

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



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

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

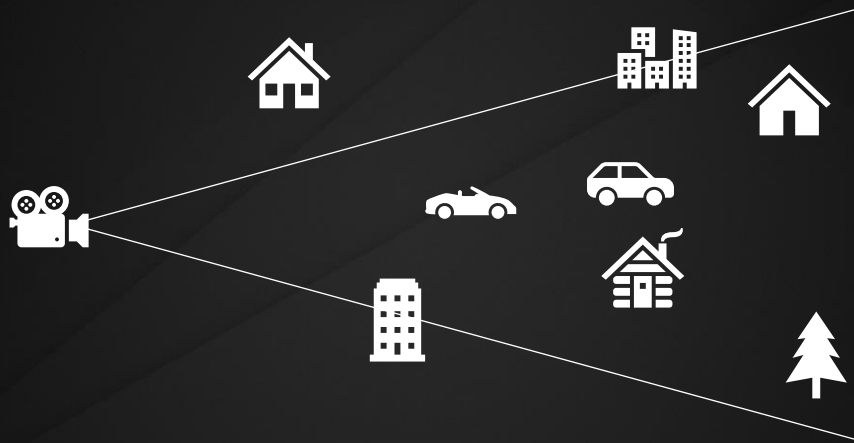
オブジェクトカリングとは

- 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？

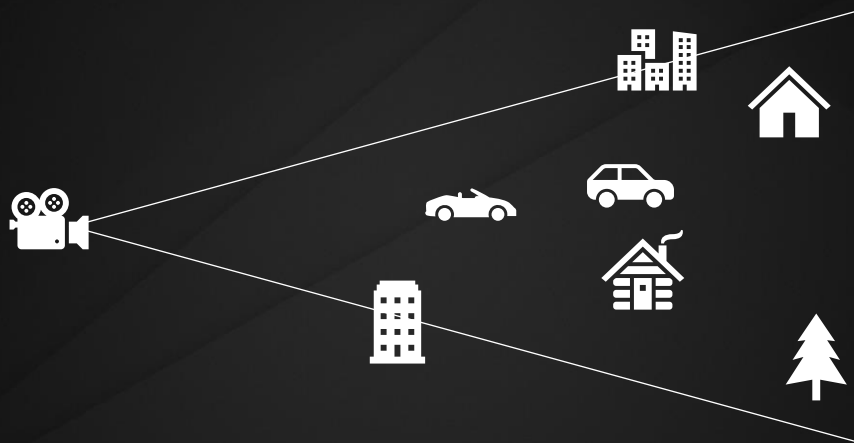
- 軽量になるから
 - 処理が減るので、CPUが軽くなる
 - 描画が減るので、GPUも軽くなる

典型的なシーンを考える



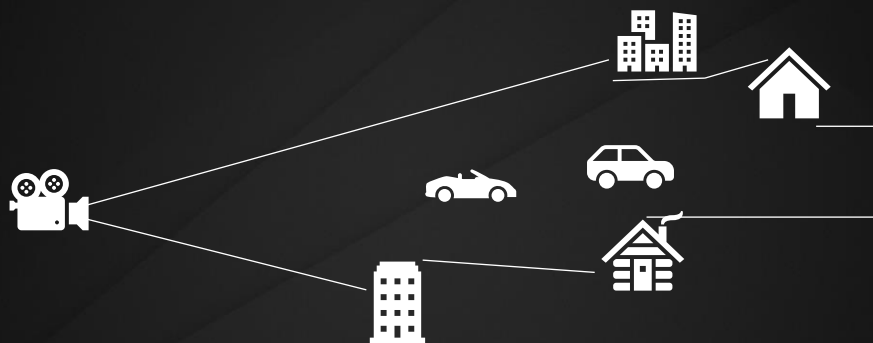
POLYPHONY™
DIGITAL

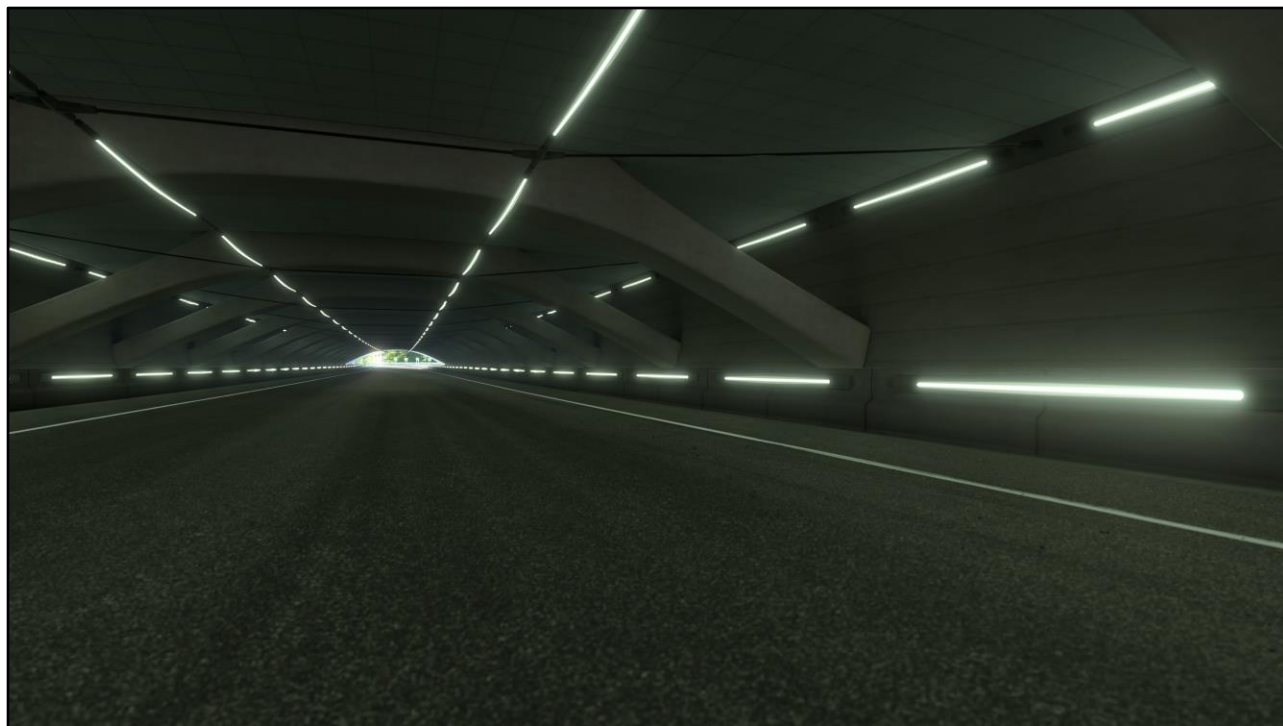
カメラの範囲外は描画しない
(フラスタムカリング)



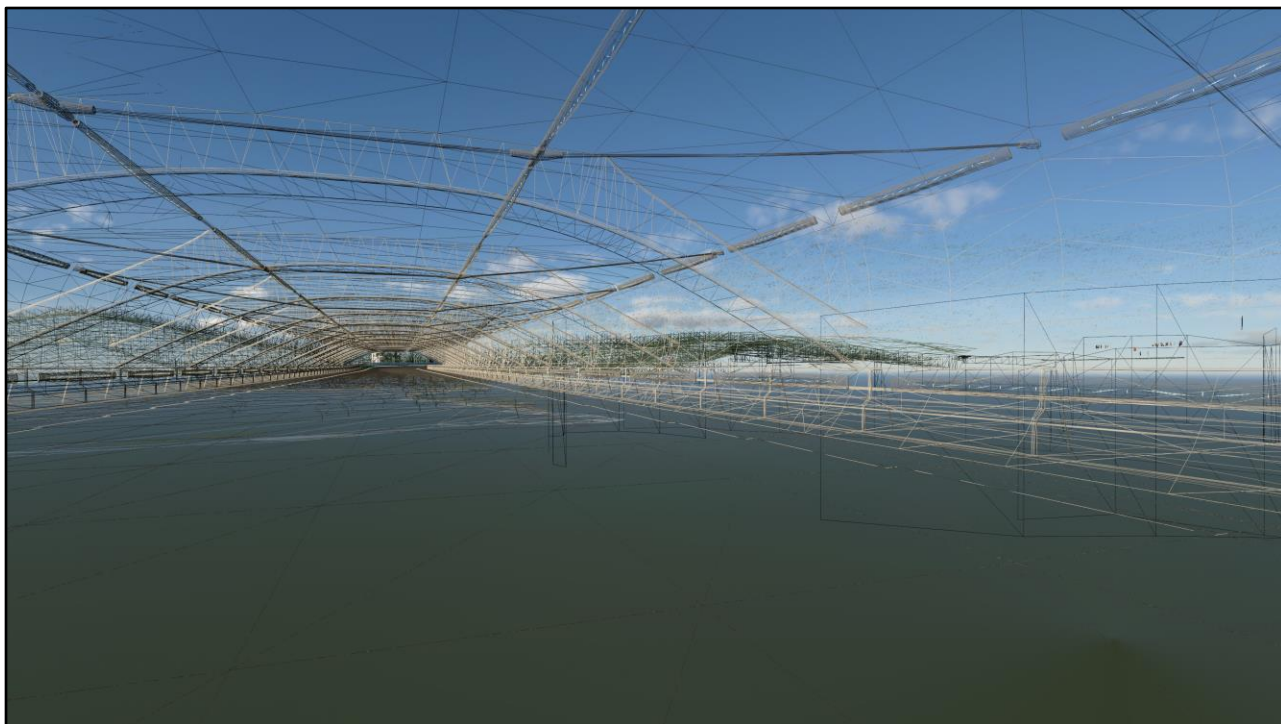
POLYPHONY™
DIGITAL

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)

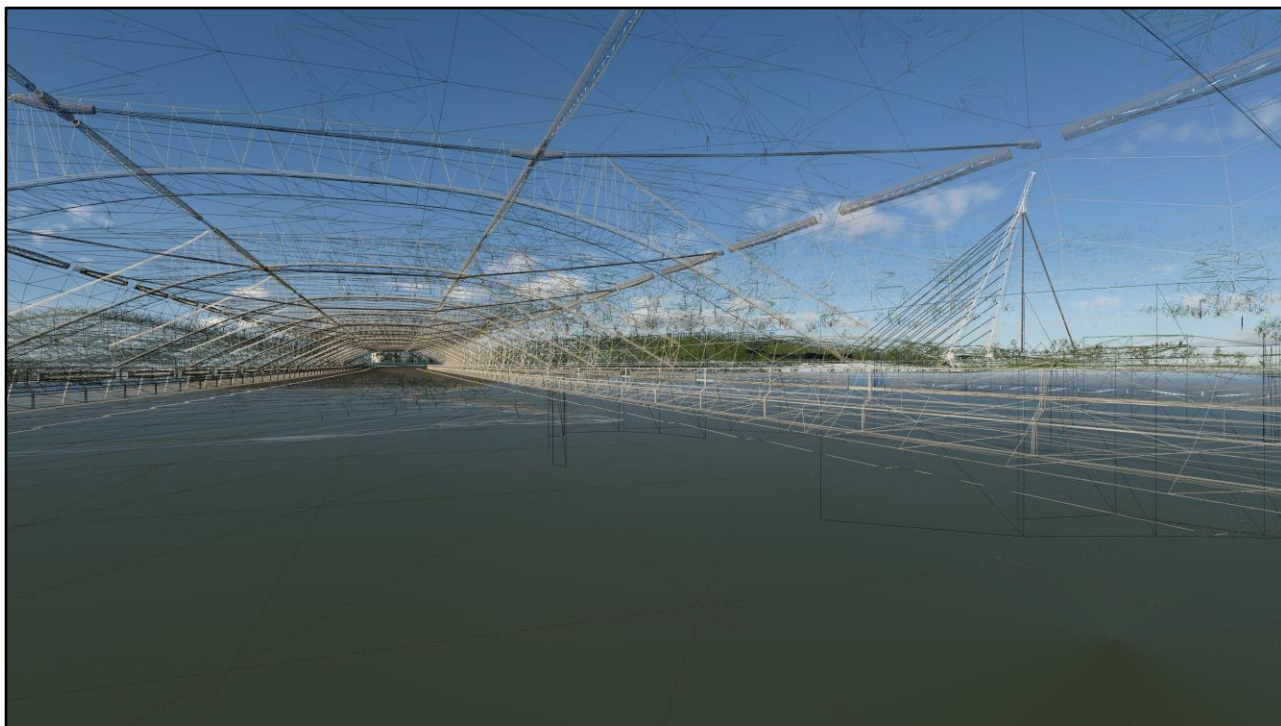




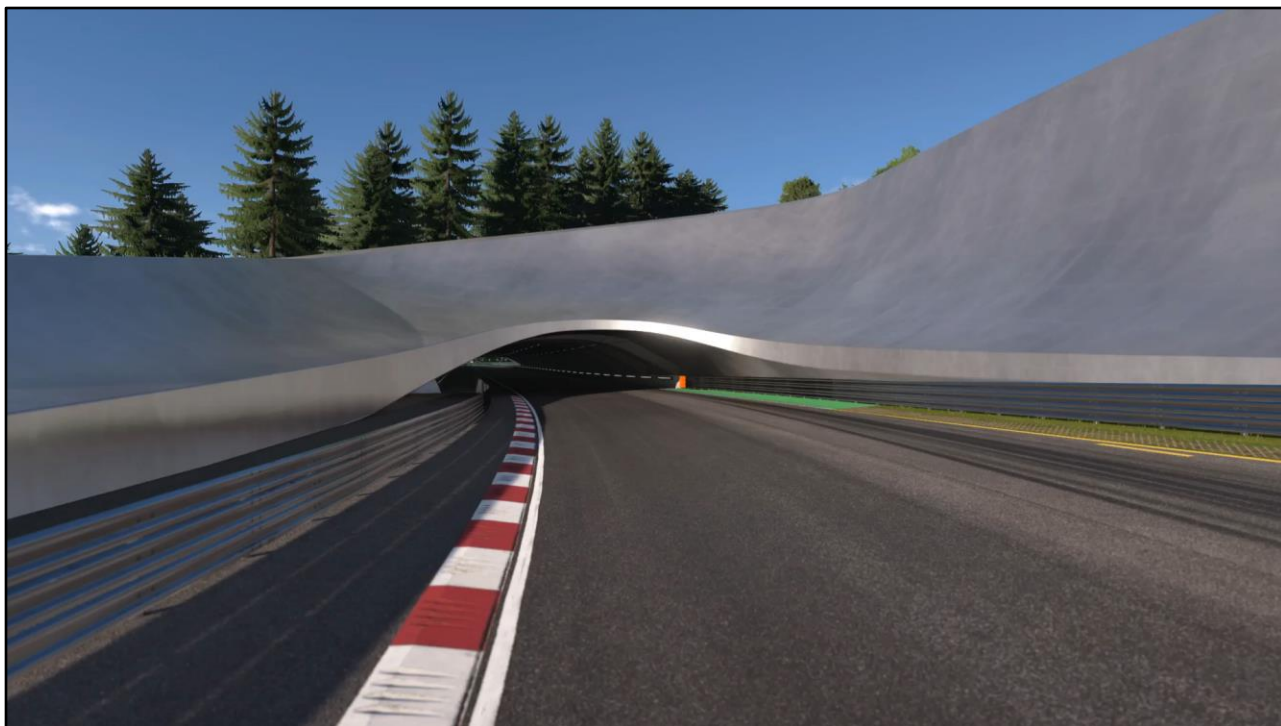
わかりやすくするために極端なシーンを
用意しました。トンネルです。



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”

