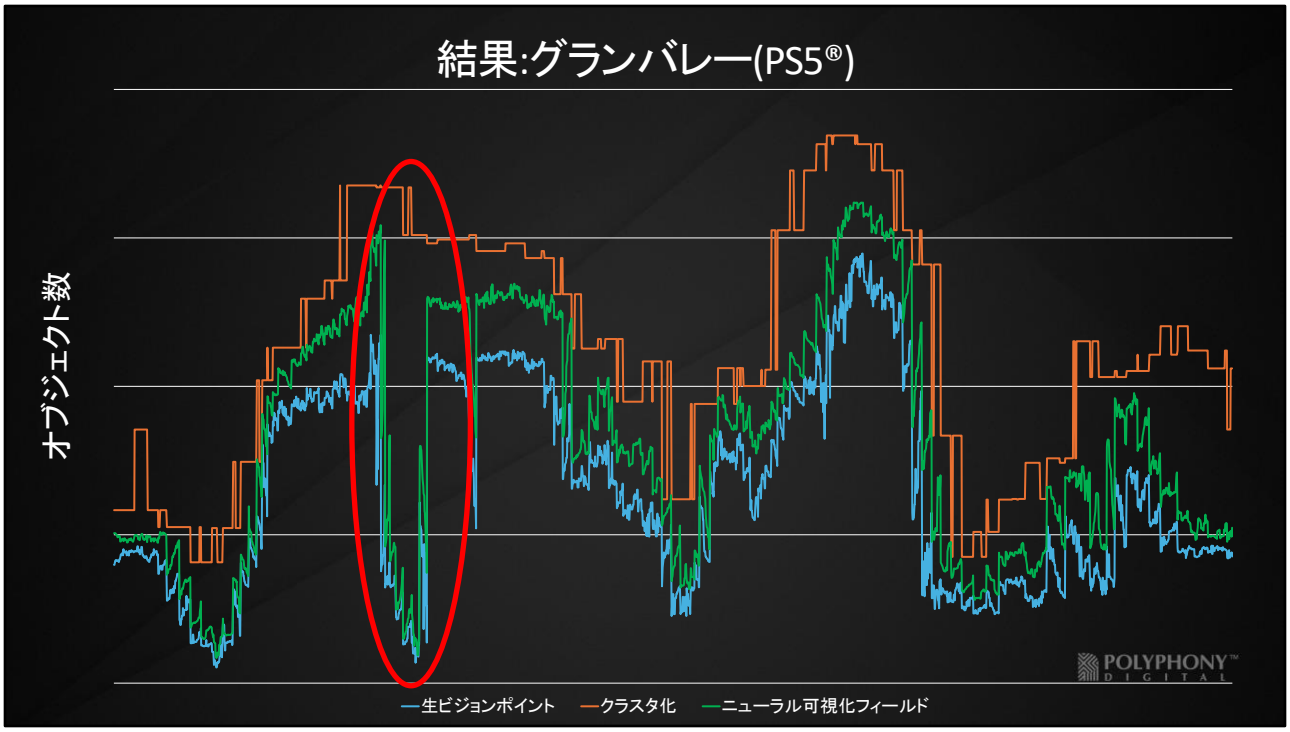


グランバレーのコース一周のオブジェクト数変動となります。


結果:グランバレー(PS5®)
オブジェクト数
POLYPHONY DIGITAL
—生ビジョンポイント —クラスタ化



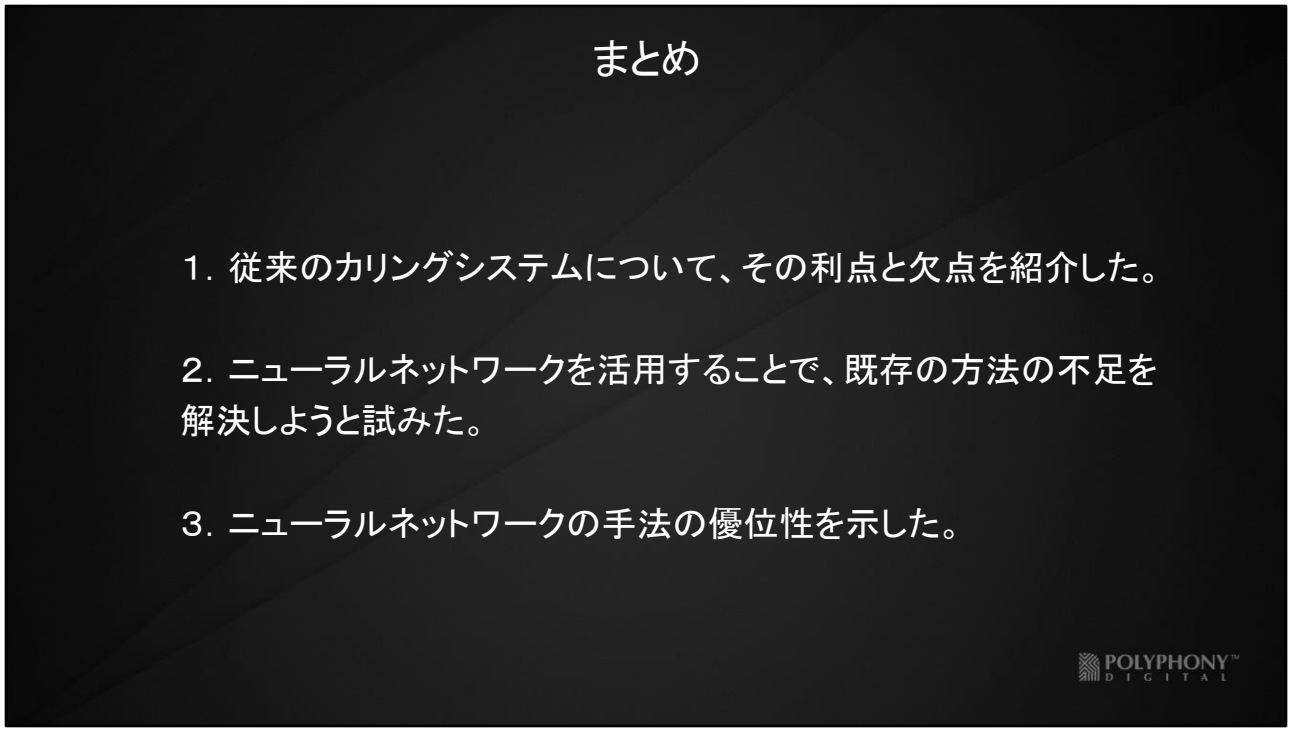
同じように、緑の線とオレンジの線の間のギャップが、ニューラルネットワークによって向上した性能を示しています。



