



# Unity用可視化フレームワークVisAssetを用いた 可視化アプリケーション開発

○ 川原慎太郎（海洋研究開発機構）  
宮地英生（東京都市大学）

# 目次

- [本日御用意頂くもの](#)
- [VisAssetsの概要](#)
- [VisAssetsのダウンロード](#)
- [新規プロジェクトの作成](#)
- [Unityエディタの画面構成](#)
- [VisAssetsを導入するための準備](#)
- [VisAssetsの導入](#)
- [可視化モジュールの場所](#)
- [データを読み込む](#)
- [カラースライスを表示する](#)
- [任意の要素のカラースライスを表示する](#)
- [複数のカラースライスを表示する](#)
- [等値面を表示する](#)
- [ユーザインターフェースの追加](#)
- [ビルド前の準備](#)
- [Windows用実行ファイル作成](#)
- [\[参考\] PC-HMDでの立体視表示](#)
- [\[参考\] Android用ビルド](#)
- [\[参考\] スタンドアロンHMDでの立体視表示](#)
- [ベクトル場の可視化](#)
- [バイナリデータの読み込み](#)
- [新たなモジュールの開発](#)
- [まとめ](#)
- [謝辞](#)
- [\[付録\] VFIVE用データの読み込み](#)
- [\[付録\] GrADS用データの読み込み](#)

# 本日御用意頂くもの

※本資料はWSのページからダウンロードできます

- UnityをインストールしたWindowsPC (Windows7 SP1+以上, 64bit)  
※ Macでも動くかとは思いますが、十分な動作確認をしていません
- Unityのバージョンは 2021.2 を使用します  
※ Unity Hubからのインストールをおすすめします
- ハンズオン後半で使用するサンプルデータ（下記URLから取得してください）

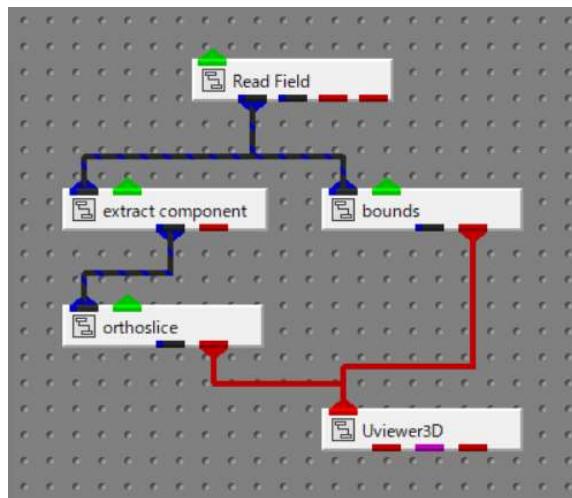
<https://www.jamstec.go.jp/ceist/aeird/avcrg/vfive.ja.html>

ダウンロードの項にあるサンプルデータABC Flowの**ビッグエンディアン版(sample\_big2.tar.gz)**を使用します

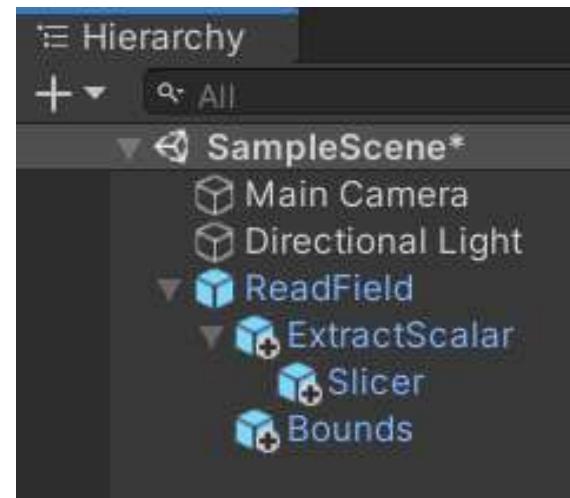
※ リトルエンディアン版データはアーカイブファイルが壊れしており、展開ができません

# VisAssetsの概要(1)

- Unity用可視化フレームワーク
- 宮地英生教授(東京都市大学)と川原との共同開発
- AVS/Expressのような可視化ネットワークの構築をUnity上でも簡単に、  
をコンセプトにオープンソースで開発中