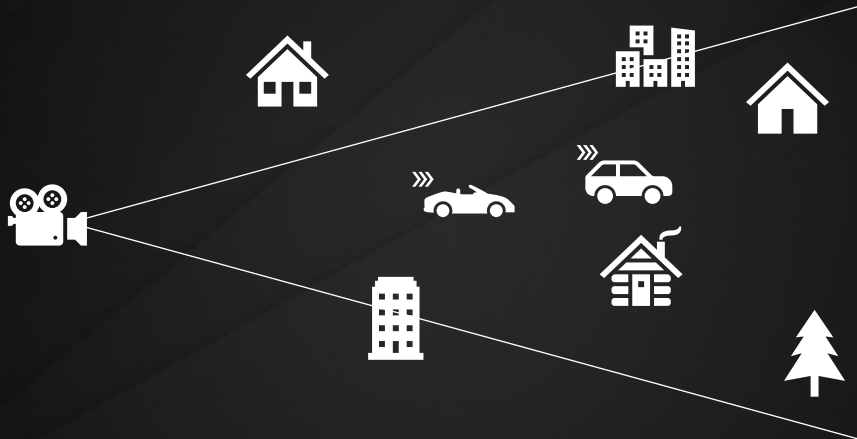
基本的なアイデア

 POLYPHONY™
DIGITAL

基本的なアイデア

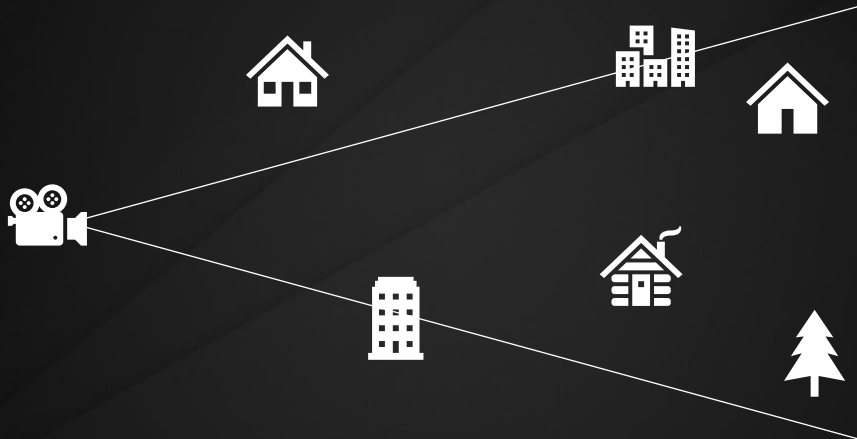
1. コース全体をいろいろな位置と角度からレンダリングする
2. レンダリング結果から、可視オブジェクトのリストを得る
3. 可視オブジェクトのリストを、シンプルな実機データにする

先ほどのシーン



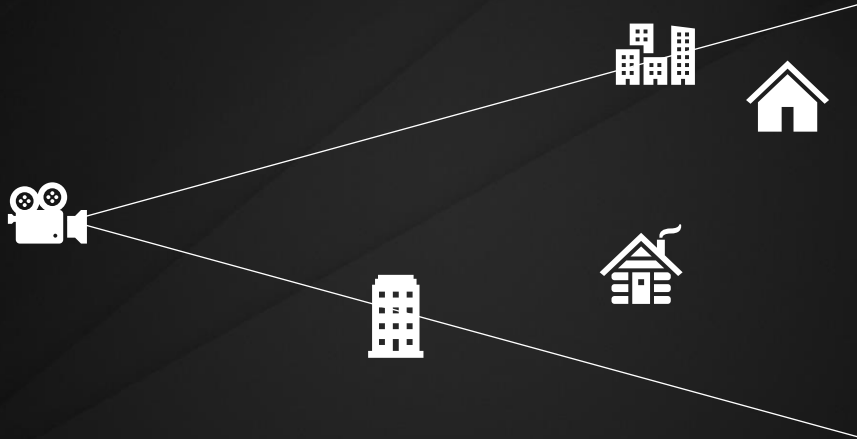
POLYPHONY™
DIGITAL

事前計算は静止物のみを対象とする



POLYPHONY™
DIGITAL

可視判定する



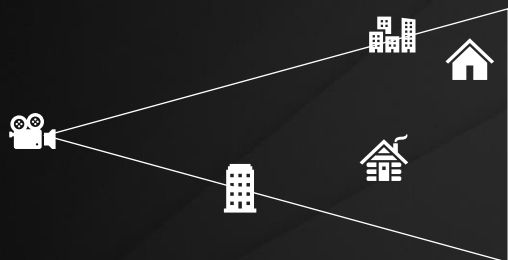
POLYPHONY™
DIGITAL

可視判定結果の表



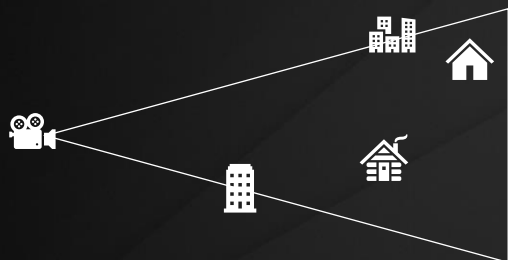
モノ	可視？
	✓
	✓
	✗
	✓
	✗

可視リストにする



カメラ	リスト
	   

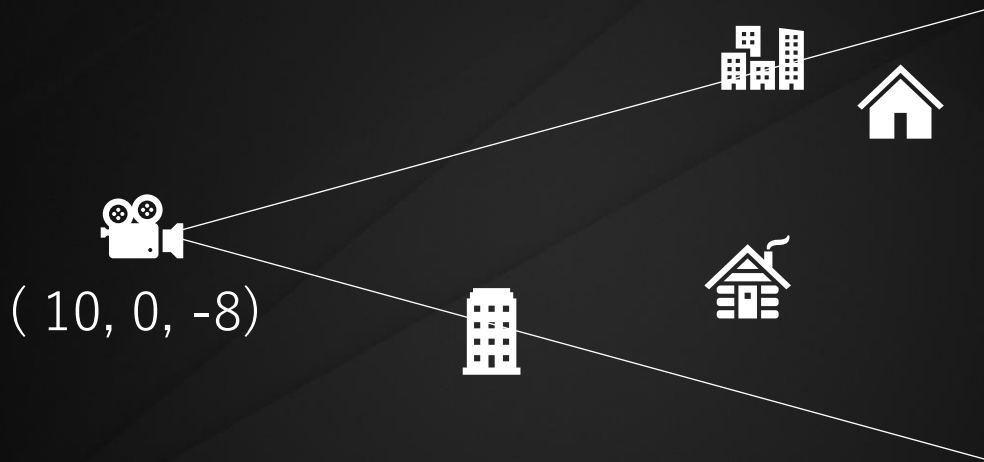
オブジェクトは
IDで管理しておく



モノ	ID
	1
	2
	3
	4
	5
	6

 POLYPHONY™
DIGITAL

カメラは座標情報を持つ



POLYPHONY™
DIGITAL

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

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



 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
...	...

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

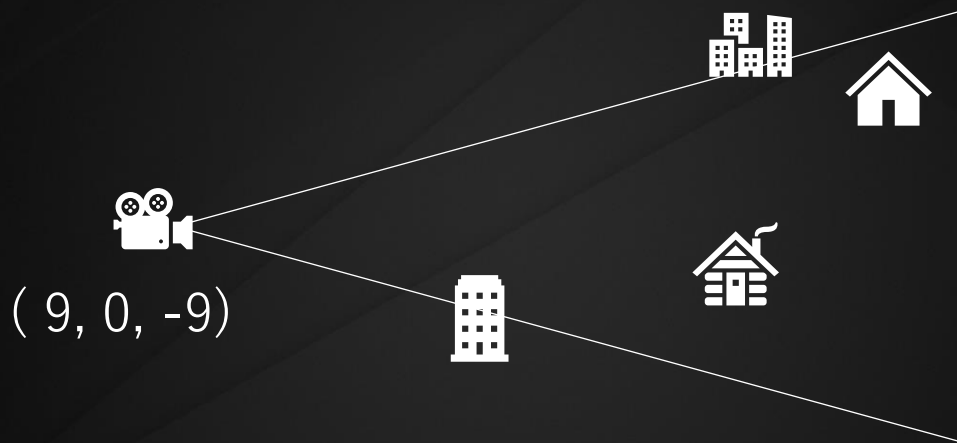


ID	モデル
1	
2	
...	...

描画すべきオブジェクト

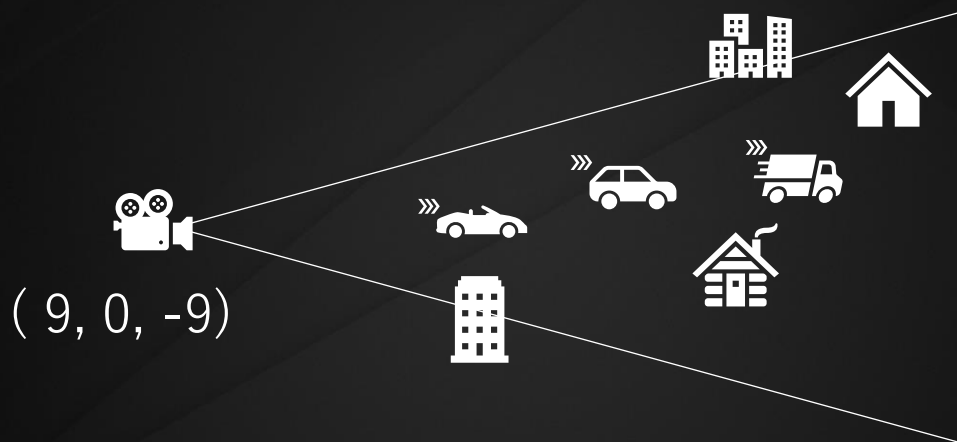


見えるオブジェクトだけ描画する




POLYPHONY™
DIGITAL

カリング対象外の
オブジェクトを追加描画する



POLYPHONY™
DIGITAL

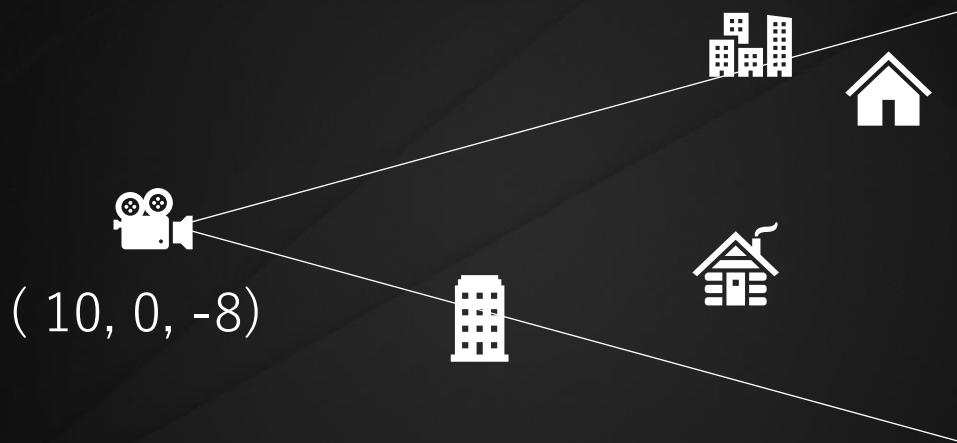
あくまでも近傍なので、誤差が発生する


(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
...	...

事前計算結果はこうだが



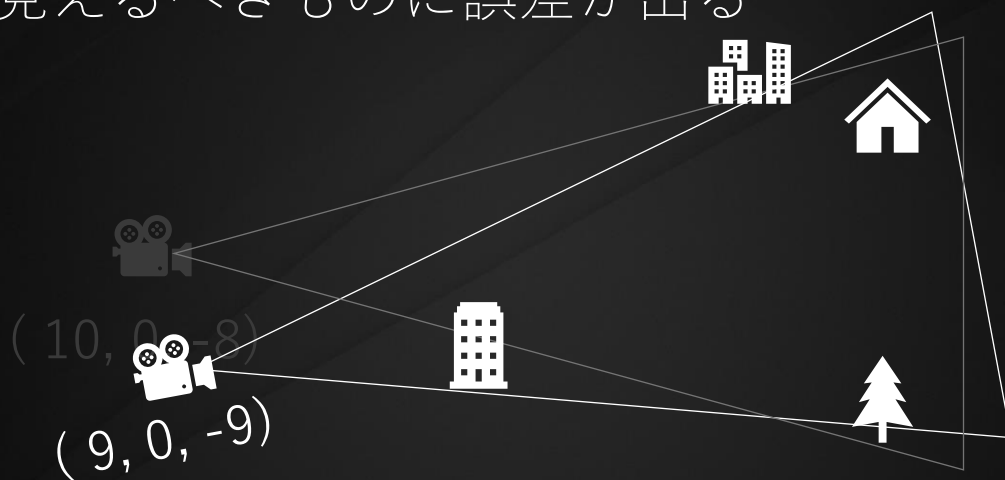
POLYPHONY™
DIGITAL

実際のカメラとのズレがあるので



POLYPHONY™
DIGITAL

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

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



POLYPHONY™
DIGITAL

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

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


$$\frac{\text{Building Icons}}{\text{Building Icons} + \text{Tree Icon}} = 75\%$$

POLYPHONY™
DIGITAL

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

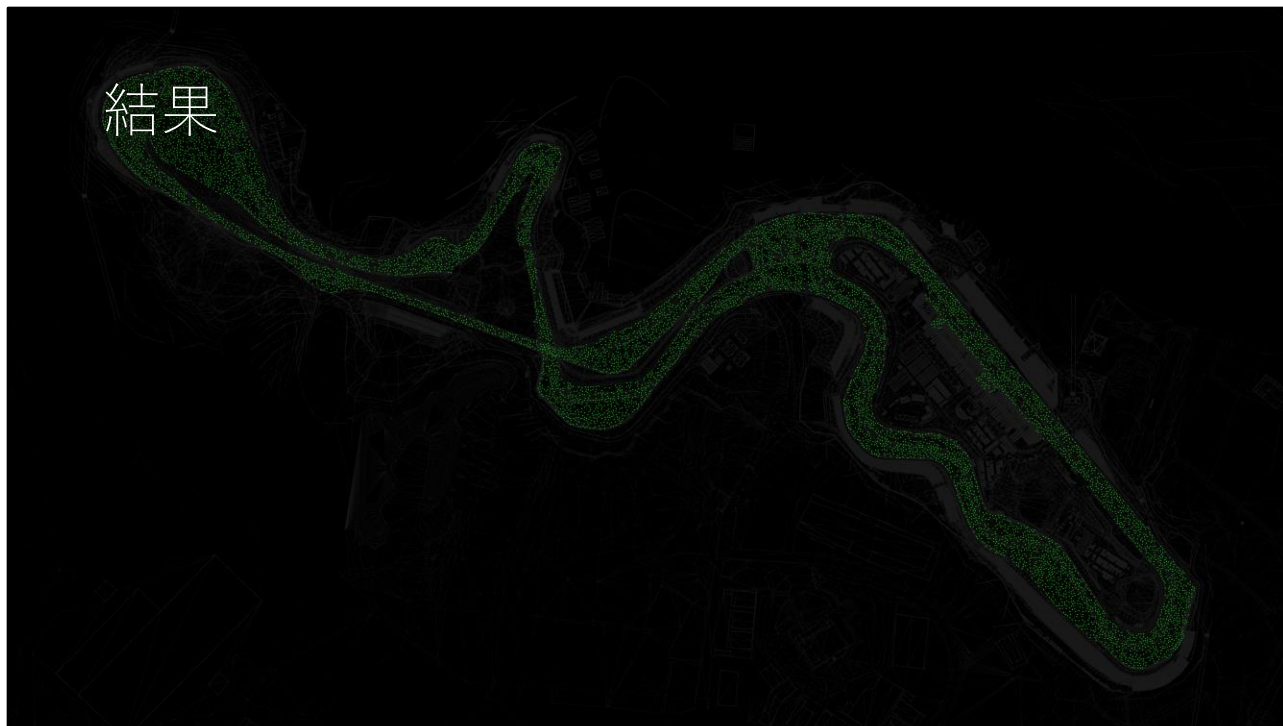
用語	意味
ビジョンポイント	ある座標と、そこから見えるオブジェクトのリスト
ビジョンリスト	ビジョンポイントのリスト
母点	カメラ座標から最も近いビジョンポイントの座標
リコール率	低いと画像がバグる
精度	高いとパフォーマンスが上がる

