『グランツーリスモ７』開発における USD 活用事例

2022年8月23日　(株) ポリフォニー・デジタル エンジニア 安藤恵美

POLYPHONY™ DIGITAL

自己紹介

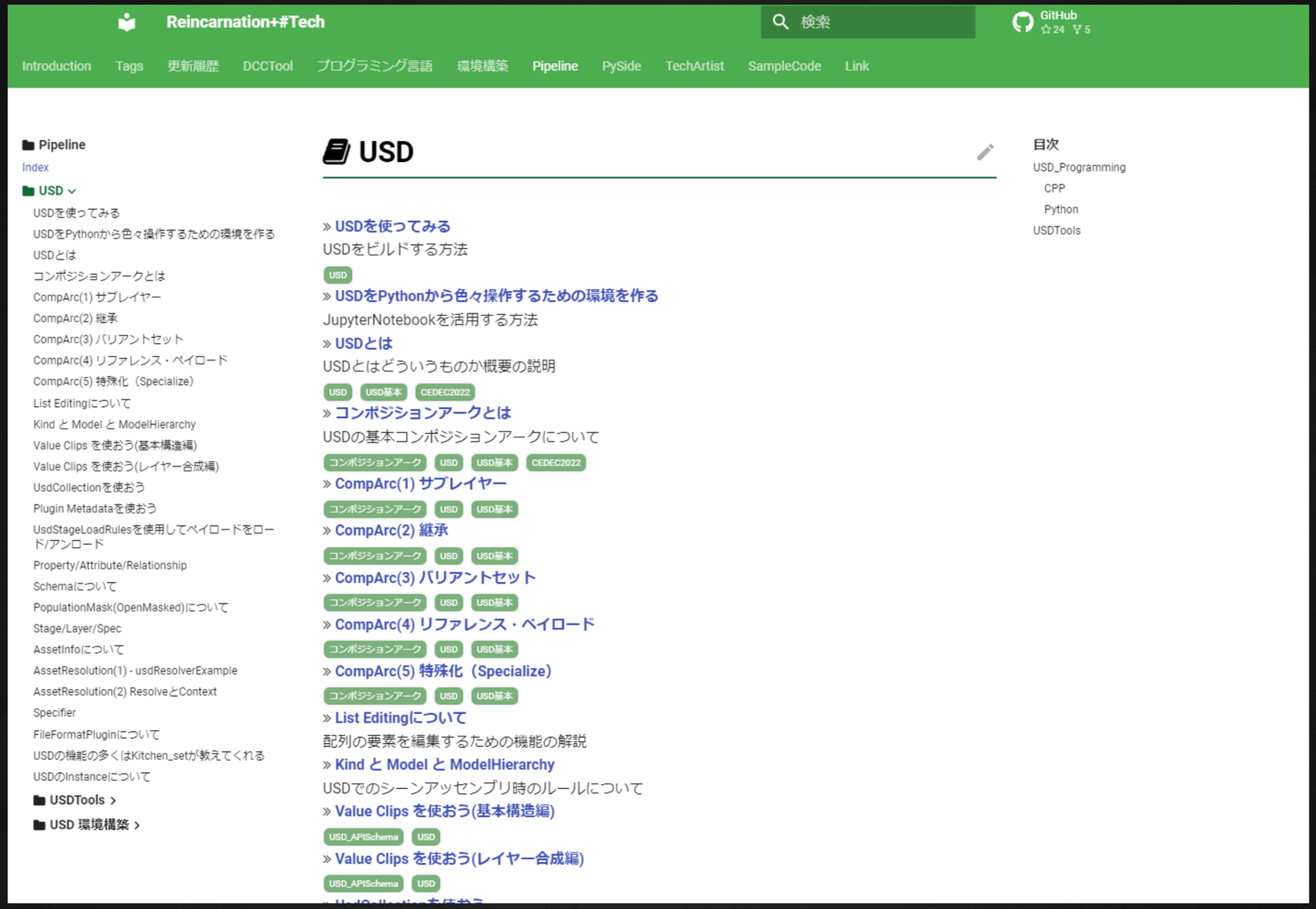
安藤　恵美 （あんどう　めぐみ）
ツールパイプラインエンジニア

テクニカルアーティストとしての業務経験を経て、
2019年6月　ポリフォニー・デジタルに入社
GT7 ではアセットビルドパイプラインの構築、アーティストのサポートなどを担当

趣味は一人旅とドライブとカメラと USD
2021年末にオープンカーを購入するくらいオープンカー大好き
2021年 USD アドベントカレンダー主催　USD 関連の記事多数執筆

POLYPHONY™ DIGITAL

Tech Blog



https://fereria.github.io/reincarnation_tech

POLYPHONY DIGITAL

USD 学習のきっかけ

- FBX があるのに、なぜ新しいフォーマットが話題になるのだろう？
  何を学習する必要があるんだろう？
  - 必要性がそこまで理解できなかった

- 2019年 の SIGGRAPH に参加したのがきっかけで USD の学習を開始
  - プロダクションでの使用例をセッションを通じて理解
  - いろいろできそうだから、調べてみる
  - コンポジションすごい Python API がすごい やりたいことが何でもできることに気づく
  - 開発された経緯を理解して、さらに USD にのめりこむ
  - USD 完全に理解した

今日話すこと

- GT7 について
- GT7 の3つのチャレンジ
- USD について
- GT7 と USD
  - アセットビルドパイプラインでの活用
  - コースのレイアウトワークフロー
  - クルマカスタムパーツの作成
- まとめ
  - USD が描く未来

POLYPHONY™ DIGITAL

Gran Turismo 7

POLYPHONY™ DIGITAL

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR


Menu Book
--
Credits
100,000
NSX Type R '92
MR / 315 BHP / 1,230 kg / NA
PP 473
Daily Workout
0.0 Miles
To Next Level
1,489 Exp.
15
17:28
2 Sep
CAR CUSTOMISATION
Captured on PS5
Learn More
Wheels
Paint Colour
Custom Parts
Racing Items
Other
Livery Editor
Create Style
Load Style


Trial Mountain
Captured on PS5
GRAN TURISMO®
THE REAL DRIVING SIMULATOR

東京駅丸の内口駅前広場

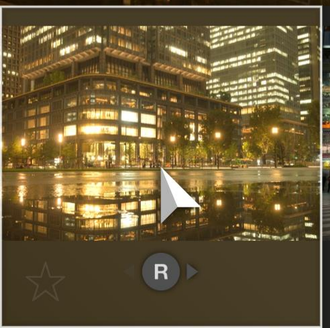
東京 - 日本

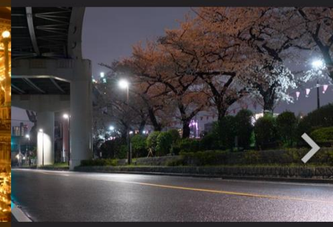
☆お気に入り

東京駅丸の内口広場から見た丸の内のビル群。明治初期には陸軍省と関連施設が置かれたが、移転によって三菱商会率いる岩崎家に払い下げられ、以降ビジネス街として発展を遂げた。正面は丸の内ビルディング。

1/12

選択する　お気に入りに追加





+2MB
+16GMB

L1 クルマ カメラ エフェクト R1

撮影する

カメラ設定

絞り F 1.4
フォーカス シングルAF
背景ぼかし 2 ふつう
露出補正 EV 0.0
シャッタースピード 1 / 60 秒

流し撮り設定

流し撮り設定 オフ
ターゲット車 86 GT '15
カメラ追尾率 90%

環境設定

アスペクト 3:2
構図 横構図
解像度 高解像度
グリッド オン
ガイダンス オン
詳細設定

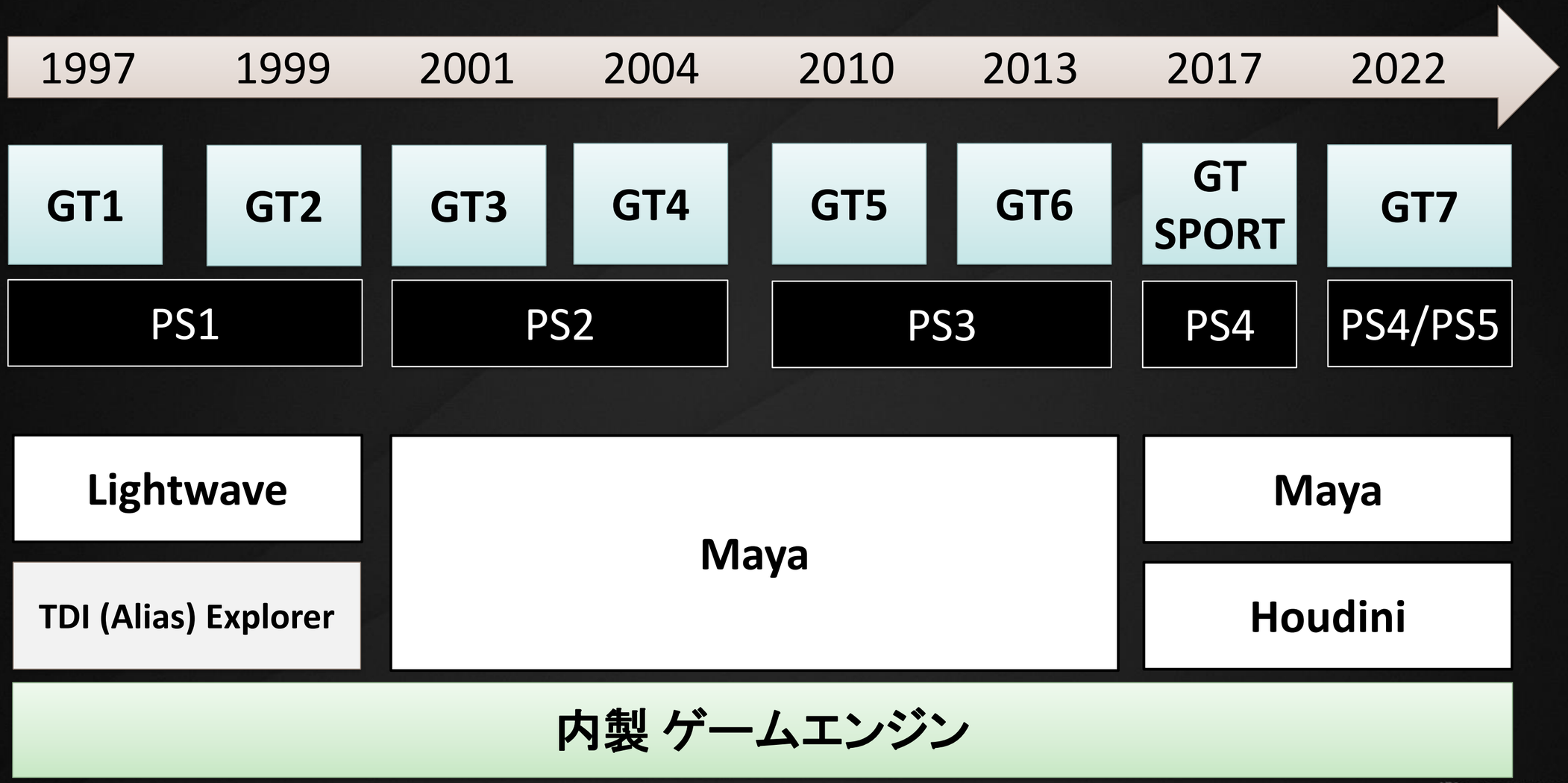
メニュー操作 ズーム／ロール 方向 撮影する フォーカス 拡大


GRAN TURISMO
AISIN
KOITO
DENSO
ZENT
KYB
SARD
NPR

Gran Turismo 7　アセット制作

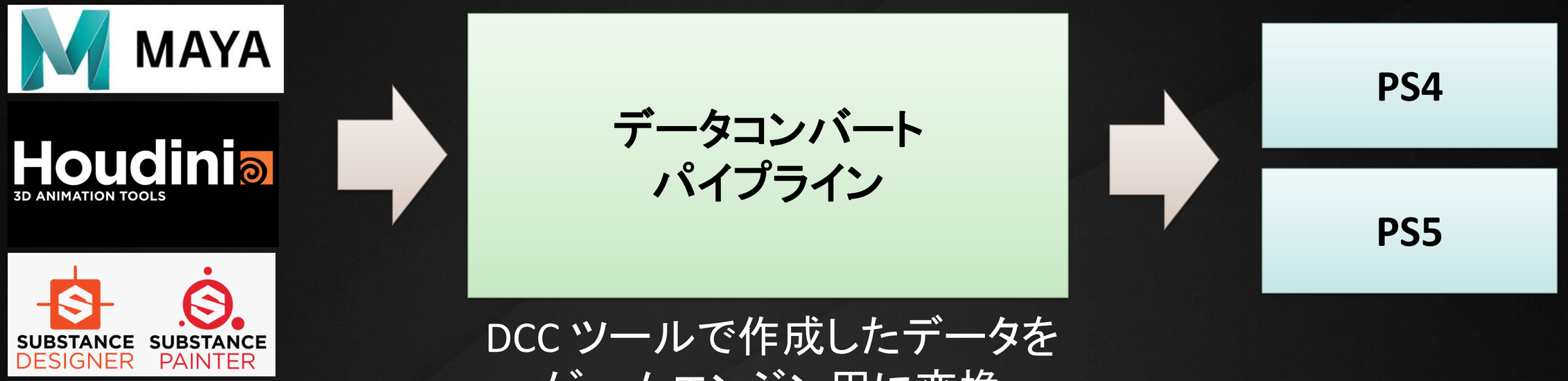
POLYPHONY™ DIGITAL

Gran Turismo シリーズ



POLYPHONY™ DIGITAL

Gran Turismo 7



DCC ツールで作成したデータを
ゲームエンジン用に変換

- 内製エンジンと、独自の巨大なデータコンバートパイプライン
  - クルマやコース、ドライバーなど　アセットの種類ごとに個別のパイプライン
  - プロジェクトの進行とともに拡張・進化
  - マルチプラットフォーム

POLYPHONY™ DIGITAL

GT7 アセット制作の3つのチャレンジ

チャレンジ1：様々な独自フォーマット・歴史的仕様の資産・負債

チャレンジ2：膨大な物量

チャレンジ3：大規模なチーム

POLYPHONY™ DIGITAL

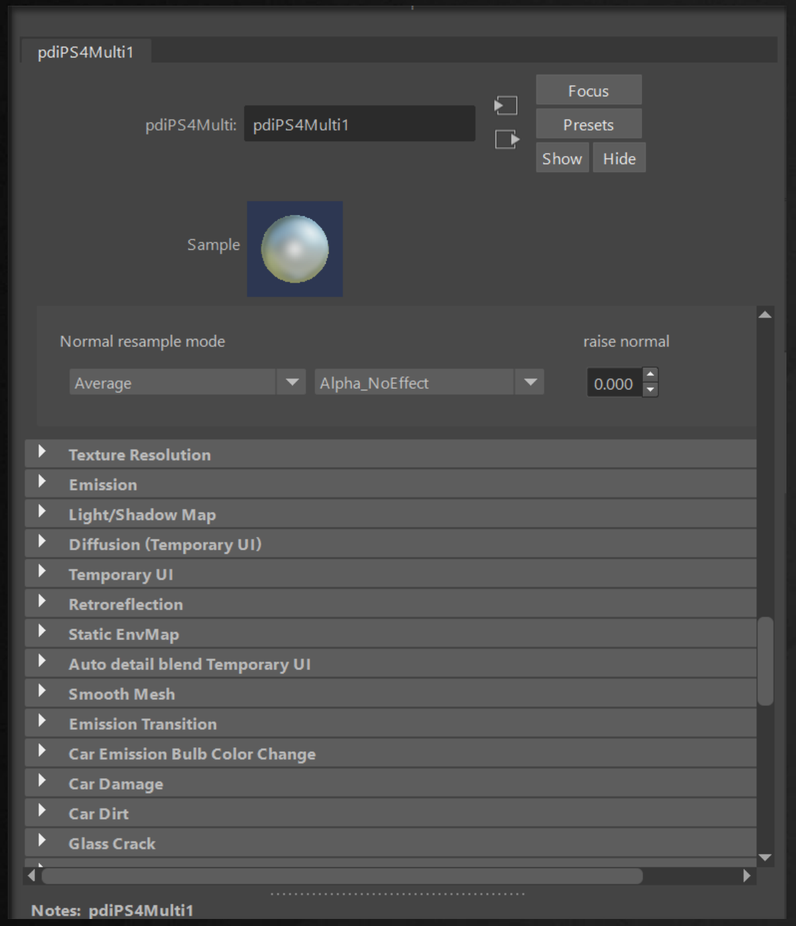
チャレンジ1 歴史的仕様の資産・負債

- 25年の歴史の中で積み重ねられた様々な仕様やデータ形式が混在
- 過去の資産と負債が多く存在
  - 複雑なデータ仕様
  - これまで制作された大量のアセット

