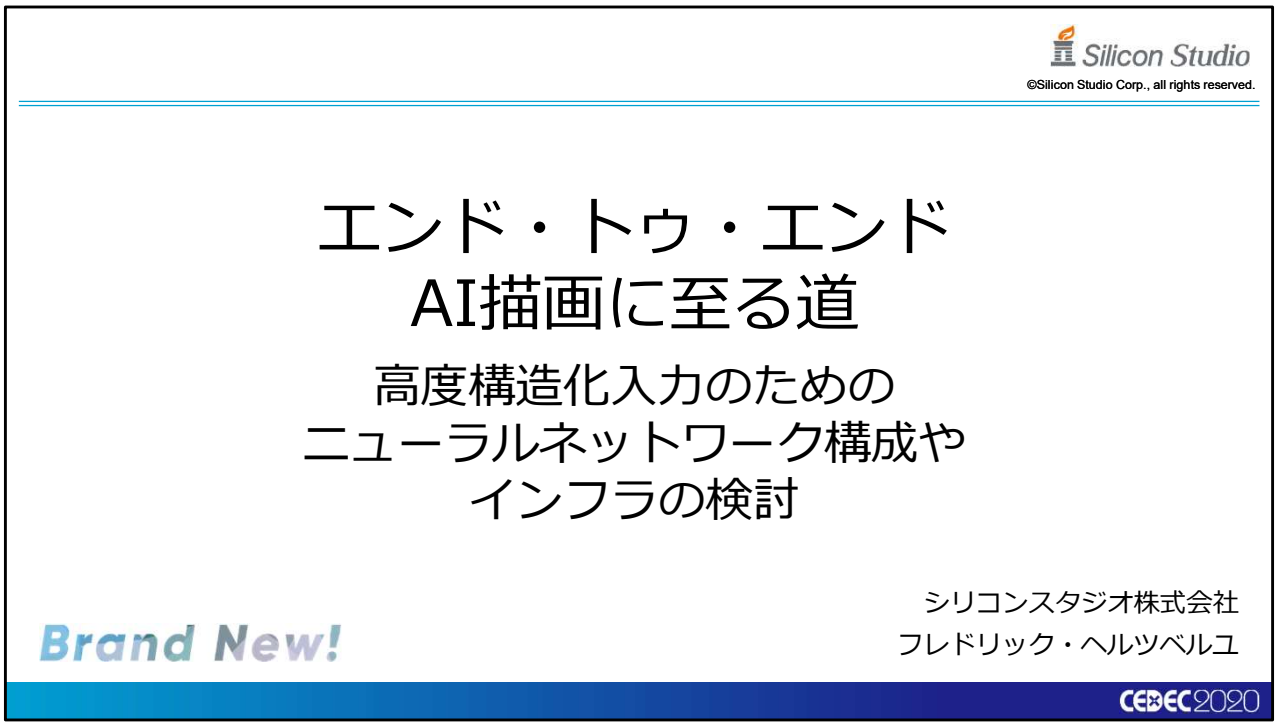


シリコンスタジオ株式会社のフレドリック・ヘルツベルユです。
「エンド・トゥ・エンドAI描画に至る道」と題して講演させていただきます。
よろしくお願いします。

講演内容

Silicon Studio

1. 前置き
2. 背景
3. 既存の神経回路網の非効率さについて
4. リアルタイム描画での条件による効率化の機会
5. 非効率さを解決するアーキテクチャの提案
6. 重要な論文のアーキテクチャの紹介
7. 今後の予定

本セッションは以下のような内容になっています。

講演内容

Silicon Studio

1. 前置き
    1. 自己紹介
    2. このセッションについて
2. 背景
3. 既存の神経回路網の非効率さについて
4. リアルタイム描画での条件による効率化の機会
5. 非効率さを解決するアーキテクチャの提案
6. 重要な論文のアーキテクチャの紹介
7. 今後の予定

CEDEC2020

まず、自己紹介とこのセッションの前提について話させていただきます。

1-1. 自己紹介

Silicon Studio

- フレドリック・オット・マックス・フォルケ・ヘルツベルユ
- スウェーデン出身
- シリコンスタジオ株式会社
  - 入社6年目、研究開発室所属
- 前職：北京红剑公司，量子化学のGPGPU開発（北京）
  Ericsson（ストックホルム）

CEDEC2020

はじめまして、私はフレドリック・オット・マックス・フォルケ・ヘルツベルユと申します。
スウェーデン人です。
シリコンスタジオに入って6年目で、その前は北京で量子化学のGPGPU開発をしていました。
その時中国語の発表も一度したことがあるので、本日も日本語の講演頑張ります。
質問は日本語でも英語でも受け付けます。
よろしくお願いいたします。

1-2. このセッションについて

Silicon Studio

- エンド・トゥ・エンドAI描画に至る道
  - 神経回路網でのリアルタイム描画を目標とする
- リアルタイムを前提として、手法について議論したい
  - デファクトスタンダードはまだ定まっていない
  - アーキテクチャの選択ができる貴重な機会
- 第一原理計算のアプローチで
  - 基本の命題から出発して、正しいアプローチであることを確認したい
  - 結果論ではない

CEDEC2020

このセッションは「エンド・トゥ・エンドAI描画に至る道」というタイトルです。
現在、物理ベースレンダリングなど計算によって行われている、CGのリアルタイム描画処理を
AIで直接描画するように置き換えたい、という話です。
AIと言っても色々ありますが、本セッションではニューラルネットワークのことをさします。

今のところ、リアルタイムAI描画のLegacy Systemはまだありません。
まだまだ、アーキテクチャを自分たちで選択することができる状況です。
普通じゃないでしょう、一般的には既存のデファクトスタンダードに従うしかありません。
これはめったにない貴重な機会です。
業界としてこの機会を無駄にしないように、本セッションで皆さんと一緒に考えてみたいと思っています。
よろしくお願いします。

前提は重要なことですね。
最初に選んだ前提によって最終的な結論が全く変わります。

絶対にリアルタイムで描画したいと決まったら、
どんなアーキテクチャが最適に成るのかと言うことについて議論したいです。

AIによる画像処理の研究では、
結果のスクリーンショットを比較して議論することが多く、
それも面白いのですが、今日はそうではありません。
第一原理計算とは、量子化学でよく使われる概念です。
測定結果に依るのでなく、本当に基本から、基本の構成だけから分析します。
ラテン語のab initioと同じです。
結果論でなく、基本の命題から出発して、正しいアプローチであることを論理的に確認したいのです。

1-2. 命題

Silicon Studio

- 下記の趣旨を主張したい：

① （現状描画に使われている）
神経回路網の構成にはGPU使用率の非効率がある　…3章

② リアルタイム描画なら非効率をなくせる可能性がある　…4章

③ そのための「ハッシュ分業」アプローチを提案する　…5章
  - タイル内容のハッシュによって神経回路網の重みを選択して読み込む

CEDEC2020

GPUと神経回路網の構成を分析することによって、これら3つの趣旨を主張したいです。

i.　現状の神経回路網の構成のままでは、GPU使用率に非効率があります。

ii. リアルタイム描画に応用する場合、そのような非効率は必要ないはずです。

iii. 「ハッシュ分業」アーキテクチャを提案します。このアプローチによって非効率をなくせます。
「ハッシュ分業」とは、タイルごとにその内容をハッシュ化し、神経回路網の重みを選択して読み込むアプローチです。

講演内容

Silicon Studio

1. 前置き
2. 背景
3. 既存の神経回路網の非効率さについて
4. リアルタイム描画での条件による効率化の機会
5. 非効率さを解決するアーキテクチャの提案
6. 重要な論文のアーキテクチャの紹介
7. 今後の予定

CEDEC2020

あらためて、本セッションの流れを説明します。
まず、背景を簡単に述べた後、先ほどの3つの趣旨についてそれぞれ詳しく説明します。
今回は、基本的なアーキテクチャの提案のみなので、結果をお見せするようなものではありません。
それから、提案手法を今後発展させていく上で、重要になる論文のアーキテクチャを紹介します。
最後に、結論と今後の予定を述べます。

講演内容

Silicon Studio

CEDEC2020

本セッションの目標と、従来手法の問題について、簡単に説明します。

2-1. AIによる人間の顔の描画

Silicon Studio

- 人間は顔については非常に感受性が高い
  - 不自然だとすぐに気づいてしまう
  - 自然な顔なら映像の魅力が大幅にUp
- 顔の正確な表現は難しい
  - 肌の質感、瞳の表現
  - 超高精細なアセットが必要
- 人間の顔の実写映像は大量にある
  - 教師データに使える
- 顔の描画をAIで行うことは非常に有望

CEDEC2020

本セッションでは、AIによってリアルタイムに描画する対象として、人間の顔を取り上げます。

人間は顔については非常に感受性が高く、不自然だとすぐに気づいてしまいます。
逆に、正しく描画できれば、映像の 魅力が大きく増すことになります。

近年、リアルタイム・レイトレーシングをハードウェアで サポートするようになって
鏡面反射などは 正確に計算できるようになりましたが、
肌の表面化散乱などを 正確に計算することは困難です。

もし計算が可能になったとしても、リアルな映像を作るには
人体の超高精細なアセットが必要になるため、制作コストが高くなります。

一方、人間の顔の 実写映像ならば、
大量に手に入れることが 比較的容易です。
そこで、これらを教師データとして用いてAIで描画を行うことは非常に 有望と考えられます。

2-2. 従来のAI画像生成

Silicon Studio

- 近年、AIによる画像生成技術は劇的に進歩
  - 非リアルタイムのアプリケーションでは本物の写真と見分けがつかない
- しかし、リアルタイム描画は困難
  - 計算時間がかかる
  - 元になる画像が必要
  - 顔の造形やライティングなどの細かい指定ができない
- リアルタイムでの応用はまだあまり考えられていない
  - アーキテクチャから見直す必要があると考える

Face swapping
[Naruniec et al., 2020]

CEDEC2020

AI論文として今は 黄金時代だと思います。
終わる前に大切にしないとですね。

近年 、AIによる画像生成は 劇的に進歩し、
非リアルタイムのアプリケーションでは 本物の写真と見分けがつかない画像を生成できています。

しかし、リアルタイムレンダリングでは、そこまでリアルな映像を AI で描画するということは いまだに 困難です。
また、既存の写真から加工したりランダムに作るのでなく、顔の造形やライティングなどを細かく指定できる必要があります。

現時点のAIの研究は「品質向上」に主眼が置かれ、リアルタイムでの応用はまだあまり考えられていないように思われます。
我々は、リアルタイム・レンダリングへの応用を考えるには、アーキテクチャから見直す必要があると考えます。

講演内容

Silicon Studio

1. 前置き
2. 背景
3. 既存の神経回路網の非効率さについて
    1. 非効率の概要
    2. GPUのメモリ構造
    3. 畳み込みカーネルの非効率
    4. チャンネル数とスケーラビリティ
    5. GPUの同期の構成
    6. データ依存性
    7. データ依存性と同期
4. リアルタイム描画での条件による効率化の機会
5. 非効率さを解決するアーキテクチャの提案
6. 重要な論文のアーキテクチャの紹介
7. 今後の予定

CEDEC2020

既存の神経回路網を GPUで 実装する 際の 非効率について説明します。
主に、メモリと同期の問題です。

3-1. 非効率の概要

Silicon Studio

- 神経回路網はレイヤー間のデータ依存性によって同期が必要
- 前のレイヤーの出力をまた次の入力として
  GDDR6メモリから読み込む
- 当該ピクセルに無関係なチャンネルが
  計算処理性能やメモリ容量を食い尽くす

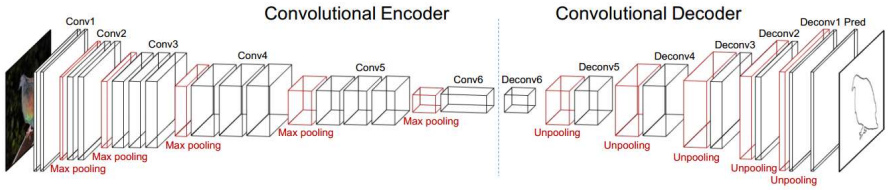


Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

https://nsarafianos.github.io/icip16

CEDEC2020

「非効率があるとは 具体的にはどう言う意味でしょう？」
本セッションで主張するのはこの３つです。

第一に、神経回路網のレイヤー間のデータ依存性があるため、レイヤーごとに同期が必要です。
神経回路網は図のように多数のレイヤーで構成されています。

第二に、前のレイヤーの出力をメモリに書き出し、また次のレイヤーの入力としてメモリから読み込む必要があります。

第三に、各レイヤーで多数のチャンネルを持っており、神経回路網の実装には分岐がないので入力チャンネルの全部を処理しないとならないです。
当該ピクセルに無関係なチャンネルが、計算処理性能やメモリ容量を食い尽くすことになります。