AVS/Expressのビジュアルエディタ



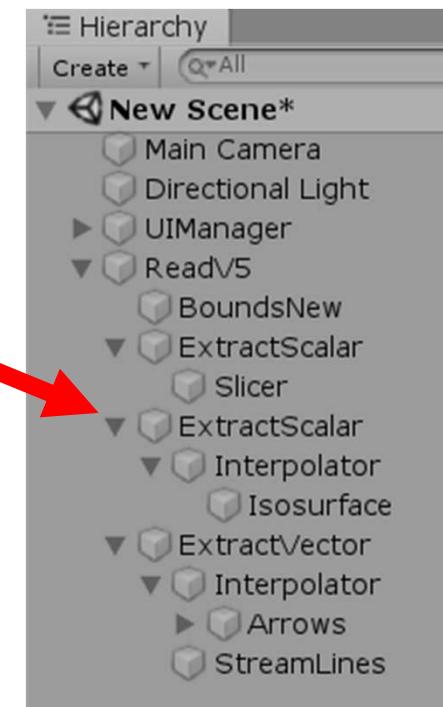
Unityのシーンヒエラルキー上での  
VisAssetsによる可視化ネットワーク構築

# VisAssetsの概要(2)

- 可視化フローを構成する小要素を機能毎にモジュールアイコン化
- Unityのシーンツリーへのモジュールアイコンのドラッグアンドドロップによるプログラミングレスでの可視化アプリケーション開発を実現



必要なモジュールを  
D&D

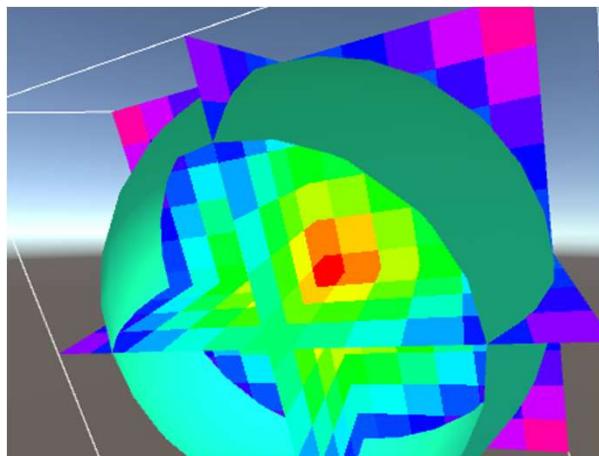


Modularized visualization elements

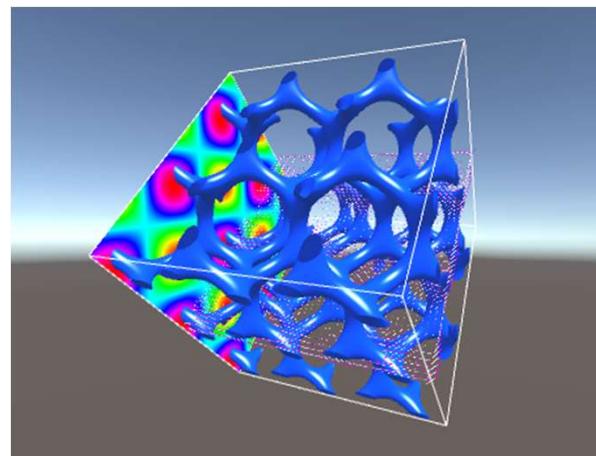
Visualization network on Unity

# VisAssetsの概要(3)

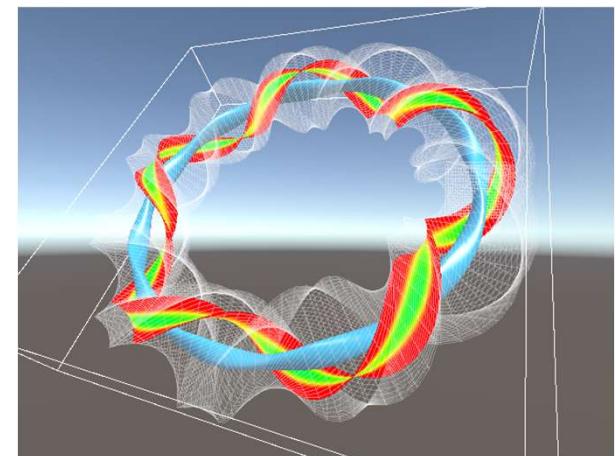
- ・構造格子データの読み込みに対応
- ・データ読み込み用モジュールを作成すれば、可視化モジュールは共用可