この部分をご覧ください。
クラスタリングされた結果は、生ビジョンポイントの結果とは大きく異なります。
しかし、ニューラルネットワークの結果は生ビジョンポイントに非常に近づくことができます。

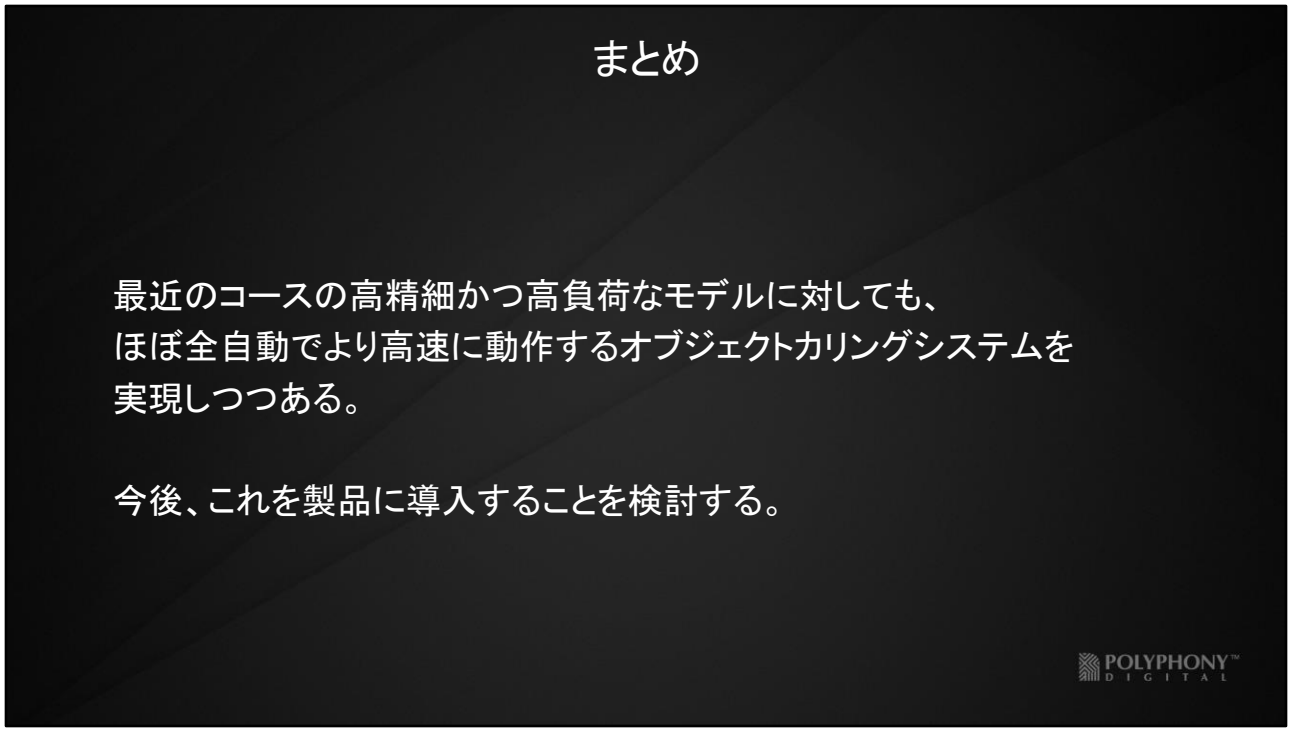
赤い円の部分では、本来、上部は山の斜面によって視界が遮（さえぎ）られ、見えないはずです。
しかし、コース間の間隔が狭すぎるため、クラスタリングが誤（あやま）って行われ、2つの部分が一つに統合されてしまいました。