具体的な実装

カメラの移動範囲をどう求めるか

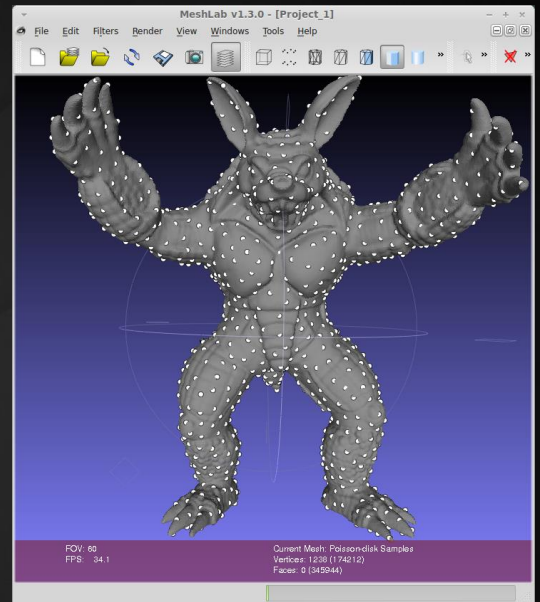
- カメラの範囲 = 走行可能な範囲
- 走行可能な範囲 = 路面メッシュ
- ポワソンディスクサンプリングする

結果



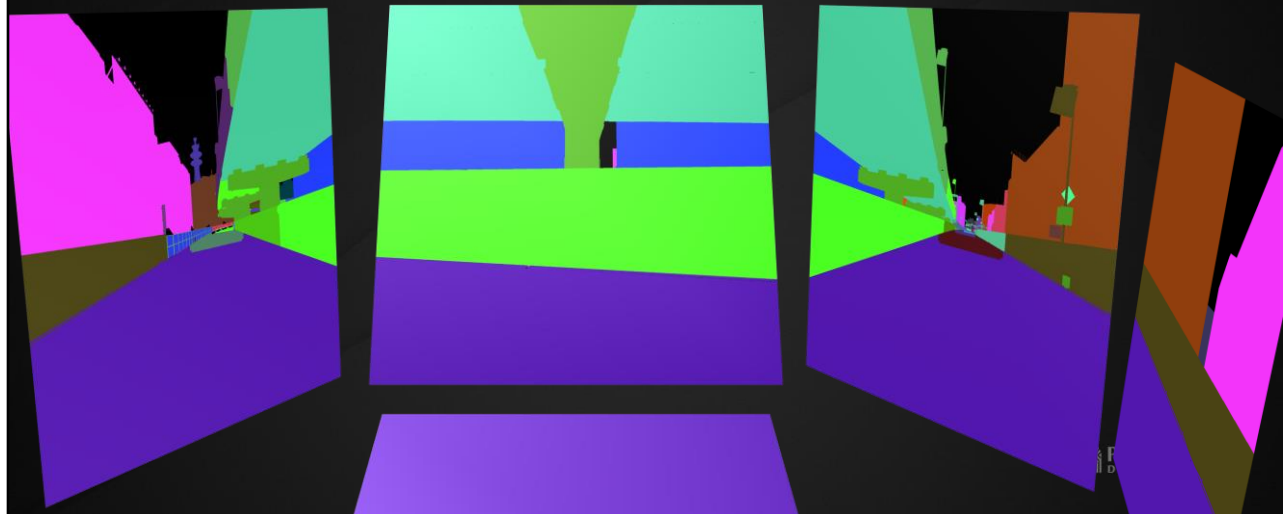
ポワソンディスク サンプリングとは

- 均一にサンプル点を生成する
アルゴリズム



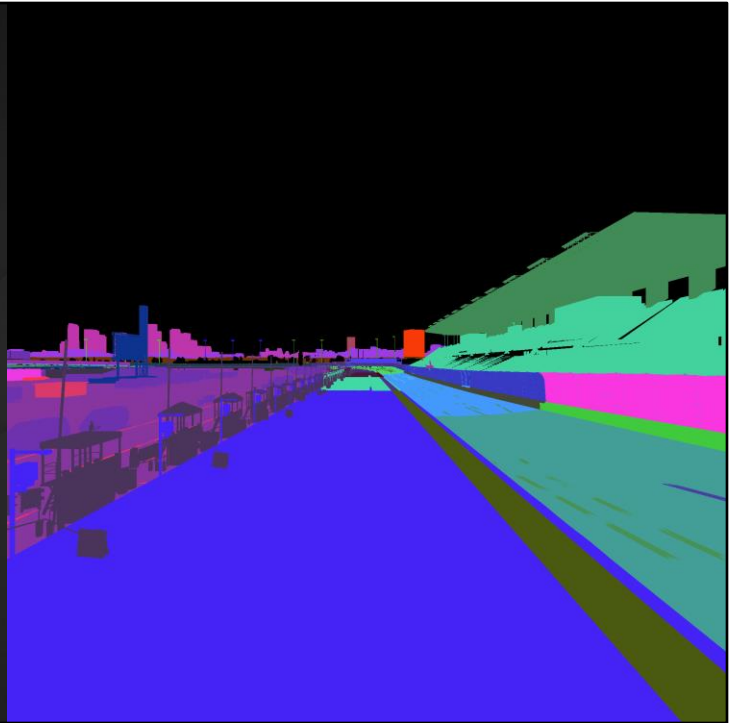
<https://rodolphe-vaillant.fr/entry/37/surface-code-for-poisson-disk-sampling-of-a-triangle-mesh>

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を使う

Name

glBeginQuery — delimit the boundaries of a query object

C Specification

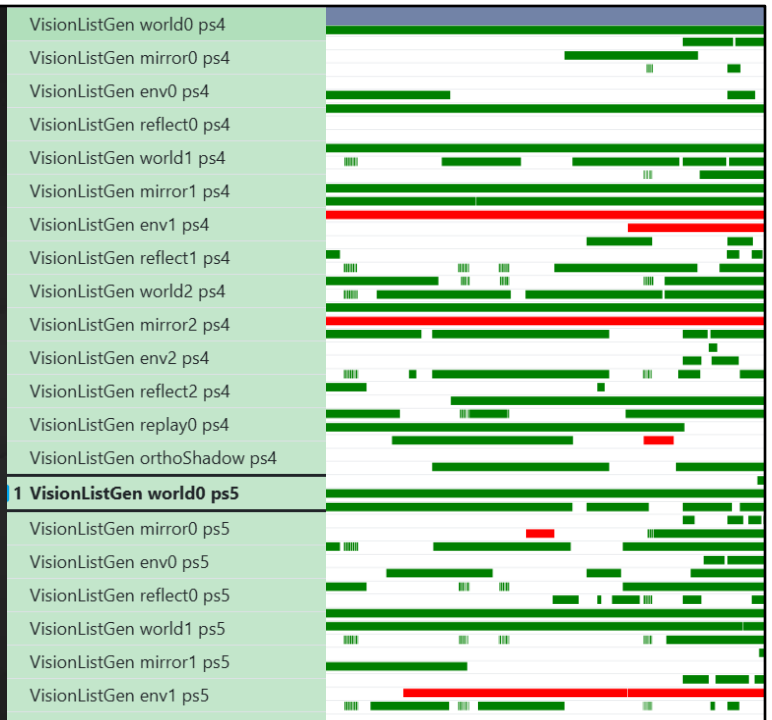
```
void glBeginQuery(GLenum target,  
                  GLuint id);
```

```
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

