ランタイム＆パイプラインの構築と最適化

テクノロジー推進部
岩崎 浩
2012/11/23

本日の内容

- データサイズ
- データフロー
- 最適化

データサイズ

あるシーンのデータサイズ

あるシーンのデータサイズ

- 頂点数 約570万 (158MB)
  - 背景 530万
  - 牛 6万
  - 車 14万
- テクスチャ 800 MB (約450枚)
  - 背景 260MB
  - 牛 120MB
  - 車 140MB
  - GI 100MB
  - その他 170MB

※この頂点数はデータのサイズです。
※実際にGPUで処理される頂点数はPass数によって増えます。

あるシーンのデータサイズ

あるシーンのデータサイズ

- 頂点数 500万 (154MB)
  - 背景 340万
  - シドロ 34万
  - 召喚獣(骨) 24万
  - その他のキャラ16万 x 6
- テクスチャ 1GB (約920枚)
  - 背景 190MB
  - シドロ 125MB
  - 召喚獣(骨) 70MB
  - その他 200MB
  - GI 350MB

あるシーンのデータサイズ

あるシーンのデータサイズ

- 頂点数 約920万 (270MB)
  - 背景 860万
  - アグニ 40万
  - 召喚獣30万
- テクスチャ 1.5GB (約1300枚)
  - 背景 430MB
  - アグニ 340MB
  - 召喚獣420MB
  - GI 180MB

あるシーンのデータサイズ

あるシーンのデータサイズ

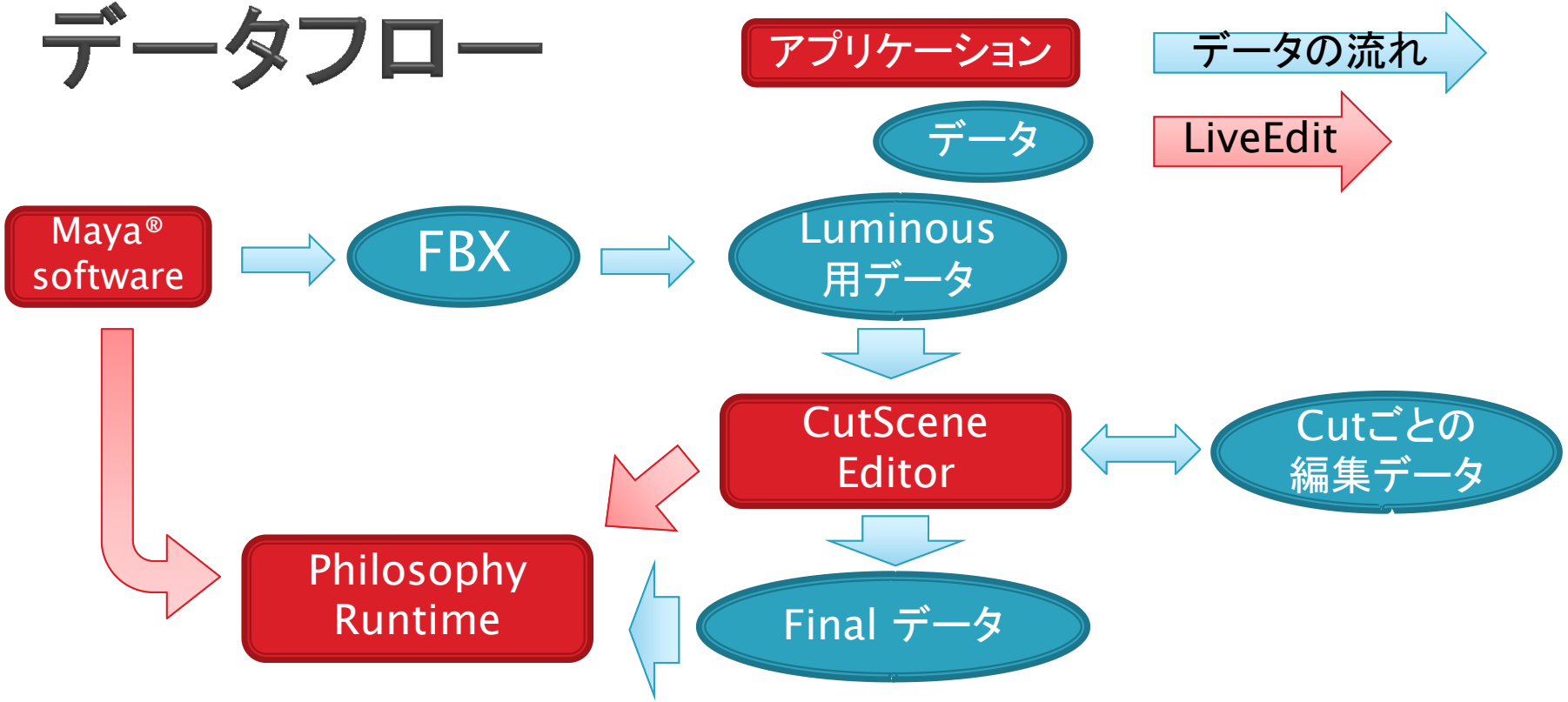
- 頂点数 510万 (150MB)
  - 背景 350万
  - 召喚獣(骨) 45万
  - アグニ 45万
  - シドロ 34万
  - その他の人 16万 x 2
- テクスチャ 1.4GB (約1000枚)
  - 背景 220MB
  - アグニ 340MB
  - シドロ 125MB
  - 召喚獣(骨) 70MB
  - その他の人 190MB
  - GI 420MB

本日の内容

- ~~データサイズ~~
- データフロー
- 最適化

データフロー

データフロー



CutSceneEditor

Maya® softwareでしていたこと

- Modeling
- Animation
- CubeMapのAreaの配置、設定
- Irradiance Volumeの配置、設定
- GI の設定
- Shader, Materialの設定
- Model, Camera, Lightの配置、設定
- Maya® softwareのReference の設定