3-1. GPUのニーズ

Silicon Studio

- 作業データセットはオンチップメモリに
  適合したサイズ(<32KB)が望ましい
- 独立したスケジューラユニット内で
  同期が行われることが望ましい
- 特徴量マップは、ハードウェアの積和演算
  (multiply-add)で容易に得られることが望ましい

CEDEC2020

「非効率がだめならどうすればいいの？」
こうするとよいです。

第一に、作業データは独立した命令スケジューラーのレジスターとSMの共有メモリに入るサイズが望ましいです。
すなわち、32KBより小さいことが求められます。

第二に、同期/シンクロはスケジューラーの中に行われることが望ましいです。
すなわち、SM外部までのデータ依存性がない状態です。
SM間の同期はオーバーヘッドが大きいためです。

第三に、
特徴量マップは、ハードウェアの積和演算 (multiply-add)で容易に得られることが望ましいです。
神経回路網で主に用いられる畳み込みカーネルには、命令分岐がないので、GPUに適していると言えます。

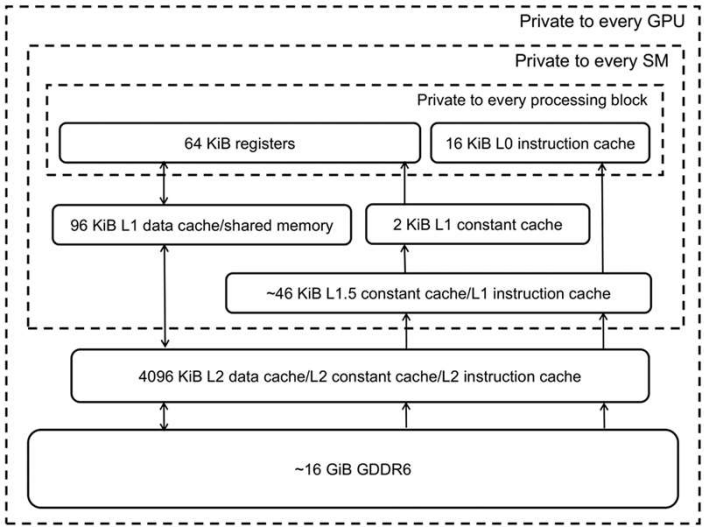
3-2. GPUの構造１：メモリ

Silicon Studio

- SM当たり
  4つの命令スケジューラ
- 命令スケジューラ当たり
  64KBのレジスタ
- メモリ読み込み遅延：

GDDR6	- 616 clock cycles
L2	- 188 clock cycles
L1	- 32 clock cycles
SMEM	- 19 clock cycles
Register	- 0 clock cycles

Zhe Jia, Marco Maggioni, Jeffrey Smith and Daniele Paolo Scarpazza. "**Dissecting the NVidia Turing T4 GPU via Microbenchmarking**", 2019



CEDEC2020

この図はTuring T4 GPUのメモリ構成を 表しています。

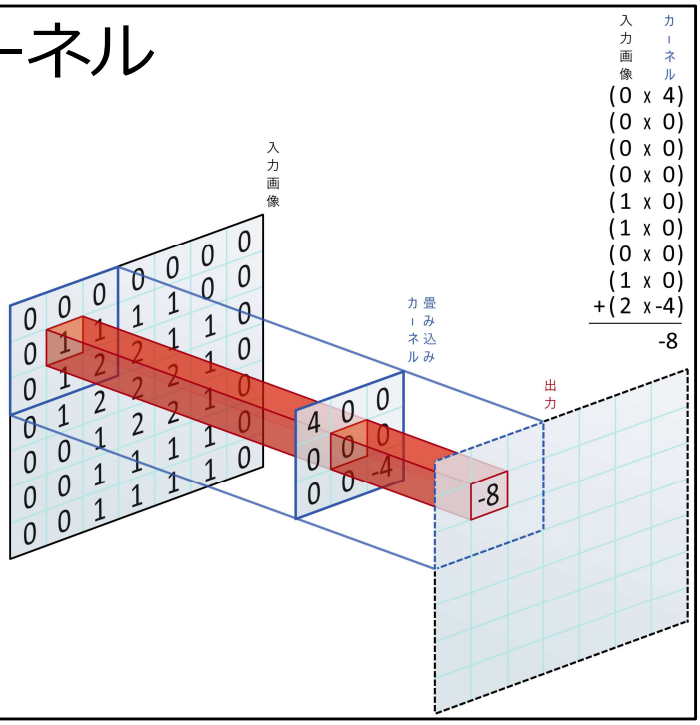
SM(Streaming Multiprocessor)当たり4つの独立した命令スケジューラがあります。

一番内側の“Private to every processing block”という点線の四角が見えますか？
1つの命令スケジューラが64KBのレジスタを持っています。
どうしても、入力情報と出力情報が ともにレジスタに 収まる必要があります、計算処理の瞬間で。

ビデオメモリ（GDDR6）からの読み込みは 最も 遅延が大きく、
共有メモリやL1キャッシュからの読み込みは 比較的高速です。

3-3. 畳み込みカーネル

- 畳み込みカーネルが
  各ピクセルに適用される
- カーネルサイズが≥2なら
  入力が重ね合わされる
- 画像に示す3 x 3のカーネルが
  標準的に用いられている



「畳み込みカーネル」は「畳み込み神経回路網」の基本です。

「畳み込みカーネル」が 各ピクセルに適用されて出力されます。

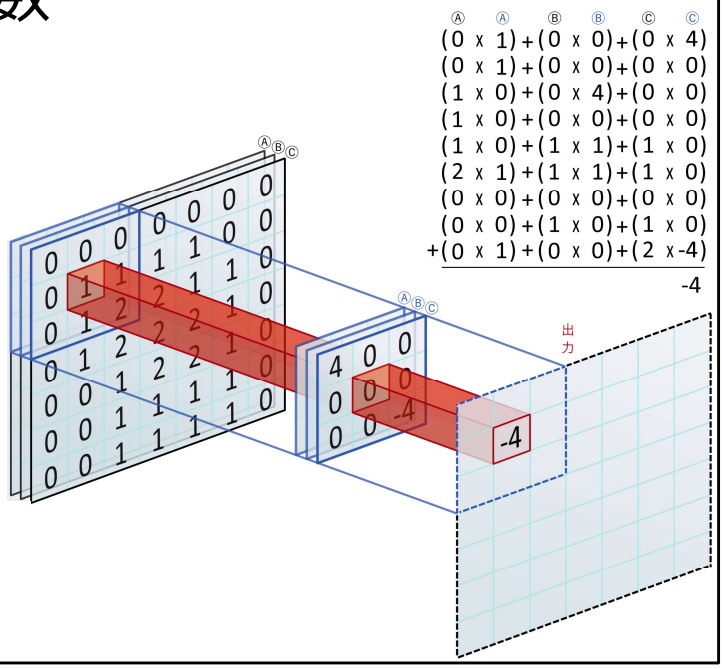
カーネルサイズが2以上なら入力が 重ね合わされます。
この図に 示す 3 x 3 のカーネルが 標準的に 用いられています。

具体的な計算を説明します。

まず、出力ピクセルの位置を 中心として、カーネルのサイズと 同じ範囲だけ、入力画像を読みます。
そして、互いに 対応する位置の、入力画像の値とカーネルの値をかけ、
それらを合計したのが出力です。
左上は0かけ4、右下は2かけ-4、出力は-8です。

3-3. チャンネル数

- チャンネルが多いほど
  計算処理が増える
- 例えば、入力が3チャンネルなら
  (3x3x3)のカーネルになる
- チャンネルが多いほど
  ピクセル当たりの
  メモリ使用量が増える



入力のチャンネルが多いほど 計算処理が増えます。

例えば、先ほどの例(3x3)で 入力が3チャンネルなら(3x3x3)のカーネルになります。

出力が複数ある場合には、カーネルが複数になります。
例えば、この例でもし出力が 10チャンネルなら(3x3x3)のカーネルが 10種類になります。

このように、チャンネルが多いほどピクセル当たりの計算量もメモリ使用量も増えます。

3-3. 表現力

- 各チャンネルがそれぞれ異なる情報を持っている
- 例えば、
- tableに属しているか、chairに属しているかなど
- 神経回路網に表現力を持たせるため、多数(512 など)のチャンネルが用いられる
- しかし、各ピクセルにおいてはほとんどのチャンネルが無関係(値が0)になっている場合が多く、非効率

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B. Tenenbaum, William T. Freeman, Antonio Torralba.
"**GAN DISSECTION: VISUALIZING AND UNDERSTANDING GENERATIVE ADVERSARIAL NETWORKS**", 2018

入力のカラー画像ならRGBですが、中間層の 各チャンネルはそれぞれ 異なる情報を持っています。

ここに示す例では、そのピクセルが table に 属しているかを 表す チャンネル、Chair に属しているかを 表すチャンネルなどですね。
もちろん、かならずしも人間が 解釈して 言葉で 表せる 情報とは限りません。

神経回路網に 表現力を持たせるため、多数のチャンネルが用いられます。
例えば 256 とか 512 とかが用いられます。

有効のチャンネルが出力に影響します。
無効のチャンネルは値が0なので影響しませんが、有効・無効に関係なく、
チャンネルの計算処理は全部一緒です。
個別のピクセルにおいて、多数のチャンネルの内、
有効なのが ごく一部 しかないとしたら、メモリ使用量と 演算コストの両面で非効率です。

3-4. チャンネル数とスケーラビリティ

Silicon Studio

- 画像の品質を求めるほどチャンネル数が増える
- 表現力を求めるほどチャンネル数が増える
- チャンネル数による非効率化はスケーラビリティのボトルネックになる

CEDEC2020

画像の品質や 描画できるドメインの拡大を求めるほどチャンネル数が増えます。
チャンネル数が増えれば、無関係なチャンネルによる非効率も大きくなり、スケーラビリティのボトルネックになってしまう可能性があります。
実行速度の最適化は、最終的には 速度のためでなく、スケーラビリティのためです。

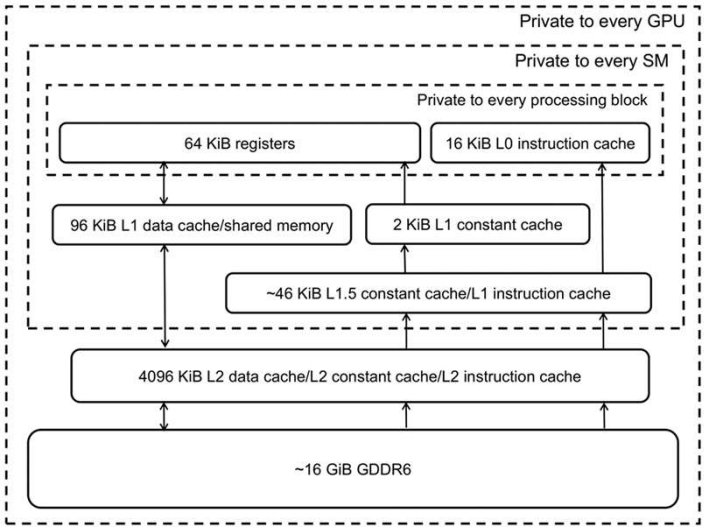
3-5. GPUの構造２：同期

Silicon Studio

- 内部のシンクロは速い
- 外部のシンクロは遅い
- 別SMとのシンクロは
  基本的にシェーダ再起動
- 依存性の同期(概数):

  GPU内　- 20,000 clock cycles
  SM内　- 100 clock cycles
  Warp内　- 20 clock cycles
  Thread内 - 0 clock cycles