```
(-80.544342, 15.506147, 39.682064) : 313a2d4dfca3d9a, cb30bf5ab22c92e1, 9e3704848a357b38, 523aaab20ecce6897, 603737d24a26fadd, 12391605c9cd40d5,
743411395ffa95a6, fe388c13f9471c9d, 9d3907d4a123f7fe,
(-70.130470, 14.925819, 91.462791) : b21a91870b9a5d341, 283b887ccf96cc4a, 693164444d76a5aa, 4c3fbe236d6015c8, b03cac0aea8dc807, 1636e52f4c9ef1c7,
d371820e9c4285e, 9338b29c4c94099f, cd3080b01f1b39e1,
(66.580383, 17.996874, -180.515060) : 12391605c9cd40d5, 1336a2c31e9c3e4d, 433b6c8108af78ec, 293b216947eb6365, 2c3019d11dc58834, 5433b9198fe43276
2, c23f84ce92737361, 7b3d34b62ee03323, 363b8019942fc8d5,
(-219.245804, 14.786657, -51.287498) : 943a86c2e28fa8uc, 4d3cc2b8dc3cda24, 2d39013251892806, 3b37de98c2778a58, a439faf105a277fd, b93fd6e97fd3b7f
e8, bf3c1942e57002fa, ca38013783ae9b8f, f93233048fd8cd8d,
(-31.678360, 12.437248, -99.001411) : c33ff37c4bb9990, 3c34e230a2ec4078, a3b31c317b60638, 903b34f29cd3bdf1, 9737ab35202aa433, b636aa9d044160ad,
4335e61c68a1a3b, a238805370afc33b, a83e29412fafbdal,
(-45.484364, 13.499858, 9.298882) : 12391605c9cd40d5, 1f30eae0b15346a2, ae327f72c4e73f6c, 65382b50f3c579f8, f135e46083b345fe, 2531862941abde20, 4
743b3f87d798586c, 743411395ffa95a6, 9d3907d4a123f7fe,
(-132.902423, 141.344069, -94.542999) : 930d02a20013b93, b93fd6e97fd3b7f7, 523aaab20ecce6897, 603737d24a26fadd, af39fd33754205a, bf31f1475f10ef8b
a, f93233048fd8cd8d, dc39b9cca788811f, c2352571f7a62181,
(106.252930, 15.763781, -125.148094) : 30343621cae5e0e0, 4e39c41409191859, 4332295fe42662de, 3322f50c6263ee, a3b31c317b60638,
(-52.421387, 15.250546, 41.459705) : 313a2d4dfca3d9a, cb30bf5ab22c92e1, ae327f72c4e73f6c, 1f30eae0b15346a2, 443ef70401e8993c, d83e51623257b8d5,
fe388c13f9471c9d, 9d3907d4a123f7fe, ca380960bb80001,
(-229.830689, 14.251721, -177.471485) : 93346d79f3d636b1, 1b324e4ccc9ecc20, 4d3cc2b8dc3cda24, b93fd6e97fd3b7f7, a439faf105a277fd, 2d390132518928
4ee, 883dd56116b3235e, 3f3fc8b2c028f1a2, 553978c5bb22a9c6,
(-294.597015, 14.997087, -13.534410) : 2344d4fb292aa0b, b037ec0ee72453f6, b93fd6e97fd3b7f7, 12391605c9cd40d5, b4380b7ec00a4b27, d83e51623257b8d5
6, 87338259e4261af5, 743b3f307d798586c, b3930be56d31897,
(-253.832916, 15.985291, -42.700991) : 943a86c2e28fa8uc, 4d3cc2b8dc3cda24, 2d39013251892806, 3b37de98c2778a58, a439faf105a277fd, b93fd6e97fd3b7f
e8, bf3c1942e57002fa, ca38013783ae9b8f, f93233048fd8cd8d,
(-7.316224, 13.129834, -112.400726) : c33ff37c4bb9990, 2c3019d11dc58834, 3c34e230a2ec4078, 5433b9198fe43276, a3b31c317b60638, 903b34f29cd3bdf1,
4e39c41409191859, b5336b37d2c24a83, a238805370afc33b,
(174.657181, 16.495752, 149.732849) : 6d3f4fb08d53c905, 9c39e1ec99813ebd, fc3c522b0e14818f, b93015998b94a31e, e339a15bed11ccb1, 733b51ff3488a9a2
293b216947eb6365, 493ef11daa69a9c, c93f03f62fcc2945,
(-9.404469, 16.368994, -104.403496) : 9e3704848a357b38, 523aaab20ecce6897, 603737d24a26fadd, bf31f1475f10ef8b, 12391605c9cd40d5, f135e46083b345fe
c, 2531862941abde20, 433b6c8108af78ec, 293b216947eb6365,
(-77.380569, 14.657792, -22.430290) : 523aaab20ecce6897, 9e3704848a357b38, 603737d24a26fadd, bf31f1475f10ef8b, 12391605c9cd40d5, f135e46083b345fe
4, 433b6c8108af78ec, d03112de6d6b3bcc, c2352571f7a62181,
(-198.576226, 14.806186, 41.647839) : 12391605c9cd40d5, 234e45992fd02ca, 363b8019942fc8d5, b03cac0aea8dc807, b23a91870b9d5d41, 4c3fbe236d6015c8,
ba32443a6d4f65b1, eb3bacb117d59b4, 1039850e7756cd58,
(55.781197, 15.118961, 143.103699) : cb30bf5ab22c92e1, d53029256c88e50, c23a06fe2eead11c, a635cb281133e6ef, 1336a2c31e9c3e4d, a338ac29b05433d6,
1a38aad51aad5f8e, 7e3c36551073d2de, 993036c4ee0bee73,
(170.756836, 16.498599, 194.003635) : 2c3019d11dc58834, 3c34e230a2ec4078, 5433b9198fe43276, 293b216947eb6365, e9339d3ecf8ecb43, b53bf7b3b592d62d
e, 733b51ff3488a9a2, e339a15bed11ccb1, 3d1a91cb37bbaa,
(-60.390053, 12.517113, -35.056850) : 9d3907d4a123f7fe, fe388c13f9471c9d, b9330be56d31897, a43c9023791e4d91, aa3e046d28c5881c, 733b51ff3488a9a2,