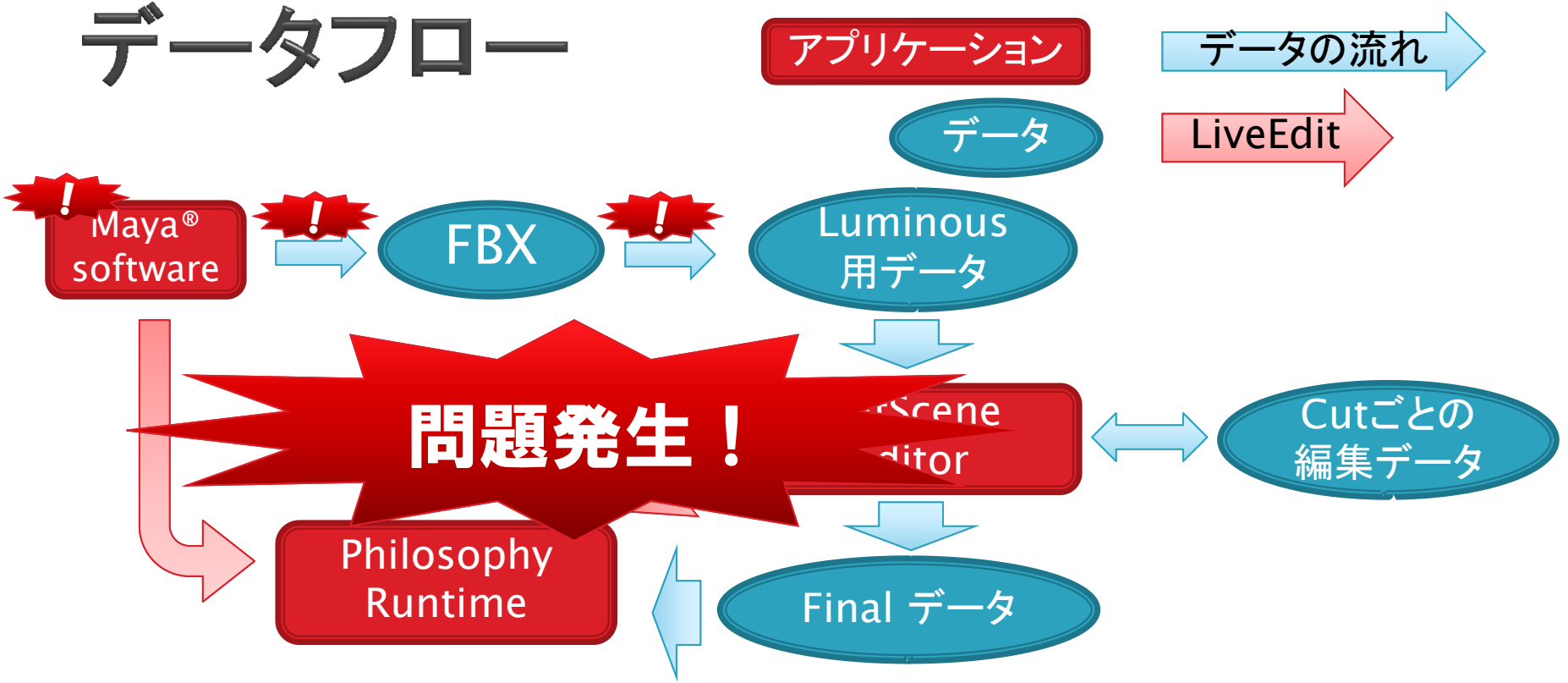
CutScene Editorでしていたこと

- 一部のAnimation
  - PostEffectのParameterなど
- Sound Trackの配置
- Cutのつなぎ
  - Cutのフェードなどもここで制御
- このツール上でCamera, Lightなど色々編集することもできましたが、今回は使いませんでした。

ワークフローの方針

- Maya® softwareでできることはMaya® softwareで行う。
  - CutScene EditorとMaya® softwareどちらでも行える部分はMaya® softwareで。
- Luminous全体の設計思想としては･･･
  - Maya® software側でも、Luminousのツールでも、なるべくどちらでも編集できるようにする、というのが最終的なゴール
  - ただ今回は、Maya® software上でのEditをメインにしました。

データフロー



ワークフローで発生した問題

ワークフローで発生した問題

- Maya® softwareのデータを開くのに時間がかかる。
- Exportに時間がかかる。
- データの配布

ワークフローで発生した問題

- Maya® softwareのデータを開くのに時間がかかる。
- Exportに時間がかかる。
- データの配布

この2つについて実例を紹介

ワークフローで発生した問題

あるシーン全体のExport時間

- FBXサイズ 200MB
- Maya® softwareの
  データを開くのにかかる時間
  - 2分
- Maya® software→FBXのExportにかかる時間
  - 1分
- FBX→Luminous用データのコンバートにかかる時間
  - 5秒
- RuntimeでLoadするのにかかる時間
  - 10秒

このシーンは許容範囲

以下全て開発環境 SSD上での計測
FBXはBinary版

別のシーン…

あるシーン全体のExport時間

- FBXサイズ 800MB
- Maya® softwareの
  データを開くのにかかる時間
  - 6分
- Maya® software →FBXのExportにかかる時間
  - 45分
- FBX→Luminous用データのコンバートにかかる時間
  - 1時間半 (FBXのLoad 50分, Convert処理 40分)
- RuntimeでLoadするのにかかる時間
  - 20秒

Exportに合計約2時間！！

Exportに合計約2時間

- ディスクスワップが発生しているわけではない。
  - ひたすらSingleThreadでBusy状態。
    - 8コアのマシンで、CPU負荷12～13%で安定

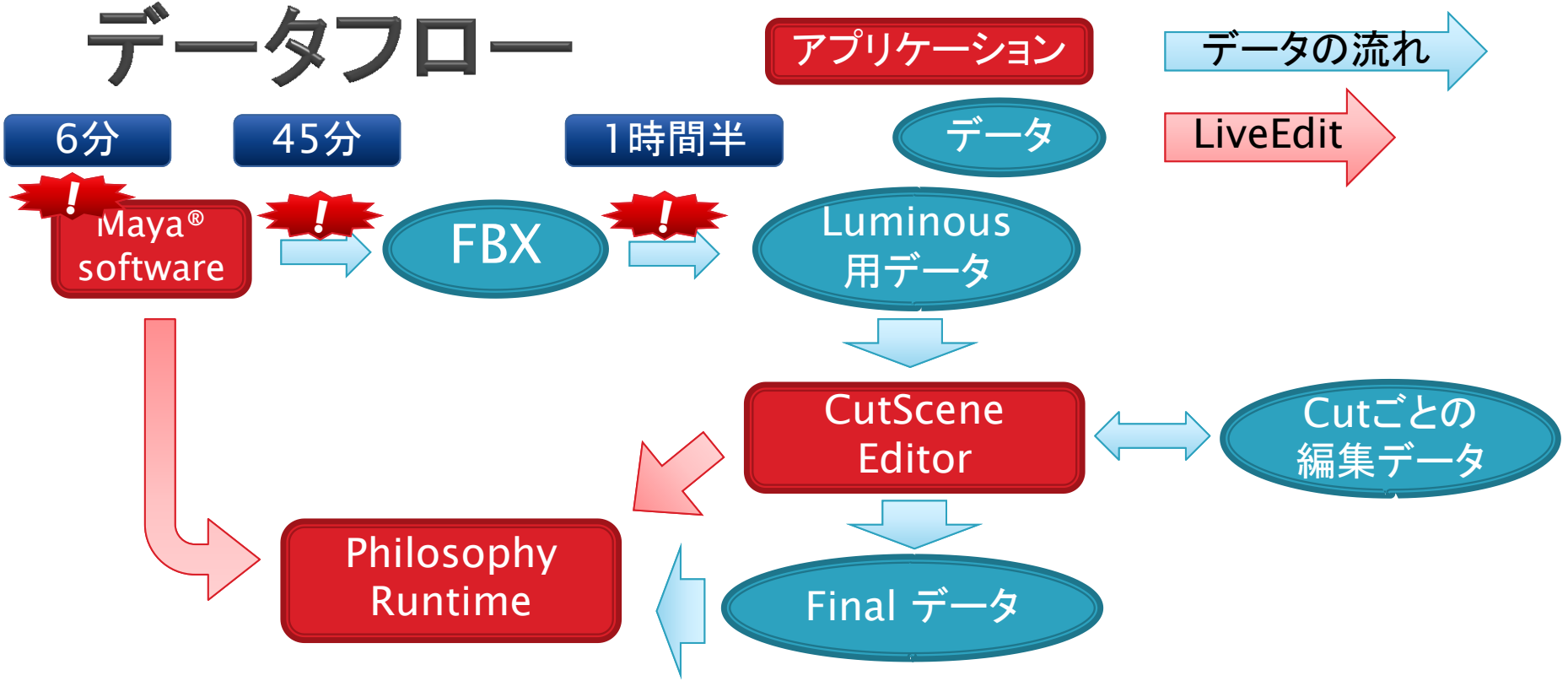
これは許容範囲を遙かに超えている！

対策が必要

ワークフローで発生した問題

- Maya® softwareのデータを開くのに時間がかかる
- Exportに時間がかかる。
- データの配布

データフロー



Maya® softwareのデータを開くのに時間がかかる

- Maya® softwareのデータを開くのに時間がかかる。
  - 6分というのは実は改善後。
  - 当初は、開くのに30分以上、かかることもあった。
    - 原因
      - Maya® softwareのデータを開くたびにShaderのビルドが走っていた。
    - 対策
      - ShaderをビルドしたBinaryがあれば、そのBinaryを使うように修正して、高速化。