→ 一度に新しい環境への移行が難しい

→ 複雑な仕様を把握しつつ、パイプラインを拡張するのが難しい

チャレンジ1 資産と負債:シェーダー



- PS4/PS5 用のシェーダー
- アーティストは Maya のビューポートでエディット
- 最適化したものをゲームエンジンで描画
- 描画用以外にも路面やコリジョン情報など、ゲームプレイにかかわる設定も存在
- 大量の設定があり、非常に複雑
- 新たな DCC ツール導入の障壁

POLYPHONY™ DIGITAL

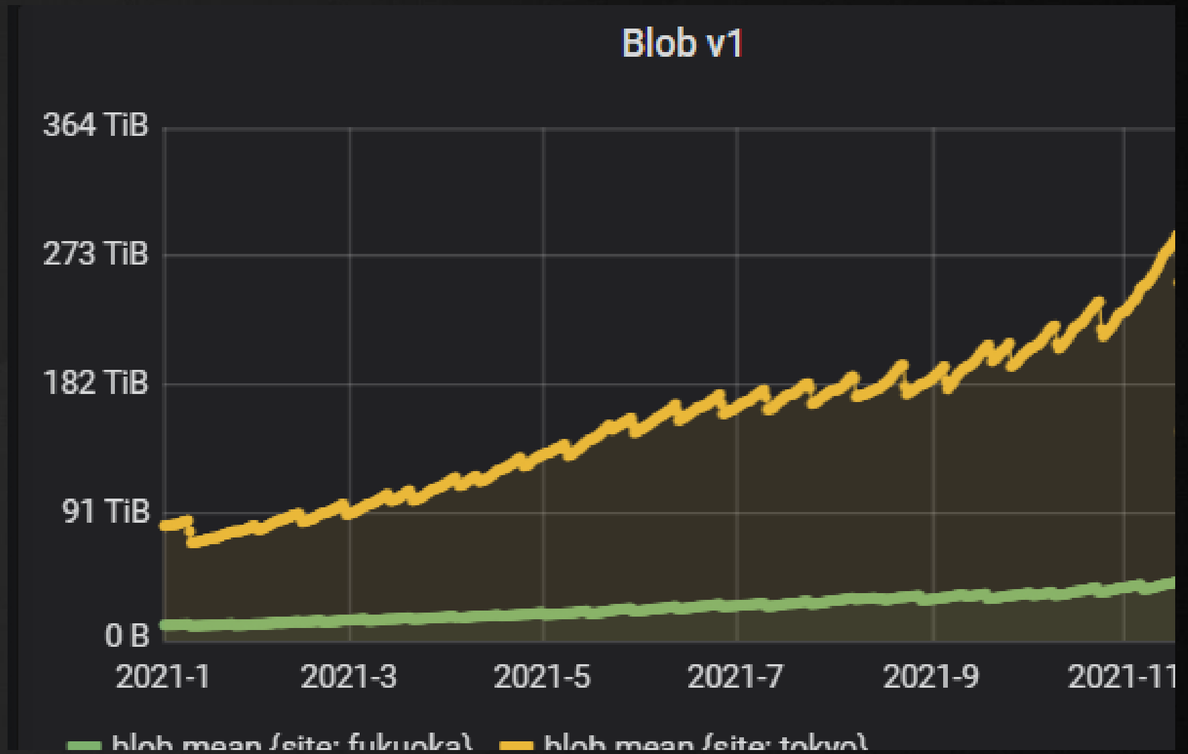
チャレンジ1 資産と負債:独自のファイル形式

- BMF という独自の 3D データファイルフォーマット
- GT5 の頃から拡張を重ねて使用
- 主にコンバートパイプラインの中間ファイルとして使用

- 独自フォーマットでエコシステムが組み立てられており、
  新たな DCC ツールへの入出力対応や、プレビュー環境を更新するのが大変
- AOS 表現のためデータの肥大化につれ、ロードに時間がかかる

チャレンジ2　物量

- 100 レイアウトのコース
- 400 以上のクルマ
- 2500 以上のスケープススポット
- PS4 PS5 とも 100GB のパッケージサイズ
- 毎日の perforce の submit が 2TB
- 毎日のアセット全ビルドで生成される
  ファイル数は 10万単位



POLYPHONY DIGITAL

チャレンジ3 多人数・同時並行・アジャイル開発

- 複数拠点・多人数開発
  - 東京、福岡にメインのスタジオ
  - 57人のエンジニア
  - 133人のアーティスト
  - 北米、欧州にも複数の拠点
  - リモートワーク
- 同一アセットに対して複数人で同時作業
  - 例：クルマの場合
  - 3D スキャン・測色・Maya ボディモデリング・内装モデリング・Houdini プロシージャル...
- マスター直前までエンジニアと共同作業し、ゲーム仕様を練り直し、ゲームエンジンを拡張しながら日々進化を継続
  - 仕様ありきのウォーターフォール型開発は適さない

POLYPHONY™ DIGITAL

クルマコンバートに必要な処理

- Maya シーンのエクスポート
- ドライバー制御情報生成
- ライトパラメータ生成
- 測色データインポート
- カラーバリエーション生成
- 動的ライティング情報の事前ベイク
- デノイズ
- タイヤコリジョン SDF 計算
- シャドウモデル生成
- ワイパー領域生成
- LOD 生成
- スムーズメッシュ変換、リダクション
- シェーダコンパイル
- 頂点データ変換
- テクスチャ変換
- パーツフィッティング
- 諸元計算
- コリジョン生成
- サイズチェック、バリデーション

とにかく処理が多い

POLYPHONY™ DIGITAL

コースコンバートに必要な処理

- Maya エクスポート
- ゲーム仕様の追加処理
  - LOD
  - 影モデル
  - デジタルフラッグ
  - アニメーション
  - BoundingBox
- 路面コリジョン・物理属性
- オクルージョンカリング情報
- 可視物体の事前計算（PVS）
- ベイク
- ライトプローブ
- 木・草のレイアウト
- コースエフェクト（煙、火花、花火）
- 天候関係の情報追加
- ストリーミングデータの作成
- インスタンス処理
- 動的コースレイアウト切り替え
- ・・・

とにかく処理が多い

POLYPHONY™ DIGITAL

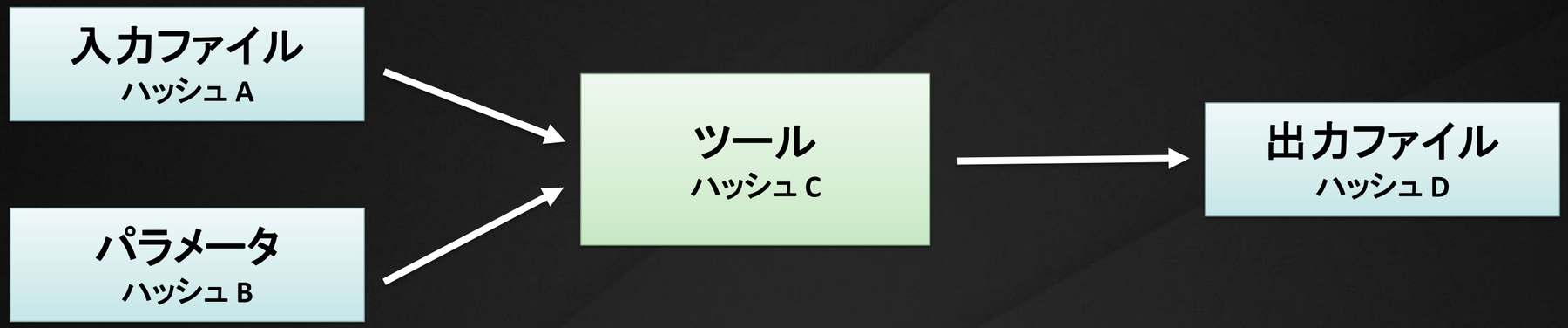
Transform：パイプラインの最小単位

ベイク処理の例

```
python –m iris --parallel 8,0 --iter 1024 --iterAO 512 ... input.bmf output.bmf
```

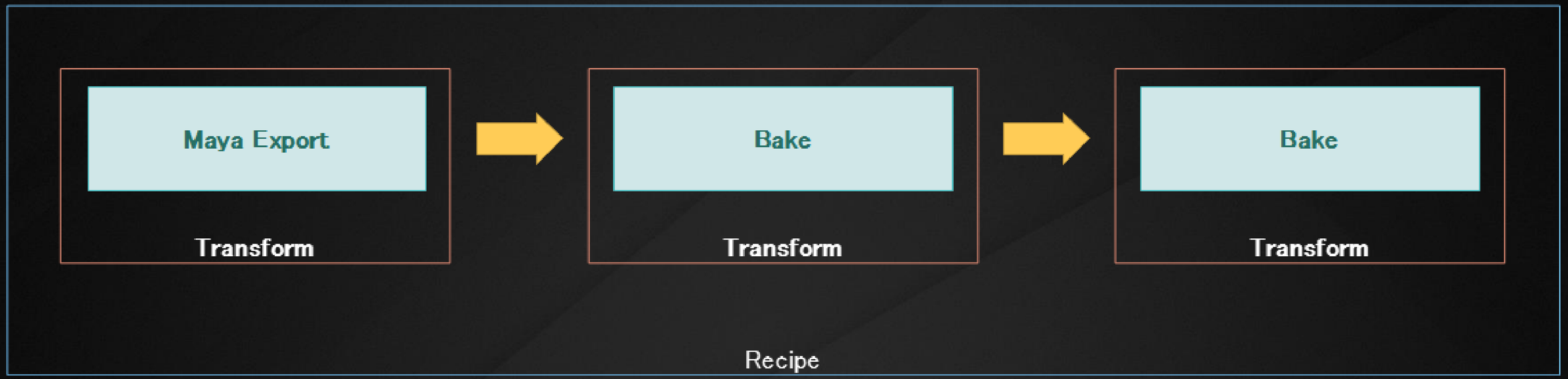
- アセットビルド処理の最小実行単位
- 典型的には実行ファイルと入力ファイル、パラメータから成る
- Json でシリアライズする

Transform : 再現可能な処理単位の定式化



- 同一の入力に対しては同一の結果を出力することを保証
  - ハッシュ A, B, C からは必ず D ができる
  - シリアライズ json には全入力データの fingerprint が含まれる
- KVS に入れてコンバート処理の部分的な再利用や共有が可能
- パイプラインのデバッグが容易になる

Recipe： Transform の依存関係の記述

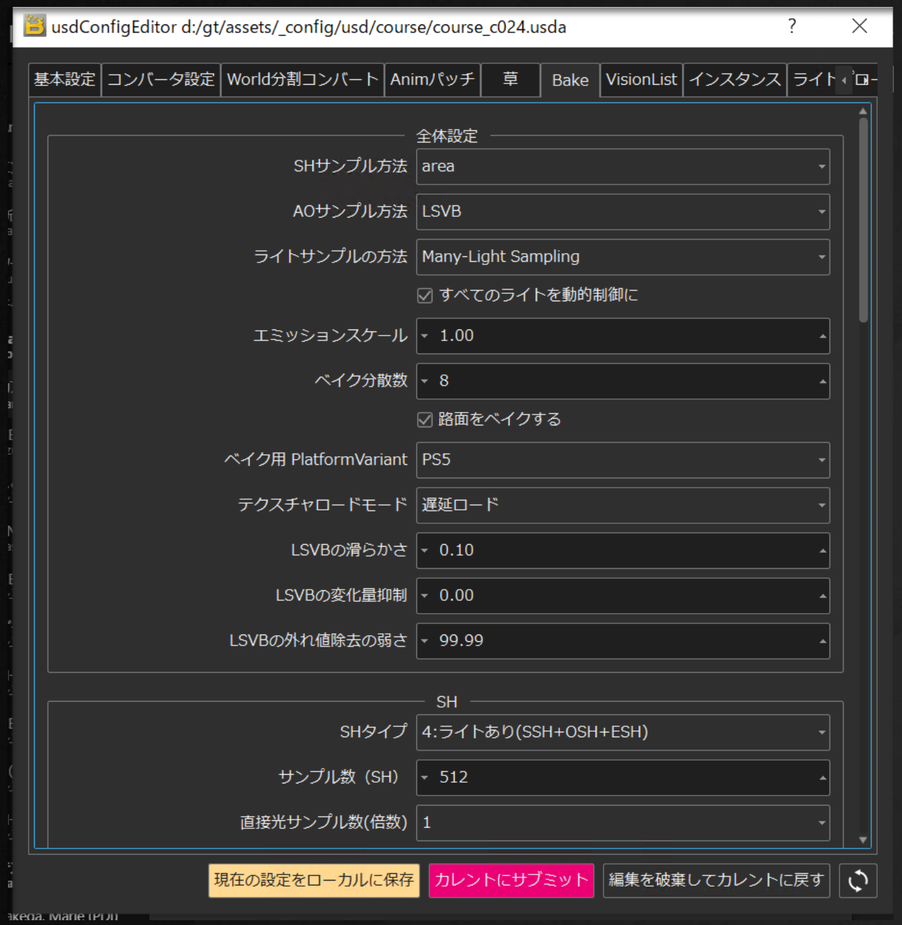


- あるアセットをゲームエンジン用データに変換するための複数の Transform とその依存関係の記述
- 例） CarRecipe、CourseRecipe、DriverRecipe
- transform の集合として json で記述する

レシピの構築

```
def InstanceBakeProbe(options, toolchain):

    if IsInstanceSuport(options) and not options.RuntimeLightProbeBind.value:

        transforms = []

        builder = TransformBuilder(Transform('Bake Instance Probe'))
        builder.setExe(toolchain.Hython, dispatchable=True, houdini=185696, rank=8,
                       cpuMem=options.kobitMinimumRequirementCPUMem.value)
        builder.muteIf(options, 'lock_tree')  # tree_layout.bin をロック

        builder.addInputs("<SRC>/houdini/houdini-py3-18.5/otls/PDI_crs_IBLProbeInstanceBake.hda")

        worldLightPath = "<BUILD-IN>/world_light.bmf"
        if options.CourseMaker.IsCourseMaker.value and len(options.LightProbePartials.value) < 1:
            worldLightPath = "<BUILD-IN>/all_rail_light_opt.bmf"

        builder.addArguments('-m gt.pipelineCourseIBLProbeInstance')
        builder.addArguments('--world <BUILD-IN>/world_opt.ibl.bmf')
        builder.addArguments('--rail <BUILD-IN>/all_rail_opt.ibl.bmf',
                             condition=options.CourseMaker.IsCourseMaker.value)
        builder.addArguments(f'--light {worldLightPath}')
        builder.addArguments('--instance_json <BUILD-IN>/tree.json')
        builder.addArguments('--instance_bmf <BUILD-IN>/tree.bmf')
        builder.addArguments('--instance_lp <BUILD-OUT>/tree_lp.json')
        builder.addArguments('--ignore_group', condition=options.BakeInstanceProbeIgnoreGroup.value)

        transforms += builder.transforms

        if IsLayoutBasedInstance(options) or (options.InstanceSupportTree.value and options.InstanceSupportTreeHoudini.va
            builder = TransformBuilder(Transform('Bake Instance Probe ps4'))
```