等間隔直交格子



不等間隔直交格子 ※1



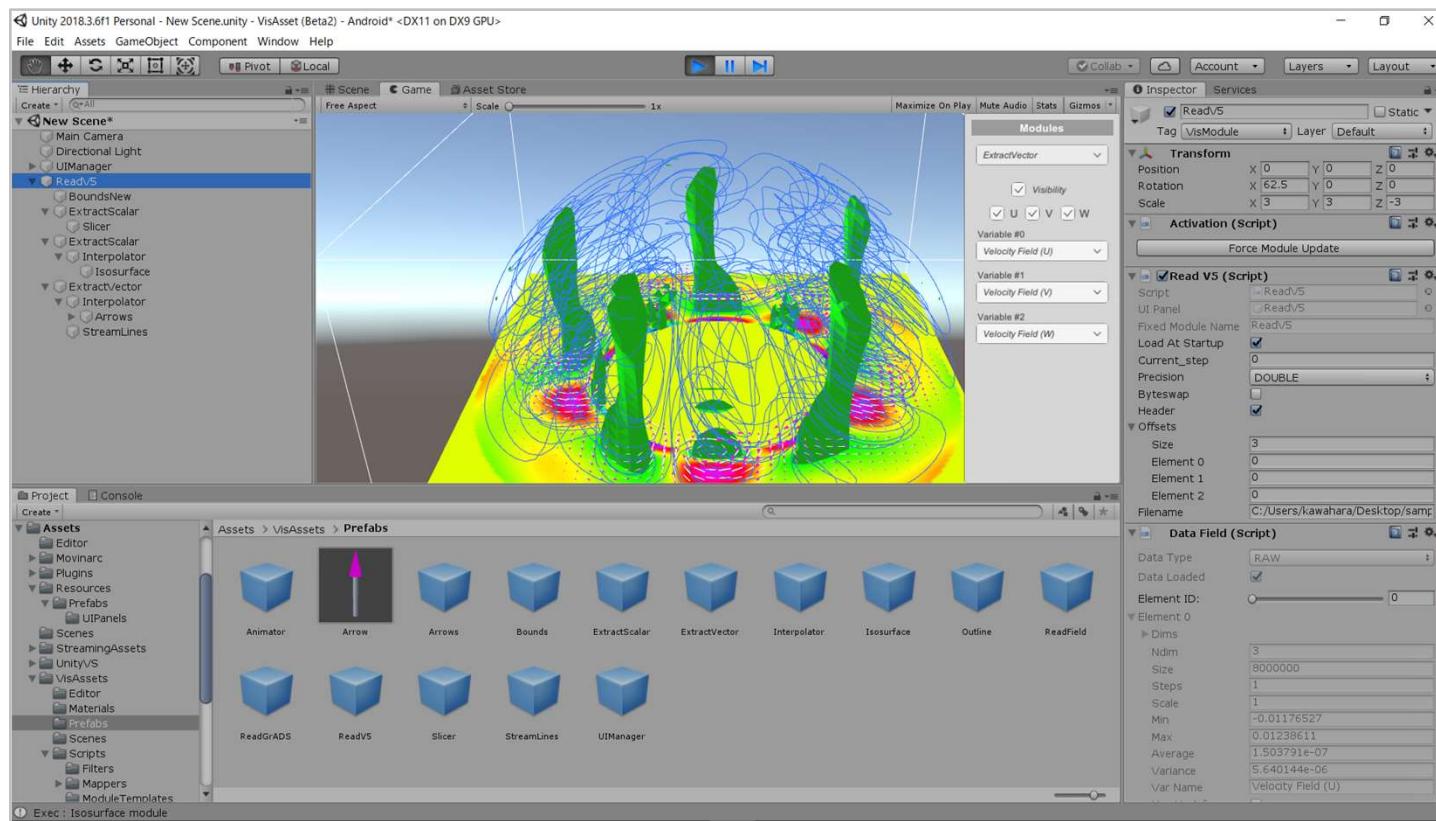
非直交格子 ※2

※1 データ提供：神戸大学 陰山聰 教授

※2 データ提供：核融合科学研究所 大谷寛明 准教授