        ※ Maya® softwareのCgFXShader Pluginはソースが提供されている。

Maya® softwareのデータを開くのに時間がかかる

- ただし改善後も、平均約2分。時には、10分を超えることも・・・

まだ対策が必要

Maya® softwareのデータを開くのに時間がかかる

- Lightのパラメーター一つ調整するのにも、Maya® softwareを開かないといけないため、効率が悪かった。
  - 対策
    - Lightなど頻繁に調整するパラメータは、Maya® softwareのデータをLightとSceneのデータに分けた。
    - Maya® software側はLightのファイルを開き、CutScene Editor側は、Scene全体のデータを開いてLiveEditで結果を確認した。

Maya® softwareのデータを開くのに時間がかかる問題まとめ

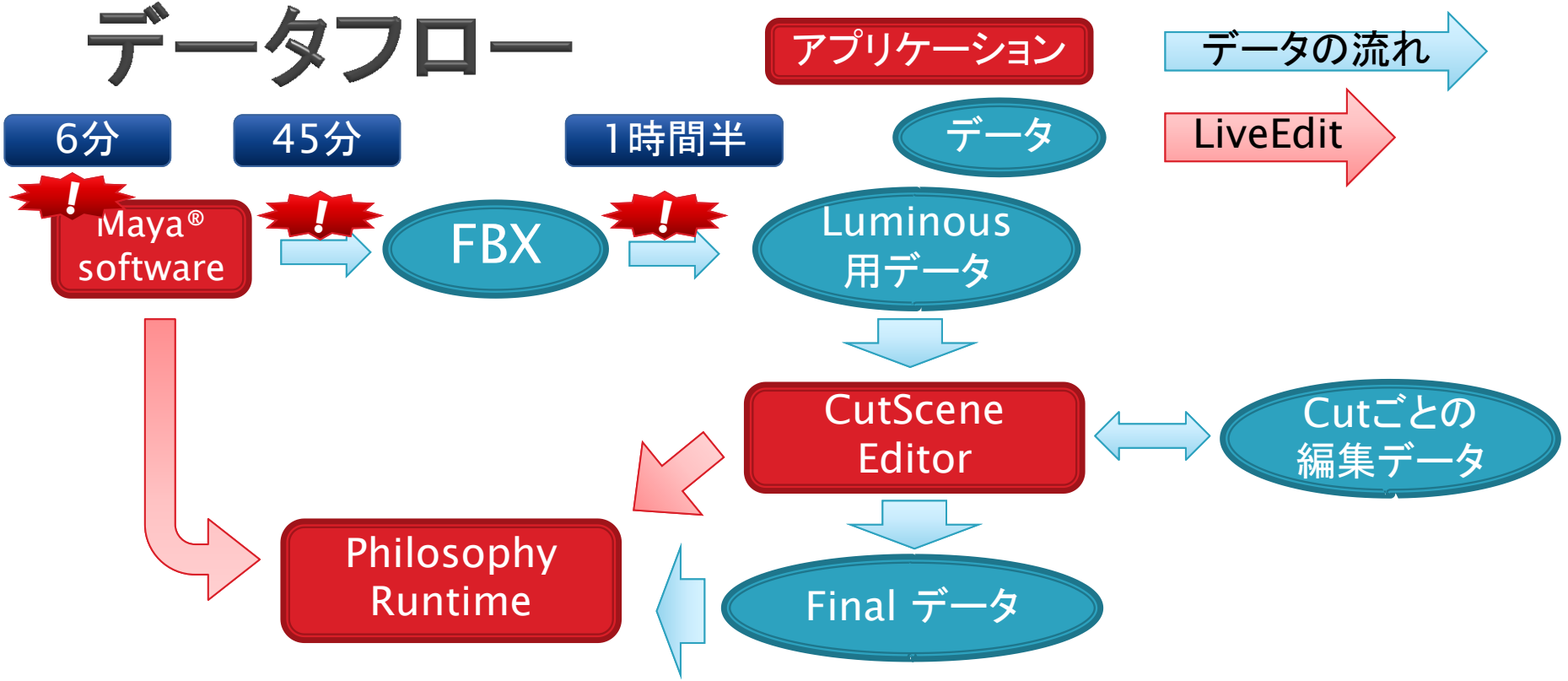
- Shader Binaryの再利用
- Maya® softwareのデータをLightとSceneに分ける

問題は解決！

ワークフローで発生した問題

- ~~Maya® softwareのデータを開くのに時間がかかる~~
- Exportに時間がかかる。
- データの配布

データフロー



Exportに時間がかかる。

日々増加するExport 時間

Exportに時間がかかる。

- 「 Maya® software → FBXファイル→Luminous用データ」という2段階のデータフローが後半ネックになった。
  - Maya® software → FBXファイル
    - Exportが重かった。
      ※特にSkinningデータ
  - FBXファイル → Luminous用データ
    - FBXファイルのロードにも非常に時間がかかった。

後半、最悪のケースでは、
3時間以上・・・

Exportに時間がかかる。

- 対策
  - 同一シーンの中のMaya® softwareファイルをEdit単位で分割
  - その分割したファイルはRuntime側で統合

Exportに時間がかかる。

- 結果
  - 一度にExportする量が減ったので、高速化できた。

問題は(ある程度)解決！

Exportに時間がかかる。

- 残った課題
  - シーン全体のExportには時間がかかるという問題は残ったまま
  - 全体データのExportには、1台のPCで一晩以上
    - 複数PCで分散するという力業で乗り切った。
    - 4台のPCで約4時間くらい。

Exportに時間がかかる問題 まとめ

- Export単位を細かく分けることで高速化。

ある程度解決

ワークフローで発生した問題

- ~~Maya® softwareのデータを開くのに時間がかかる~~
- ~~Exportに時間がかかる。~~
- データの配布

データの配布

- Agni’s Philosophyではデータの管理にPerforceを使っていた。

データの配布

- Perforceが一時期非常に重くなった。
  - 原因
    - PerforceのMetaDataをNFS上に置いていたこと。
    - 当時はまだ情報システム部にもPerforce運用のノウハウが十分になかった。

データの配布

- Perforceが一時期非常に重くなった。
  - 対策
    - 途中でServerのLocal Storage上にMetaDataを移動したら快適になった。

しかし・・・

Perforceが重かった問題

- NFSを解決後も、後半は重くなった。
  - 一つには、Perforceの使い方の問題
    - Workspaceを必要以上に作ったり・・・
    - Filteringをせずに全部ファイルを持ってきていたり・・・

負荷の高い使い方をしている人が結構いた。

Perforceが重かった問題

- さらに・・・
  - 巨大なデータをアップするときに問題発生
    - 一晩たってもアップが終わらず、結局一部のファイルはPerforceでの管理をあきらめた。
      - 例えば、一部のMaya® softwareのAnimationファイルなど

データフローのまとめ

- いったんFBXにExportすると言うのはネックになっていた。
- FBXの使い方がまずかった。
- データサイズの見積もりが甘かった。

本日の内容

- ~~データサイズ~~
- ~~データフロー~~
- 最適化

最適化

描画最適化

- 実行環境のマシンスペック
  - Geforce GTX 680 1枚
  - Main Memory 32GB
  - CPU Intel® Core™ i7-3770K 3.5GHz
    - Main Memory, CPUはとりあえず最高スペックのマシンにしたという感じで、フルに使い切ってはいない。