コンバートパラメータ



ベイクパラメータの設定 GUI

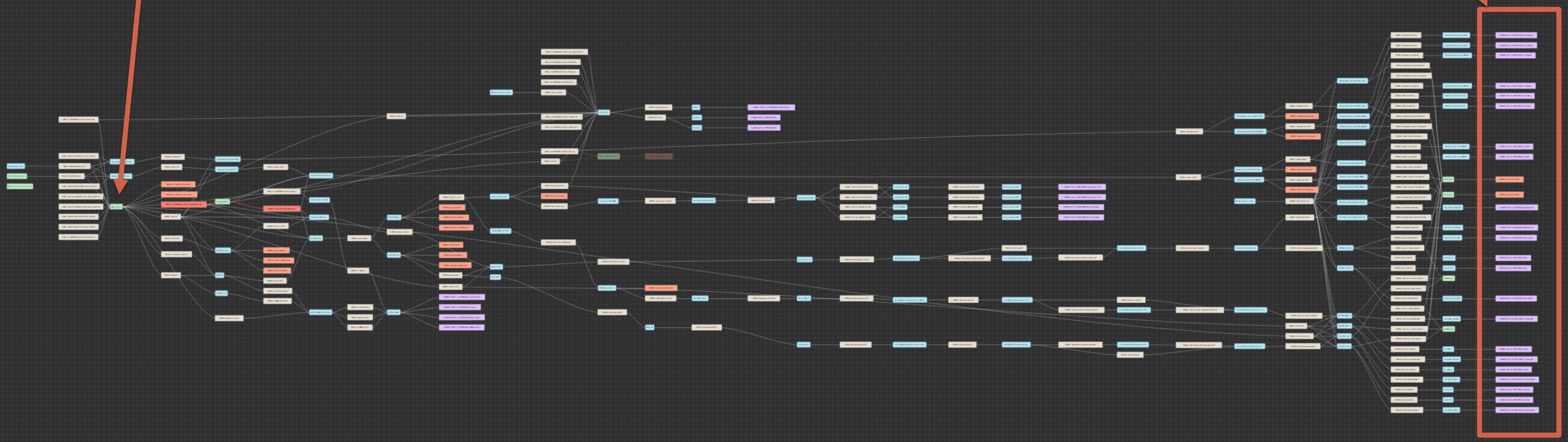
- DCC ツールの出力するデータ以外に、Transform のパラメータもアセットごとに設定
- Transform パラメータを調整する人
  - アセットの担当アーティスト
  - TA
  - ツールを実装するエンジニア
  - トラブルシューティングするエンジニア
- 同時並行で編集
- 新たな機能は日々追加されていく

POLYPHONY™ DIGITAL

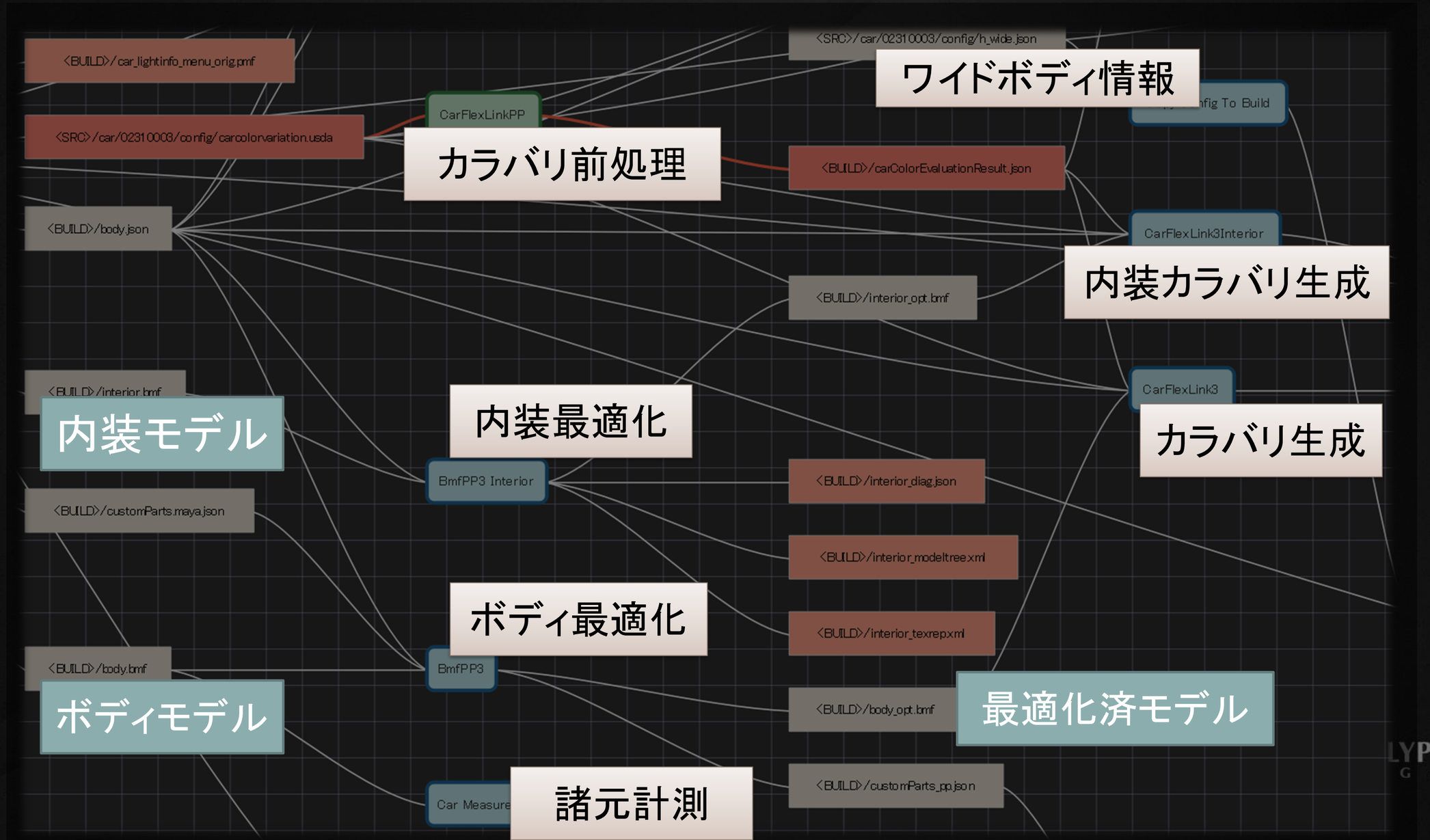
クルマコンバートの例

Maya
シーン

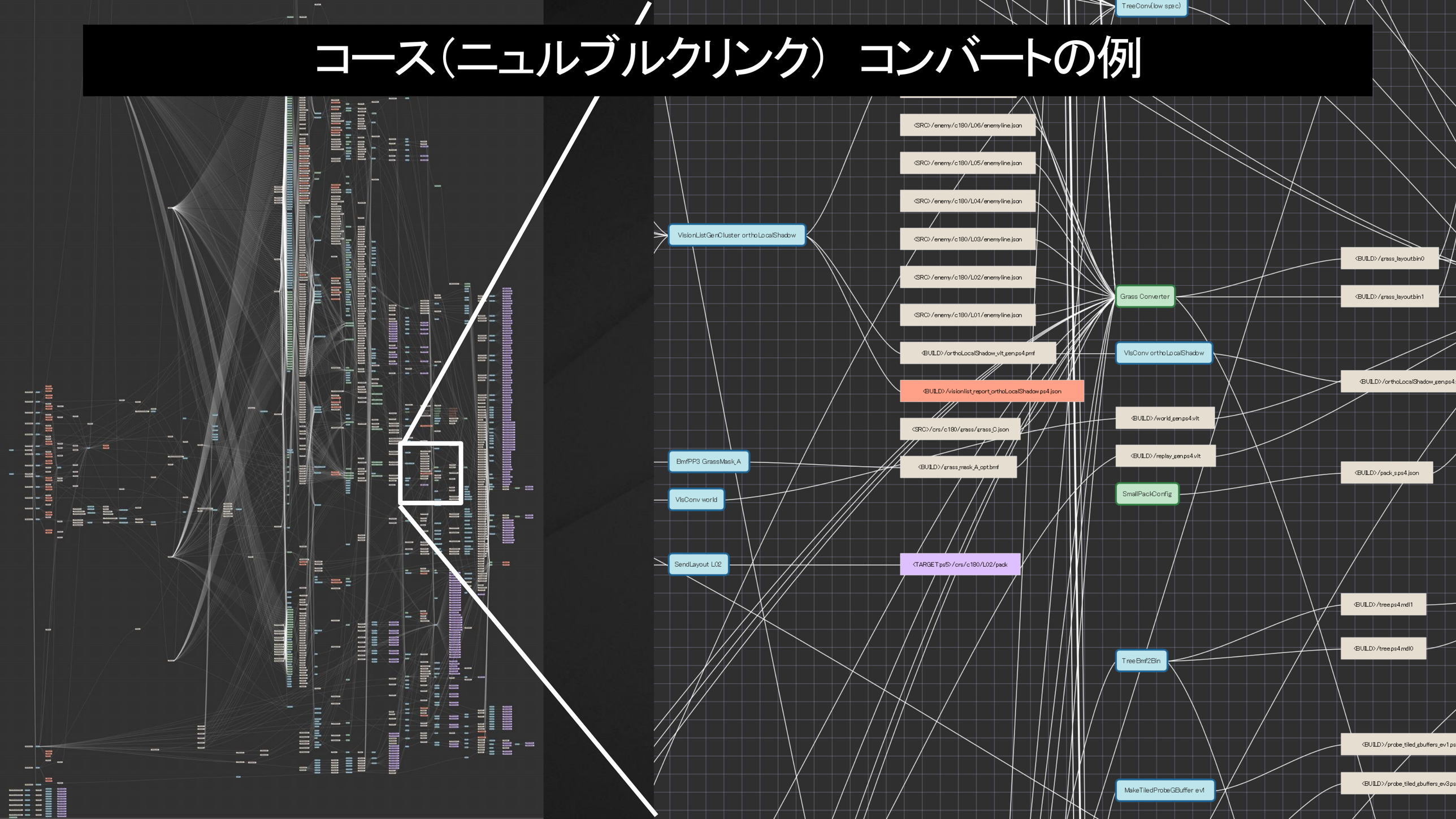
PS4/PS5
データ



クルマコンバートの例


<BUILD>/car_lightinfo_menu_orig.pmf
<SRC>/car/02310003/config/carcolorvariation.usda
<BUILD>/body.json
<BUILD>/interior.bmf
<BUILD>/customParts.maya.json
<BUILD>/body.bmf
Car Measure
CarFlexLinkPP
BmfPP3 Interior
BmfPP3
<SRC>/car/02310003/config/h_wide.json
<BUILD>/carColorEvaluationResult.json
<BUILD>/interior_opt.bmf
<BUILD>/interior_diag.json
<BUILD>/interior_modeltree.xml
<BUILD>/interior_texrep.xml
<BUILD>/body_opt.bmf
<BUILD>/customParts_pp.json
nfig To Build
CarFlexLink3Interior
CarFlexLink3
ワイドボディ情報
カラバリ前処理
内装カラバリ生成
内装モデル
内装最適化
カラバリ生成
ボディ最適化
ボディモデル
最適化済モデル
諸元計測

コース（ニュルブルクリンク） コンバートの例



コンバートパイプラインの最適化

- コンバートキャッシュ
  - 一度実行した transform を誰かが実行済みならば、結果だけ利用
- 分散実行
  - コンバートに必要な transform を、複数の PC に分散して実行
- コンバートパラメーターのプリセットの活用
  - 軽量ビルド版　のようなプリセットを使用することでプレビュー用に高速化

それでも解決しきれない

- パイプラインはエンジニアが毎日のように拡張
- 古くからある資産も互換性を維持しつつ動かさなければいけない
- 複数人が同時並行でアセットビルドをする
- それを数千数万のアセットに矛盾なく適用する

- 当初は TOML を使用して設定を管理しようとしたが失敗
  - 互換をとりにくい
  - GUI 定義などの拡張がしにくい
  - 複数人で編集すると設定ファイルが競合してしまう

正しく動かすには「インターチェンジ」「スケーラビリティ」「同時編集」を求められる

POLYPHONY™ DIGITAL

USD

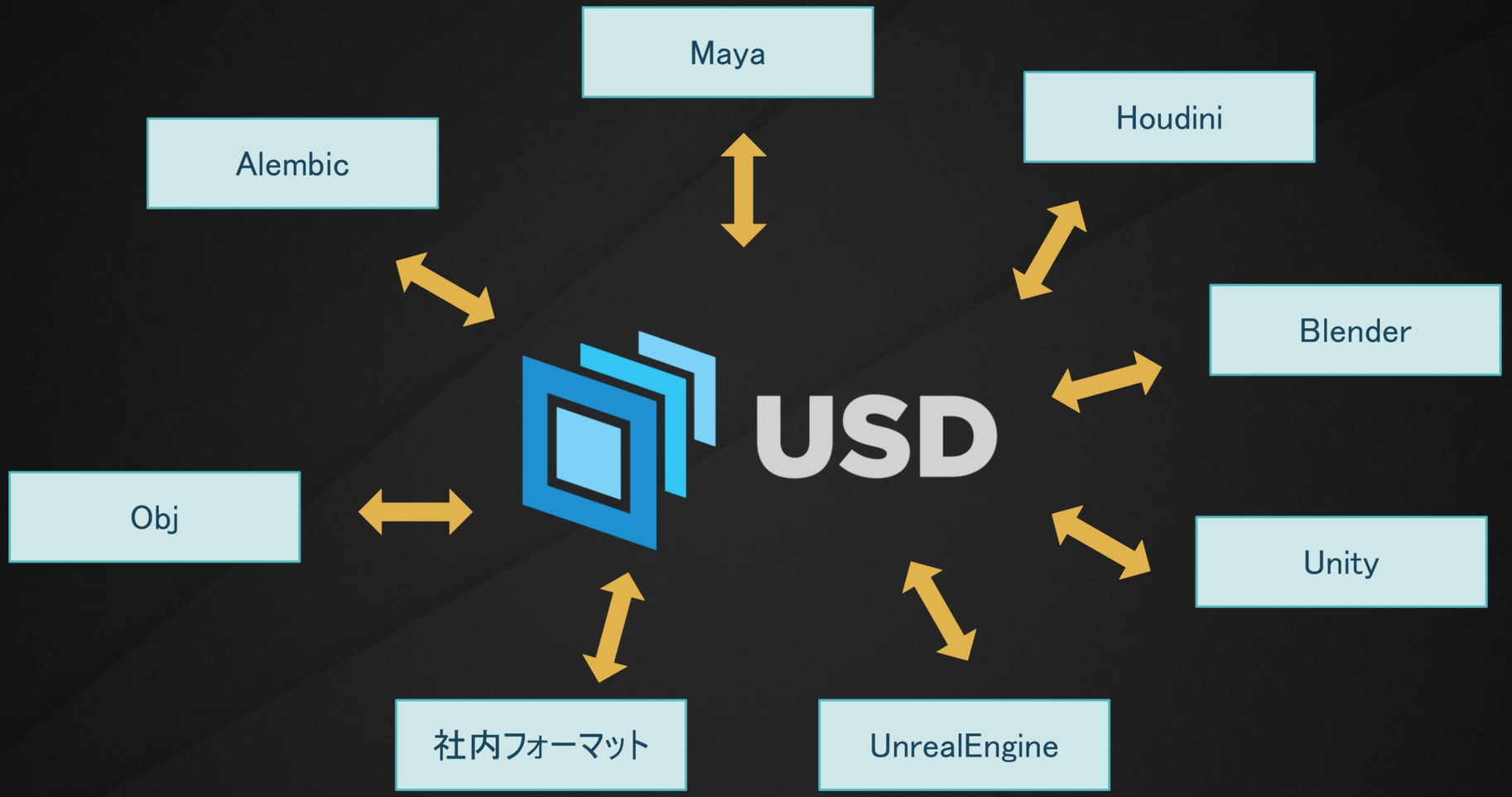
Universal Scene Description

映像制作用の 大規模・多人数・多様なアセットパイプラインを支える新技術

- 「インターチェンジ」
- 「スケーラビリティ」
- 「同時編集」

この3つを備えた仕組み
ここ数年で主に映像制作のためのテクノロジーとして急速に普及

インターチェンジ



汎用コンテナとしてのUSD

Prim と Schema

- Prim
  - USD のコンテナオブジェクト
  - アトリビュートを定義することができる

- Schema
  - Prim に対して、「メッシュ」や「トランスフォーム」などの 3DCG で一般的に使われるデータ構造の定義
  - Schema を、各 DCC ツールが解釈することで
    DCC ツール内では異なる表現であっても、シーングラフを相互互換して利用できる
  - Custom Schema を定義することで
    ゲームエンジン固有の情報も扱うことができる