Zhe Jia, Marco Maggioni, Jeffrey Smith and Daniele Paolo Scarpazza. "**Dissecting the NVidia Turing T4 GPU via Microbenchmarking**", 2019



CEDEC2020

プロセス間の同期のオーバーヘッドについて考えてみます。
原則的に、内部のシンクロは速く、外部のシンクロは遅いです。

別SMの間でのシンクロは基本的にシェーダを再起動するので、非常に遅いです。

GPUに詳しい方は Warp内 同期 の 20 Clock Cycles を見れば驚くかも知れません。
でも最近の Ampere 世代で新しい命令が出ています。Warp全体の一発で変数が共有できます。

同期の Clock Cycles は色々な条件に影響されるので、ここに出したのが正確な数字ではないです。

3-6. データ依存性

Silicon Studio

- 認識力のためにはデータ依存性が必要だと考えられる
- 最適化のためにはデータ依存性は最低限にしたい
- まず依存性の原因を見せたい

CEDEC2020

ここで、考えたいデータ依存性とは、あるピクセルの出力が 入力のどの範囲に依存しているかと言うことです。

1つのピクセルだけを見ても、RGB画像を 理解することはできません。
そのため、認識力のためにはデータ依存性が必要になります。

しかし、最適化のためにはデータ依存性は 最低限にしたいです。

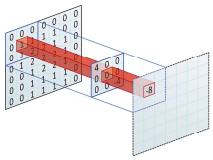
まず、畳み込みカーネルにおける依存性の 原因 をお見せします。

3-6. データ依存性の拡大

Silicon Studio

- 畳み込みカーネルによるレイヤーを多数重ね合わせてゆく
- 畳み込みカーネルの出力は次のレイヤーの入力になる
- 重ね合わせるためデファクトの入力領域が徐々に広がってゆく

- 「緑」はデータ依存性ありの入力領域を意味する



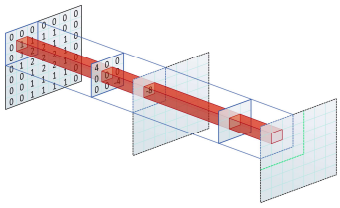
「畳み込み神経回路網」では、畳み込みカーネルによるレイヤーを多数、重ね合わせてゆきます。
あるレイヤーにおける畳み込みカーネルの出力が、次のレイヤーの入力になります。
重ね合わせるため、実質的な入力領域は 徐々に広がってゆきます。

先ほどの3x3カーネルの図ですね。

3-6. データ依存性の拡大

- 畳み込みカーネルによるレイヤーを多数重ね合わせてゆく
- 畳み込みカーネルの出力は次のレイヤーの入力になる
- 重ね合わせるためデファクトの入力領域が徐々に広がってゆく

- 「緑」はデータ依存性ありの入力領域を意味する

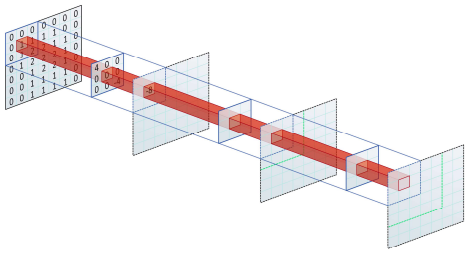


レイヤーを連続してかけてみます。

3-6. データ依存性の拡大

- 畳み込みカーネルによるレイヤーを多数重ね合わせてゆく
- 畳み込みカーネルの出力は次のレイヤーの入力になる
- 重ね合わせるためデファクトの入力領域が徐々に広がってゆく

- 「緑」はデータ依存性ありの入力領域を意味する



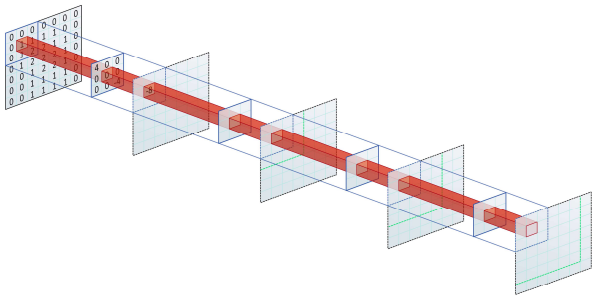
緑色はデータ依存性の範囲です。
赤を中心としたカーネルから見た場合です。

3-6. データ依存性の拡大

Silicon Studio

- 畳み込みカーネルによるレイヤーを多数重ね合わせてゆく
- 畳み込みカーネルの出力は次のレイヤーの入力になる
- 重ね合わせるためデファクトの入力領域が徐々に広がってゆく

- 「緑」はデータ依存性ありの入力領域を意味する



このように、レイヤーが重なるほど、データ依存性が拡大していくため、
遠くまで同期しないとならないです。

3-7. データ依存性と同期

Silicon Studio

- 大域的なデータ依存性があると、各レイヤーの推移毎にシェーダを再起動する必要がある
- 同期として、シェーダ再起動は遅い（ほぼ１０μ秒？）
- ドライバーとAPIの設定があればシェーダ再起動より少し速い方法もある（CUDAのみ？）
- 独立したスケジューラ内の同期がもっとも速い（ほぼ１０ナノ秒？）

大域的なデータ依存性があると、各レイヤーの推移毎にシェーダを再起動する必要があります。

シェーダ再起動による同期は非常に遅いです。
独立した命令スケジューラー内部の同期の千倍くらい違います。

ドライバーと API の設定があれば シェーダ再起動よりもっと速い方法もありますが、
やはり独立したスケジューラ内部の同期が もっとも速いです。

3. 非効率さのまとめ

Silicon Studio

- 神経回路網は
  レイヤー間のデータ依存性によって同期が必要
- 前のレイヤーの出力をまた次の入力として
  GDDR6メモリから読み込む
- 当該ピクセルに無関係なチャンネルが
  計算処理性能やメモリ容量を食い尽くす

CEDEC2020

このように、畳み込みカーネルのレイヤーを　重ね合わせた畳み込み神経回路網をGPUでそのまま実装すると、
データ依存性のため、出力をメモリに書き出し、次のシェーダを起動し、そのメモリを読み込む、という実装になります。

メモリの読み書き、シェーダ再起動による同期、どちらも非常に遅いです。

しかも、当該ピクセルに無関係なチャンネルに対しても処理が行われるのですから、とても非効率です。

講演内容

Silicon Studio

1. 前置き
2. 背景
3. 既存の神経回路網の非効率さについて
4. リアルタイム描画での条件による効率化の機会
    1. 大域的なデータ依存の必要性
    2. リアルタイムレンダリングの事情
5. 非効率さを解決するアーキテクチャの提案
6. 重要な論文のアーキテクチャの紹介
7. 今後の予定

次に、神経回路網をリアルタイム描画に応用する場合の条件について考えてみましょう。
この条件下においては効率化できる余地があることを主張したいです。

4-1. 大域的なデータ依存の必要性

Silicon Studio

- 求めている情報は当ピクセルの中になければ
  他のピクセルも参照する必要がある（＝依存性）
- 色情報(RGB)のみの入力では、
  当該ピクセルの情報としては基本的に足りない
- 画像認識や画像変換の場合なら
  データ依存性は仕方ないことと言える

CEDEC2020

求めている情報がそのピクセルの中になければ、当然ながら他のピクセルも参照する必要が出てきます。
すなわち、データ依存性が生じます。

色情報(RGB)のみの入力では、当該ピクセルの情報としては基本的に足りません。

画像認識や、画像変換の場合なら、データ依存性は仕方ないことと言えます。

4-2. リアルタイム描画では事情が異なる

Silicon Studio

- 入力は内容不明の写真ではなく、
  カメラやメッシュなどのバイナリデータから生成される
  - そのため、ピクセルに入れる情報は自由であり、
    必要であれば追加も可能
- この利点を活用するには手法の革新が必要

CEDEC2020

CGのレンダリングにAIを用いる場合には、事情が違います。

入力の構造や内容は不明ではありません。
カメラやメッシュ、ライティング情報などのバイナリデータから我々がフレーム毎に用意するんですね。

そのため、神経回路網の入力画像のピクセルに入れる情報は自由であり、必要に応じて追加することができます。
入力情報を明示的に支配すれば、大域的なデータ依存性は必要ありません。

この利点を活用するには手法の革新が必要です。

講演内容

Silicon Studio

1. 前置き
2. 背景
3. 既存の神経回路網の非効率さについて
4. リアルタイム描画での条件による効率化の機会
5. 非効率さを解決するアーキテクチャの提案
    1. 提案手法の概要
    2. 畳み込みカーネル
    3. ハッシュ分業
    4. 高度構造化入力
    5. 教師データ生成
    6. トレーニング
    7. システムの実装
6. 重要な論文のアーキテクチャの紹介
7. 今後の予定

リアルタイム描画に特化した、GPUの非効率さを解決するアーキテクチャを提案します。

5-1. 提案する対策

Silicon Studio

- 下記の対策を講じる：
  - ① １ｘ１の畳み込みカーネルとすることでデータ依存性の問題を解決
  - ② １６ｘ１６の独立タイルとすることでデータがレジスタに収まり、レイヤー推移の同期の問題を解決
  - ③ タイルで応用される神経回路網が、そのタイルに関係あるチャンネルだけを利用する

CEDEC2020

ここまでの議論で問題がはっきりしました。
問題が分かれば、対策もそんなに複雑ではないです。

第一に、１ｘ１のカーネルであれば、当該ピクセル以外の依存性がありません。
つまり、１ｘ１の畳み込みカーネルとすることでデータ依存性の問題を解決できます。

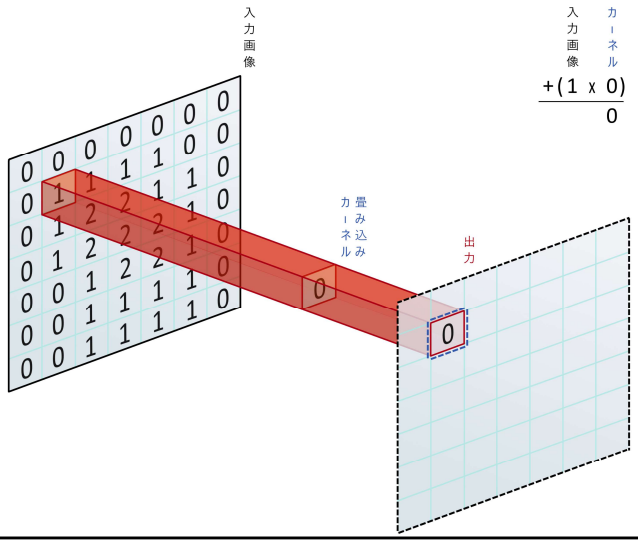
第二に、レジスターの容量が少量なら、担当するピクセルも少量にすればよいのです。
１６ｘ１６の独立タイルとすることでデータをレジスタに収めることにより、レイヤー推移の同期の問題を解決します。

第三に、各タイルで適用される神経回路網が、そのタイルに関係しているチャンネルだけを利用することで、メモリと演算の無駄をなくします。
実装は正直もう少し複雑になりますが、でも基本的に無効チャンネルが少ない神経回路網を使うようにします。

5-2. １ｘ１の畳み込みカーネル

- 計算処理が減少する
- データ依存性がなくなる
- ピクセルのチャンネルを吟味する必要がある

- 周囲の情報を用いない単純な神経回路網
  - 顔の、ある特定の位置を描画することにのみ特化



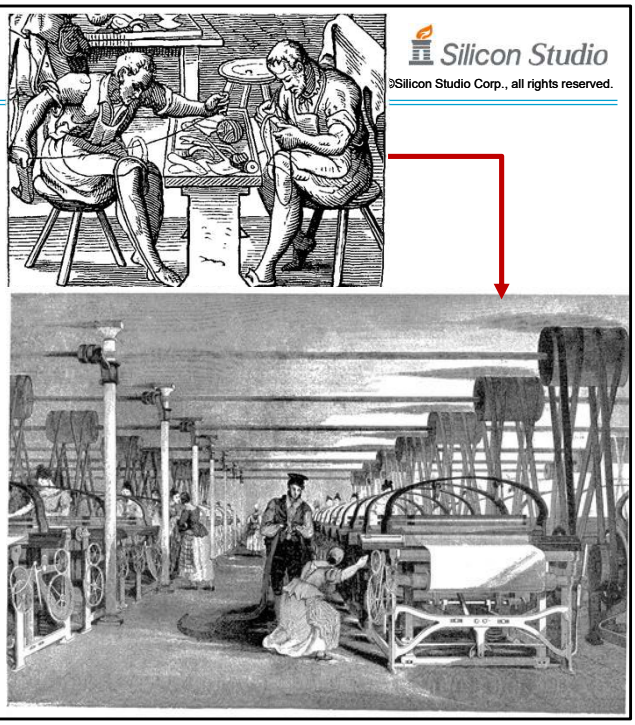
入力画像の当該ピクセルの情報のみを用い、周囲のピクセルは用いません。
これにより、計算処理が 減少し、データ依存性がなくなります。