最適化の方針

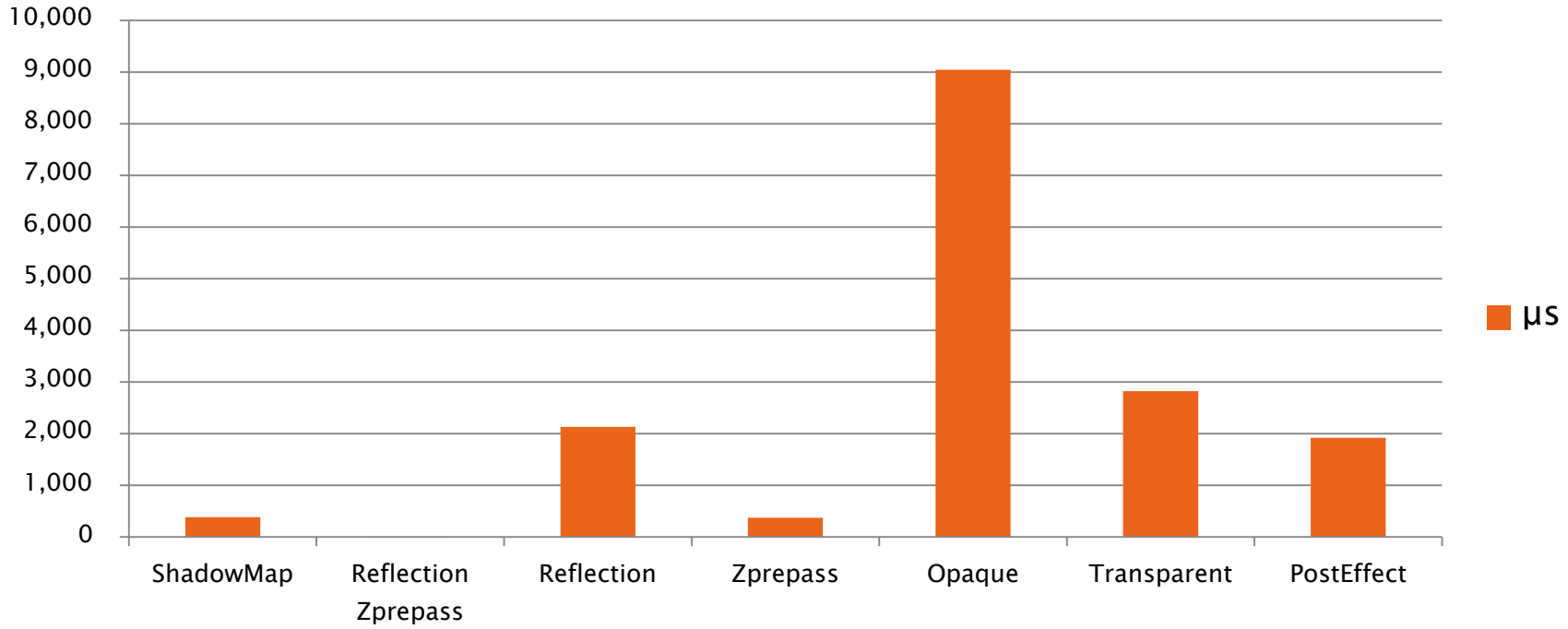
- Shaderには極力手を加えない。
  - 表現はまだまだ手を加えている最中だったため、変更するのが危険だった。
  - いったん、よく使われるShaderでFullScreenQuadを描いてみた。
    - 結果 5.8msだったので、このままでなんとかいけるはず、と判断。

とあるシーンのパフォーマンス

とあるシーンのパフォーマンス

- ShadowMap : 382μs
- Reflection Zprepass : 6μs
- Reflection : 2,132μs
- Zprepass : 370μs
- Opaque : 9,042μs
- Transparent : 2,825μs
  - ほとんど煙
- PostEffect : 1,921μs

とあるシーンのパフォーマンス



最適化でしたこと

- 効果が出たもの
  - Drawcall数を減らす
  - 頂点サイズを小さくする
  - Indexの並びの最適化
  - PreSkinning
  - ShadowMap用のMeshを用意
    - Artisにデータを用意してもらった。
  - Textureのサイズを減らす
    - 特にMain MemoryとのSwappingが発生しているケースで対応
- 効果があまり出なかったもの
  - Modelデータの頂点数を減らす。

最適化でしなかったこと

- Shaderの最適化
  - まだ開発が継続中だったので、Shaderをいじるのは危険だった。
- Modelデータの頂点数を減らす
  - やってもらっても効果が出にくかったので、途中でやめた。
- 頂点処理の最適化
  - LOD
  - Occlusion Culling
  - 最適化目的のTessellation
    - 背景に関しては頂点ネックになることがほとんどなかったため。

最適化でしたこと、の概要

最適化でしたことの概要

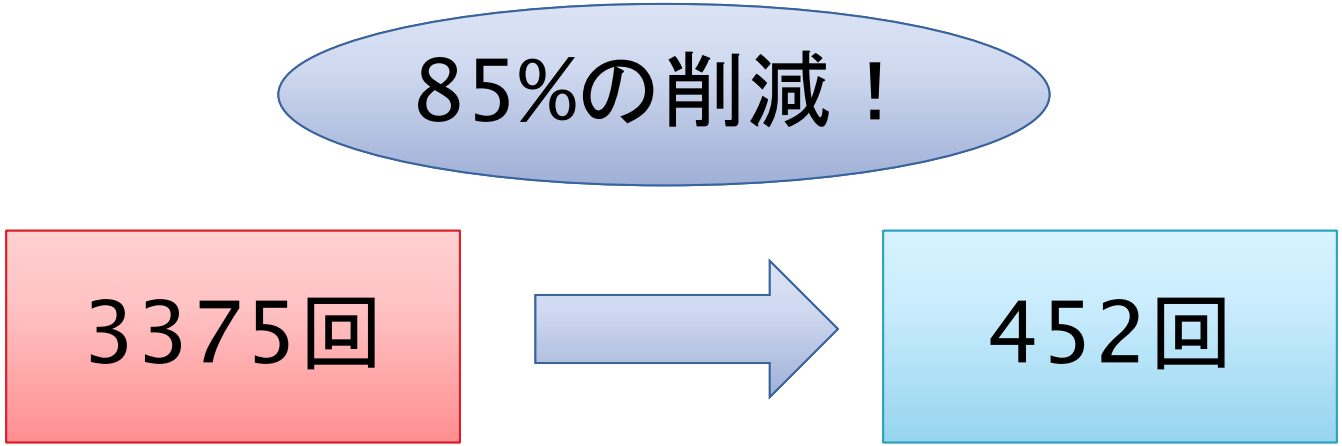
- Drawcall数を減らす
- 頂点サイズを小さくする
- Indexの並びの最適化
- PreSkinning

DrawCall数

- とにかくこれがネックだった。
  - 最適化前、DrawCall数は最初のシーンで3375回あった。

DrawCall数を減らす

- できるだけDraw Callをまとめた。



DrawCall数を減らす

- どうやって減らしたのか？
  - 単純な処理です。
  - 同じMaterialのものはまとめて描画するようにしただけ。

DrawCall数を減らす

- まとめるために、少し特別な処理をしたもの
  - GI Textureのみが違う場合
    - GIをTexture Arrayに入れて まとめて描画
  - Node Animationをしているもの
    - One Bone Animationとしてまとめて描画
  - Bone Animatoinをしているもの
    - 詰めるだけBoneを積んでまとめて描画

DrawCall数を減らす

- Unified Shaderがメインだったので、DrawCall数が劇的に減りました。
  - Texture Atlas化をすれば、さらに減らせる。
    - 事前の分析によると、最大93%
    - ただ、それをする前に最適化の目標値に達したので、しませんでした。