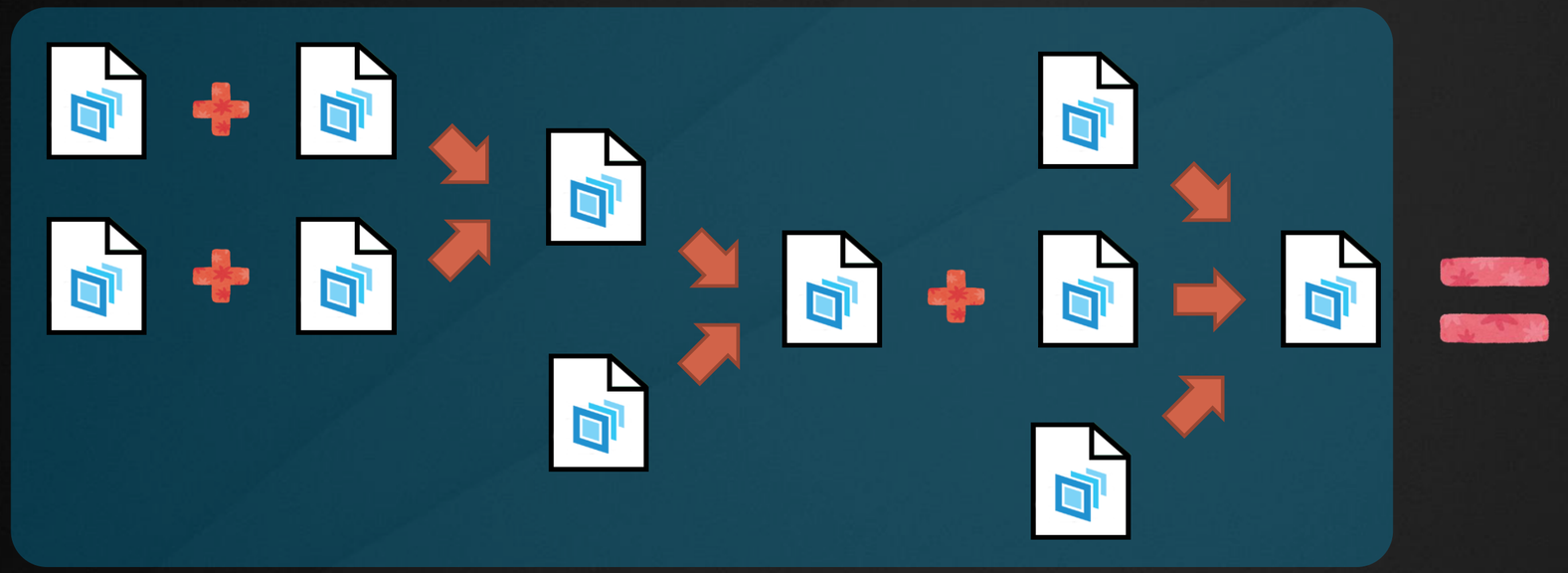
参考: Schema について https://fereria.github.io/reincarnation_tech/11_Pipeline/01_USD/18_schema/

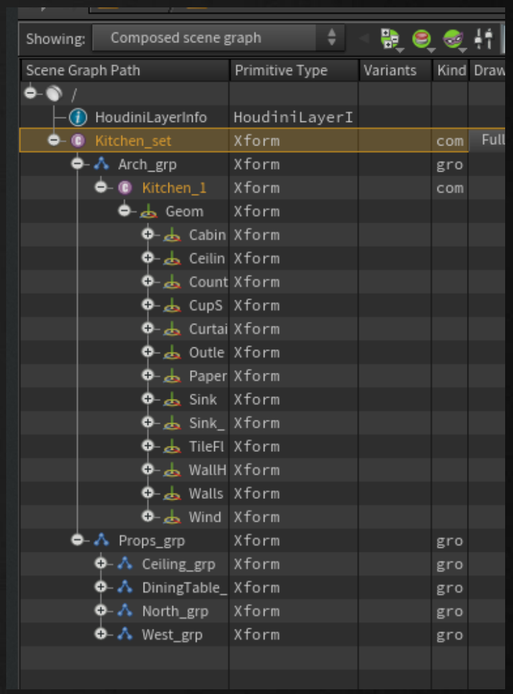
スケーラビリティ

- 長編映画のプロダクションレベルの大規模データの取り扱いができる
  - 数万ファイルから合成される 百万 単位の Prim Attribute も
    十分な速度で読み書きできる

同時編集

- コンポジション
  - コンポジションアーク（SubLayer/Inherits/Reference/Payloads/VariantSet…）
    - 複数のファイルに書かれた Prim を合成するときのルール
  - ファイルを分割・合成することで、同時編集を可能にする





POLYPHONY™ DIGITAL

USDの利点

「インターチェンジ」「スケーラビリティ」「同時編集」
を実現するための数多くの USD の機能がある

- コンポジション　　複数のファイルを合成するしくみ
- Asset Resolver　　アセットの Path 解決を拡張するしくみ
- File Format Plugin　　異なるフォーマットを USD として読み込むしくみ
- Custom Schema　　Prim の Schema を拡張するしくみ
- デバッグ環境 (usdview usdcat usdrecord etc…)

参考:
CEDEC2022関連USD記事　https://fereria.github.io/reincarnation_tech/tags/#cedec2022

Gran Turismo 7　と　USD

根本的な問題は映像制作でも現代のコンソールゲーム制作でも同じ

- チャレンジ１：様々な独自フォーマット・歴史的仕様の資産・負債
  インターチェンジ
- チャレンジ２：膨大な物量
  スケーラビリティ
- チャレンジ３：大規模なチーム
  同時編集

USD で **GT7 の３つチャレンジが解決できる**
ただし、映像制作とは違ったアプローチが有効

POLYPHONY™ DIGITAL

映像制作とゲーム制作での USD の位置づけの違い

DCC ツール

レンダラー

画像

映像制作：UsdGeom などの確立したスキーマを利用

DCC ツール

コンバート
パイプライン

ゲームエンジン用
最適化データ

モデル
シェーダ
テクスチャ

ゲーム制作：ゲームエンジン固有のデータも全て USD に載せる
コンバートパイプラインをレンダラーとみなせる

POLYPHONY DIGITAL

ゲーム制作で USD を使う利点

- 既存のデータをシームレスに統合できる
  - USD は拡張性がある汎用フォーマット、どんなデータも入れられる
  - 入出力プラグインで長年利用している独自社内形式とも自由に変換
  - アセットリゾルバでライブラリ化やバージョン管理も自由

- スケーラブル
  - 数万、数百万のアセットを一つのツリーで表現可能

- 多様なワークフローに合わせて、複数人の編集を確実に非破壊で合成できる
  - ジオメトリ・アニメーション・シェーダに限らない
  - コンバート用・ベイク用パラメータ、コリジョン、ゲーム用カスタムデータすべて

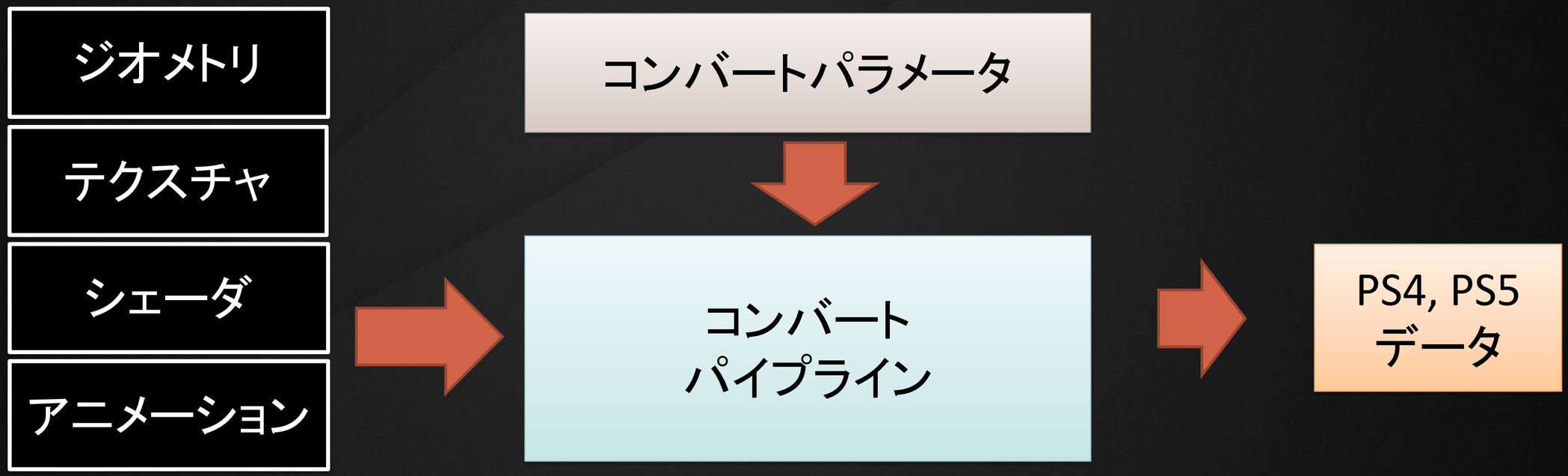
USD は GT7 の3つのチャレンジ全てを解決する

POLYPHONY™ DIGITAL

事例1：コンバートパラメータ

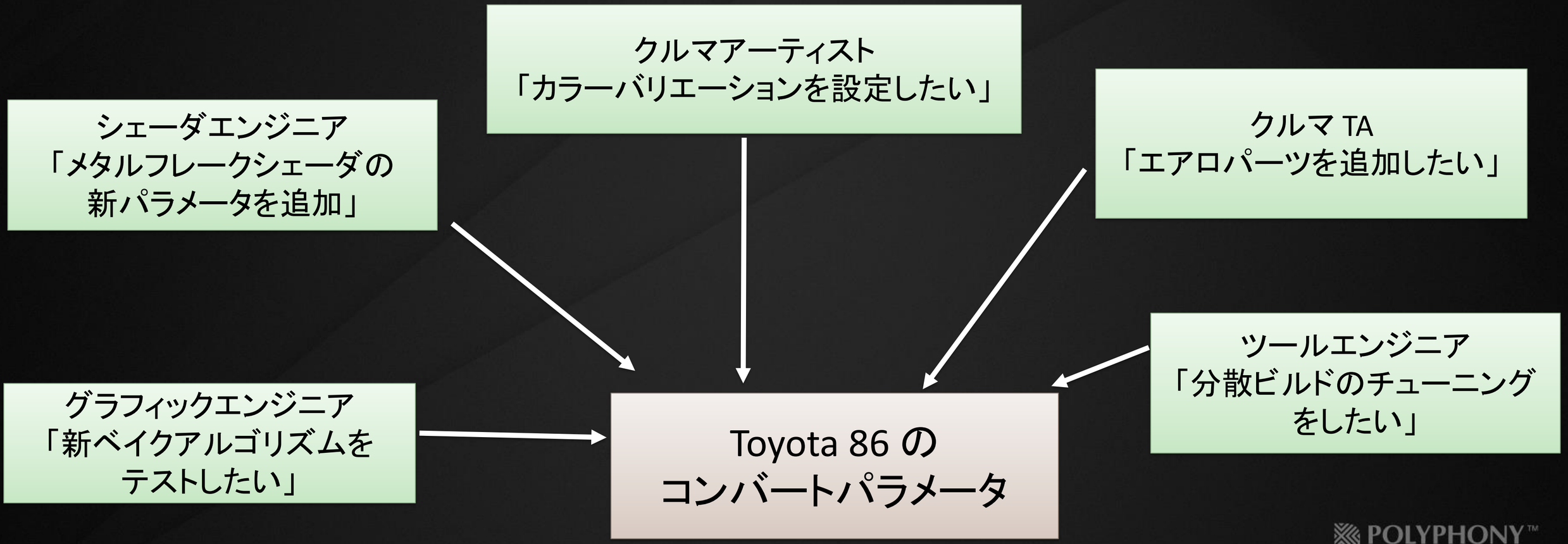
コンバートパラメータを USD で扱う

- GT7 では、レシピシステムの transform パラメータ
- DCC ツールの出力をゲームエンジン向けに最適化されたデータに変換する指示
- ベイク、オクルージョン、LOD 設定、カスタムパーツ情報・・・
- アセットごとに数百～



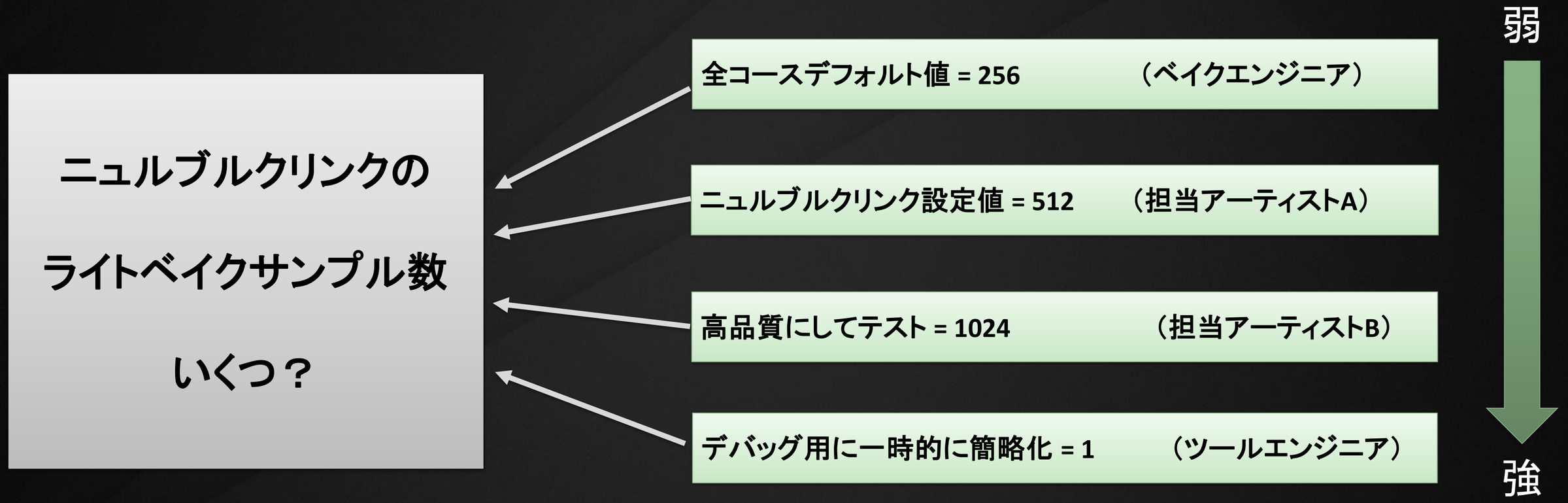
コンバートパラメータを USD で扱う

1） 同じアセットに対して、様々な人がエディットする



コンバートパラメータを USD で扱う

2） 同じパラメータについて、様々な優先度で非破壊に編集する



コンバートパラメータを USD で扱う

- USD コンポジション
  - 複数の複雑な意見を正しい優先度で調停する仕組み
  - どのように組み合わせるかはプロジェクトのワークフローによって注意深く設計する必要がある
- 設計の参考例
  - ModelHierarchy
  - Fragment-Based Composition / Entity-Based Composition (Usd at Scale)
  - サンプルアセット
    - KitchenSet
    - Moana Island Scene

参考 :

コンポジションアークとは https://fereria.github.io/reincarnation_tech/11_Pipeline/01_USD/05_comp_arc/

Pixar USD 入門 https://www.slideshare.net/takahitotejima/usd-79288174

Model Hierarchy https://graphics.pixar.com/usd/release/glossary.html#usdglossary-modelhierarchy

USD at Scale https://dl.acm.org/doi/10.1145/3543664.3543677

Moana Island Scene https://www.disneyanimation.com/resources/moana-island-scene/

コンバートパラメータを USD で扱う(5)

- パラメータを定義する USD class

```
#usda 1.0

class "course"
{
    string generator = "pdi.pipeline.GT.course"
    string options:Bake:ManyLightsSampleMethod = "random" (
        displayName = "ライトサンプルの方法"
        displayGroup = "Bake"
        customData = { string[] values = [ "random", "SPI" ] }
    )

    int options:Bake:NumDistributed = 4 (
        displayName = "ベイク分散数"
        documentation = "ベイクの計算を何台分散させて同時に行わせるか"
        displayGroup = "Bake"
    )
    bool options:Bake:DenoiseEnableAO = false (
        displayName = "AO"
        documentation = "AOのデノイズを有効化"
    )
    …
}
```