周囲の情報を用いないため、RGB画像を深く理解する汎用的な神経回路網を構築することはできません。
このような単純な神経回路網が何の役に立つのか疑問に思われるかも知れません。

それは、顔の、ある特定の位置をレンダリングすることのみに 特化した 専用の神経回路網にするのです。

5-3. 分業

- 産業革命の時、全体的に優秀な職人が狭い知識の専門家に差し替えられた
- 神経回路網にもそのようなDeskilling(脱スキル化)を応用する
- トレーニングの入力と応用の入力が合っているなら狭いドメインでも問題ない



人類の歴史にも似ている発展が有りました。
産業革命以前は、全体の工程、全部の部位に対応できる、
汎用的に 優秀な 職人が、一人や 少人数で 製作していました。

しかし、産業革命によって、特定の作業にのみ 特化した
狭い知識の専門家が分業する体制が確立され、
非熟練労働者や、機械にも作業を任せられるようになりました。

神経回路網にもそのような Deskilling (脱スキル化) の考え方を応用できると思います。
トレーニングの入力とアプリケーションの入力が合っているなら、狭いドメインでも問題ないはずです。

5-3. 「ハッシュ分業」

Silicon Studio

- ドメインを管理するため、
  タイルのピクセル内容からハッシュを計算
- 疑似コード：

```
for( pixel in tile ) { hash_u |= (1<<pixel.u) }
for( pixel in tile ) { hash_v |= (1<<pixel.v) }
```

- ３Dモデルのもつテクスチャ UVは理想的
  - 基本は０→１
  - その上にスケーリングする（例えば８倍）
- 「配列index＝ハッシュ」で重みを効率的にロードできる

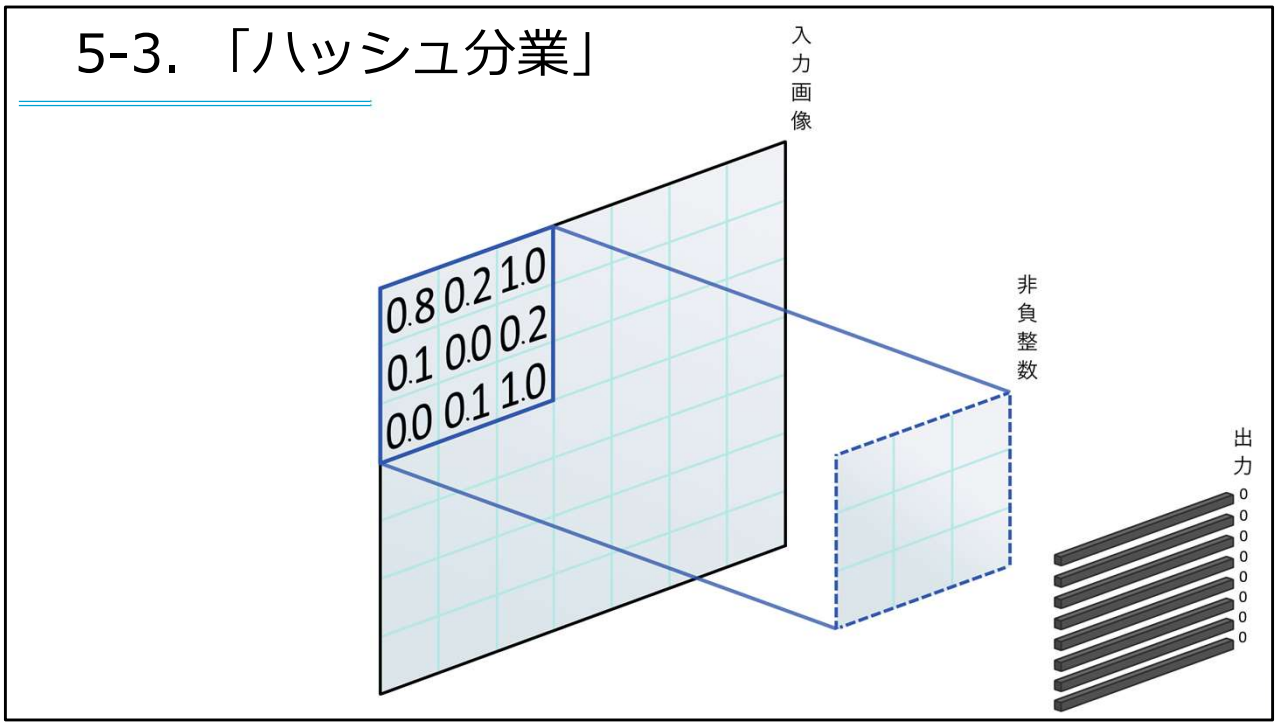
CEDEC2020

まず、画面全体をタイルに分割します。
そして、各タイル内の 各ピクセルについて、その内容からハッシュを計算します。

ハッシュ値に応じて、異なる神経回路網の重みをロードします。
最も単純な実装は、「配列 index＝ハッシュ」とすることです。

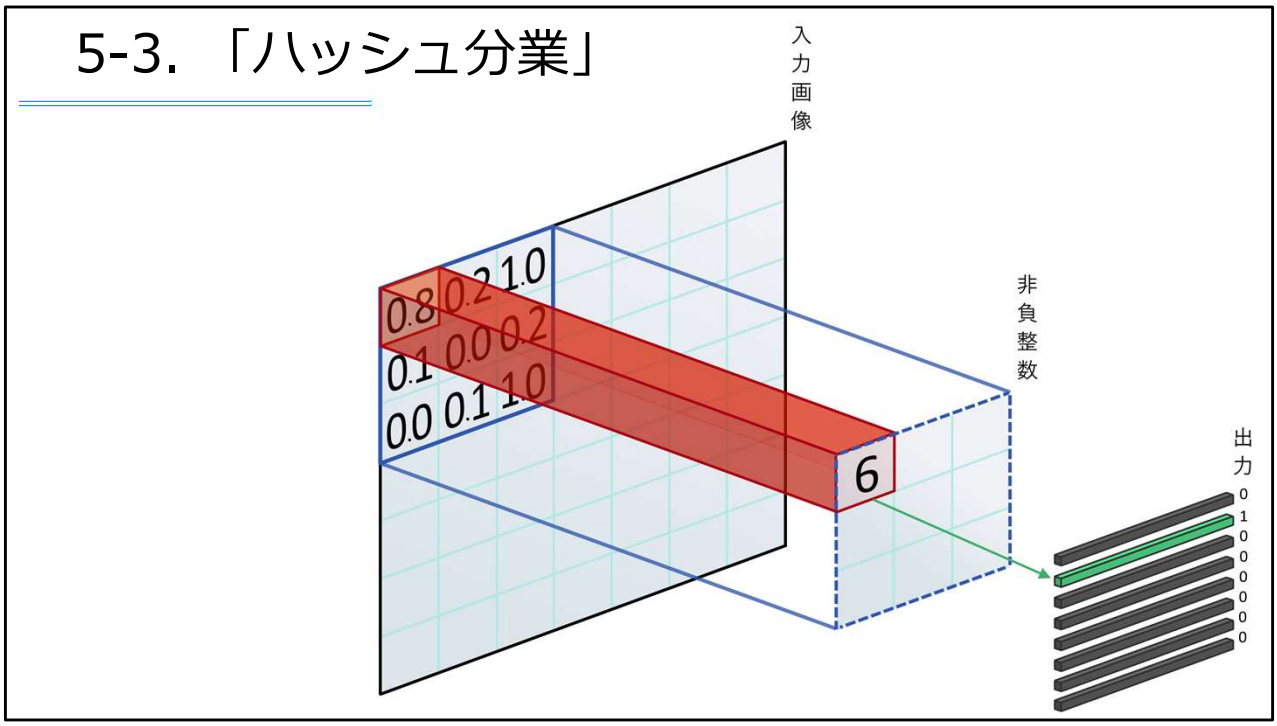
3Dモデルのもつテクスチャ UVは、そのピクセルが顔のどの位置に当たるのかを
一意に表しているので、ハッシュ値としてふさわしいです。
もちろん、それ以外の情報をハッシュ値として用いることもできます。

5-3. 「ハッシュ分業」



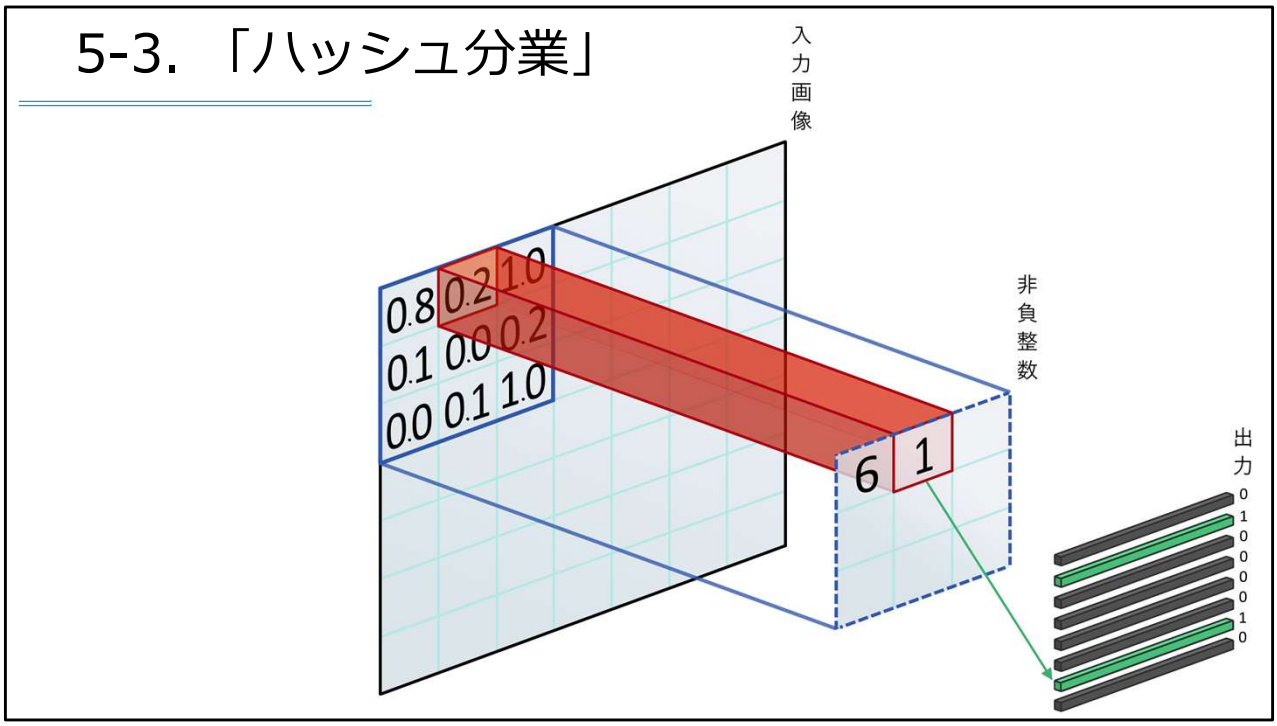

```
出力 = 0
for( pixel in tile )
{
    出力 |= 1<<((static_cast<int>( pixel.u * 8.0f )) % 8)
}
```