DrawCallをまとめた後

赤はBefore
青はAfter

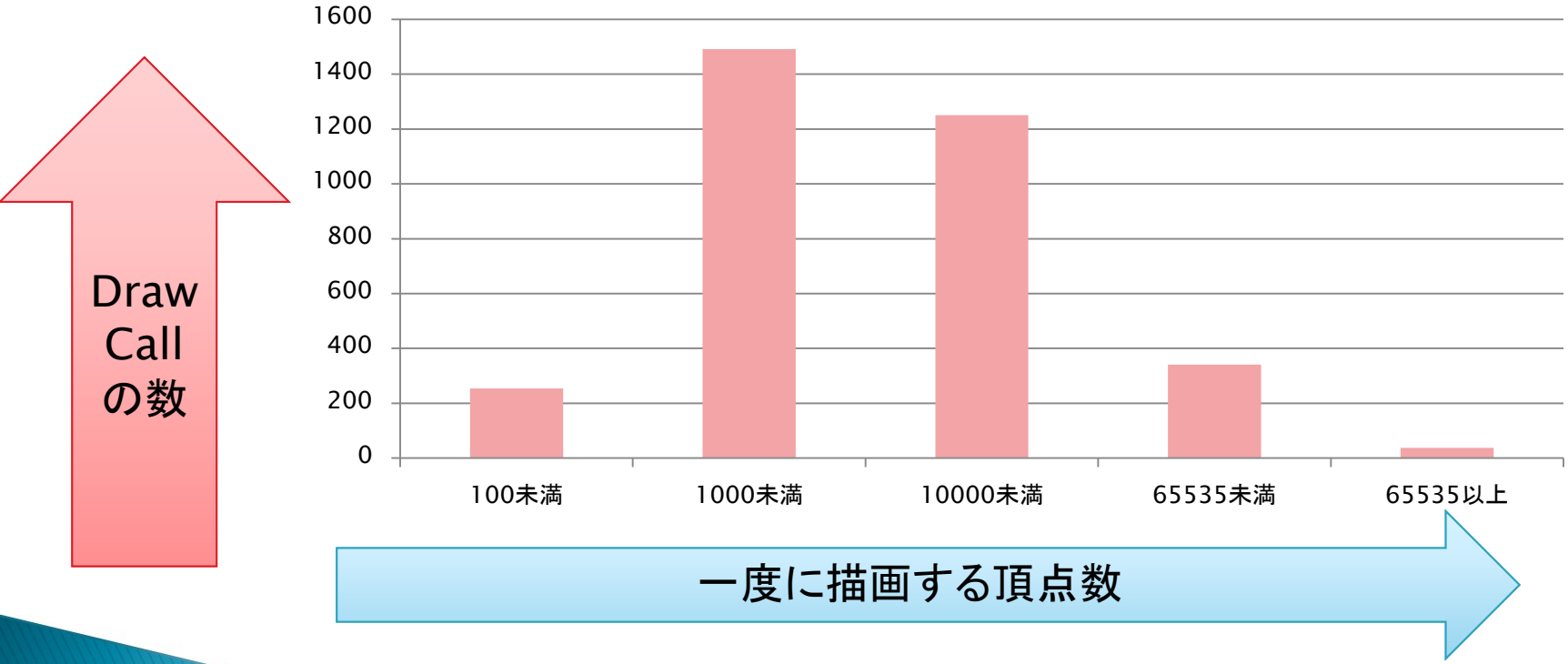
DrawCallをまとめた後

赤はBefore
青はAfter

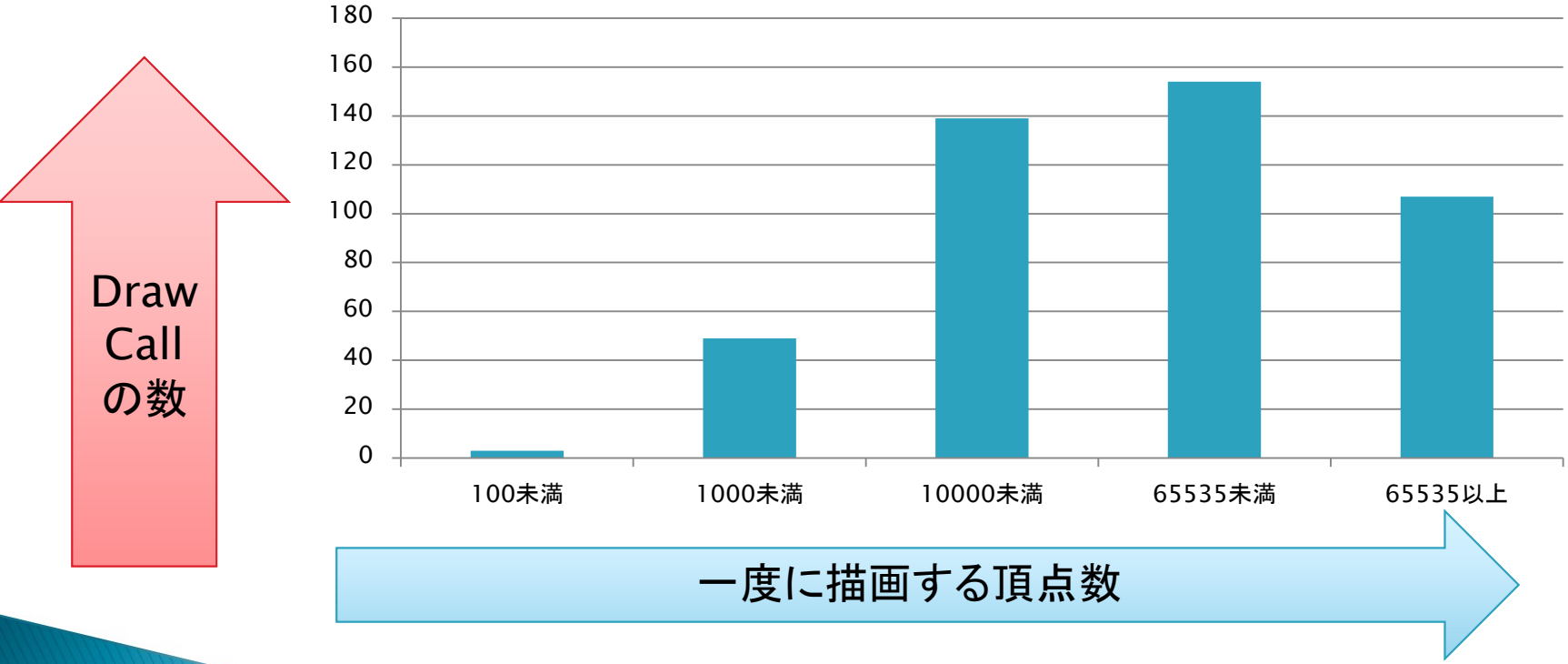
DrawCallをまとめた後

赤はBefore
青はAfter

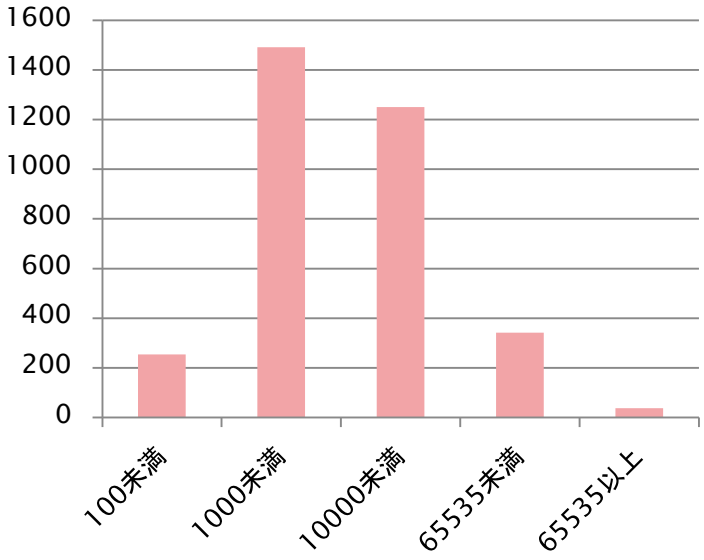
最適化前のDrawCall数



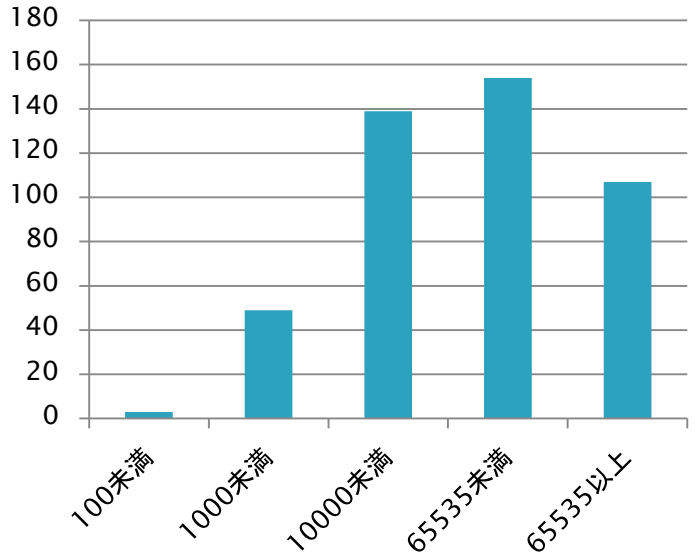
最適化後のDrawCall数



DrawCallをまとめる前後の比較

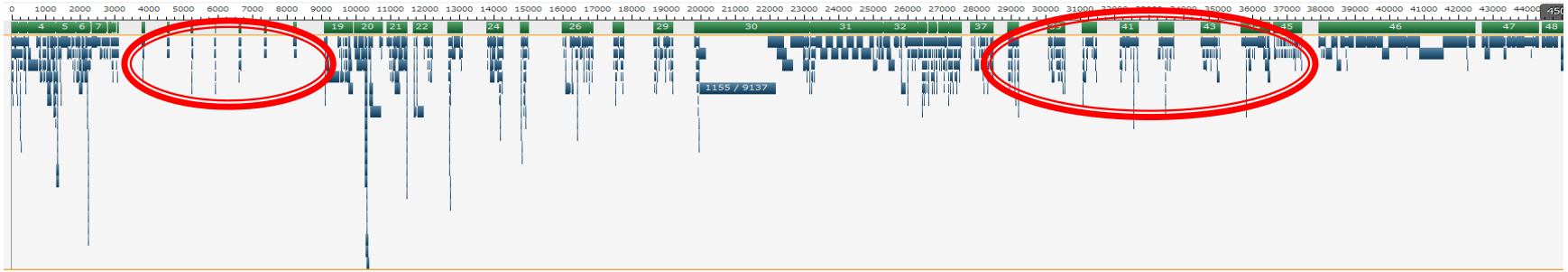


Before



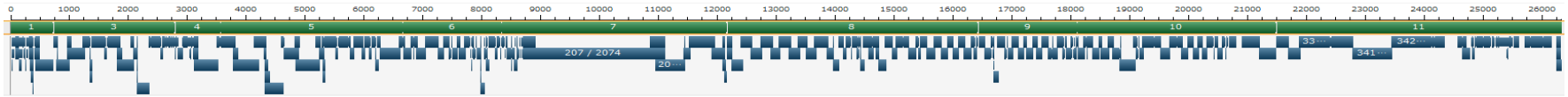
After

DrawCallを減らす前



GPUのStallが多く、スカスカ

DrawCallを減らした後



Stallがほぼなくなった。

DrawCall数

- DrawCall数を減らした結果
  - パフォーマンスは44msから25msへ
    - 40%くらい高速化

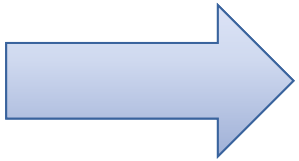
最適化でしたことの概要

- ~~Drawcall数を減らす~~
- 頂点サイズを小さくする
- Indexの並びの最適化
- PreSkinning

頂点サイズ

変わったところ

- 最初は56Byte
  - Position : float3
  - UV : float2 x 2
  - Normal : float3
  - Tangent : float3
  - Binorma sign : float
    - Normal Mapを反転して貼り付けたとき、法線が反転するかどうか？のフラグ



- 28Byteに縮小
  - Positoin : float3
  - UV : half2 x 2
  - Normal : byte3
  - Tangent : byte3
  - Binormal sign : byte
  - GI Index : byte
    - GI違いのModelをまとめて描画するため、GI用のTexture Array内Index

10%くらい早くなった。

最適化でしたことの概要

- ~~Drawcall数を減らす~~
- ~~頂点サイズを小さくする~~
- Indexの並びの最適化
- PreSkinning

Indexの並びの最適化

- 頂点キャッシュが効きやすいように事前処理でIndexを並び変えた。