グラフィックス・ツールエンジニアが
パイプラインに合わせ手書きで用意

- デフォルト値
- GUI 用のメタデータの定義
  - 日本語名
  - ツールチップ
  - 特殊 Widget の呼び出し情報

参考:
USD は手書きするもの https://qiita.com/takahito-tejima/items/ee0332bfb5c9baed3b09

コンバートパラメータを USD で扱う（6）

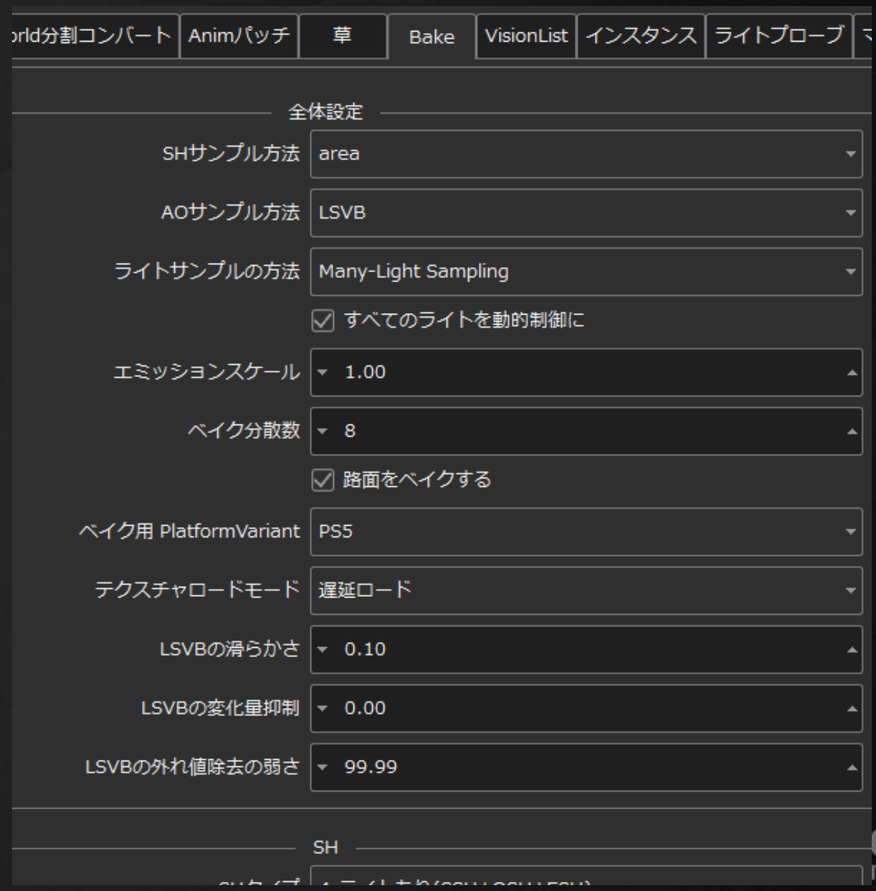
- USD 中のメタデータから、各パラメータ用の設定 GUI を PySide2 で自動生成

エンジニアが記述する共通クラスレイヤー

```
#usda 1.0

class "course"
{
    string generator = "pdi.pipeline.GT.course"
    string options:Bake:ManyLightsSampleMethod = "random" (
        displayName = "ライトサンプルの方法"
        displayGroup = "Bake"
        customData = { string[] values = [ "random", "SPI" ] }
    )

    int options:Bake:NumDistributed = 4 (
        displayName = "ベイク分散数"
        documentation = "ベイクの計算を何台分散させて同時に行わせるか"
        displayGroup = "Bake"
    )
    bool options:Bake:DenoiseEnableAO = false (
        displayName = "AO"
        documentation = "AOのデノイズを有効化"
    )
    …
}
```

GUI
自動生成

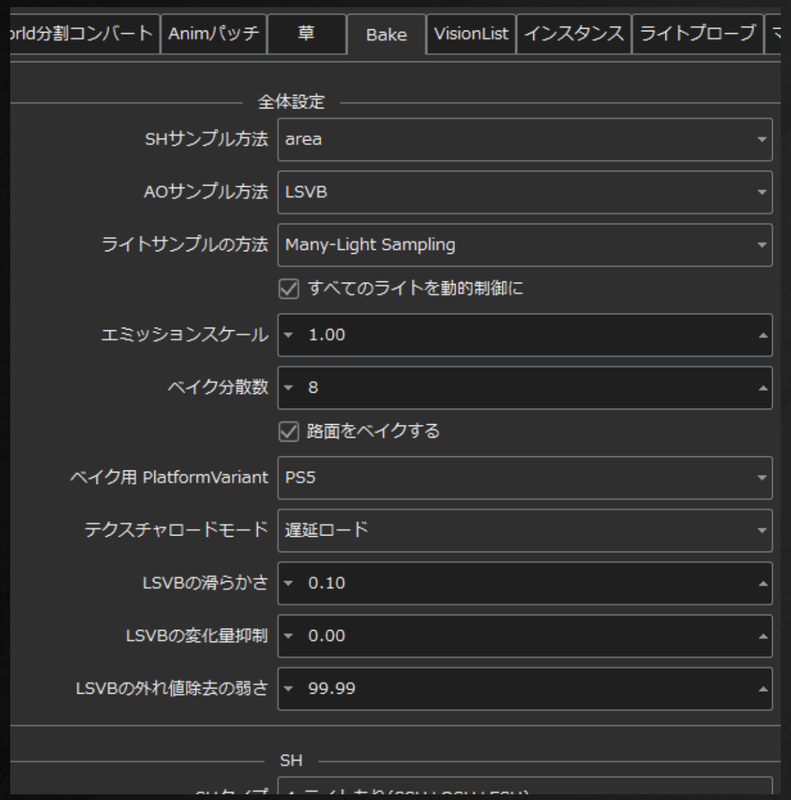
アーティストの設定 GUI



コンバートパラメータを USD で扱う（7）

- アーティストが GUI で個別設定したパラメータを USD レイヤーとして ファイルに保存し、perforce で管理

アーティストの設定 GUI



書き出し

GUI で設定したアーティストレイヤー

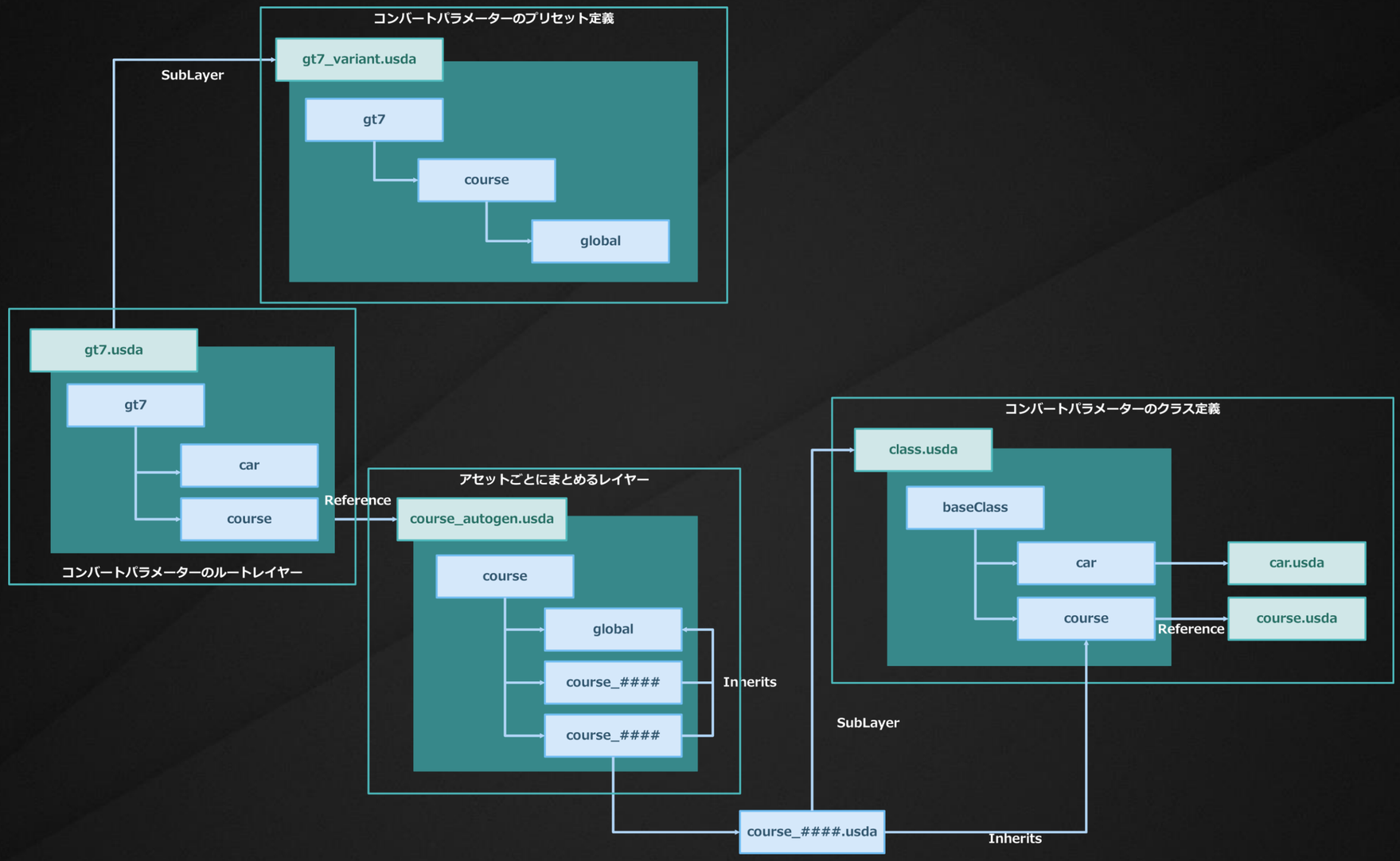
```
#usda 1.0

def RecipeGen "course_nurburgring" (
    prepend inherits = </RecipeGenClass/course>
)
{
    float options:Bake:AODistance = 30
    int options:Bake:AOIteration = 512
    int options:Bake:AOPathLenMax = 8
    string options:Bake:BakeAOSampleMethod = "LSVB"
    float options:Bake:BakeEmissionScale = 1
    int options:Bake:BakeIteration = 512
    string options:Bake:BakeSHSampleMethod = "area"
    int options:Bake:BakeSHType = 4
    int options:Bake:BakeTreeIteration = 256
    int options:Bake:dbgLightingSampleNum = 1
    bool options:Bake:ForceDynamicLights = 1
    float options:Bake:LsvbEdgeRegWeight = 0.1
    float options:Bake:LsvbSigma = 1000
    string options:Bake:ManyLightsSampleMethod = "SPI"
    int options:Bake:NumDistributed = 8
    string options:Bake:PlatformVariant = "PS5"
    ...
```

コンポジションデザインのポイント

- 誰が、どのタイミングで更新するのかを決める
- どのような優先順序で反映すべきなのか決める

- デフォルト定義を用意したい
- Jenkins を使用したオートインテグレーションをしたい
  - データベースにある情報をコンバートパラメータとして使用したい
  - GT SPORT のコンバートパラメータ（データベースで管理）から GT7 に移行したい
- 「すべてのクルマ」「すべてのコース」に対して一律で反映したい
- 「テストビルド用」「軽量ビルド用」などのパラメータープリセットを使いたい