5-3. 「ハッシュ分業」



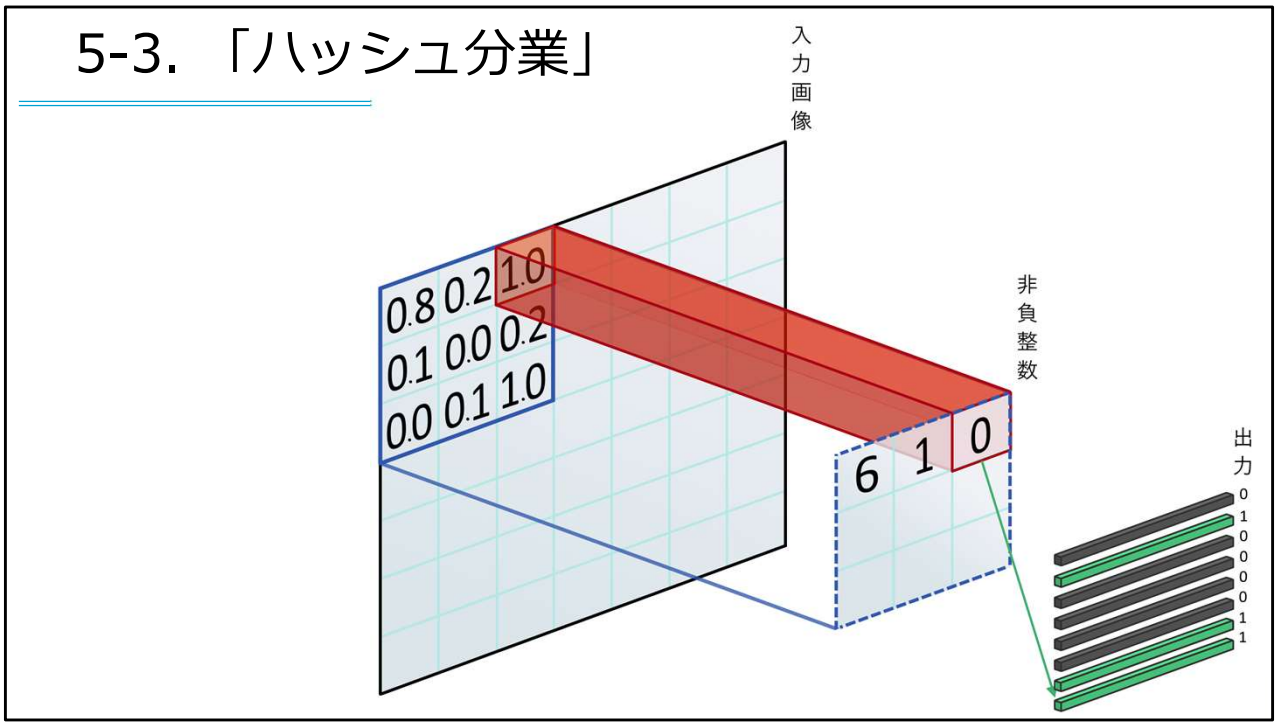
出力 |= 1<<((static_cast<int>(0.8f * 8.0f)) % 8)

5-3. 「ハッシュ分業」



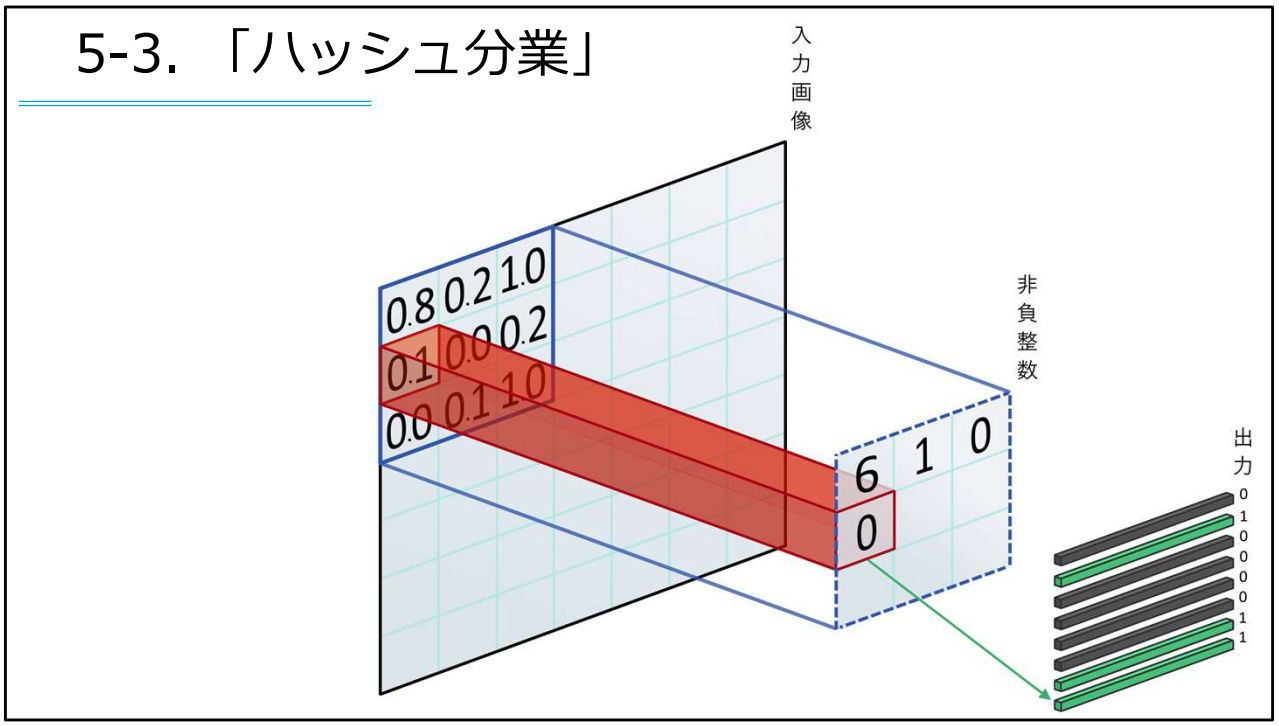
出力 |= 1<<((static_cast<int>(0.2f * 8.0f)) % 8)

5-3. 「ハッシュ分業」



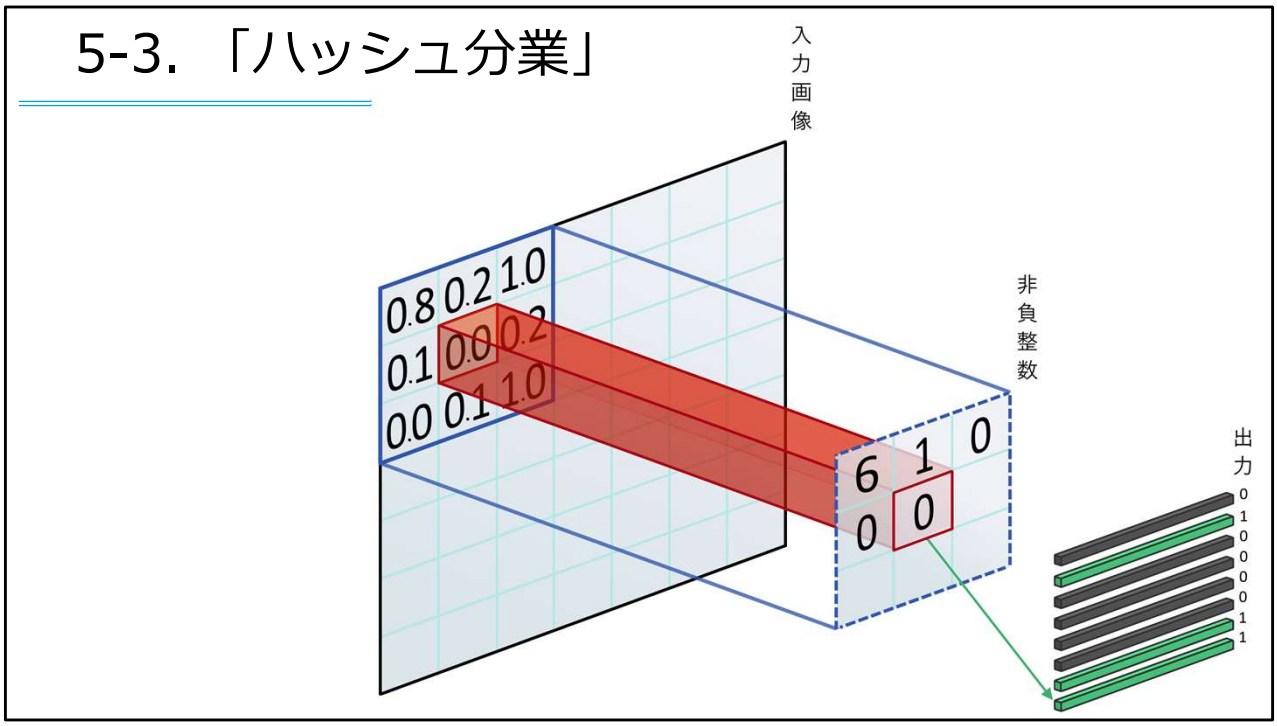
出力 |= 1<<((static_cast<int>(1.0f * 8.0f)) % 8)

5-3. 「ハッシュ分業」



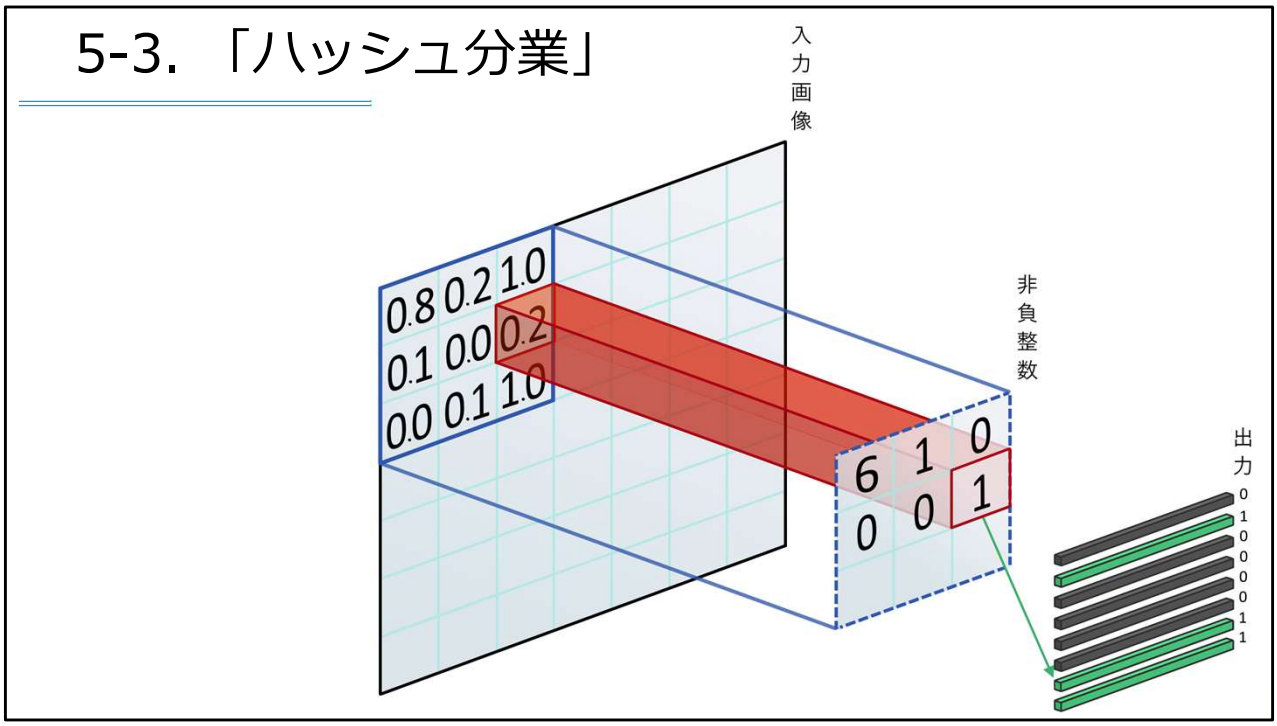
出力 |= 1<<((static_cast<int>(0.1f * 8.0f)) % 8)