8f332c7e13b60245, 643041fe4a43235a, d43112de6d6b3bcc,
(13.234009, 13.870998, 138.506744) : 313a2d4dfca3d9a, cb30bf5ab22c92e1, 1336a2c31e9c3e4d, 8e3b3685ea259afc, f23359b02cbfe931, a338ac29b05433d6,
7e3c36551073d2de, 9b3c5886c7aa58bc, c23e7d0e3c9465ec,
(-225.889954, 14.302652, -60.872234) : 943a86c2e28fa8uc, 4d3cc2b8dc3cda24, 2d39013251892806, 3b37de98c2778a58, a439faf105a277fd, b93fd6e97fd3b7f
fa, ca38013783ae9b8f, f93233048fd8cd8d, 65382b50f3c579f8,
(-79.227341, 11.325275, 82.232590) : 363b8019942fc8d5, 1636e52f4c9ef1c7, 7e323bde6cae6d0c, b03cac0aea8dc807, 4c3fbe236d6015c8, 283b887ccf96cc4a,
```



山のよう に得られた



点が多すぎる

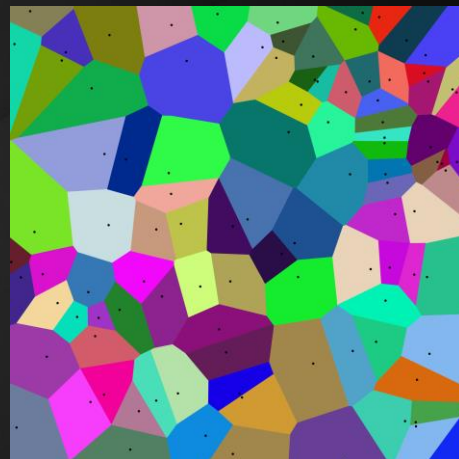
- コース全体で10万以上のビジョンポイント
 - データサイズ
 - クエリ速度
- このままでは非常に使いにくいデータなので簡略化したい

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

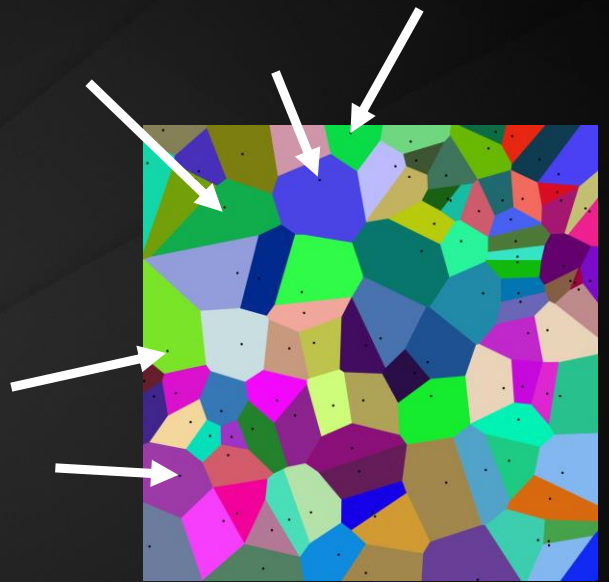


POLYPHONY™
DIGITAL

Mysid (SVG), Cyp (original) - Manually vectorized in Inkscape by Mysid, based on Image:Coloured Voronoi 2D.png,
CC 表示-継承 3.0, <https://commons.wikimedia.org/w/index.php?curid=4290269>による

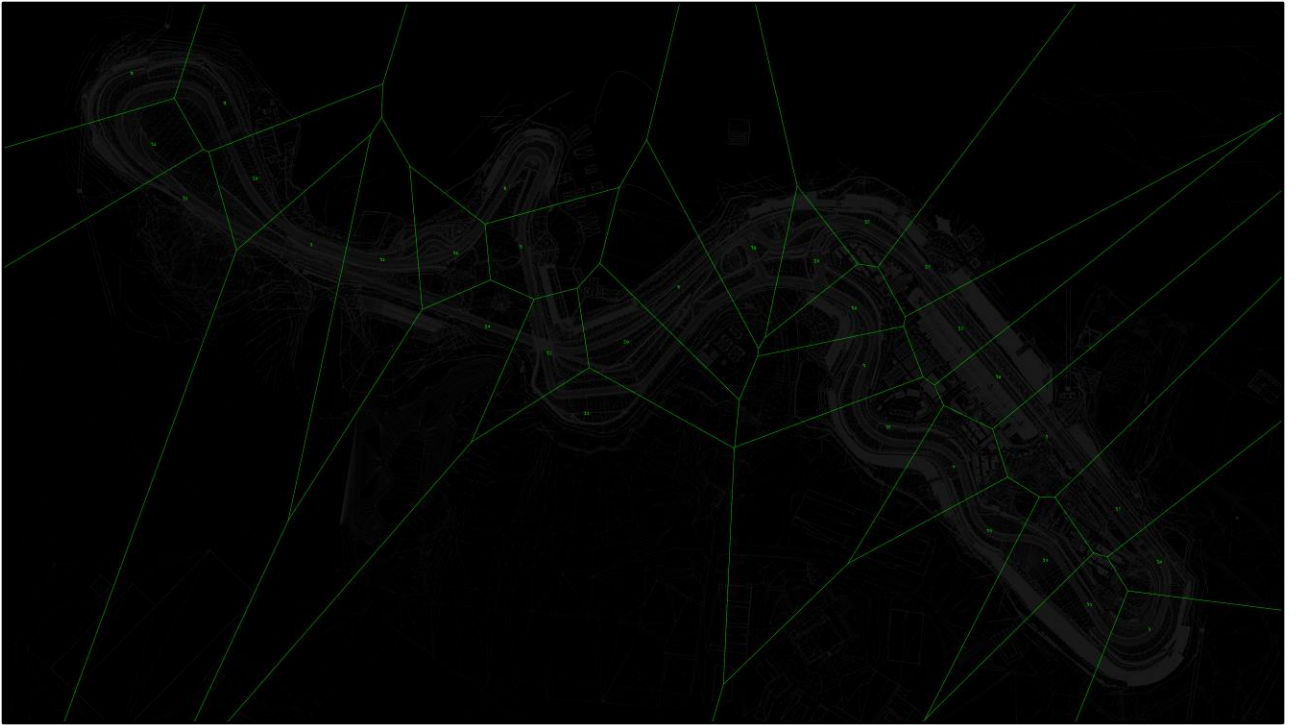
ボロノイ領域とは

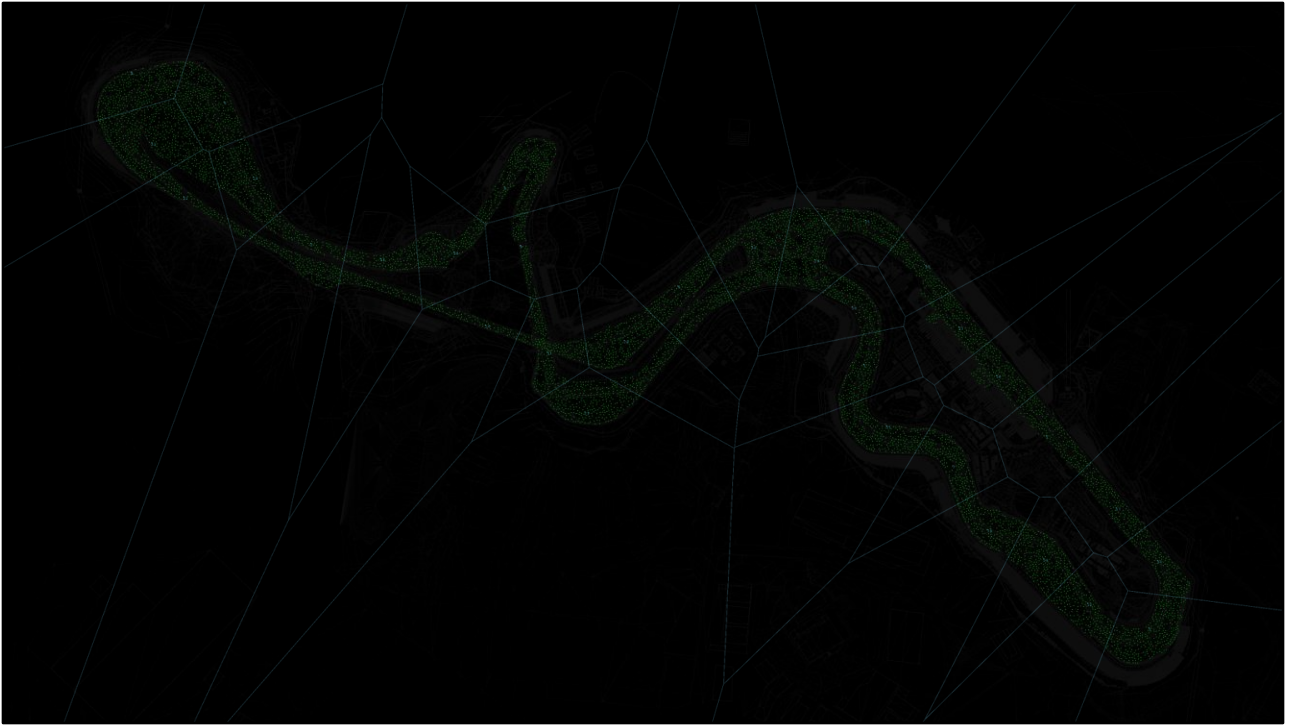
- それぞれの代表点を「母点」と呼ぶ

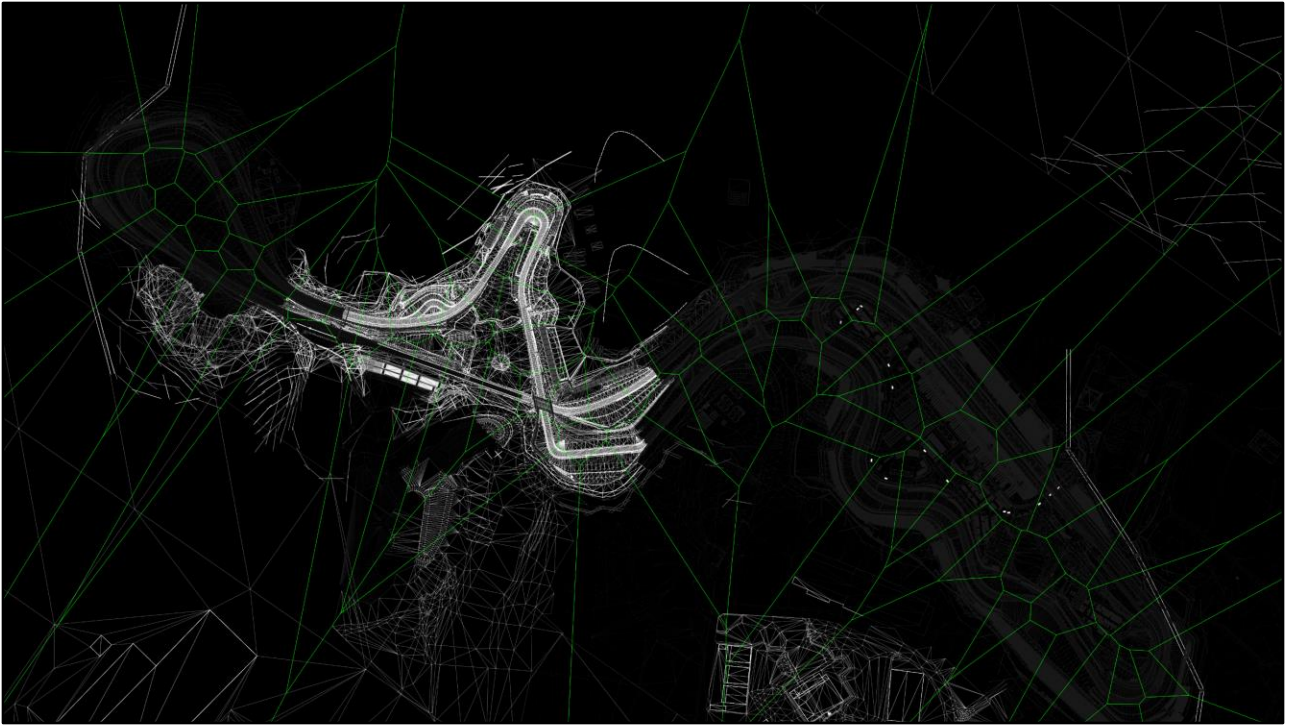


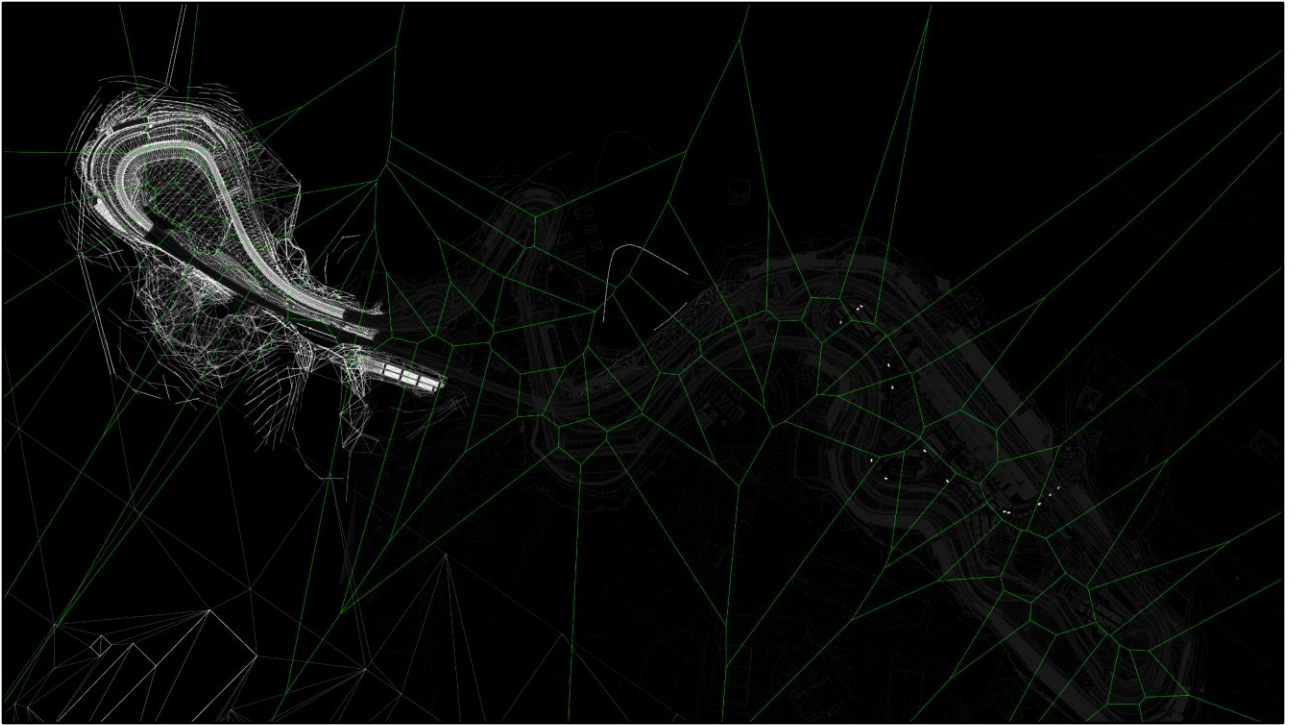
POLYPHONY™
DIGITAL

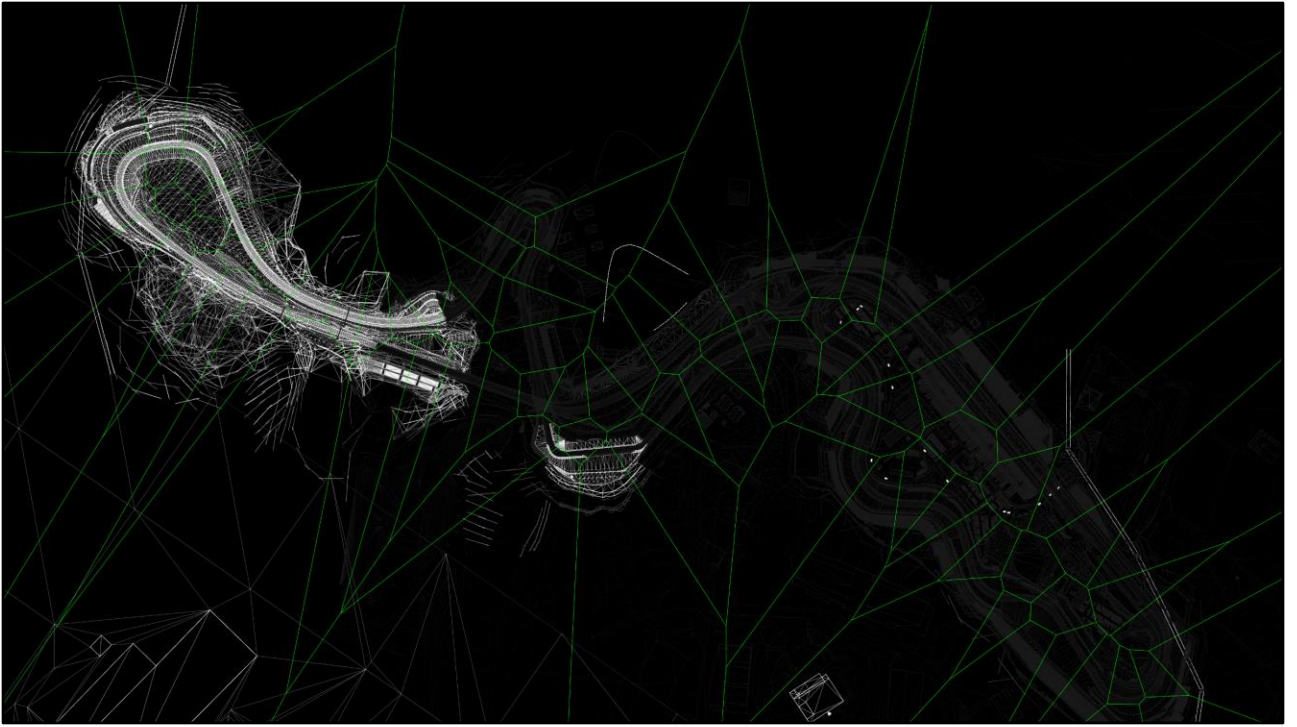
Mysid (SVG), Cyp (original) - Manually vectorized in Inkscape by Mysid, based on Image:Coloured Voronoi 2D.png,
CC 表示-継承 3.0, <https://commons.wikimedia.org/w/index.php?curid=4290269>による











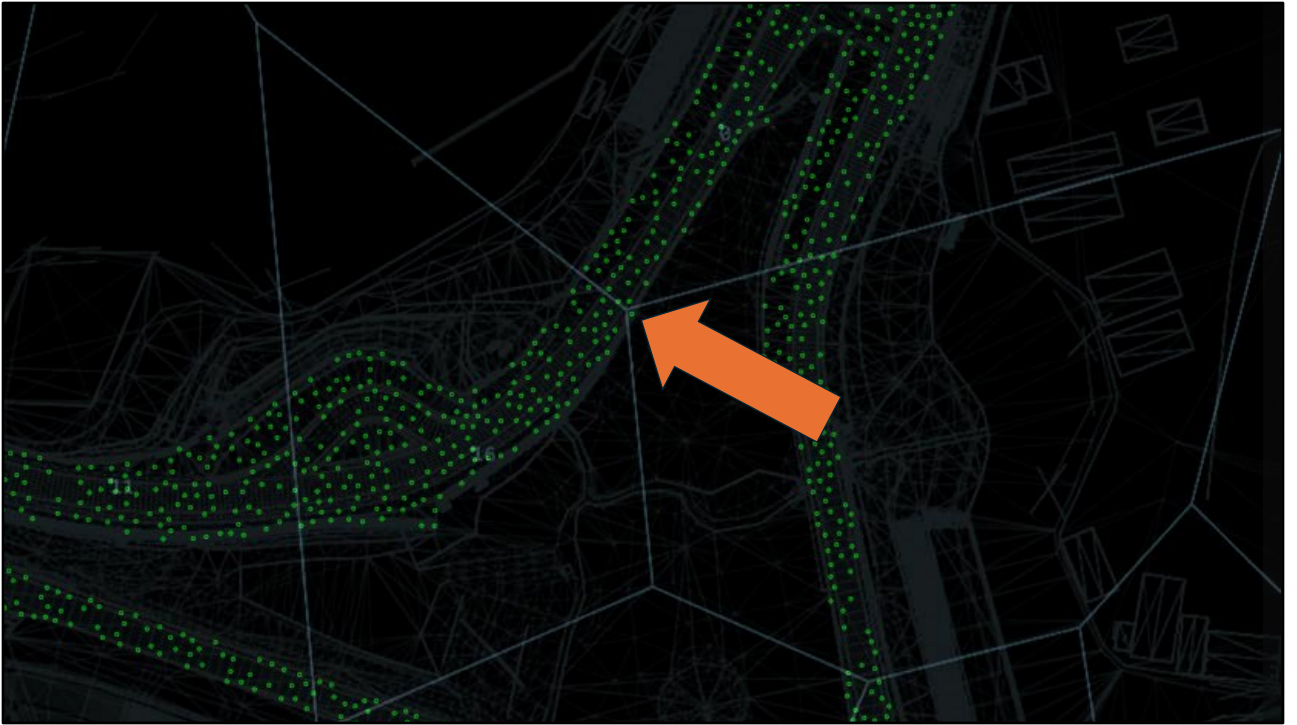


クラスタ化の目的

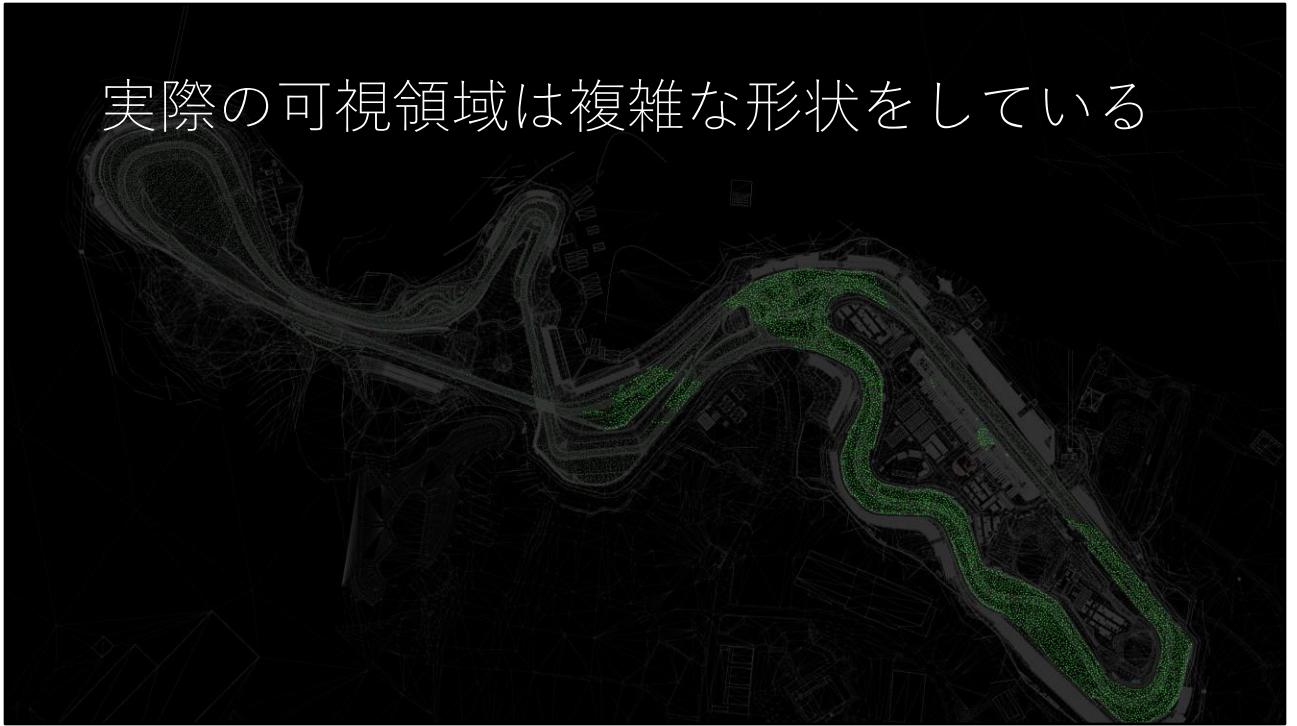
- データサイズの削減
- カメラ位置とビジョンポイントの位置の差による誤差の低減

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。

コース一周のオブジェクト数変動

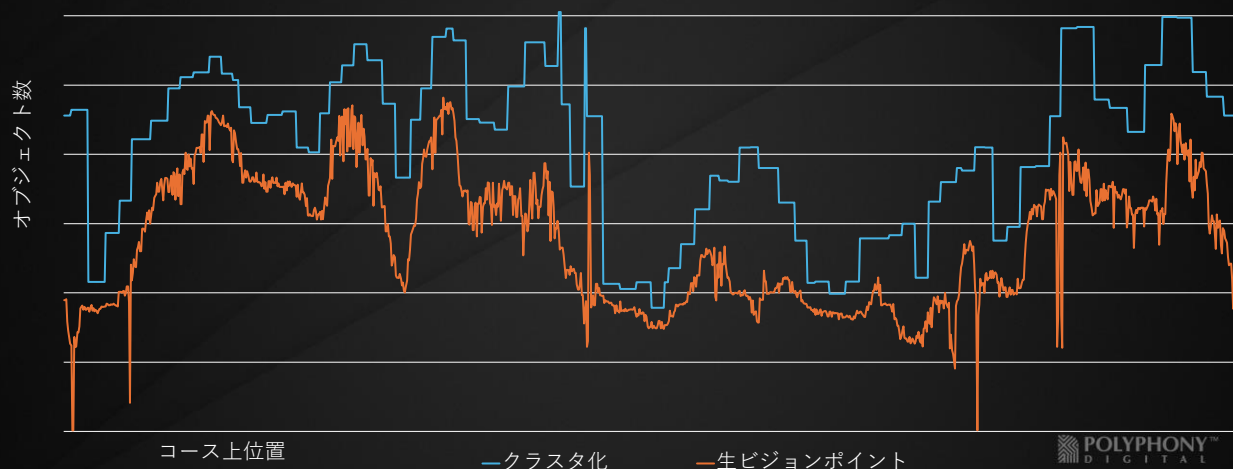


まず、計算したままクラスタリングしていないビジョンポイントを用いて、コースを一周するカメラから、見えるオブジェクトの数をプロットしてみました。

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。

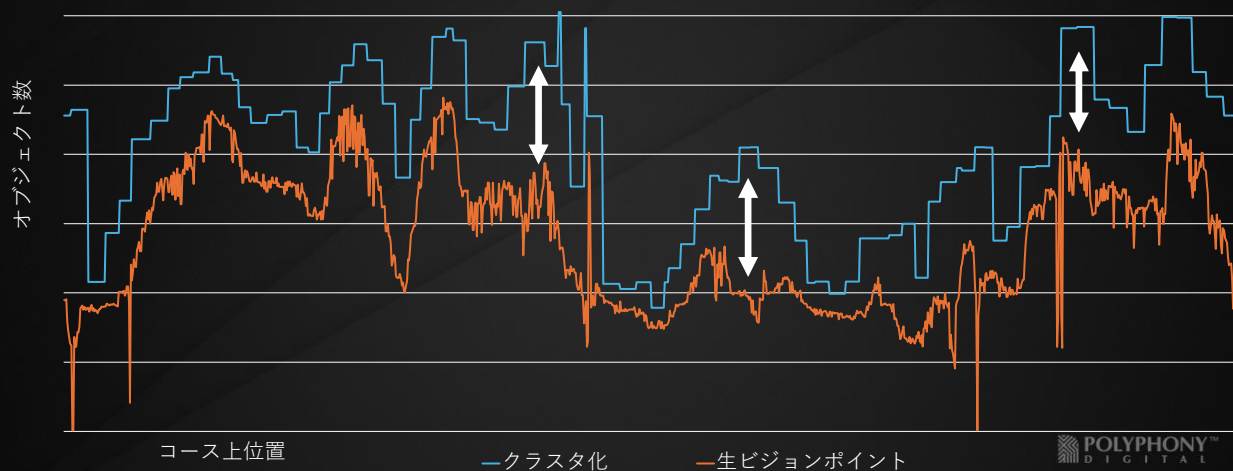
コース一周のオブジェクト数変動



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プング等を防ぎつつ、
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。

コース一周のオブジェクト数変動



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

- 事前計算キャッシ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

ニューラル可視化フィールド



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。

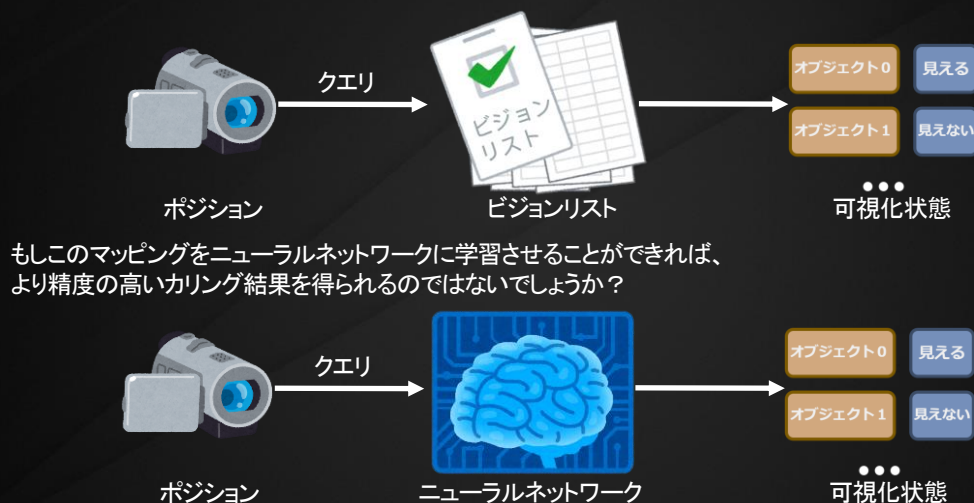
POLYPHONY™
DIGITAL

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。

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。



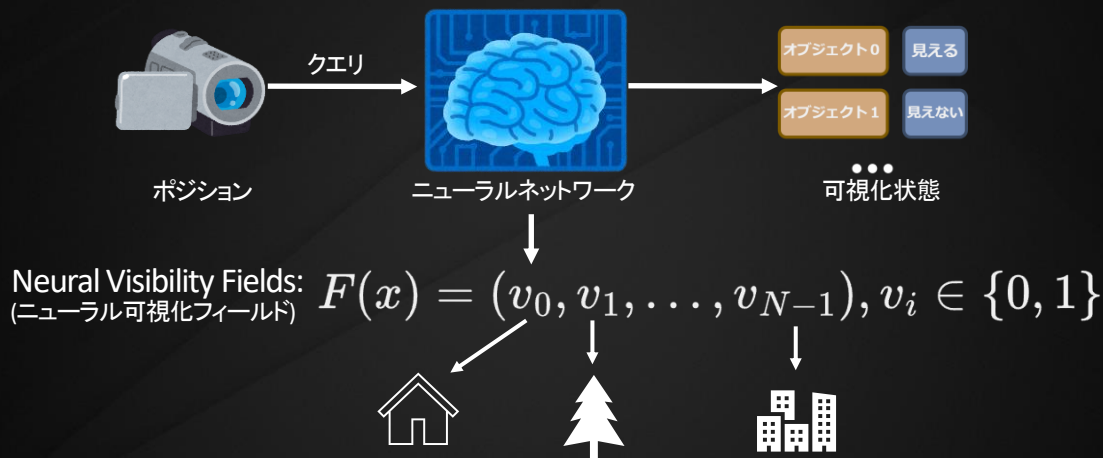
観察を通じて

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。

ニューラル可視化フィールド

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)¹の形に似ている。



[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>

POLYPHONY™
DIGITAL

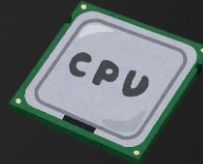
ポジションと各オブジェクトの可視化状態
のマッピングは、NeRF (Neural Radiance
Fields) の形に似ています。

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。

実装の概要



必要なツール



SSE

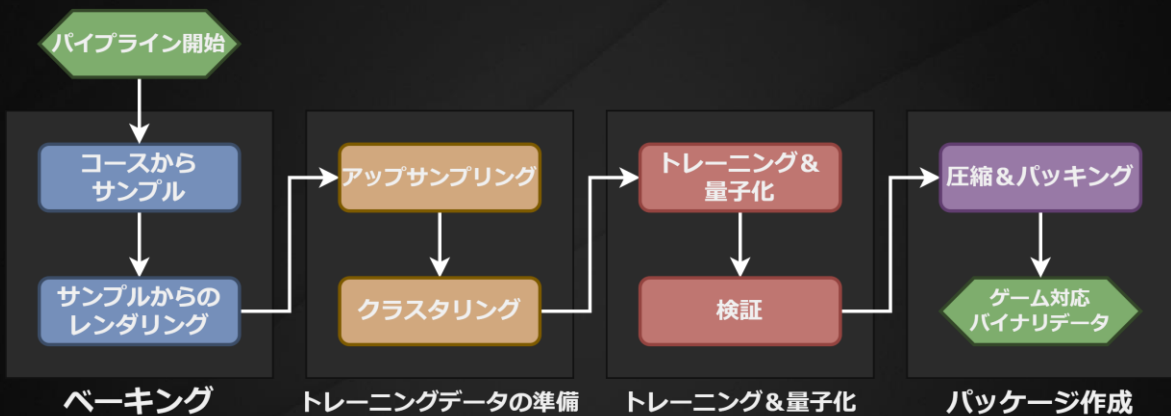
POLYPHONY™
DIGITAL

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。

オフライン処理



POLYPHONY™
DIGITAL

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
ベーキング、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
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。

ベーキングの元データ

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。
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しきい値を超える場合、**可視オブジェクトリスト**に追加する。
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。



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。

トレーニングデータの準備



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

Upsampling



Clustering

トレーニングデータの準備は、**ビジョンリスト**の入力し、密度が高いバイナリデータ形式の位置、重要度、可視情報を入力する。

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。

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。