コンバートパラメータを USD で扱う

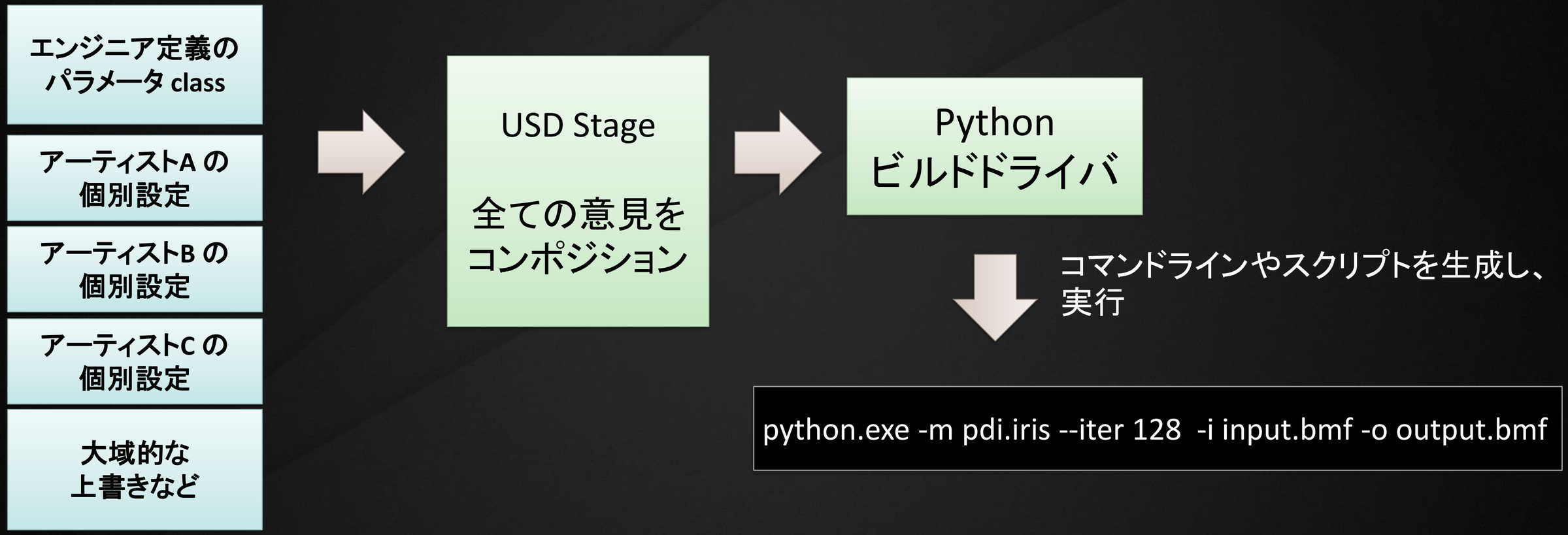


ワークフローに合わせて USD コンポジションをデザイン

POLYPHONY™ DIGITAL

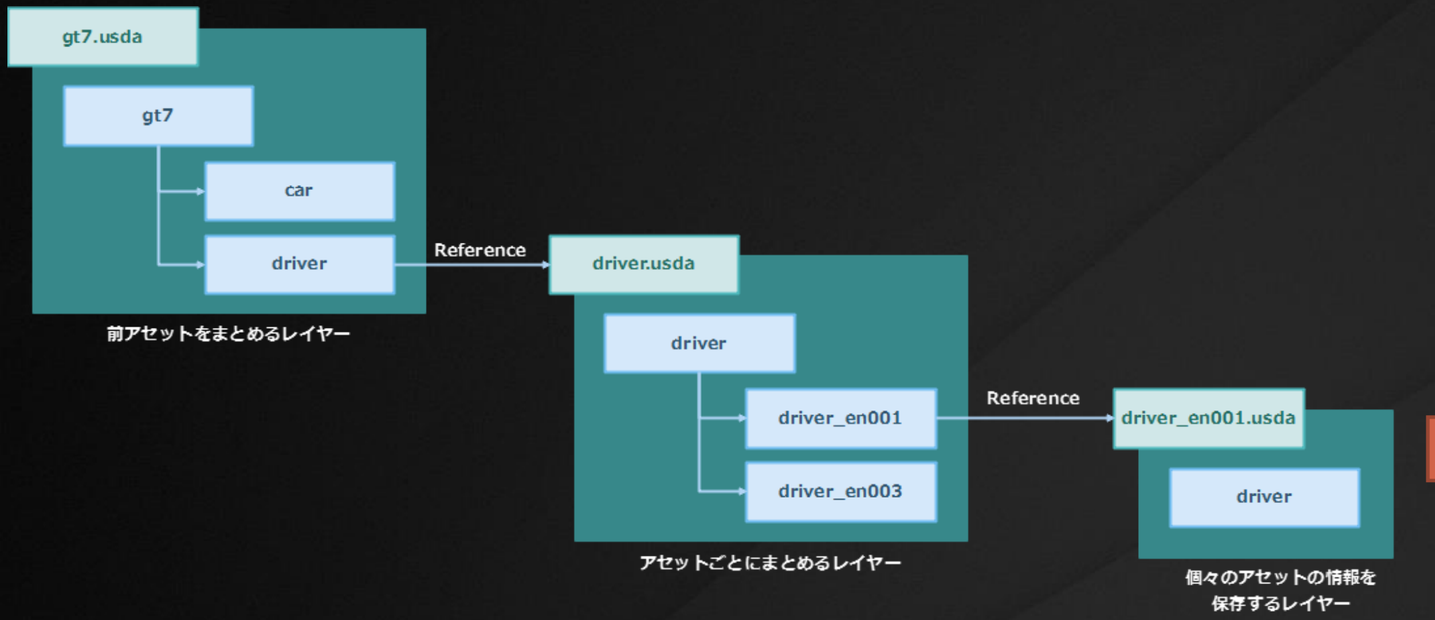
コンバートパラメータを USD で扱う（8）

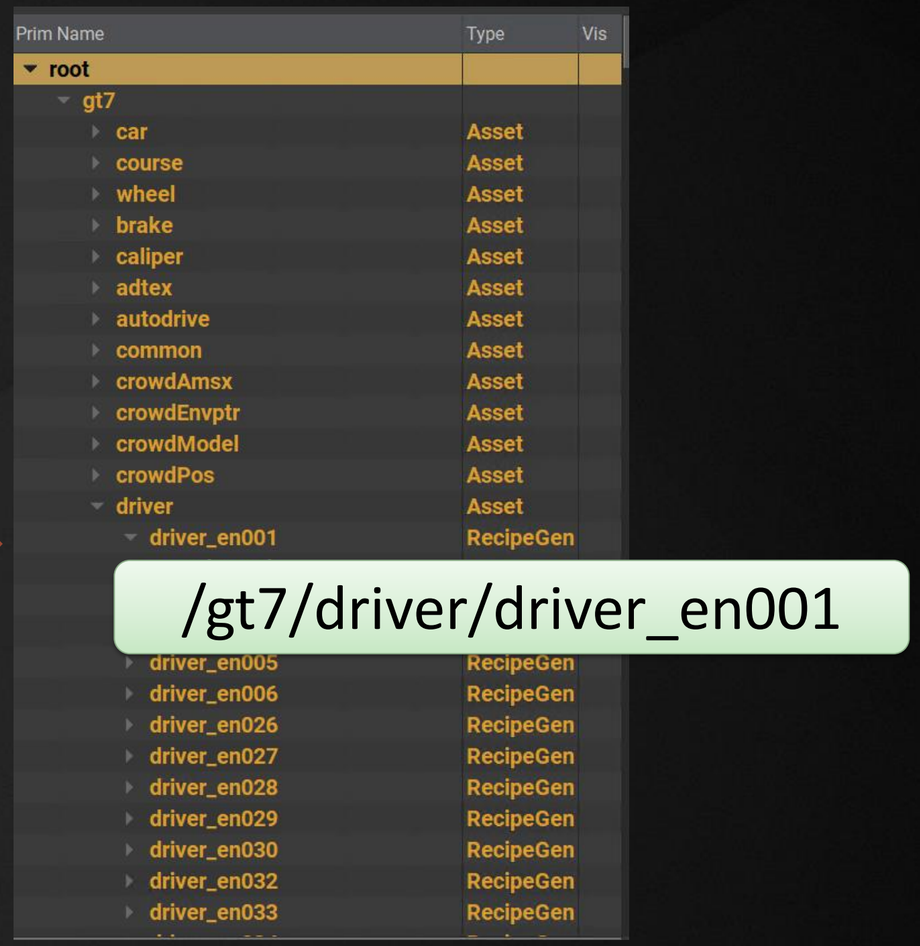
- アセットビルド時には USD ステージを評価し、各種コンバータを起動



POLYPHONY™ DIGITAL

コンバートパラメータを USD で扱う（9）





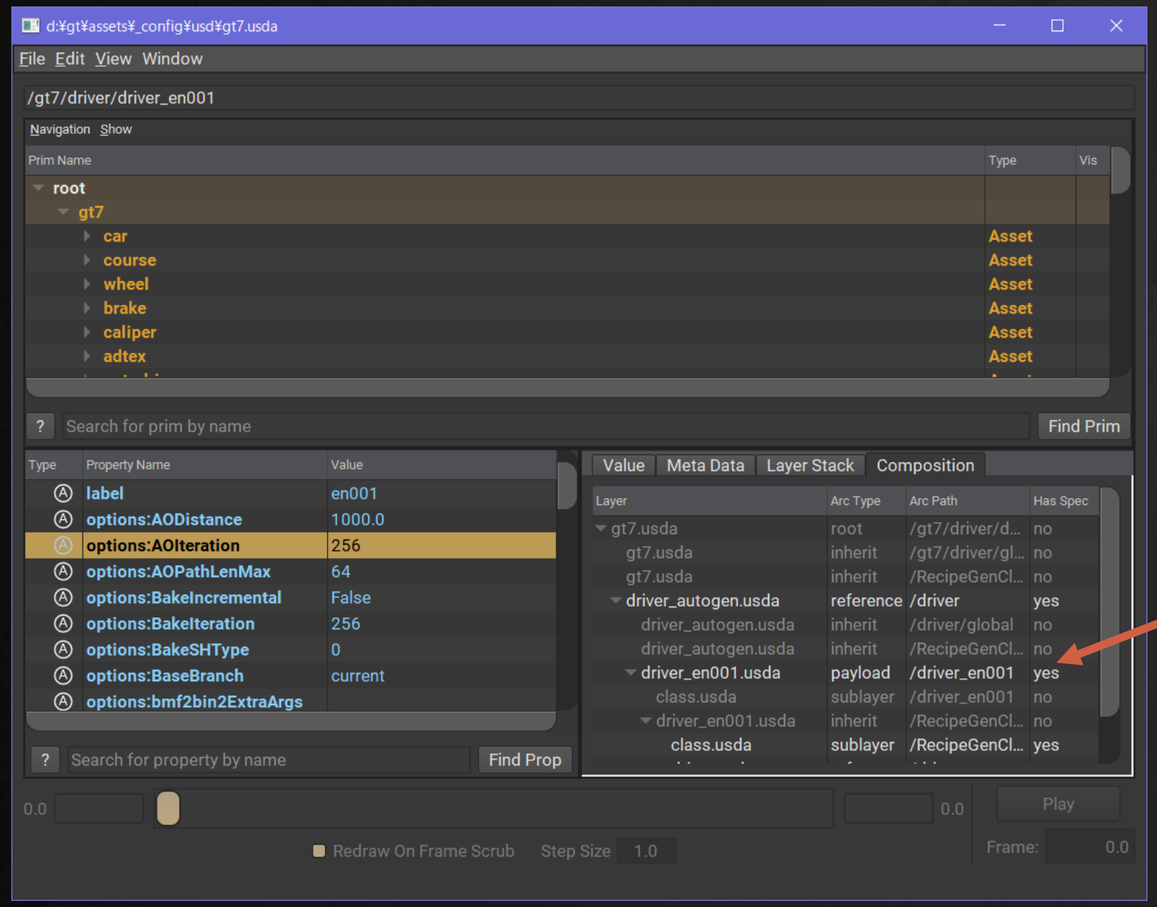
/gt7/driver/driver_en001

全 25000 アセットを USD シーングラフのパス SdfPath で表現

POLYPHONY™ DIGITAL

コンバートパラメータを USD で扱う（10）

- usdview で全アセットのコンバートパラメータを分析可能
  - 総数 200万以上のパラメータも問題なく扱える



コンポジションにより
パラメータがどう解決されたか明快

パイプラインのデバッグも容易

POLYPHONY DIGITAL

USD コンバートパラメータの成果

- 社内の PC 全てをリンクした分散ジョブシステムで実行
- GT7 の膨大なアセットに対して、パイプラインを頻繁に変更しながら毎日のフルベイク・フルビルドを実現した

- エンジニアのローカルテストも全アセットに対して実行可能

- エンジニアが .usda ファイルをエディタで書くのは楽
- 宣言的なパイプラインデザイン言語として利用
- USD の仕組み、構造の理解が深まった

事例2：背景レイアウト


35R1

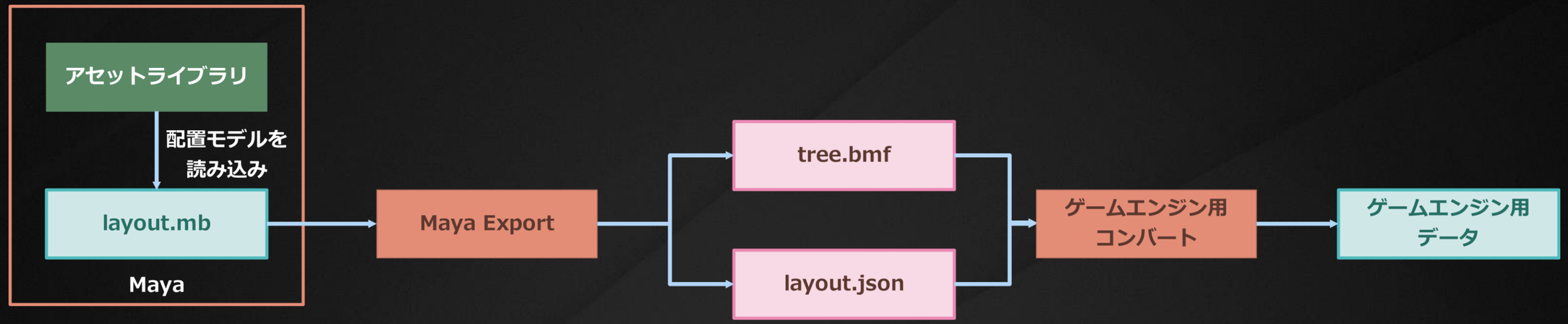
解決したい問題

Mayaでのレイアウト作業を分離したい

- PS4・PS5 世代になってから配置するオブジェクト数が増加
  - Maya でのレイアウトだとビューポートが重く作業にならない
  - シーンを開くのに時間がかかる
- モデルが変更された時の更新問題
  - Maya のリファレンスは不安定
  - 配置してはモデル更新で置きなおし

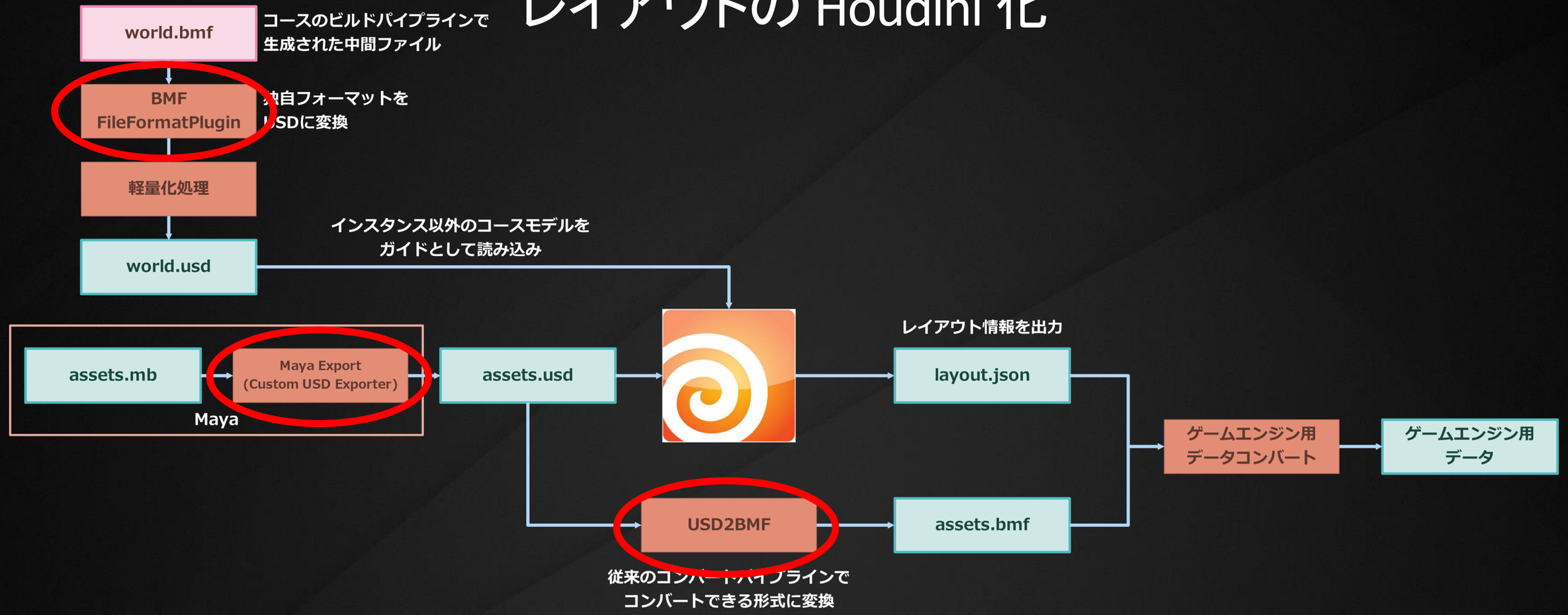
→ レイアウト作業を Houdini へ移行

今までのフロー



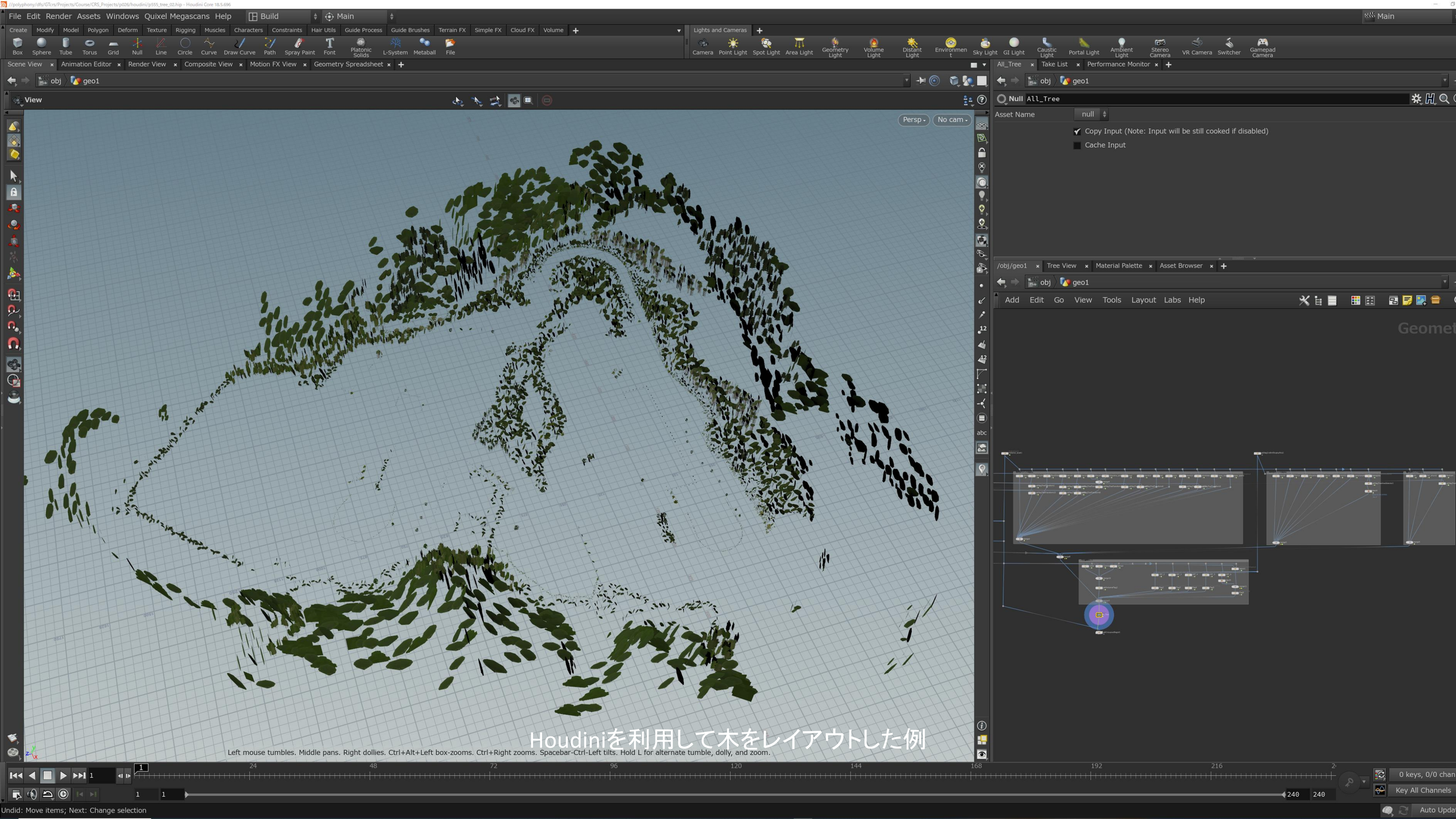
- Mayaで配置
  - モデルを複製して（Reference を使用せず）配置
  - Export 時にインスタンス化して出力
- Export 時にインスタンスの元になるモデルデータ (bmf) と配置 (json) を出力
- Bmf と json をコンソールデータにコンバート