ニューラルネットワークを使用すると、右側の広い領域（りょういき）が正確にカリングされていることが分かります。

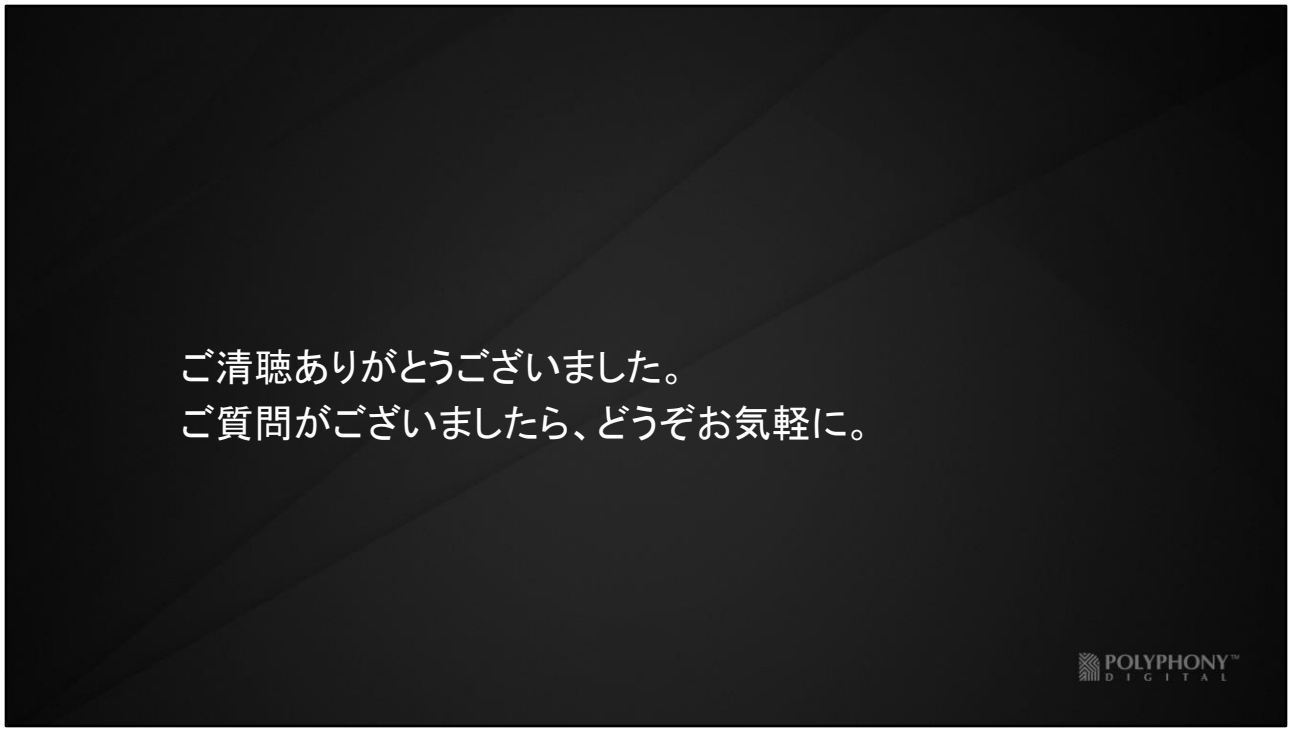


まず、我々は従来のカリングシステムについて、その利点と欠点を紹介しました。
次に、従来のカリングシステムに対して、ニューラルネットワークを活用することで、既存の方法の不足を解決しようと試み（こころみ）ました。

最後に、実験を通じて、私たちのニューラルネットが
カリングのタスクにおいて全体的な優位性を示しました。



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(じつげん)しつつあります。今後、これを製品に導入することを検討しています。



ご質問がございましたら、どうぞお気軽に
(きがるに)お聞きください。