POLYPHONY™
DIGITAL

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、可視情報を入力します。
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。

アップサンプリング

Upsampling

Clustering

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。

ビジョンポイント



アップサンプリング



重要度高い

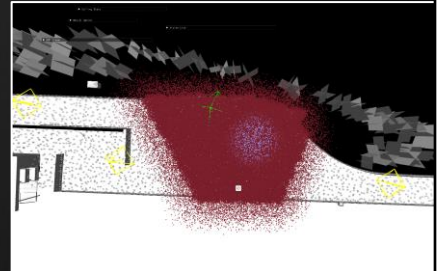


重要度低い

KNN

ビジョンリスト 重要度高い

ビジョンリスト 重要度低い



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。

アップサンプリング

Upsampling

Clustering

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。

ビジョンポイント



アップサンプリング



重要度高い

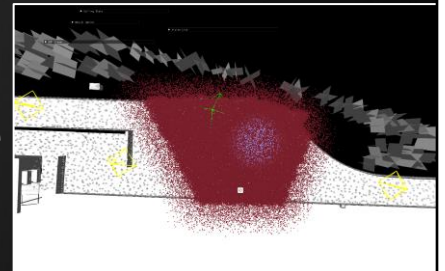


重要度低い

KNN

ビジョンリスト 重要度高い

ビジョンリスト 重要度低い



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。

アップサンプリング

Upsampling

Clustering

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にして、**重要度の低いビジョンポイント**を取得する。

ビジョンポイント



アップサンプリング



重要度高い

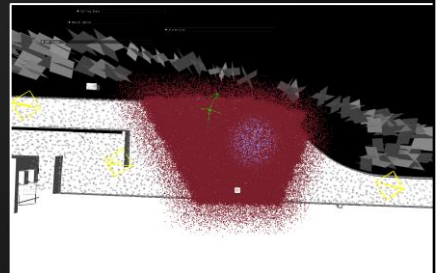


重要度低い

KNN

ビジョンリスト 重要度高い

ビジョンリスト 重要度低い



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。

クラスタリング

Upsampling

Clustering

1. 重要なビジョンポイントに対して
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。

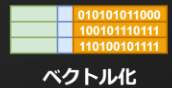
ビジョンポイント



クラスタリング



ベクトル化して
出力



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。

クラスタリング

Upsampling

Clustering

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。



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。

クラスタリング

Upsampling

Clustering

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。



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。

クラスタリング

Upsampling

Clustering

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。

ビジョンポイント



クラスタリング



ベクトル化して
出力

0	1	0	1	0	1	1	0	0	0
1	0	0	1	1	1	0	1	1	1
1	1	0	0	1	0	1	1	1	1

ベクトル化

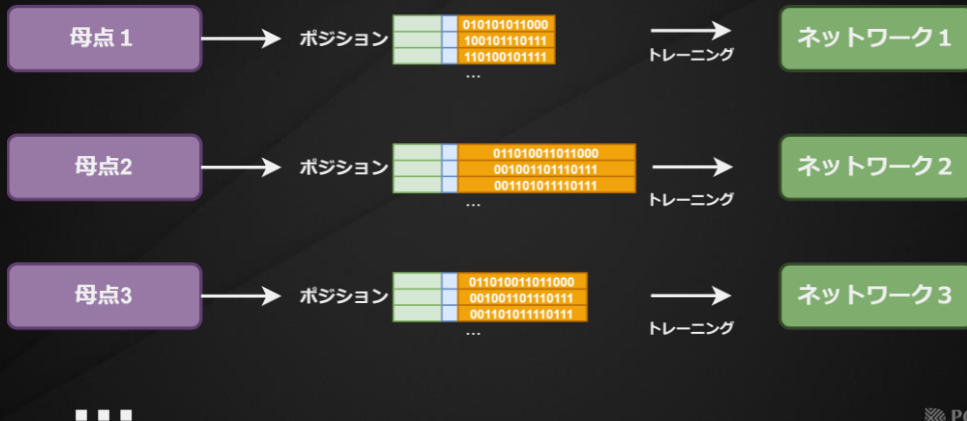


最後に4つ目として、PyTorchで読み取れるように、各母点内のビジョンポイントに関して、位置、重要度、そして0と1で表現されたビジョンリストをベクトルとして出力します。

ここでは、直接バイナリデータとして出力することになります。

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。



POLYPHONY™
DIGITAL

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。

トレーニング & 量子化

 POLYPHONY™
DIGITAL

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

トレーニング &
量子化



検証

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。

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。

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。

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。

POLYPHONY™
DIGITAL

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。

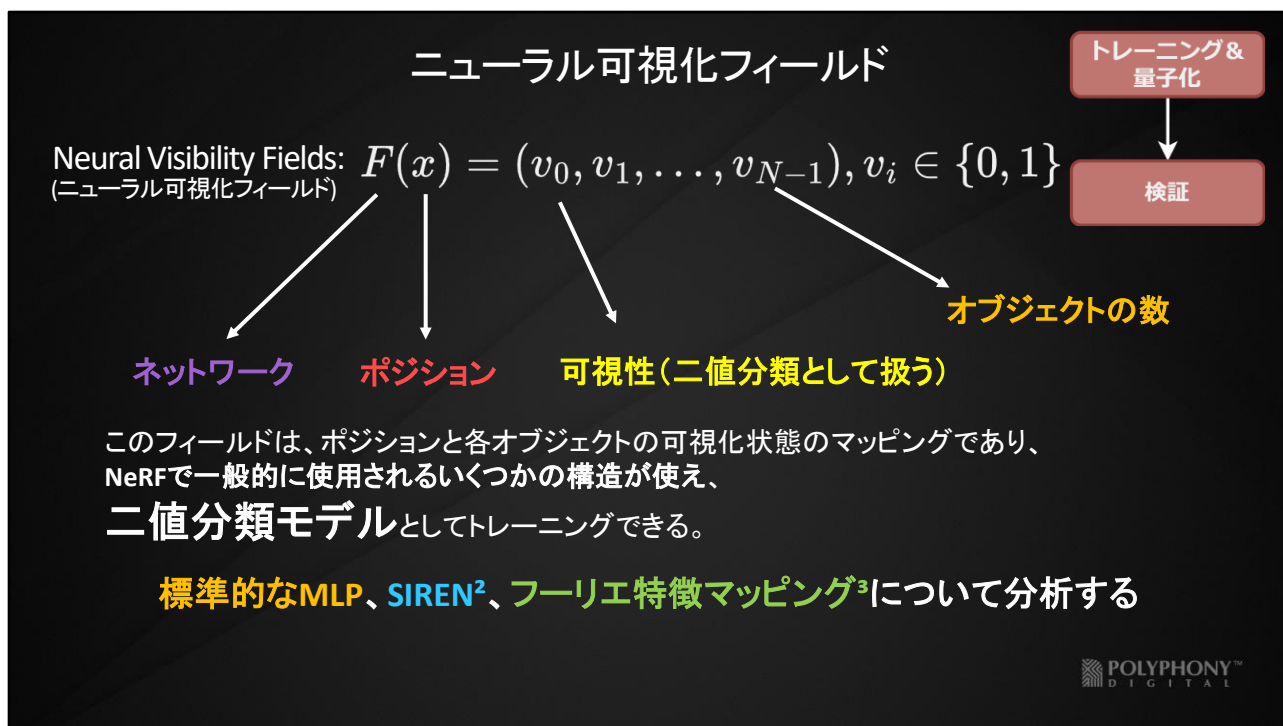
次に、2つ目は「ネットワークパラメータの探索」です。
ここでは、検索を通じてネットワークの幅や深さなど、最適なパラメータを見つけます。

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

 POLYPHONY™
DIGITAL



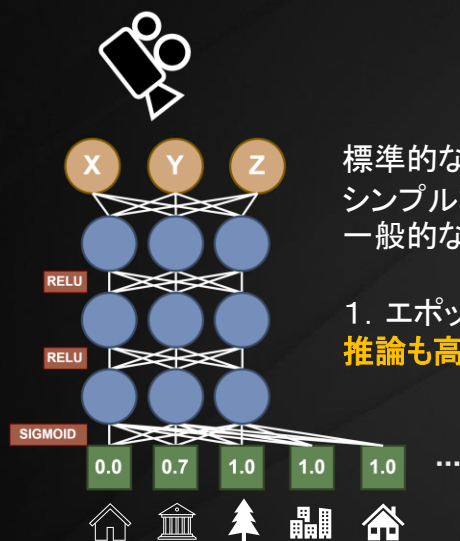
先ほど紹介されたNeural可視化フィールドは具体的に次のようになります。

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。

標準的なMLP(Vanilla MLP)



トレーニング&
量子化

検証

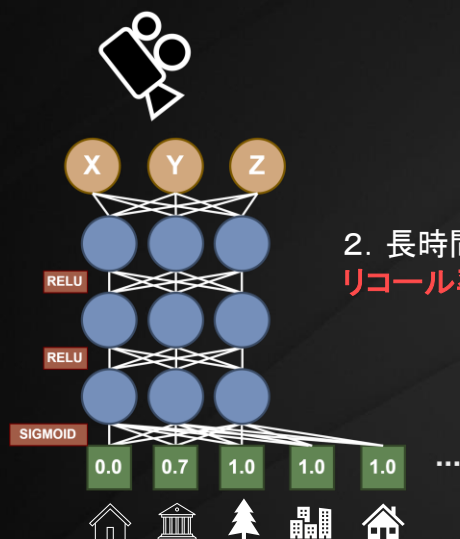
POLYPHONY™
DIGITAL

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。

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。



トレーニング &
量子化

検証

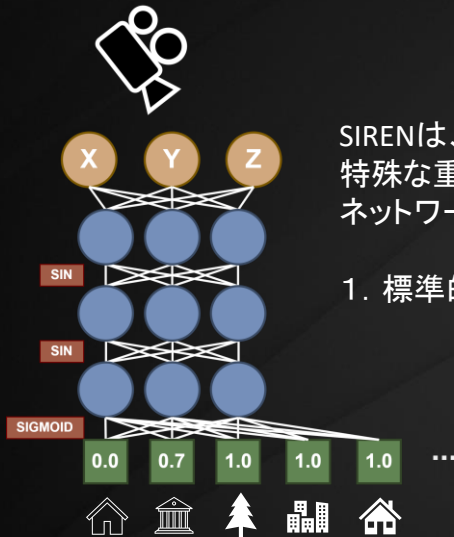
POLYPHONY™
DIGITAL

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。

SIREN²



SIREN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**。

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

POLYPHONY™
DIGITAL

トレーニング&
量子化

検証

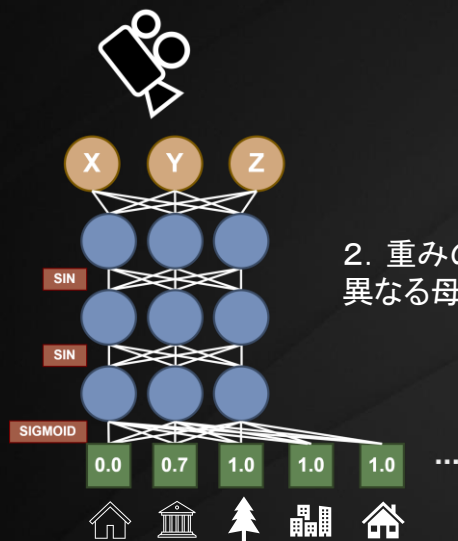
次に、SIRENについて話します。

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。

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

POLYPHONY™
DIGITAL

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。

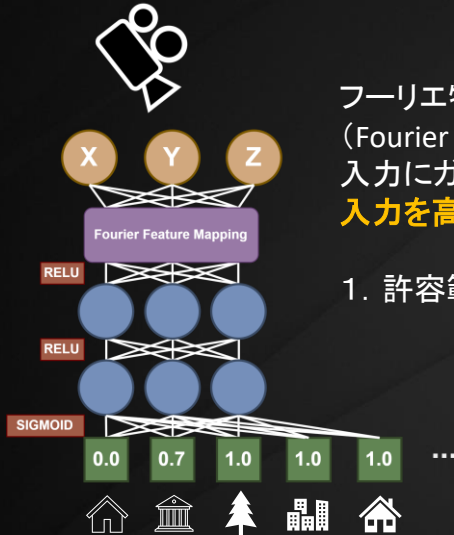


そして、3つ目に、収束が難しく、同等の品質を求めると学習速度が十分ではないです。

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。

フーリエ特徴マッピング³



フーリエ特徴マッピング

(Fourier Feature Mapping) 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**。

トレーニング &
量子化

検証

[3]: 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