レイアウトの Houdini 化



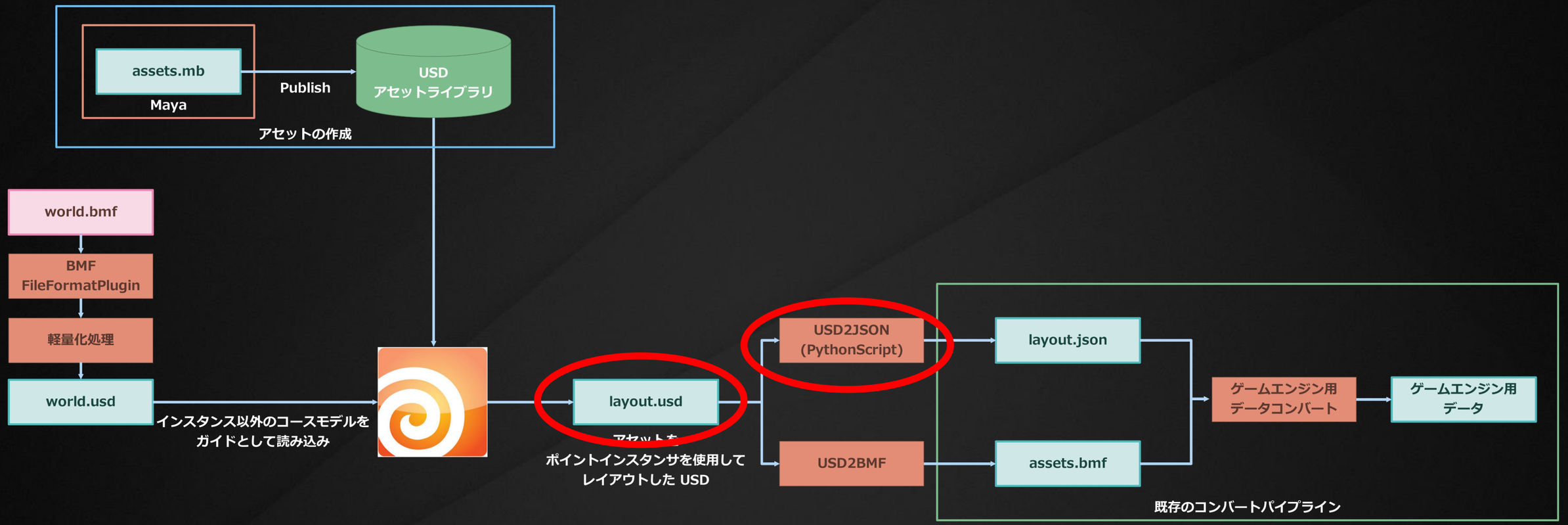
- 独自フォーマットをUSDに変換
- レイアウトするアセットを Maya から USD で出力
- Houdini でレイアウト
  - SOP で位置をモデリングし、SOLARIS で PointInstancer で配置
- レイアウト情報を json で出力
- assets.usd をゲームエンジンで利用するため 独自フォーマットへ変換

POLYPHONY™ DIGITAL


Houdiniを利用して木をレイアウトした例


35A

USD アセットライブラリ導入

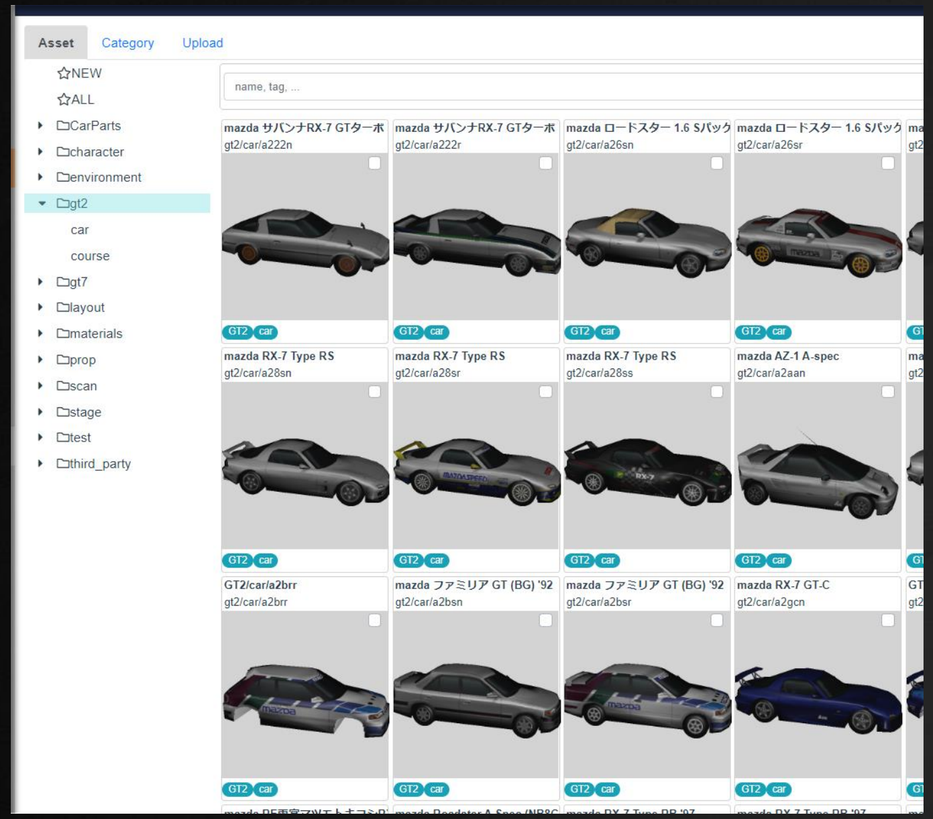


今までコース単位で管理していたアセットを、全コース共通化

参考:
Pythonコードサンプル https://fereria.github.io/reincarnation_tech/65_SampleCode/Notebook/USD/Instance/usd_pointinstancer

POLYPHONY DIGITAL

USD アセットライブラリ「Agora」



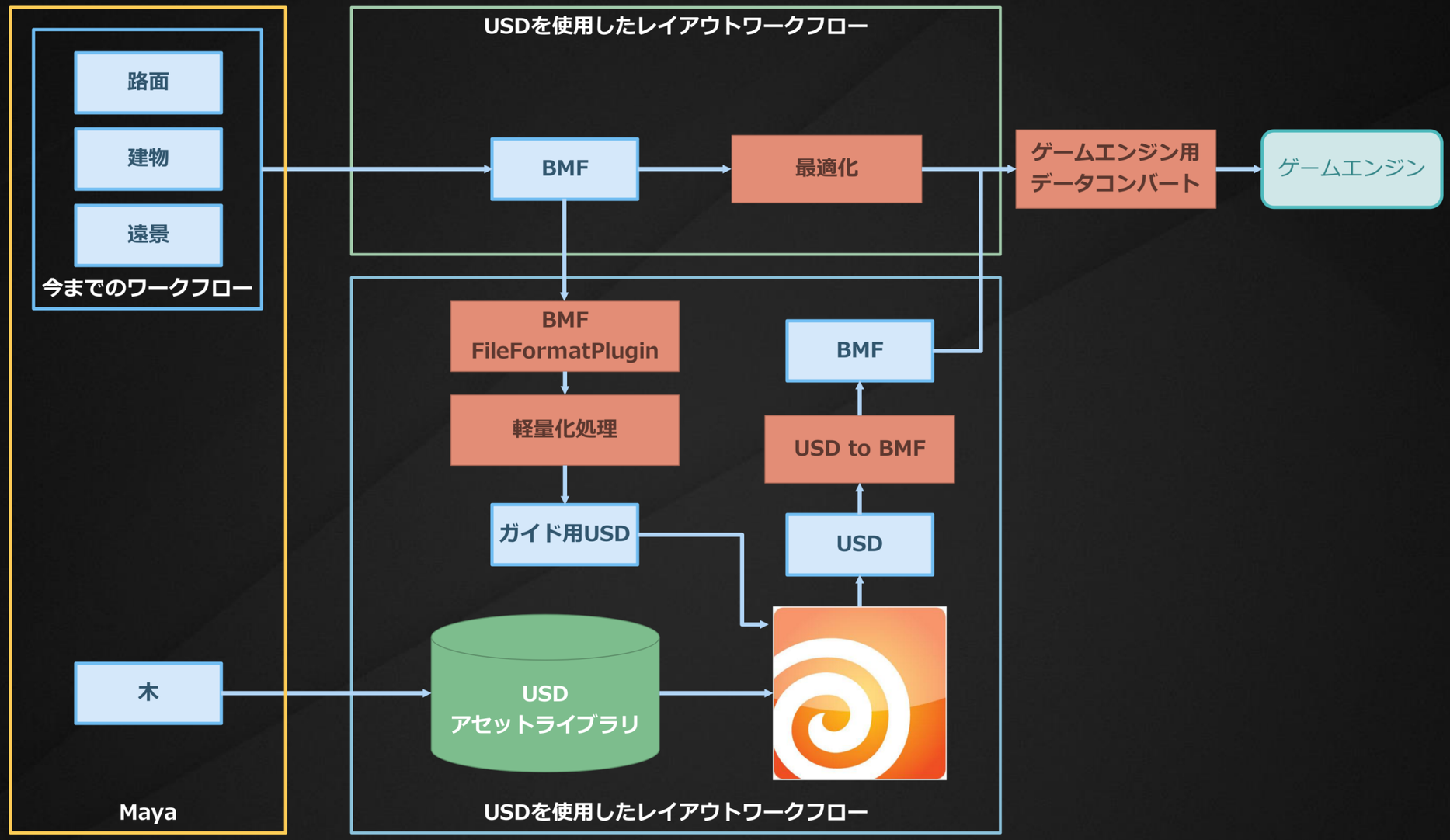
- USD アセットライブラリ
- Maya で作成したモデルを Houdini などで使用できるしくみ
- バージョン管理
- Asset Resolver

  ファイル名ではなく URI
  agora://prop/tree/sampleTree.usd?version=10
  でアクセス

参考：
Asset Resolution について https://fereria.github.io/reincarnation_tech/11_Pipeline/01_USD/24_asset_resolution/

POLYPHONY™ DIGITAL

コースレイアウト全体図



レイアウト部分を USD 化し Houdini で作業 するが、以前のパイプラインも維持

POLYPHONY™ DIGITAL

コースレイアウトの成果

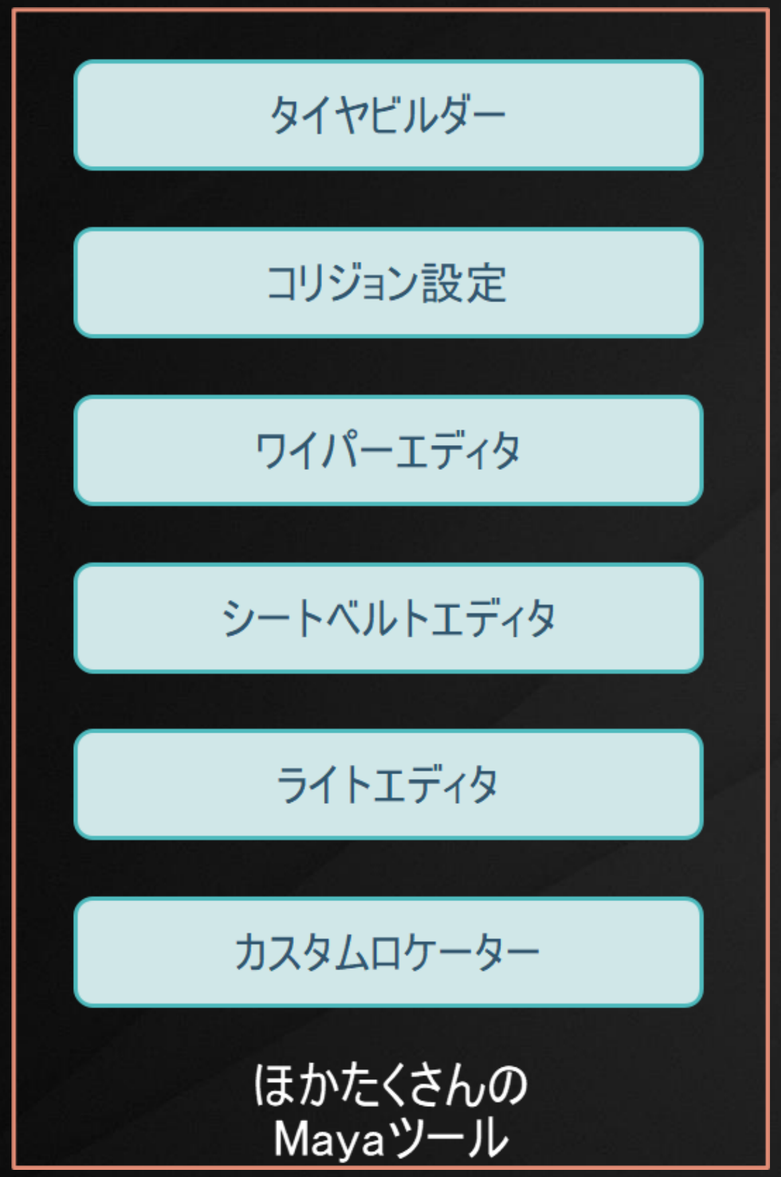
- Maya で作成したコースレイアウトのうち
  大量に配置するもの（木など）を Houdini を使用したワークフローへ移行
- 既存のパイプラインと並行しながら徐々に移行できる環境が完成
- USD ベースのアセットライブラリの構築が進んだ
- 今後は USD と Houdini を使用したコース製作環境をさらに拡大

事例3：カスタムパーツ

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR

GRAN TURISMO® 7
THE REAL DRIVING SIMULATOR

これまでの 作成環境



- 車を作る環境は Maya
  必要なツールや、アセットビルド時の仕様追加などはすべて ツールエンジニア が作成
- 複雑な Maya の仕様
- アセットビルド時にも複雑な処理
  - アーティストが手を出しにくい環境
  - ツールエンジニアの手が足りない…

Houdini を活用して TA が作業できるようにする

POLYPHONY™ DIGITAL

Houdiniの導入

- 今ある機能をすべて Houdini で再実装するのはコストが高い

  複雑な仕様、過去の遺産、大量で巨大なアセット

  → 部分的な導入から進めたい

→ USD で解決

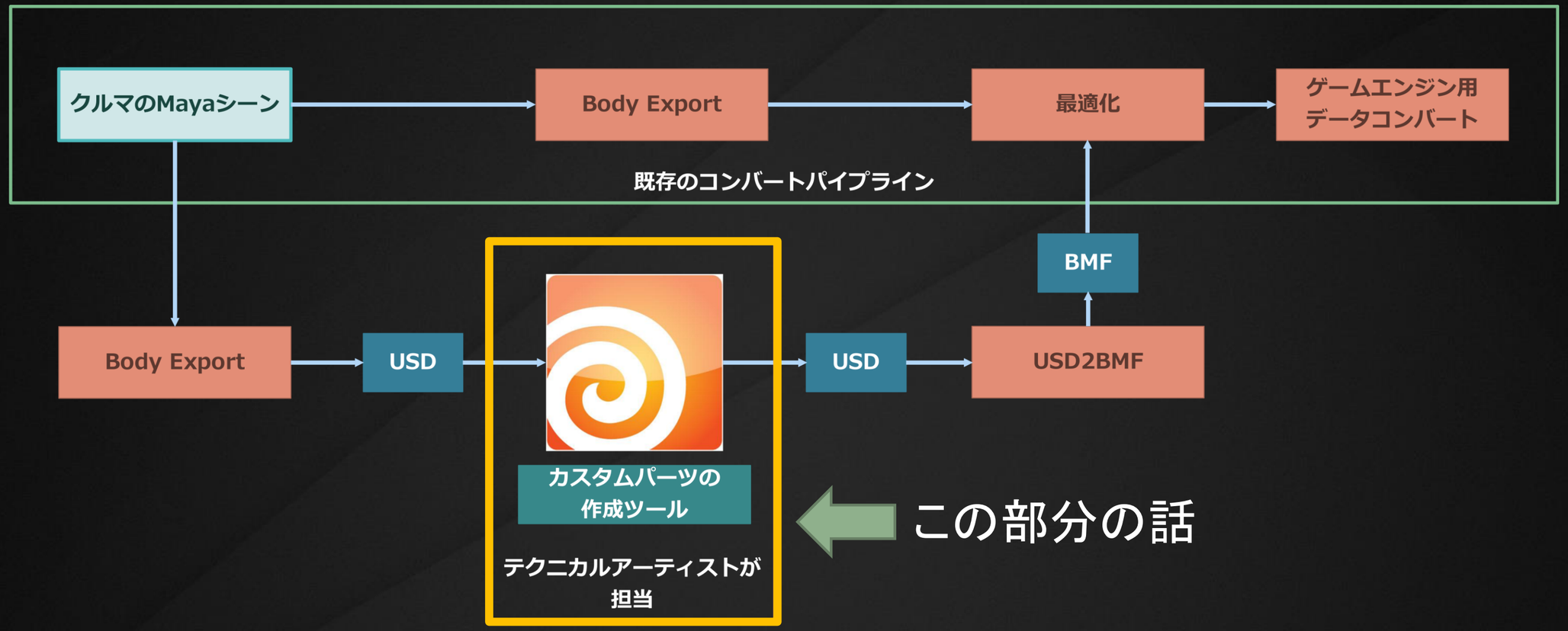
- USD をフルサポートした SOLARIS で USD の編集をする環境がある
- USD の「インターチェンジ」 ソフトウェアを超えた作業を可能にする