  - シーンによっては5%くらい早くなった。

最適化でしたことの概要

- ~~Drawcall数を減らす~~
- ~~頂点サイズを小さくする~~
- ~~Indexの並びの最適化~~
- PreSkinning

Pre skinning

- Skinning結果をいったんVertex Bufferに出力してから複数Passを描画
  - Stream outputを使いました。
  - SkinningとShadowMapが多いシーンは5%くらい早くなった。

その他、色々細かい最適化

- PositionのStreamを分離
  - ShadowMap生成高速化
- Early Z
- Z Prepass
- Constant Bufferの更新頻度の見直し

ごく一般的なものです。

最適化でしなかったこと

最適化でしなかったこと

- Shaderの最適化
- Assetの最適化

Shaderの最適化

- 典型的なShaderの命令数は約3600
  - ただし、Shadow をオフにすると750くらい。
  - 実際には、Dynamic branchなどで3600がそのまま実行されるわけではない。
    - Cascadeの数, Lightの種類などでDynamic branchがありその分命令数が増えていた。

Shaderの最適化

- 機能全部入りShaderだったので命令数が増えていた。
  - 最適化のプランがあったが、それをする前に最適化の目標値に達したので、しなかった。
    - Convert時に使っていない機能は削除したShaderにアサインし直す予定だった。

最適化でしなかったこと

- ~~Shaderの最適化~~
- Assetの最適化

Assetの最適化

- 頂点処理はむちゃくちゃ早かった。
- 結果として、Assetの最適化はあまりしていません。

Assetの最適化

- Assetの最適化の余地はまだまだある。
  - 今回Pre-render用のデータをそのまま持ってきています。
  - 見えない場所にもデータがあったり･･･
  - そもそも頂点数が多かったり･･･

まだまだ減らせます。

Assetの最適化

- ただ、闇雲に頂点を減らしてもほとんど効果はなかった。
  - マイクロポリゴンが大量に発生しない程度の細かさなら、あまりがんばって頂点を減らしても、労力に見合わない印象。
    - ただし、静的な頂点に限ります。
  - 今回に関しては、マイクロポリゴン自体も、ほとんどパフォーマンスネックにならなかった。
    - シンプルなアプリケーションで検証したが、GeforceGTX680では、マイクロポリゴンもほぼネックになっていなかった。