# VisAssetsの概要(4)

サンプルモジュールによるスカラー場、ベクトル場の可視化例

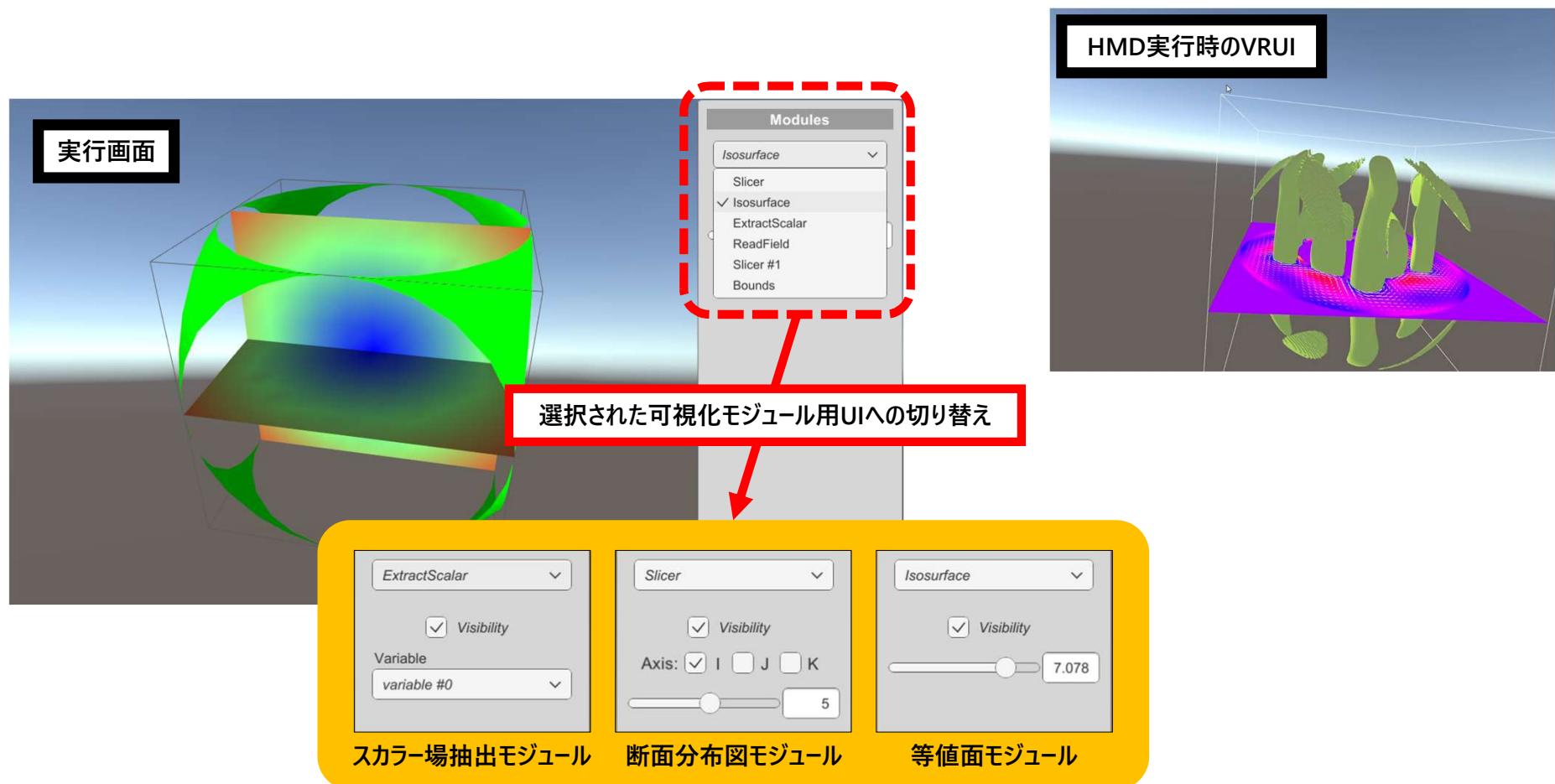


※ データ提供：神戸大学 陰山聰 教授

※ 本ワークショップで使用するバージョンには流跡線描画モジュールは含まれていません