  必要なカスタムパラメーターを定義

Houdini™

POLYPHONY™ DIGITAL

カスタムパーツでの戦略



詳細は CEDEC2022
プロシージャルゲームコンテンツ制作ブートキャンプ 2022『グランツーリスモ7』　16:10～

HoudiniのInput/Outputを　USD　化する

POLYPHONY DIGITAL

ツールエンジニアの対応事項

- ガイド用のクルマモデルの出力
  - Houdini にモデルを持ち込みたい
    - ギリギリ FBX などでもいけるが…
  - カスタムパーツを作成するときにガイドモデルとして必要
  - Mayaで作成したクルマモデルを Houdini に持ち込む経路が必要
- Maya で作成したマテリアルの出力
  - Houdini で作成したモデルに Maya で作成したマテリアルをアサインしたい
- コンソール用データにコンバート対応
  - 作成したカスタムパーツをコンソールで使用できる形にする必要がある

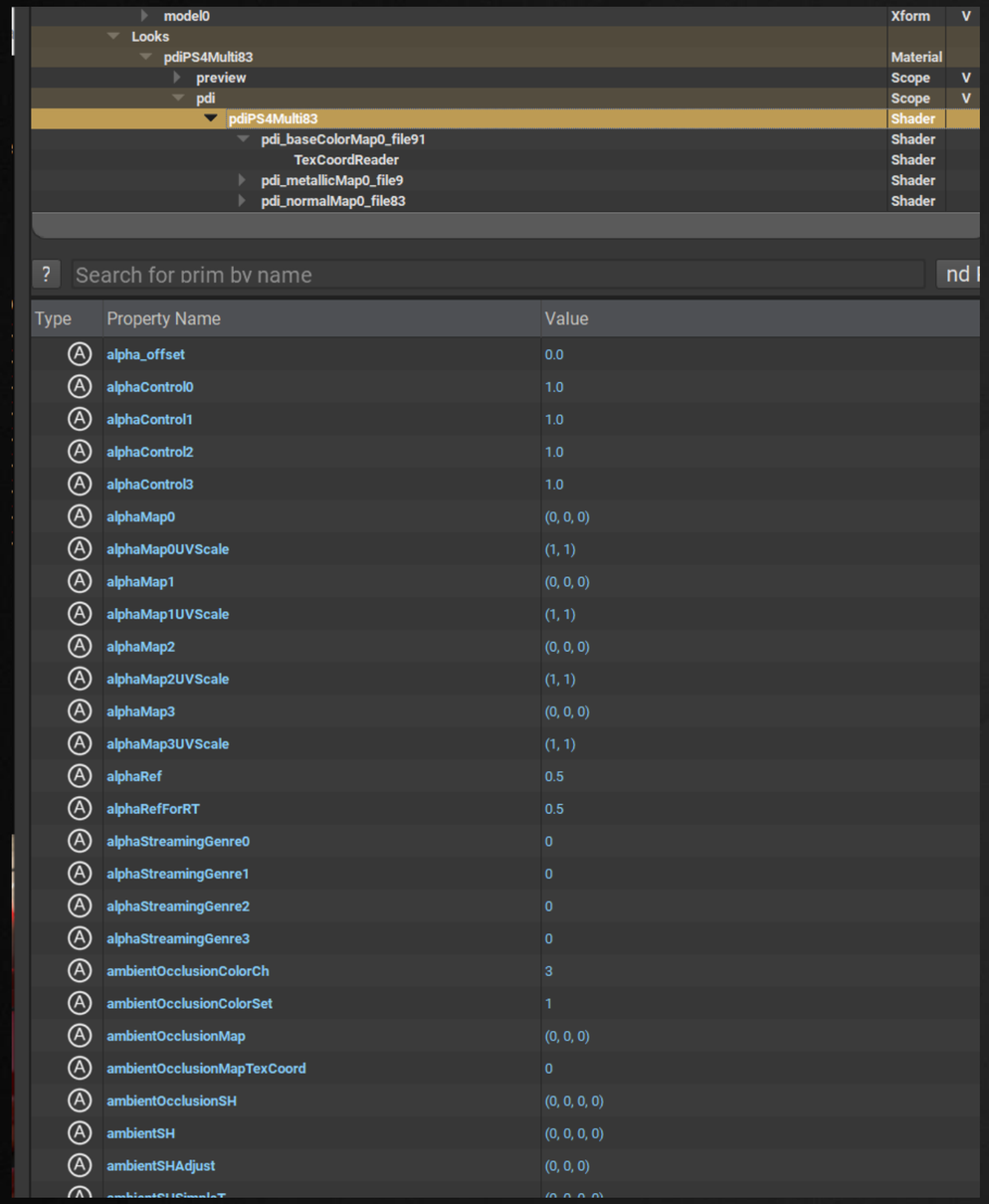
USD + Houdini SOLARISで解決

UsdPreviewSurface

- USD が標準で用意している簡易的な物理ベースのサーフェイスモデル
- Maya・Blender・Houdini などの DCC ツールから出力可能
- DCC ツールとゲームエンジンなど、異なるプラットフォームに
  移行するときなどに使用できる
- テクスチャ、頂点カラー、UV などに対応
- Specular Metalness を両方サポート

参考:
https://graphics.pixar.com/usd/release/spec_usdpreviewsurface.html
https://fereria.github.io/reincarnation_tech/11_Pipeline/01_USD/30_USD_Programming/01_Python/06_shader

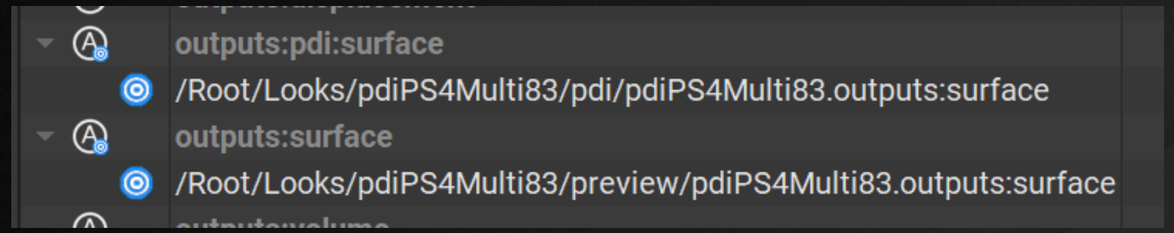
マテリアルの出力

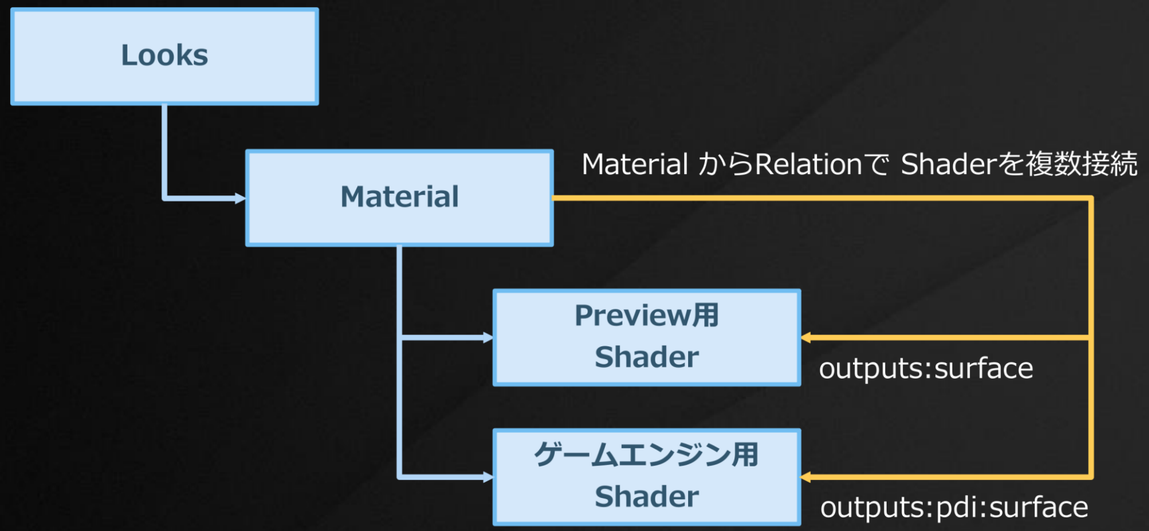


- 既存の Maya GUI でマテリアルのエディット
- Maya カスタムエクスポーターを使用して
  複雑で大量のパラメーターを持つ Shader を
  USD の Material/Shader と Attribute として出力
- 出力したマテリアル は SOLARIS で
  リファレンスして使用

POLYPHONY™ DIGITAL

USDのリレーション





- usdview や SOLARIS でテクスチャ表示をするために PreviewSurface も同時に出力
- USD のリレーションを使用して Preview用のシェーダーと ゲームエンジン用のシェーダーを 両方とも接続
  - Preview 時は PreviewSurface で簡易表示
  - 実機コンバート用には実機用 Shader を使用

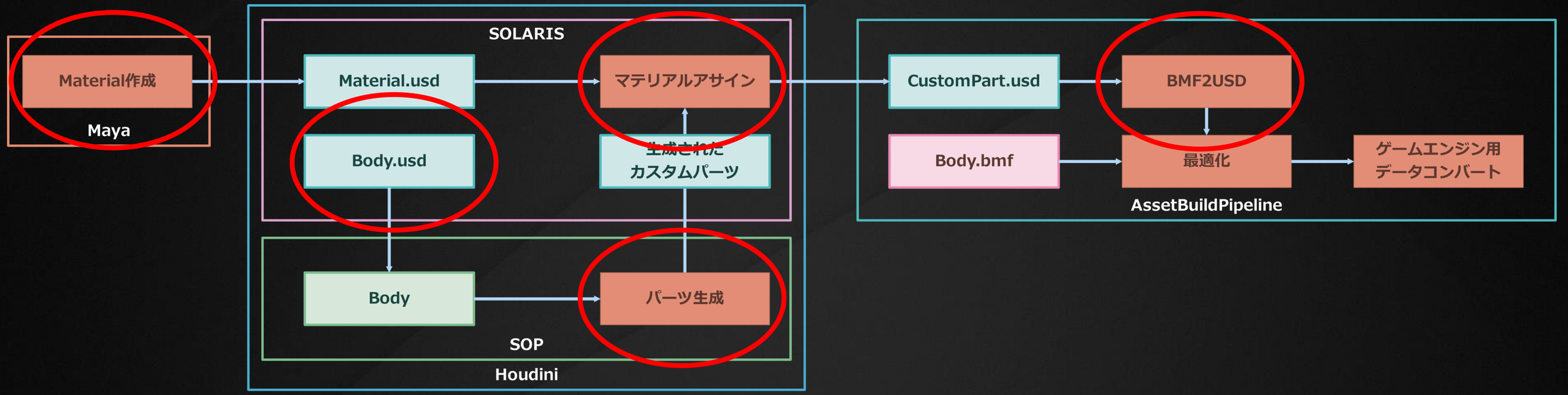
POLYPHONY™ DIGITAL

Houdini SOLARIS

- Houdini 18 で追加された
  Layout/Lighting/Lookdev を行うための機能
- USD をフルサポート
  - SOLARIS = USD の世界
- USD のシーングラフを 非破壊で Houdini のノードベースで
  プロシージャルに編集することができる
- SOP と SOLARIS(USD) を相互に行き来できる

POLYPHONY™ DIGITAL

SOLARIS での マテリアルアサイン



Material.usd を Reference することで、Mayaで作成した Material を SOLARIS で Assign

POLYPHONY™ DIGITAL

カスタムパーツ制作への USD 導入の成果

- Houdini SOLARIS を使用した
  テクニカルアーティストがツールを作成できる環境ができた
  - 既存のパイプラインを動かしながら、USD を使用して拡張できる仕組みを実現
- Maya から Houdini へ
  USD を使用することでデータを受け渡しできるようになった
  - USD の「インターチェンジ」を生かしたワークフローへ
- パイプラインの一部を USD に置き換えることができるようになった
  - 今後さらにスケーリングさせる土台が完成した

USD が描く未来

POLYPHONY™ DIGITAL

映像プロダクションへの広がり

- 公開されて以降、多くのプロダクションで採用され
  現在は共通のプラットフォームになりつつある
  - 多くのツールをつなぎ、さまざまな仕様に対応する「インターチェンジ」
  - 長編映画のレンダリングなどで使用する巨大なシーングラフを
    高速にロード可能にする「スケーラビリティ」
  - 数百人規模の、大量のデータを使用する開発を
    「同時並行」して「ロバストに」「非破壊で」扱うことができる性能
- そしてそれらを支えるパイプラインのテクノロジーの存在により
  多くのプロダクションで採用された
- USD のコミュニティの発展
  - https://wiki.aswf.io/display/WGUSD

ゲーム業界とUSD（1）

- 映像プロダクションに比べて、ゲームでの採用はまだまだこれから
- しかし、昨今の ゲーム製作は映像プロダクションが抱える問題と同じものを抱えている
  - 大量のアセットやレイアウト、ショットを
  - 大人数で同時並行して
  - 様々なゲーム仕様の追加と並行して行う必要がある
- ゲームのアセットを映像用にも使いたい
- これらの問題に、USD の持つポテンシャルを発揮することができる
  - 「インターチェンジ」「スケーラビリティ」「同時編集」はゲーム開発でも生かせる

POLYPHONY™ DIGITAL

ゲーム業界とUSD（2）

- Unity・UnrealEngine でのサポート
  - UnrealEngine5　では UsdSkelAnimation、BlendShape なども使用可能
- ゲームデータの映像パートでの利用事例などが 2022年 DigiPro で発表 ※1
  - ゲーム用データをUSDに変換しアセットライブラリ化する仕組み
- Omniverse
  - プラットフォームを超えてのライブコラボレーション
  - ゲームや映像分野以外（建築・製造・ロボティクス分野）での利用

※1 DIGIPRO 2022 BUILDING SCALABLE AND
EVOLUTIVE USD PIPELINES ON DISTRIBUTED ARCHITECTURE AT UBISOFT

GT7 での活用

- コンバートパラメータの編集というコンパクトなプロジェクトからスタート
- 同時並行して編集し、巨大なパラメーターをエディット
- 徐々にアセットライブラリまで利用範囲を広げた
- 独自仕様のデータも USD の拡張で対応
- 複雑かつ大量の仕様の情報を維持しつつ、さまざまなツールを超えて扱える「インターチェンジ」

既存のパイプラインを引き継ぎながら、徐々に USD にシフトできる柔軟さは
大規模ゲーム開発の様々な問題を解決する大きな可能性

POLYPHONY™ DIGITAL

まとめ

- USD のエコシステムは映像業界を超えて拡大し続けている
- ゲーム制作でも当たり前のように随所で利用される未来が来る
- 少しづつでも導入は可能で、十分なメリットが得られる

POLYPHONY™ DIGITAL

Q & A

POLYPHONY™ DIGITAL

Thank you!!

POLYPHONY™ DIGITAL

参考資料

- Pixar USD入門 新たなコンテンツパイプラインを構築する
  - https://www.slideshare.net/takahitotejima/usd-79288174
- Reincarnation#Tech CEDEC2022
  - https://fereria.github.io/reincarnation_tech/tags/#cedec2022
- COMPLEXITY SIMPLIFIED
  - https://indyzone.co.jp/archives/1715
- Building scalable and evolutive USD pipelines on distributed architecture at Ubisoft
  - https://dl.acm.org/doi/10.1145/3543664.3543679
- USD at Scale
  - https://dl.acm.org/doi/10.1145/3543664.3543677
- Universal Scene Description
  - https://graphics.pixar.com/usd/release/index.html
- USD Working Group
  - https://wiki.aswf.io/display/WGUSD
- nvidia Omniverse
  - https://www.nvidia.com/ja-jp/omniverse/