どれくらい頂点数があるか？
実例

Assetの頂点数

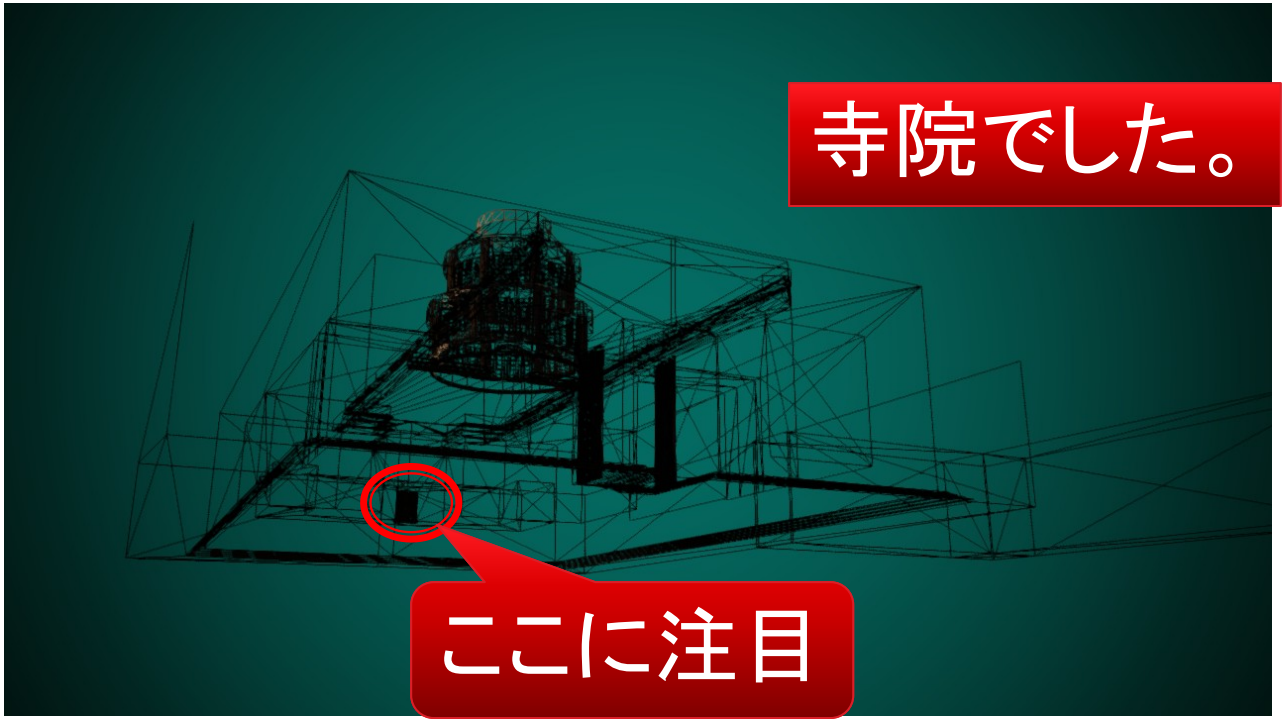
Assetの頂点数

Assetの頂点数

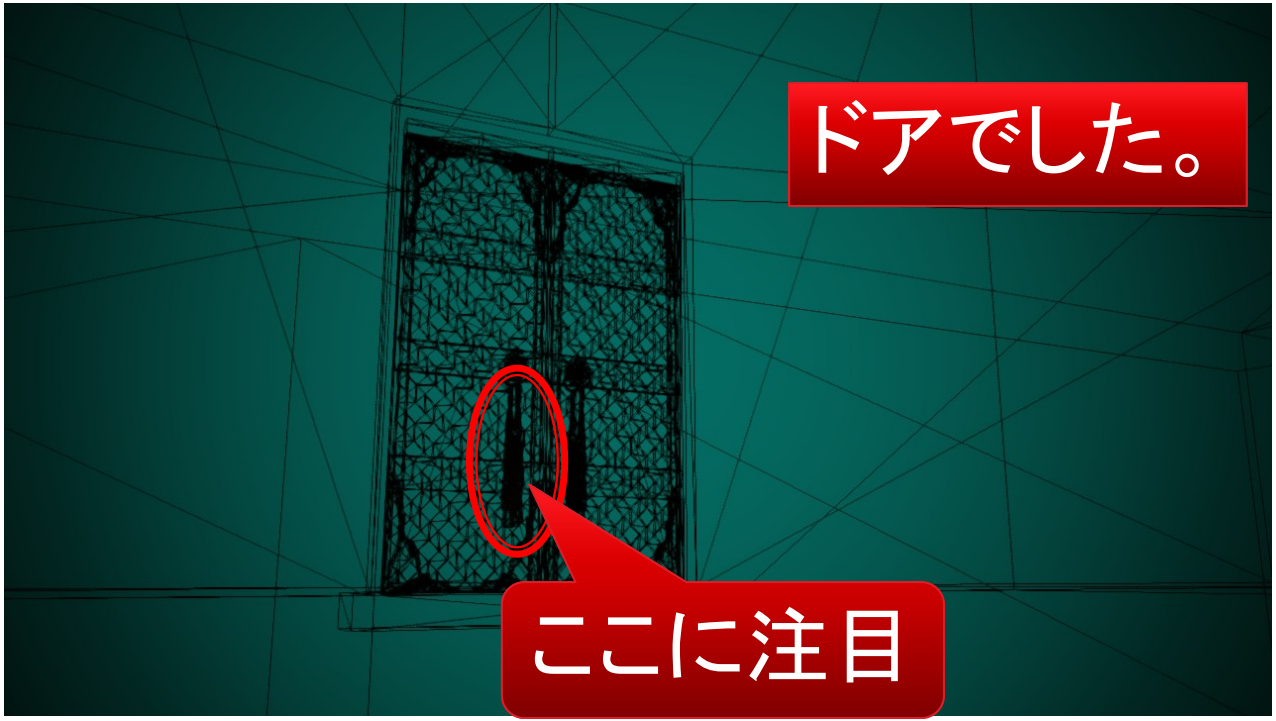
Assetの頂点数

Assetの頂点数

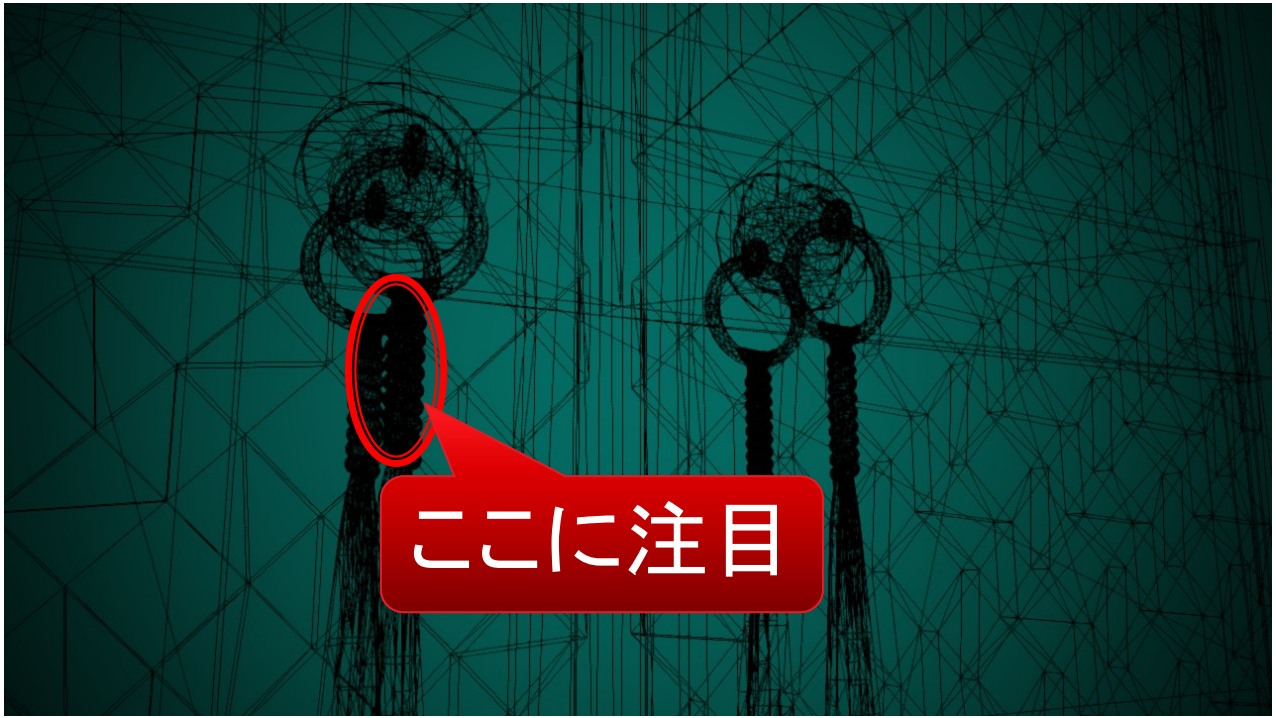
Assetの頂点数



Assetの頂点数

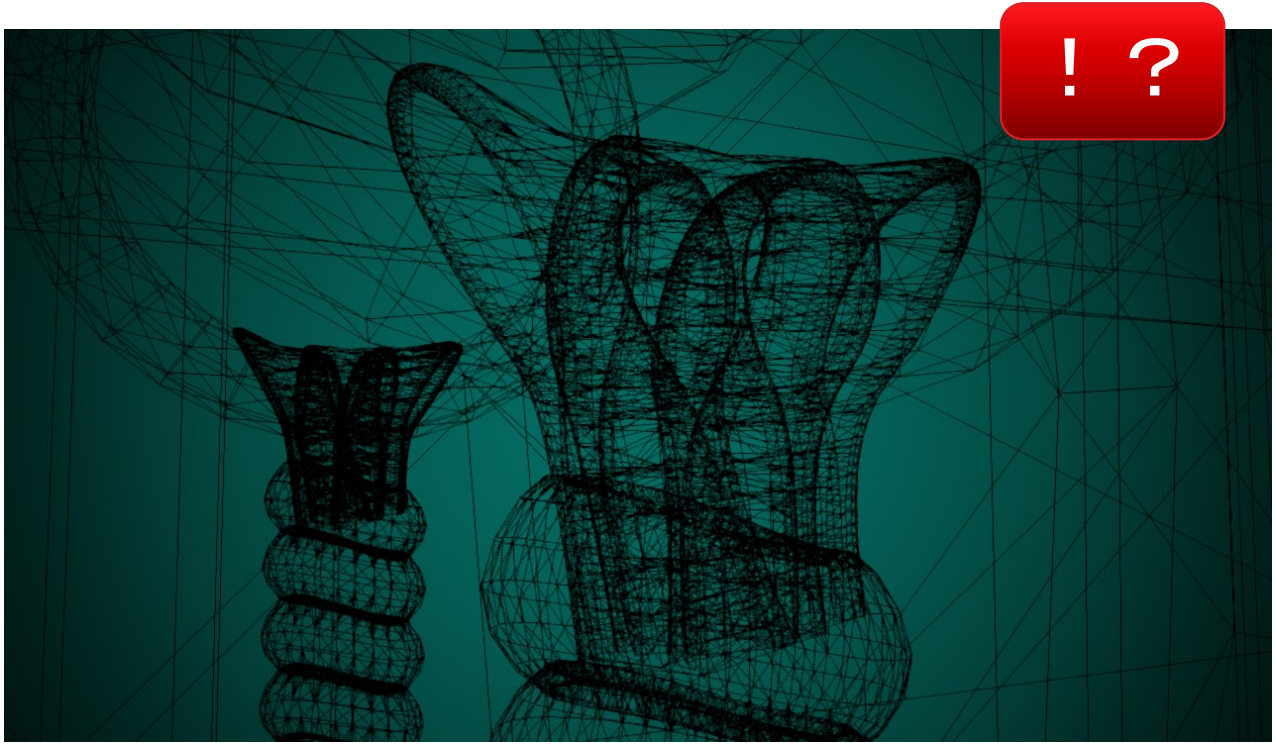


Assetの頂点数



Assetの頂点数

Assetの頂点数



Assetの頂点数

- このシーン、
  縄だけで
  約9万頂点使っています。

Assetの頂点数

きっとこの辺りに9万頂点

Assetの頂点数

- GPUの負荷は、90μs
  - ただし、全てEarlyZ Cullingされているため、PixelShaderは回っていない。
  - このシーンは既に最適化の目標値を超えていたので、そのままにしておきました。
- ALUの性能はかなり高速で、あまり神経質にMeshを削る必要がありませんでした。
  - もちろん、時間が許せば削った方が良いですが。

Assetの頂点数

- 今回はPre-render用のデータをそのままリアルタイムに持ってくるという流れだったので、無駄な頂点が多いです。
  - ただそれでも普通に動作するくらいHardwareの性能が高かったです。
  - 逆にがんばって頂点を削ってもあまり効果がなかった。
  - 今後は、静的なMeshについては、あまり頂点数を気にしなくてもいいかも？
- ただし、これはあくまでも地形の話。
  - Skinningをする場合は、頂点ネックになりましたので、頂点を削る、Pre-skinning, Tessellationなどは有効だと思います。

Assetの頂点数

- 今後は頂点を削るのは、ALU負荷が理由ではなくなるかも。
- むしろメモリ使用量、ストリーミング、作業効率の方が主な理由になるかも。