# VisAssetsの概要(5)

ビルド後のアプリケーションでのパラメータ変更用GUIを提供



# VisAssetsの概要(6)

マルチプラットフォーム対応であるUnityを基盤とすることにより、PCの他、スマートフォン等多くのデバイスで実行可能なアプリケーションの開発ができます。



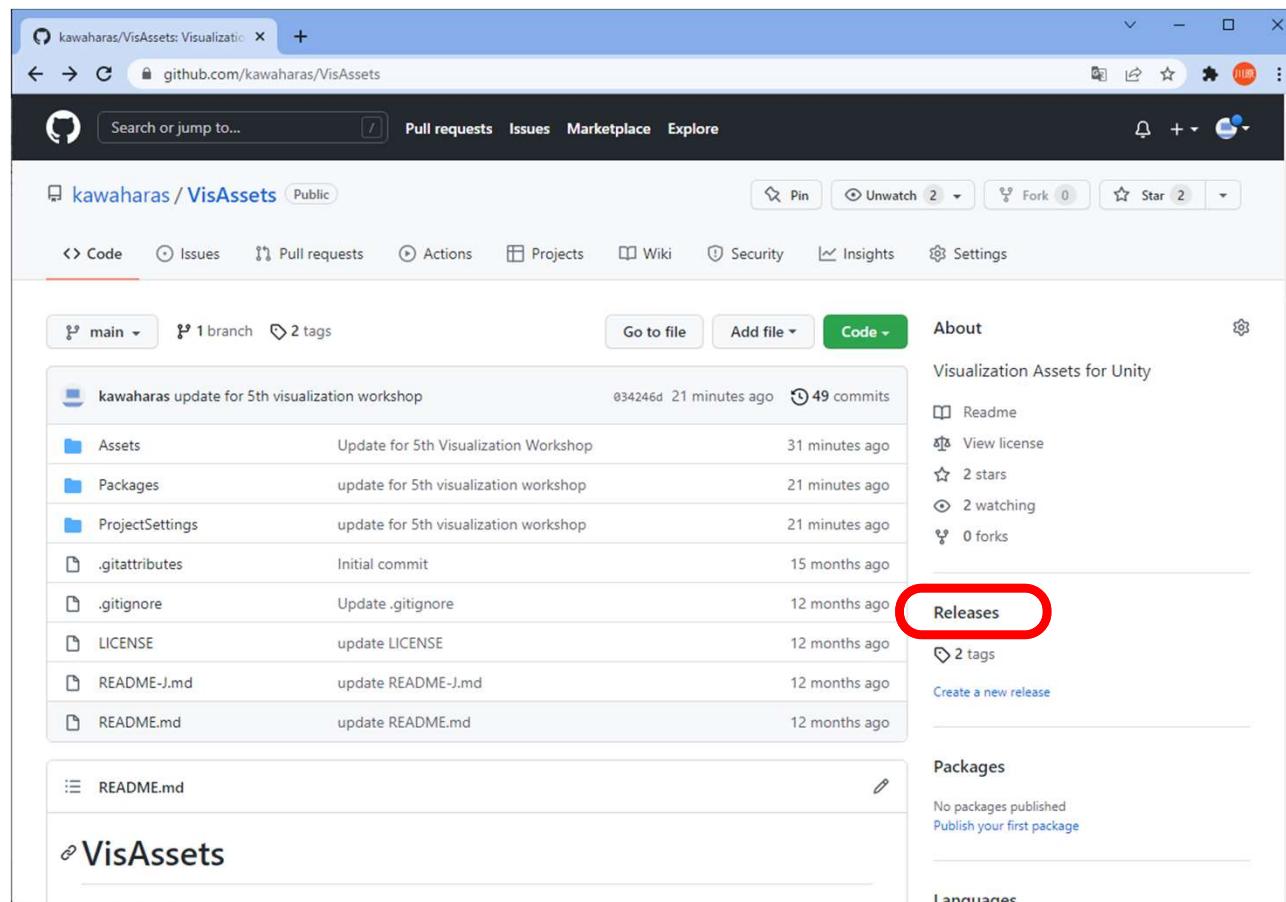
PC-HMDへの表示



Androidスマートフォンでの実行

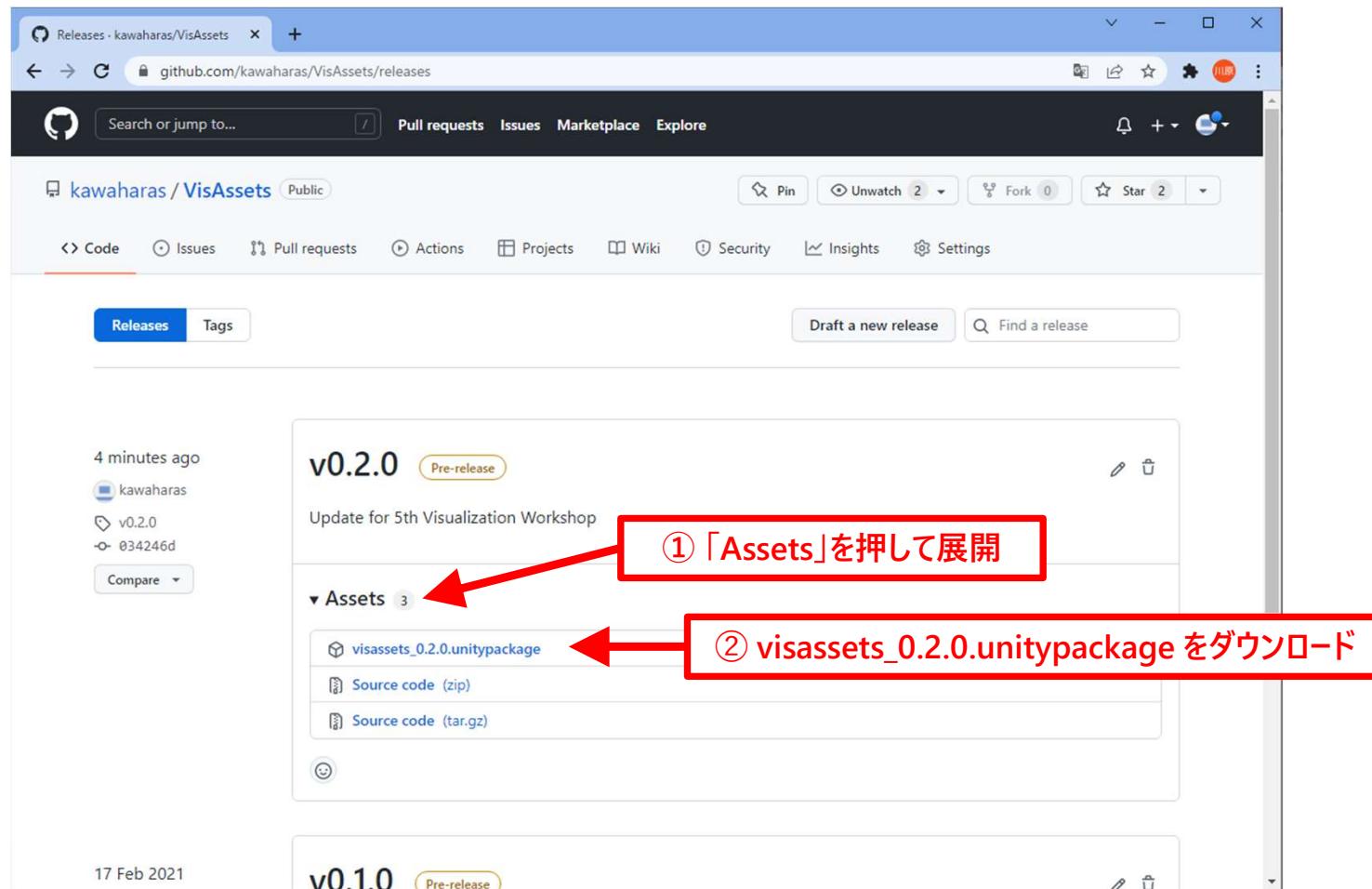
# VisAssetsのダウンロード(1)