5-3. 「ハッシュ分業」



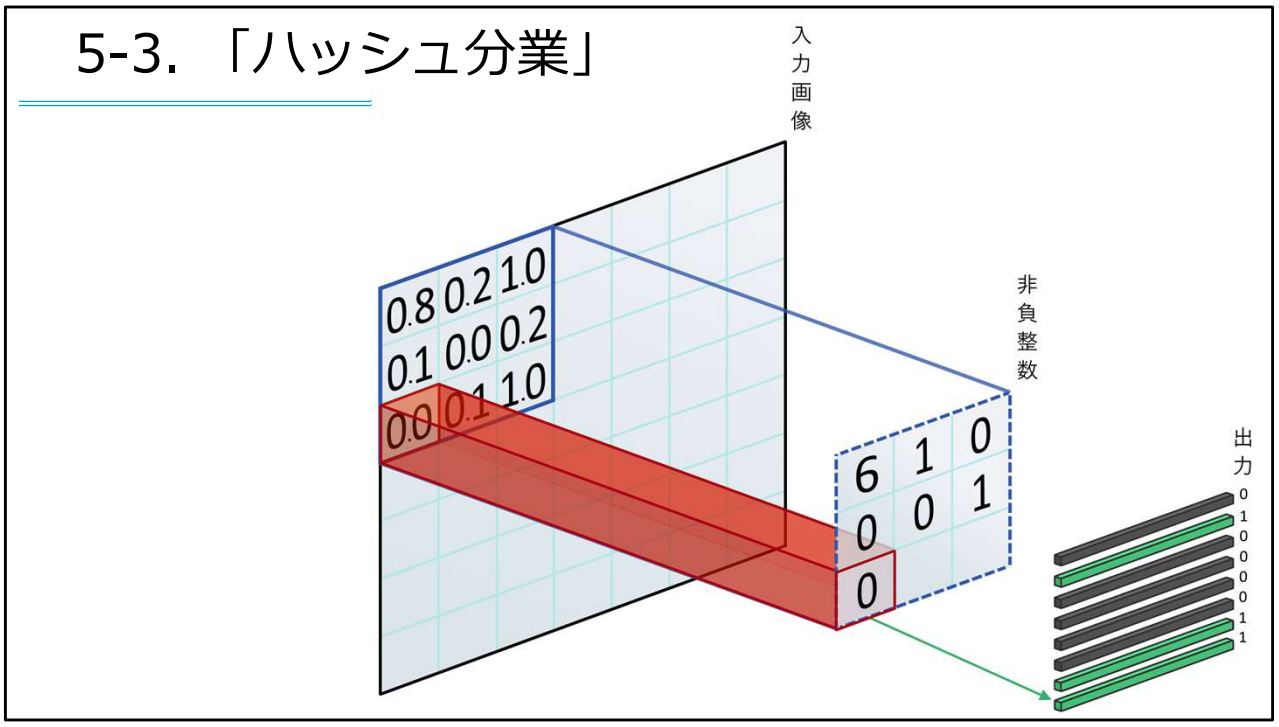
出力 |= 1<<((static_cast<int>(0.0f * 8.0f)) % 8)

5-3. 「ハッシュ分業」



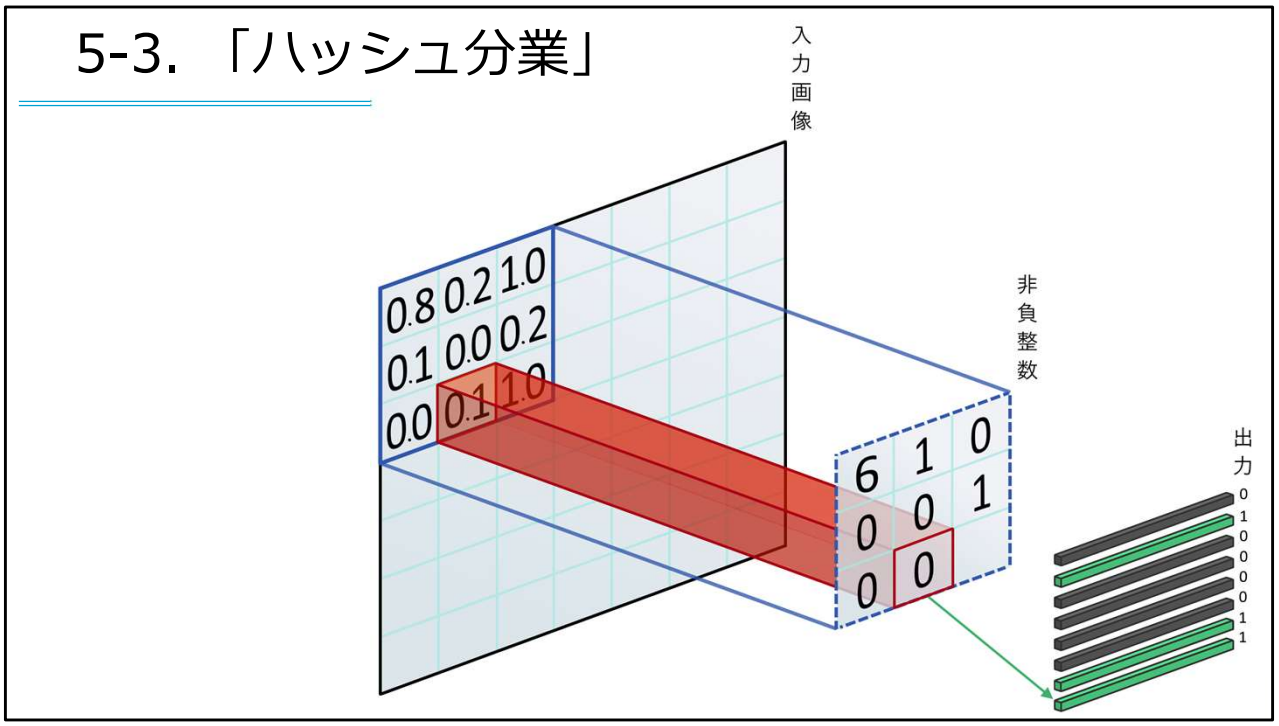
出力 |= 1<<((static_cast<int>(0.2f * 8.0f)) % 8)

5-3. 「ハッシュ分業」



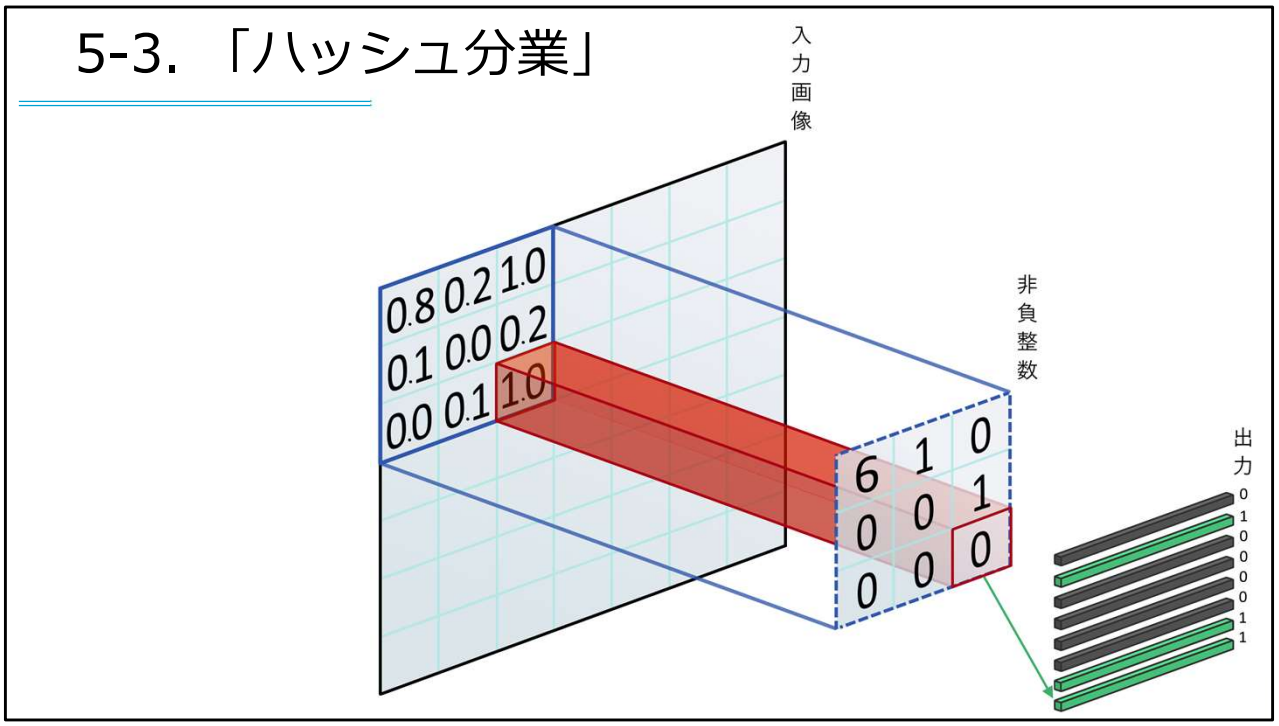
出力 |= 1<<((static_cast<int>(0.0f * 8.0f)) % 8)

5-3. 「ハッシュ分業」



出力 |= 1<<((static_cast<int>(0.1f * 8.0f)) % 8)

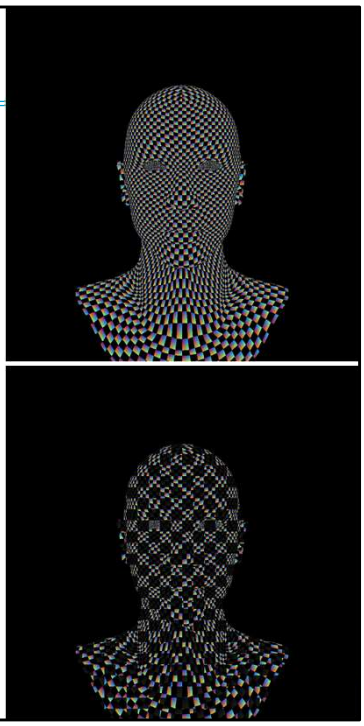
5-3. 「ハッシュ分業」



出力 |= 1<<((static_cast<int>(1.0f * 8.0f)) % 8)

5-3. 独立のタイル

- ピクセルが独立でも、
  神経回路の重みは共有
- 効率のために１６ｘ１６ぐらいの
  ピクセル（１タイル）が協力して
  一緒に計算される
- 重みは共有メモリに保存される
  （Tensor Coreと好相性）



効率のために１６ｘ１６ぐらいのピクセルを１つのタイルとして、協力して一緒に計算する実装にします。
ピクセルが独立でも、タイル内で神経回路の重みを共有します。
同じ重みを利用するので、共有メモリに読み込むことができ、効率がいいです。

5-4. 高度に構造化した入力フォーマット

Silicon Studio

- ラスタライゼーションまたはレイトレーシングを使用
- 画面上のすべてのピクセルについて、マテリアル、表面のUV座標、法線、および光源情報を事前計算
  - ディファードシェーディングのGバッファのような内容
- 入力が構造化されているため、各ピクセルは独立しており、単一のシェーダ呼び出しで計算を完結できる
- これらを明示的に事前計算で行うことで、フレーム間の不安定性につながる潜在的な原因が1つ少なくなる

CEDEC2020

ピクセルが他のピクセルを参照しないなら、そのピクセルの入力をちゃんと用意する必要があります。

そのためにはRasterizationまたはRay Tracingを使用し、
画面上のすべてのピクセルについて、
マテリアル、表面のUV座標、法線、および光源情報などを事前計算します。

Deferred ShadingにおけるGeometry Bufferのようなものと考えれば理解しやすいと思います。

このように入力が構造化されているため、各ピクセルは独立しており、単一のシェーダ呼び出しで計算を完結できます。
明示的に 事前計算で得られた情報を神経回路網の入力とすることで、
フレーム間の不安定性につながる潜在的な原因が、1つ少なくなります。

5-5. 教師データ

Silicon Studio

- 提案手法では、
  教師データ収集が大きな課題
  - 実写映像にはUV座標や法線などの情報がない
- 教師データもCGで生成
  - アーキテクチャの検証のため
  - 実写映像から学習するアイデアについては最後に紹介
- 弊社の顔画像生成ツール
  (Avatar Generator)を使用
  - パラメータを自在に変更して
    様々な顔画像を容易に生成可能
    - 異なる人種、年齢
    - 眉毛、目、鼻、口、耳の各パーツの 大きさ、位置
    - 髪の長さ、ウェーブ、ボリューム等
    - メイクアップ
    - フェイシャルパラメータによる様々な表情
    - 光源やカメラの位置、レンズの設定