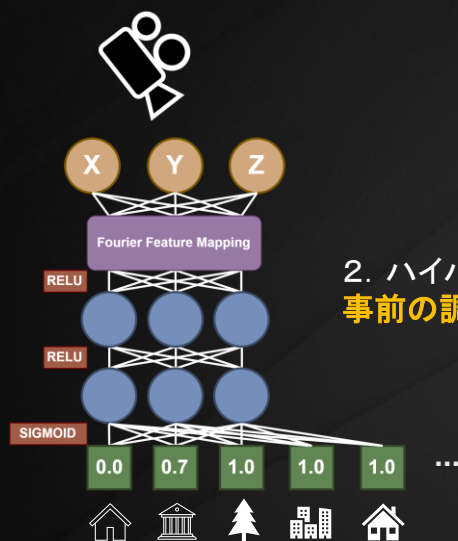
最後に、フーリエ特徴マッピングについて話します。

フーリエ特徴マッピング (Fourier Feature Mapping) は、入り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。

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。

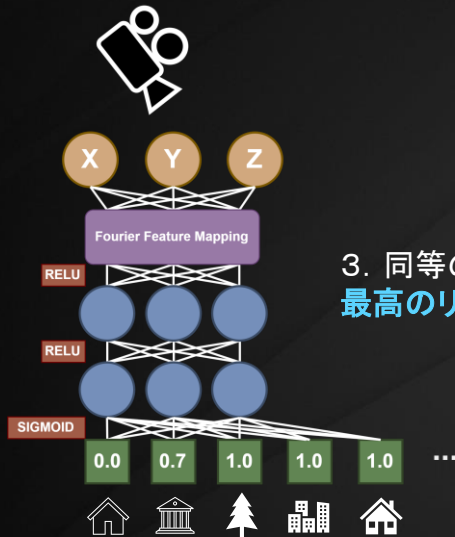
トレーニング &
量子化

検証

POLYPHONY™
DIGITAL

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。

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

POLYPHONY™
DIGITAL

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（いっつ）です。



フーリエ特徴マッピング

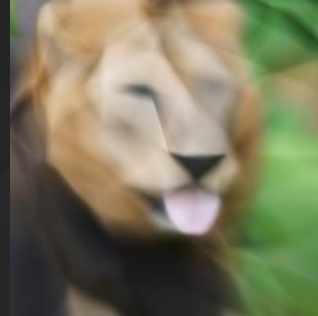
トレーニング &
量子化



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

POLYPHONY™
DIGITAL

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。

フーリエ特徴マッピング

Fourier Features Mapping

入力ベクトル

全結合層
(バイアスがない)

Sin

Cos

結合

隠れ層

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。

トレーニング & 量子化

検証

POLYPHONY™
DIGITAL

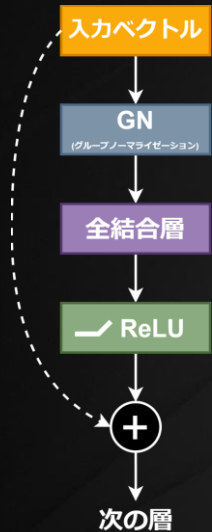
では、フーリエ特徴マッピングは具体的にどのように実装されているのでしょうか？

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。

フーリエ特徴マッピング



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。

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。

トレーニング &
量子化

検証

POLYPHONY™
DIGITAL

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わかりました。

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。

損失関数

トレーニング &
量子化

検証

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。

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

[BCEWithLogitsLoss — PyTorch 2.5 documentation](#)

POLYPHONY™
DIGITAL

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。

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、各母点の可視性フィールドの複雑さは異なる。
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。



トレーニング &
量子化



検証

POLYPHONY™
DIGITAL

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。

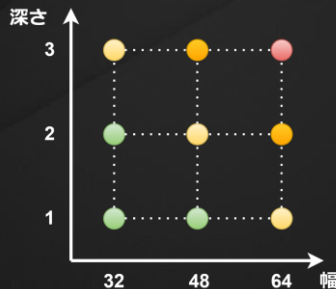
ネットワークパラメータの探索

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

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。

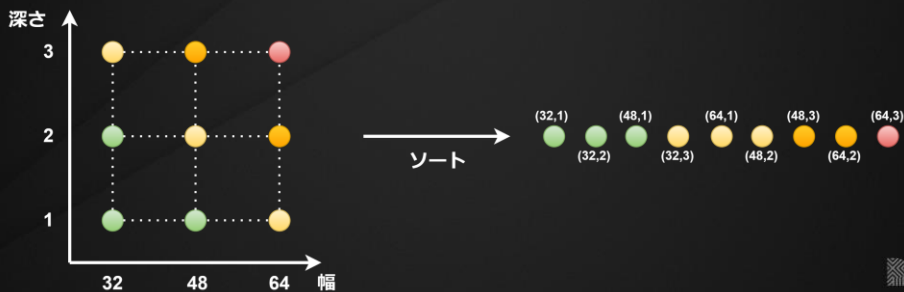
ネットワークパラメータの探索

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。



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。

ネットワークパラメータの探索

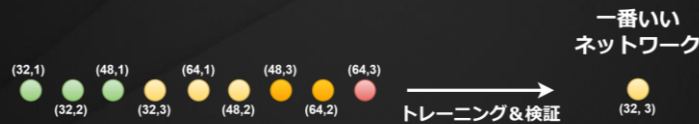
トレーニング &
量子化



検証

3. 早期終了

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。



POLYPHONY™
DIGITAL

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

 POLYPHONY™
DIGITAL

量子化

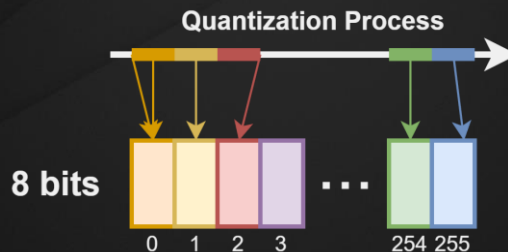
トレーニング &
量子化



検証

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

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。

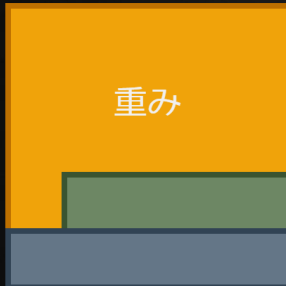
量子化

トレーニング&
量子化



検証

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。

POLYPHONY™
DIGITAL

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値(あたい)に戻す必要があつて、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。

量子化

トレーニング&
量子化

検証

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。

POLYPHONY™
DIGITAL

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。

量子化

トレーニング &
量子化

検証



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) という手法を使用する。

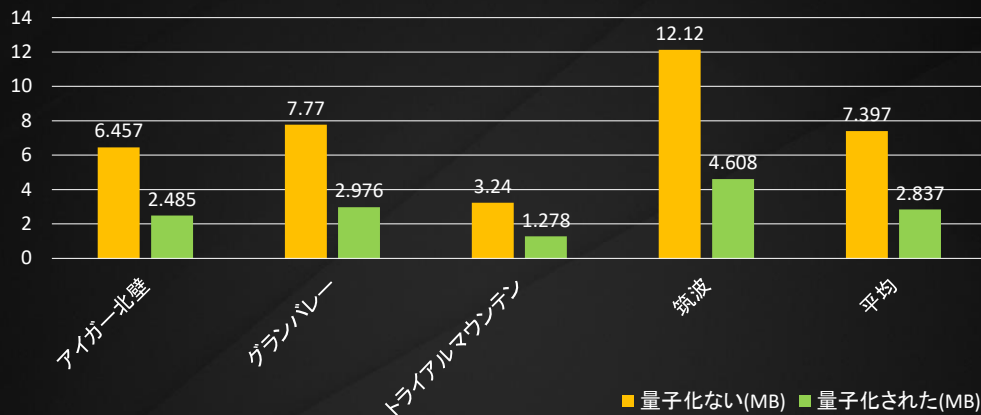
POLYPHONY™
DIGITAL

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



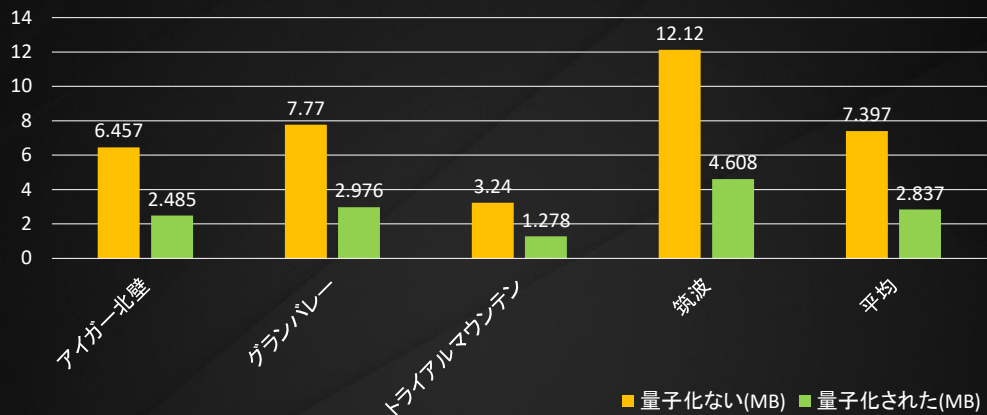
POLYPHONY™
DIGITAL

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。

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

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。



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。

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。

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(エフツー)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。

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
維持できるよういくつかの工夫を紹介する。

トレーニング &
量子化



検証

POLYPHONY™
DIGITAL

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。

検証:再現率の確保

トレーニング &
量子化



検証