最適化でしたこと(復習)

- 効果が出たもの
  - Drawcall数を減らす
  - 頂点サイズを小さくする
  - Indexの並びの最適化
  - PreSkinning
  - ShadowMap用のMeshを用意
    - Artisにデータを用意してもらった。
  - Textureのサイズを減らす
    - 特にMain MemoryとのSwappingが発生しているケースで対応
- 効果があまり出なかったもの
  - 背景のVertexを減らす。

最適化でしなかったこと(復習)

- Shaderの最適化
- Modelデータの頂点数を減らす
- 頂点処理の最適化
  - LOD
  - Occlusion Culling
  - 最適化目的のTessellation

最適化まとめ

- 今回GPUはすごく早かった。特にALUは非常に高速。
- 頂点処理がネックになることはほぼなかった。
- とにかくGPUのStallを減らすことが大事、と言う印象。
  多少無駄にALUを走らせてもStallさせるよりまし。
- メモリの帯域ネックは結構起きた。

本日の内容

- ~~データサイズ~~
- ~~データフロー~~
- ~~最適化~~

全体のまとめ

- 最適化、ワークフローの問題は以下の2つが主な原因だった。
  - データサイズが桁違い
  - GPUのStall

全体のまとめ

- データサイズが桁違い
  - 無駄な頂点を気にせず作っても案外GPU側は処理できた。
  - ただ、GPUが処理できるデータにあわせて、データサイズを増やすと、開発環境側がデータサイズの増大に耐えられないかも。
    - Authoringデータであっても、データサイズを小さくする、メモリ使用量を減らす、などの最適化が必要。

全体のまとめ

- GPUはStallさえしてなければ、すごい量のデータを処理できた。
  - Stallさせない、というのは当たり前の話ですが・・・。

本日の内容

- ~~データサイズ~~
- ~~データフロー~~
- ~~最適化~~
- 今後に向けて

今後に向けて

今後に向けて

Q&A