Avatar Generator
Tools for creating various 3D avatars

https://www.siliconstudio.co.jp/products-service/ai-cgdata/avatar/

提案手法では、入力が単なる画像ではないため、教師データ収集が大きな課題となります。

リアルな結果のために実写映像を教師データに用いたいのですが、
実写映像には、UV座標や 法線などの情報がありません。

もともは、3Dカメラのデータを使って、点群にUVを付加する方法を検討していましたが、
教師データの質の問題なのか、学習モデルの問題なのかを切り分けることが困難でした。

そこで、今回は教師データもCGで生成してアーキテクチャの検証を行います。
今後、実写映像から学習するアイデアも考えていますので、後ほど紹介します。

生成には、弊社の顔画像生成ツール(Avatar Generator)を用いました。
このツールは、
•人種、年齢
•各パーツの大きさ、位置

•髪型、メイク、表情
•光源、カメラ、レンズの設定
などのパラメータを 自在に変更して様々な顔画像を容易に生成できます。

正確な教師データが得られたことで、効果的に実験できます。

5-6. トレーニングについて

Silicon Studio

- タイル内容によってバイナリファイルを別フォルダに
  - フォルダ毎に独立に学習させる
- Batch Normalizationの影響で、
  高解像度出力にはメモリ使用量が大きくなる問題
  - ハッシュ分業の開発はある程度その問題の対策
  - 分業の神経回路網は重みが比較的少ないため、
    メモリのネックがなくなる

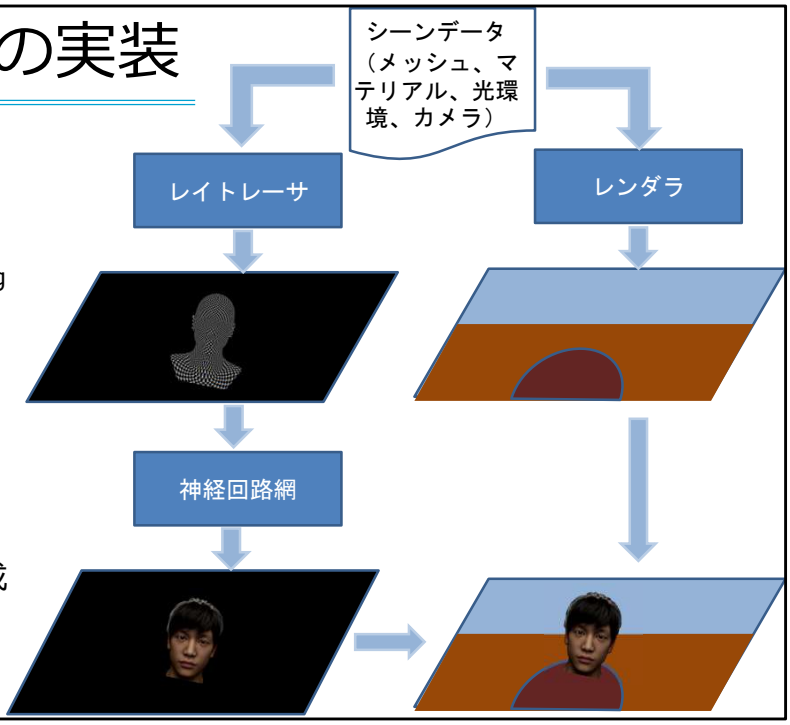
CEDEC2020

タイル内容によってバイナリーファイルを別フォルダに置いて、そのフォルダを独立に学習させます。

解像度の高い描画にはトレーニング時のメモリ使用量が大きな問題です。
ハッシュ分業にはトレーニングの最適化のメリットもあります。

5-7. システムの実装

- 各ピクセルにおいて
  データを収集し、
  神経回路網に渡す
  入力フォーマットに変換
  - Nvidia の Vulkan Ray-Tracing Extension を利用して実装
- 神経回路網のコンピュート
  シェーダを実行、
  RGBの色情報を出力
- 顔以外の部分は
  従来どおりのレンダリング
- 最後に2つの結果画像を合成



提案するリアルタイム描画システムの実装は、次のようなものです。

入力として、既存のレンダリングシステムと同様に、
顔のメッシュ、光源、カメラ、マテリアルなどの情報を与えます。

まず、各ピクセルにおいてデータを収集し、神経回路網に渡す入力フォーマットに変換します。
今回は Nvidia の Vulkan Ray-Tracing Extension を利用して実装しています。

次に、神経回路網のコンピュートシェーダを実行し、入力画像に基づいて、RGB の 色情報を 出力します。

顔以外の部分は 既存のレンダリングシステムによってレンダリングします。

最後に、2つの結果画像を 合成します。

講演内容

Silicon Studio

1. 前置き
2. 背景
3. 既存の神経回路網の非効率さについて
4. リアルタイム描画での条件による効率化の機会
5. 非効率さを解決するアーキテクチャの提案
6. 重要な論文のアーキテクチャの紹介
    1. オートエンコーダ
    2. GAN
7. 今後の予定

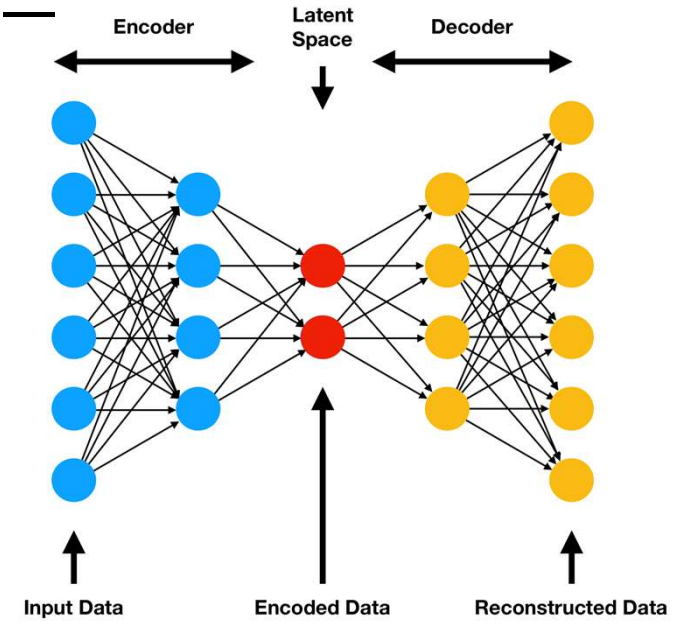
CEDEC2020

AI 描画は 最適化だけで実現できません。
最近の研究から、よくできているアーキテクチャを紹介します。

柔軟性があり過ぎて、リアルタイムに使うことは難しいですが、
高度に構造化された教師データを作るには 重要な 役割となります。
そのため、今までの内容と 少し 離れてしまいますが、ここで紹介しておきます。

オートエンコーダー

- 画像を一度低次元に
  エンコードした後、
  再度復元(デコード)し、
  入力と出力が一致する
  ようにトレーニングする
- ドメインに特化した
  高次の空間の内容を
  少ない次元で表現できる



CEDEC2020

オートエンコーダとは、「自動的にエンコードしてくれる 道具」と言う意味です。
エンコードされた情報は 真ん中の 潜在空間 にあります。
図の通り、潜在空間 は入力より小さいですが、復元・生成に必要な 本質的な情報がすべて含まれています。

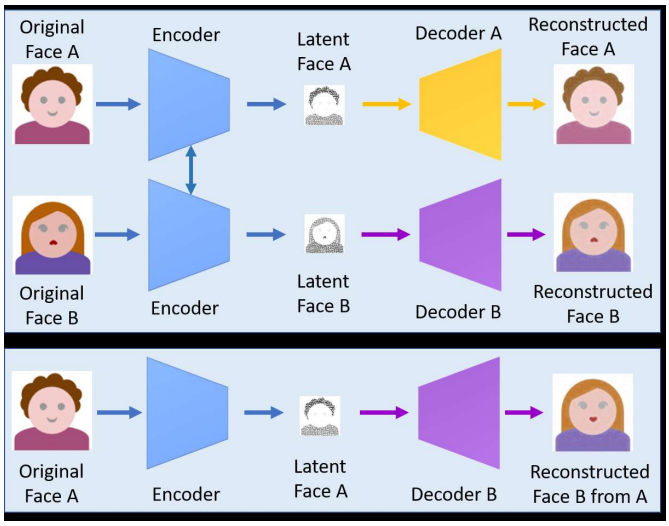
トレーニングするときの出力目標は、入力と同じです。
そのため、何でも教師データになれます。構造化する必要がないです。

共有潜在空間 オートエンコーダー

Silicon Studio

- エンコーダは複数の人間で共有
- デコーダは共有しない
- 別の人のデコーダを使えば、ある人の顔を別の人の顔に変えることができる
- ディープフェイクで使用されており、製品レベルの画質に近づいている
- フォトリアリスティックな入力が必要

Thanh Thi Nguyen, Cuong M. Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen and Saeid Nahavandi. "**Deep Learning for Deepfakes Creation and Detection**", 2019



CEDEC2020

Deep Fake では、オートエンコーダのアーキテクチャを応用しています。
製品に近づいているレベルの 画像品質だと思われます。
リアルタイム描画とは別物ですけど。

上の図のように、エンコーダは 複数の人間で共有しています。
デコーダは それぞれの人間で 個別になっています。

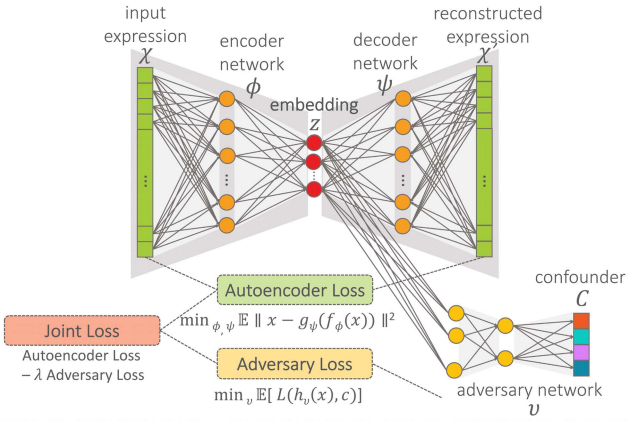
下の図のように、別の人のデコーダを使えば、ある人の顔を別の人の顔に変えることができるのです。

写真レベルの出力には 写真レベルの入力が必要なので、リアルタイム描画には向いていません。

敵対的オートエンコーダ

Silicon Studio

- オートエンコーダの潜在空間を
  もっと使いやすい形にしたい
- 別のニューラルネットワーク
  「敵対的な識別器」を
  損失関数に用いる
- 潜在空間の分布を制御できる
- 本研究では、
  教師データ生成に用いる予定



Ayse B. Dincer et al., "**Adversarial Deconfounding Autoencoder for Learning Robust Gene Expression Embeddings**", 2020

CEDEC2020

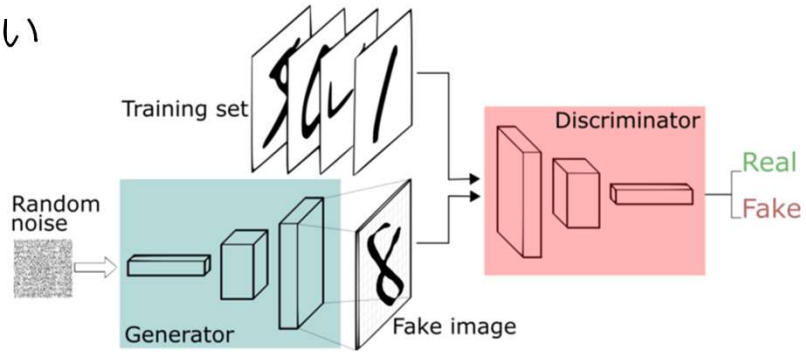
潜在空間のフォーマットが もっと使いやすい形になればいいんじゃないですか？
もう一つの「敵対的な 識別器」の神経回路網はそういう目的のためのロスを計算してくれます。