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。

POLYPHONY™
DIGITAL

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。

検証:再現率の確保

トレーニング&
量子化

検証

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。



POLYPHONY™
DIGITAL

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。

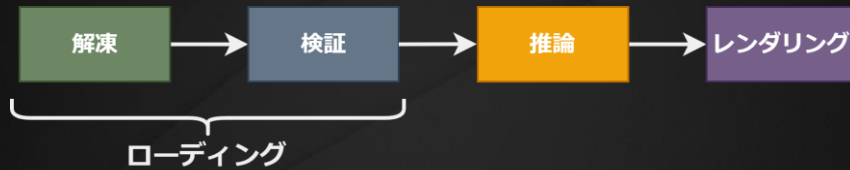
各ネットワークに対して、

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

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。

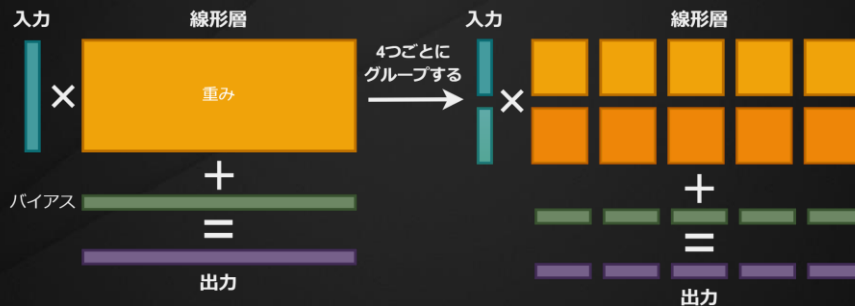
推論

 POLYPHONY™
DIGITAL

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。



POLYPHONY™
DIGITAL

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。

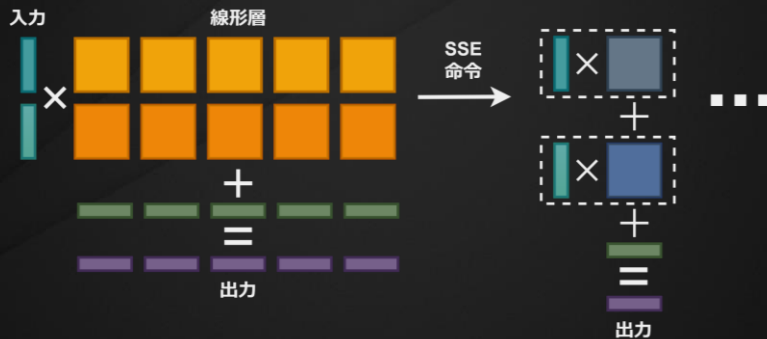
1つ目は「グループ化されたベクトル」です。SSE命令を活用するために、ベクトルは4つのスカラーでグループして、マトリックスは4x4のサイズでグループ化しています。

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。



POLYPHONY™
DIGITAL

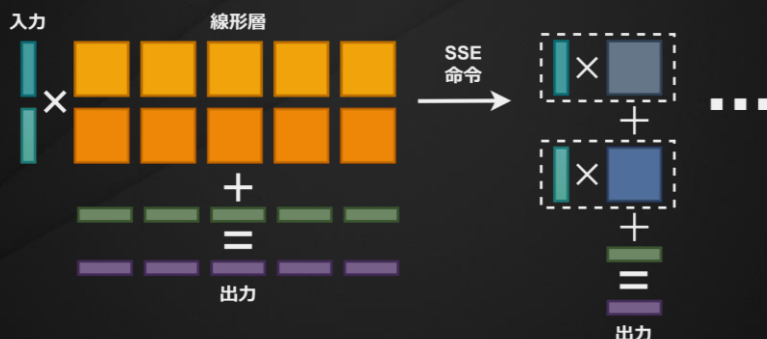
グループされた後、
線形層を4つのベクトルと4x4のマトリックスの積で構成される演算子に簡単に変換できます。

変換をされた形式は一般的なCGで使わ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。

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。



POLYPHONY™
DIGITAL

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。

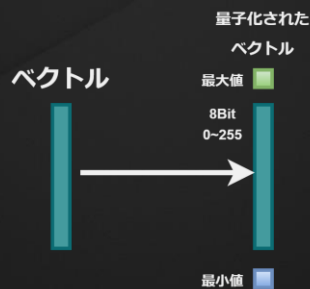
推論(量子化あり)

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。

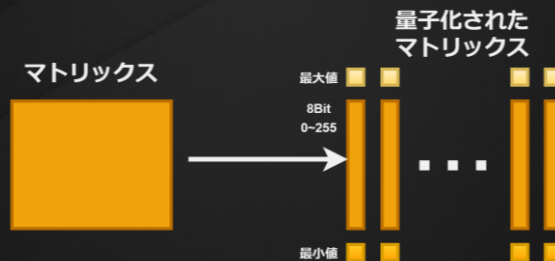
推論(量子化あり)

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。

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

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



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。

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



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。

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。

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

POLYPHONY™
DIGITAL

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。

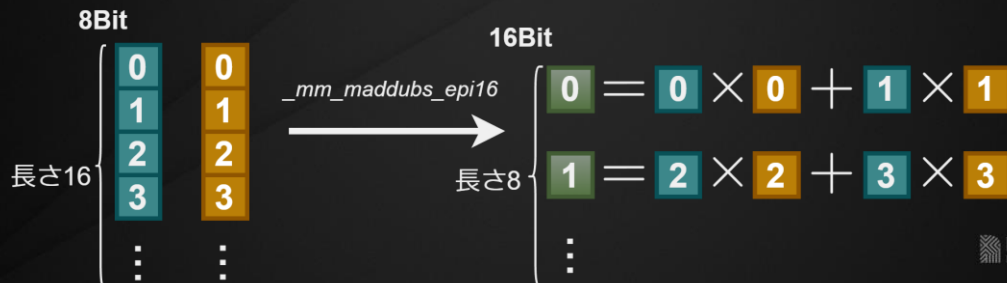
整数ベクトルのドット積

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。



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。

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。



POLYPHONY™
DIGITAL

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。

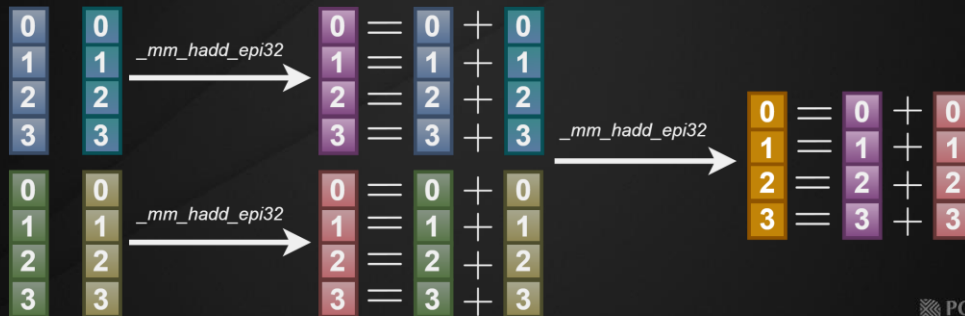
最後に、残る4つの32ビット整数の和を求める必要があります。命令の使用をさらに削減するために、より効率的な方法が必要です。

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。**



POLYPHONY™
DIGITAL

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。

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

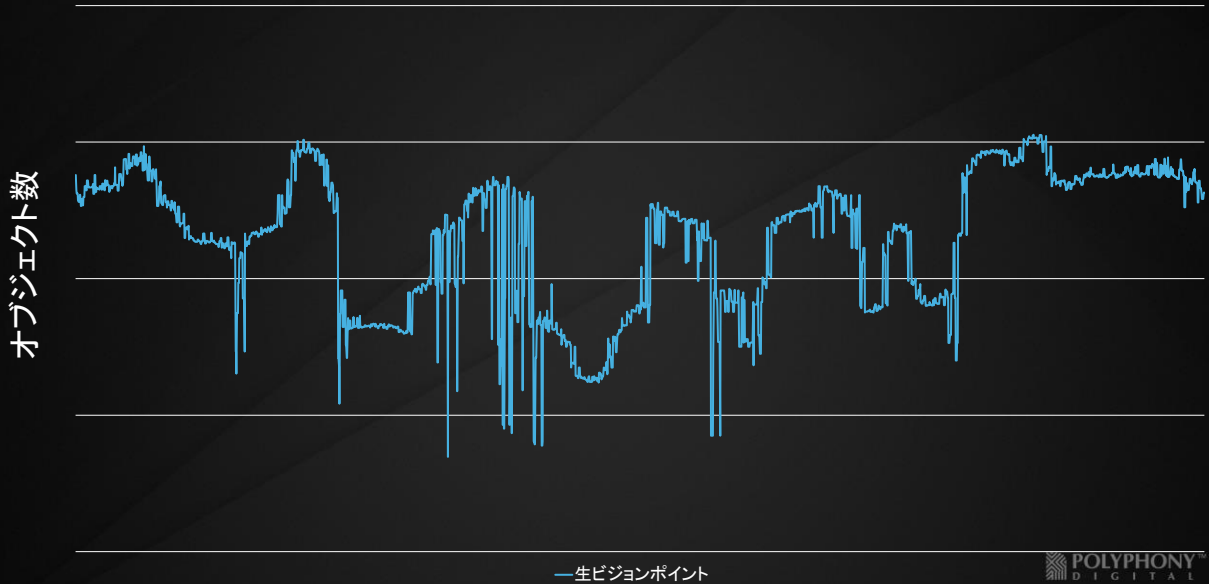
アイガーは、**遮蔽物が少ない広大なコース**である。
ニューラル可視性フィールドを使用することで、コース内のランプや丘陵などの
潜在的な遮蔽物を活用し、レンダリングの負荷をさらに軽減できる。



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。

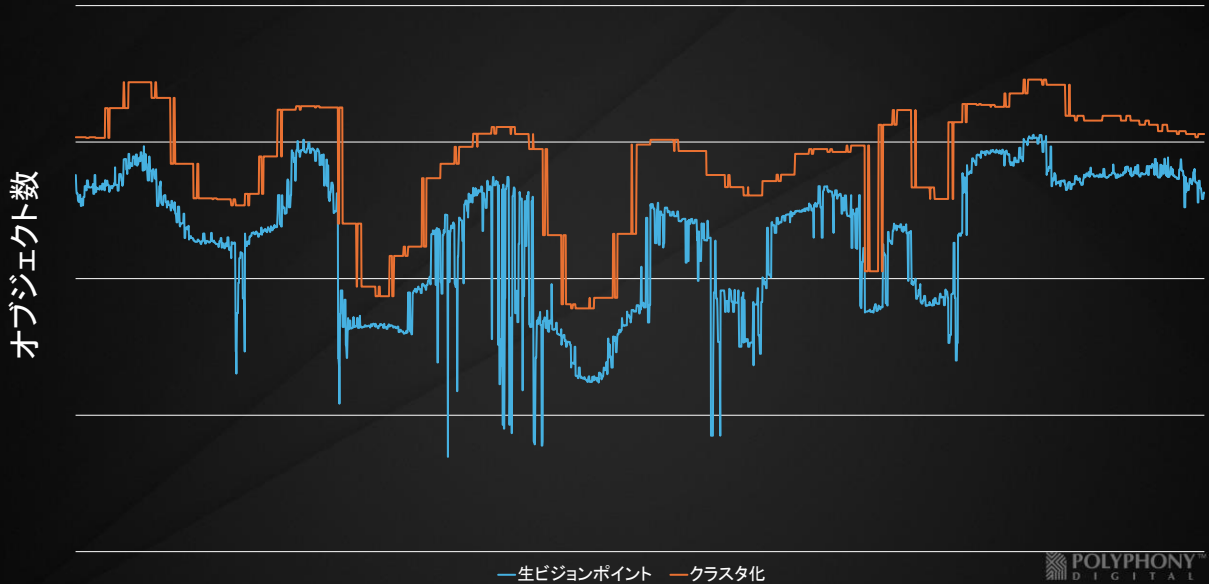
このコースでの全体的な改善は大きいです。

結果:アイガー北壁 (PS5®)



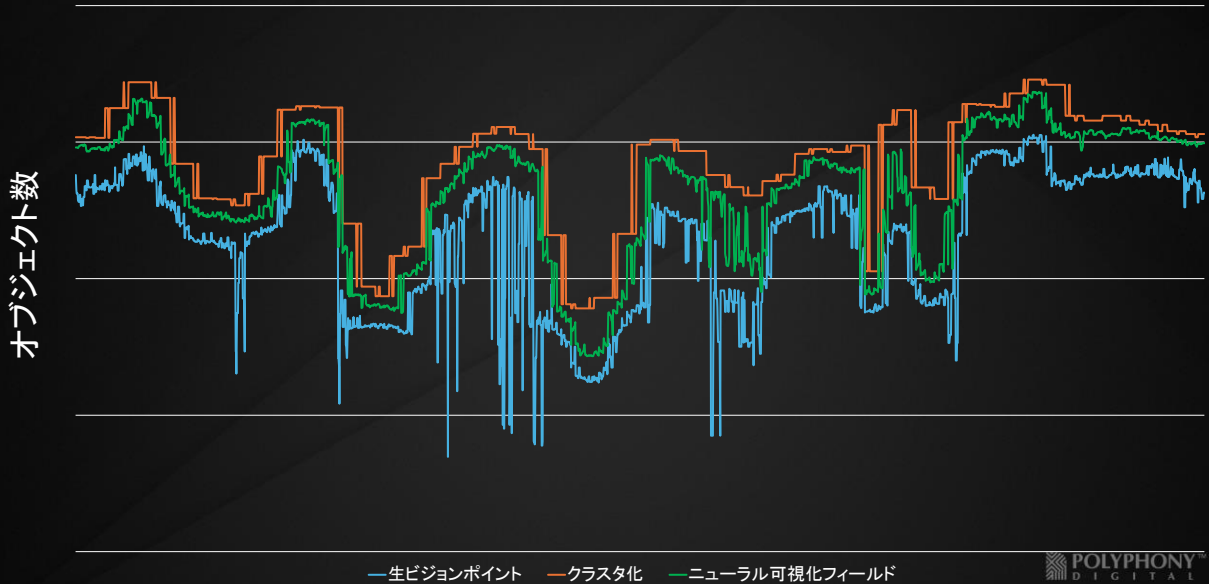
アイガーのコース一周のオブジェクト数変動となります。
青い線は生のビジョンポイントの結果です。

結果:アイガー北壁 (PS5®)



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。

結果:アイガー北壁 (PS5®)



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®)

 POLYPHONY™
DIGITAL

次にこの手法の結果について説明します。



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。

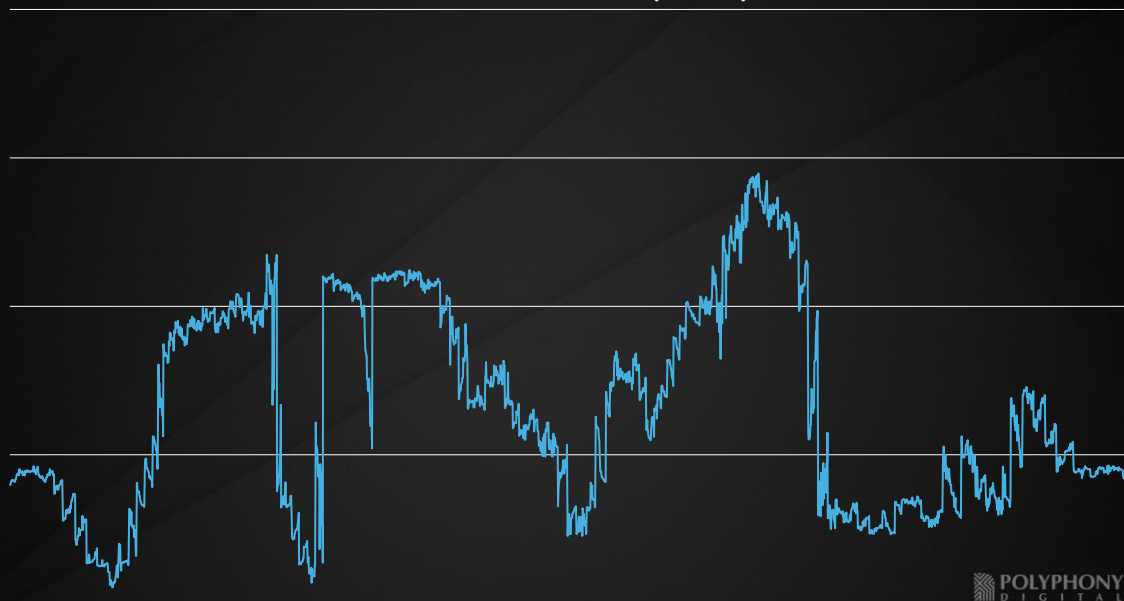
POLYPHONY™
DIGITAL

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。

結果:グランバレー(PS5®)

オブジェクト数



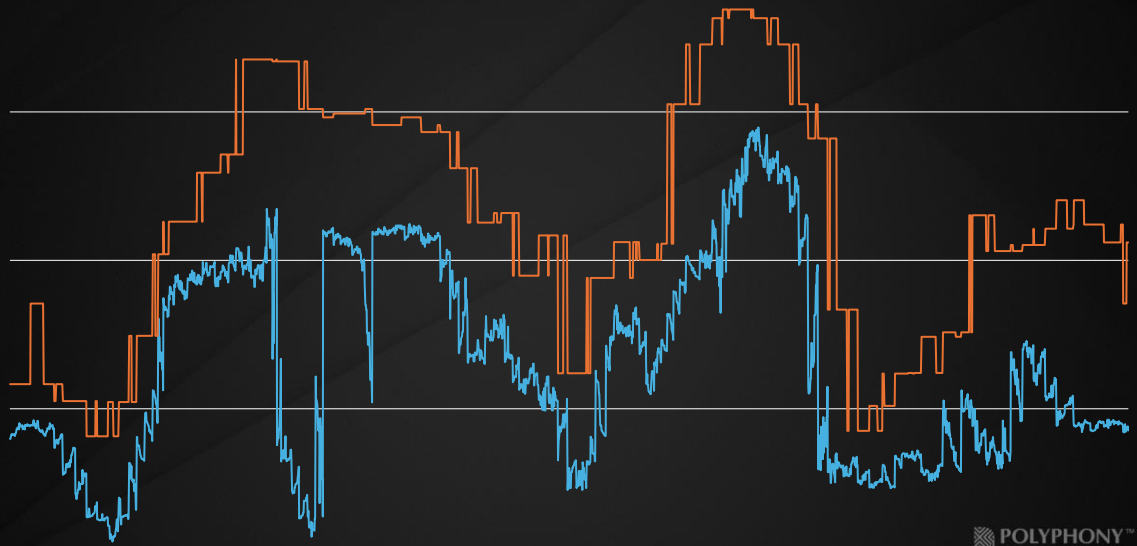
—生ビジョンポイント

POLYPHONY™
DIGITAL

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

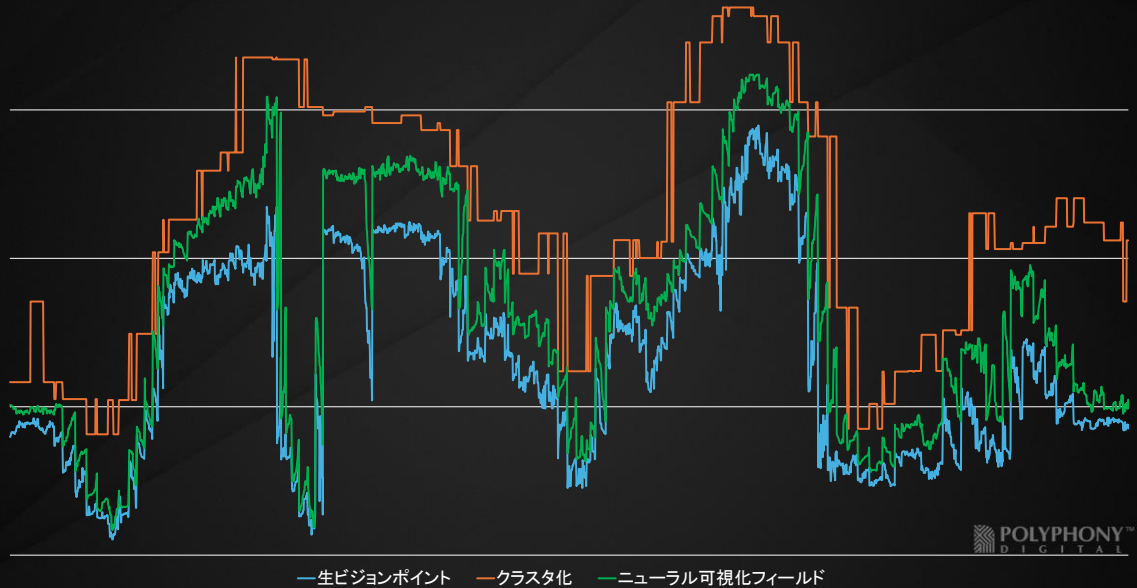


—生ビジョンポイント —クラスタ化

POLYPHONY™
DIGITAL

結果:グランバレー(PS5®)

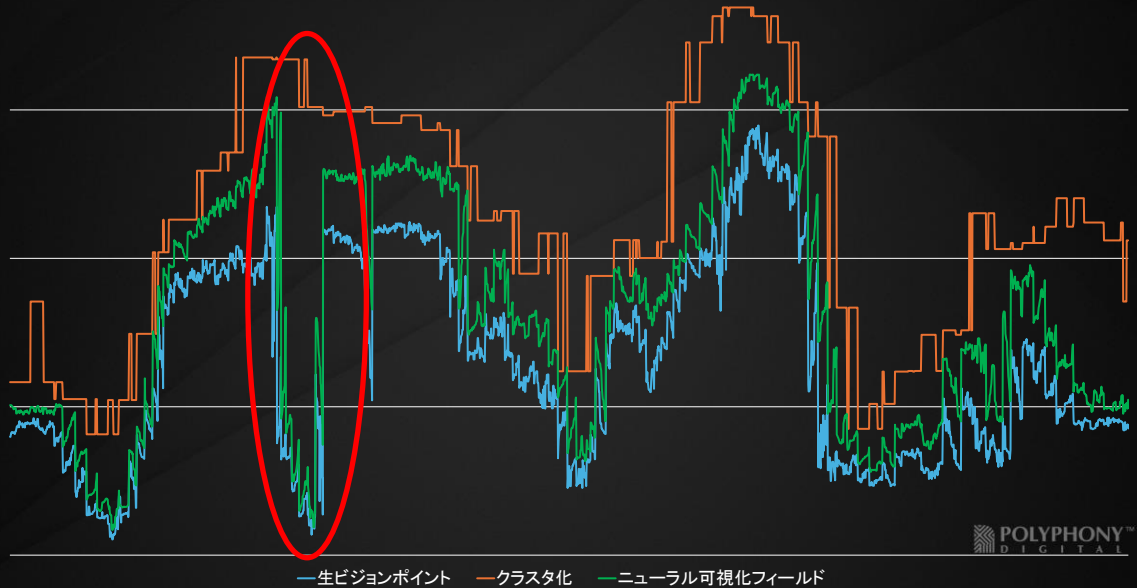
オブジェクト数



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。

結果:グランバレー(PS5®)

オブジェクト数



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。

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する。



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。

 POLYPHONY™
DIGITAL

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。