柔軟性がかなりありますので、将来的に教師データを作るために使おうと考えています。

GAN（敵対的生成ネットワーク）

Silicon Studio

- 敵対的オートエンコーダと似ているが、
  潜在空間ではなく生成器の出力に識別器を適用する
- 生成器への入力はランダムな分布
- 出力の制御が難しい



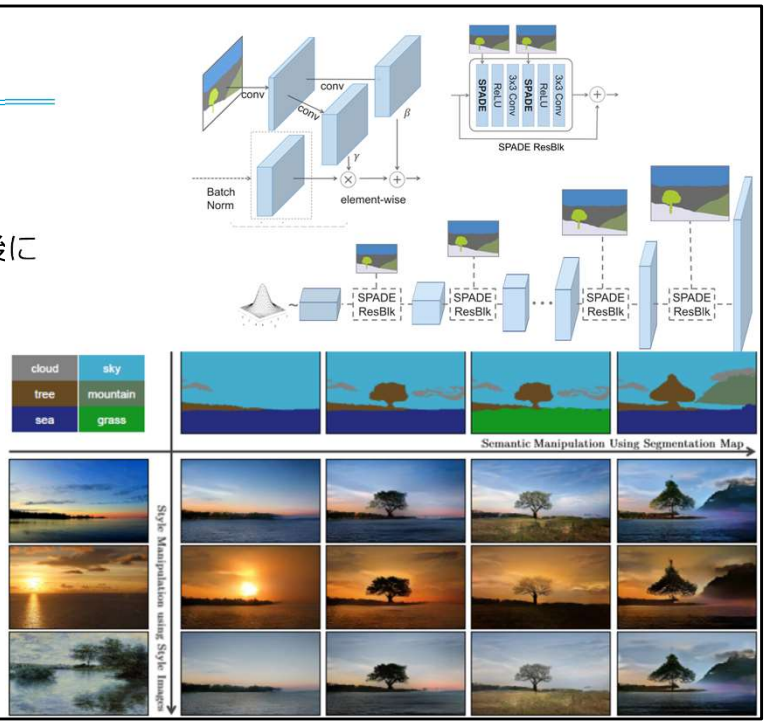
https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/

CEDEC2020

敵対的生成の 神経回路網は 最近いい画像をよく作ってくれるので、少し有名になっています。
基本的に ロス関数も 神経回路網のように トレーニングされています。
入力は 乱数です。それ重要です。トレーニングの時のドメインによって決まり、それ以外は出力を支配できません。
つまり、リアルタイム描画には そのままで使いづらいです。

条件付きGAN

- 出力を制御できるGAN
  - 生成器の各階層における Batch Normalizationの後に 整数マスク画像を入力
  - 条件付けすることで、 望ましい出力が得られる
- Batch Normalization により入力の構造が 壊れてしまう問題を 解決した



Taesung Park et al., "**Semantic Image Synthesis with Spatially-Adaptive Normalization**", 2019

入力が乱数なら、どうやって出力を支配できるでしょうか。

この手法では、各レイヤーの Batch Normalization の後に 整数マスク画像 を入れています。
これにより、条件付けされて、望んだ出力が得られます。
結果がかなりいいです。

Batch Normalization は トレーニング 安定性のためには素晴らしいですが、問題も多いです。
その一つは入力の構造を 壊してしまうことですが、
この手法はその問題点を 解決しています。

講演内容

Silicon Studio

1. 前置き
2. 背景
3. 既存の神経回路網の非効率さについて
4. リアルタイム描画での条件による効率化の機会
5. 非効率さを解決するアーキテクチャの提案
6. 重要な論文のアーキテクチャの紹介
7. 今後の予定
    1. まとめ
    2. 今後の予定

CEDEC2020

本セッションのまとめ、今後の予定です。

7-1. まとめ

Silicon Studio

- リアルタイムAI描画に至る道
  - 現状の神経回路網によるAIはリアルタイム描画に不向き
    - 実行速度が遅い
    - 顔の形や方向、ライティングなどを直接指定できない
  - 神経回路網の構成とGPUの非効率を考察
    - データ依存性、チャンネル数の問題
  - リアルタイム描画に特化したAIを提案
    - メッシュやライティング情報を構造化した入力フォーマットに変換
    - 特定の描画に特化した神経回路網を多数学習し、ハッシュによって使い分ける
    - 神経回路網はハードの制約に収まるようにする
  - 人工的に生成した学習データで検証
    - 実写映像からの学習データ生成が今後の課題

CEDEC2020

本セッションでは、リアルタイムAI描画に至る道を 示しました。
まだ道なかばですが、進むべき方向性を 明らかにできたと思います。

現状の神経回路網によるAIはリアルタイム描画に 不向きです。
実行速度が 遅いだけでなく、顔の形や方向、
ライティングなどを直接指定できないので、
既存のレンダリングをAIに置き換えることはできません。

リアルタイム描画のため、神経回路網の 構成と
GPU の非効率の 問題を 考察しました。
データ依存性により、メモリ読み書き、同期の必要が 生じることと、
チャンネル数が 増大し、計算資源を使ってしまうことです。

そこで、Real-Time描画に 特化したAIを提案しました。
MeshやLighting情報は 神経回路網のために 高度に 構造化した入力フォーマットに変換されます。
特定の描画に特化した 神経回路網を 多数学習し、タイルごとにハッシュによって使い分けるハッシュ分業アーキテクチャです。
それぞれの神経回路網はハードの制約に収まるように設計します。

現在は、人工的に生成した学習データで検証します。
実写映像からの学習データの生成が 今後の課題です。

7-2. 今後の予定

Silicon Studio

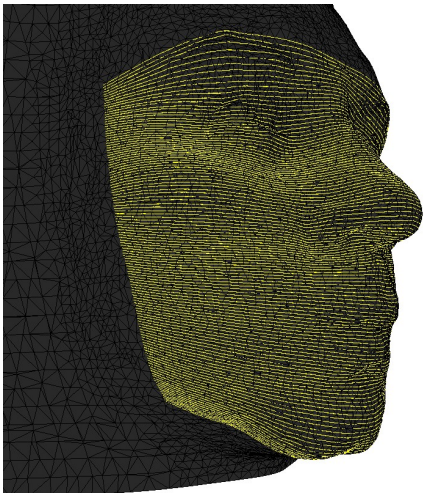
- 複数光源対応
- GPGPU最短経路探索を用いた3Dカメラ点群データのUV生成
- 敵対的オートエンコーダによるUV座標
- 速度向上
- 条件付きGANによるカスタムメイドのトレーニングデータ

今後は ここに挙げたような 課題に 取り組むつもりです。
特に、UV生成について説明します。

GPGPU最短経路探索を用いた3Dカメラ点群データのUV生成

Silicon Studio

- トレーニングデータ
  生成プログラムによって、
  顔の正しいUVマッピングが
  提供されたデータで学習を行ったが、
  可能ならば実写映像を使いたい
- 3Dカメラならば、顔の実写映像と
  同時に3D形状データも取得できる
- 3D点群データに対し、
  GPGPU最短経路探索を用いて
  UV生成する実装を行っている



CEDEC2020

本セッションでは、トレーニングデータ生成プログラムによって、
顔の正しいUVマッピングが提供されたデータで学習を行いましたが、今後は
実写映像を使いたいです。

3Dカメラならば、顔の実写映像と同時に3D形状データも 取得できます。
3D点群データに対し、GPGPU最短経路探索を用いて UV生成する実装を行っています。

敵対的オートエンコーダによるUV座標

Silicon Studio

- 2D映像のみからUV生成できれば、
  教師データがより集めやすくなる
- 顔の潜在的な変数空間を生成し、それから顔を再作成する
  敵対的なオートエンコーダを想像してほしい
- 識別器によって、トレーニングデータの顔から定義された
  正しいUVマッピングにするように、
  潜在的な空間を強制することができるはず
- これはあくまでトレーニングデータを生成するための
  ものであり、リアルタイムに実行できる必要はない

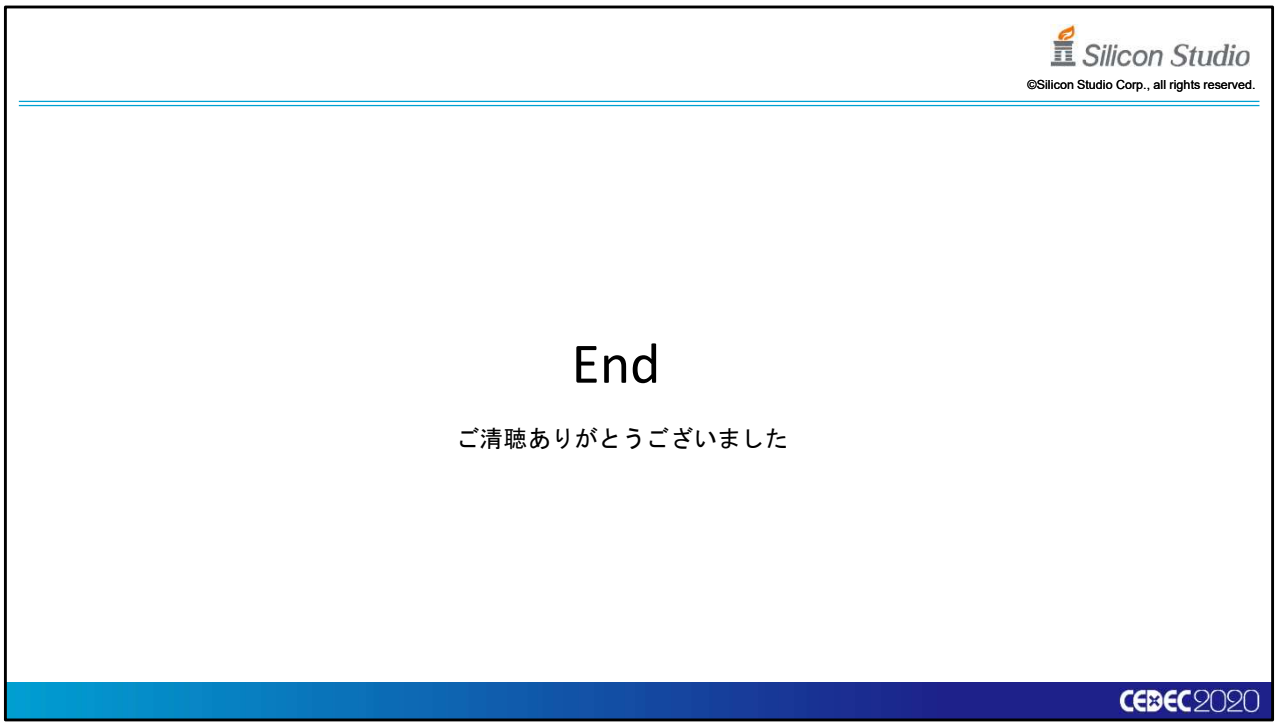
CEDEC2020

3Dカメラを用いずに、2D映像のみからUV生成できれば、教師データをより集めやすくなります。

顔の 潜在的な 変数空間を生成し、
それから顔を 再作成する敵対的な オートエンコーダを想像してください。

識別器によって、トレーニングデータの顔から定義された正しい UVマッピングにするように、
潜在的な空間を 強制することができると思われます。

これはあくまでトレーニングデータを生成するためのものであり、リアルタイムに実行できる必要はありません。



本セッションの発表は、以上です。
ありがとうございました。

ご質問、ご意見、ご感想のある方は、コメントをお願いします。

Special Thanks

Silicon Studio

- Avatar Generatorチームの皆さん
- Bosphorus 3 Dチーム

Savran, N. Alyüz, H. Dibeklioğlu, O. Çeliktutan, B. Gökberk, B. Sankur, and L. Akarun, "**Bosphorus Database for 3D Face Analysis**," The First COST 2101 Workshop on Biometrics and Identity Management (BIOID 2008), Roskilde University, Denmark, 7-9 May 2008

CEDEC2020

本実験の教師データ生成に協力してくれたAvatar Generatorチームの皆様、ならびに、顔の3Dデータベースを提供してくれたBosphorus 3Dの皆様に謝意を表明します。

Contact

Silicon Studio

- Fredrik Herzberg
  fredrik.hertzberg@siliconstudio.co.jp