<https://github.com/kawaharas/VisAssets> にアクセスし、画面右側にある「Releases」をクリックします。



# VisAssetsのダウンロード(2)

v0.2.0の「Assets」をクリックして展開し、unitypackage をダウンロードします。



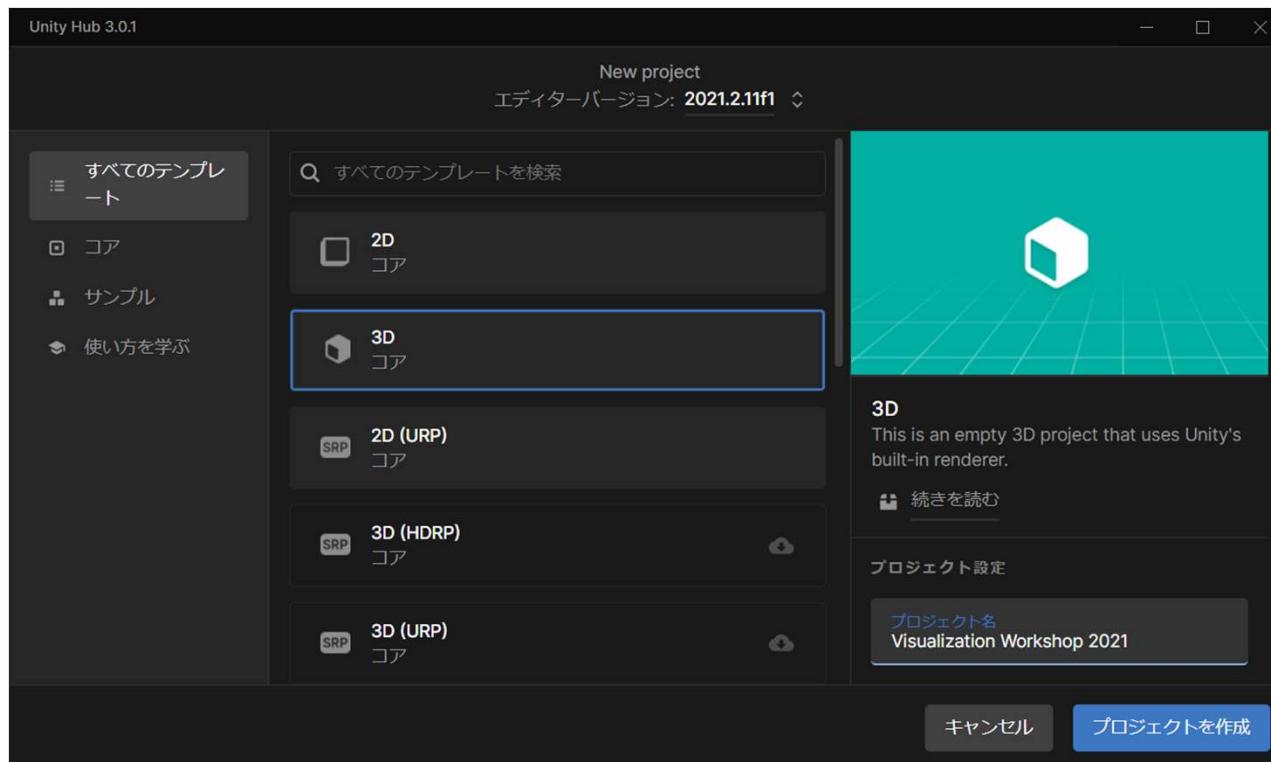
# 新規プロジェクトの作成(1)

Unity Hubを起動し、「新しいプロジェクト」ボタンを押します



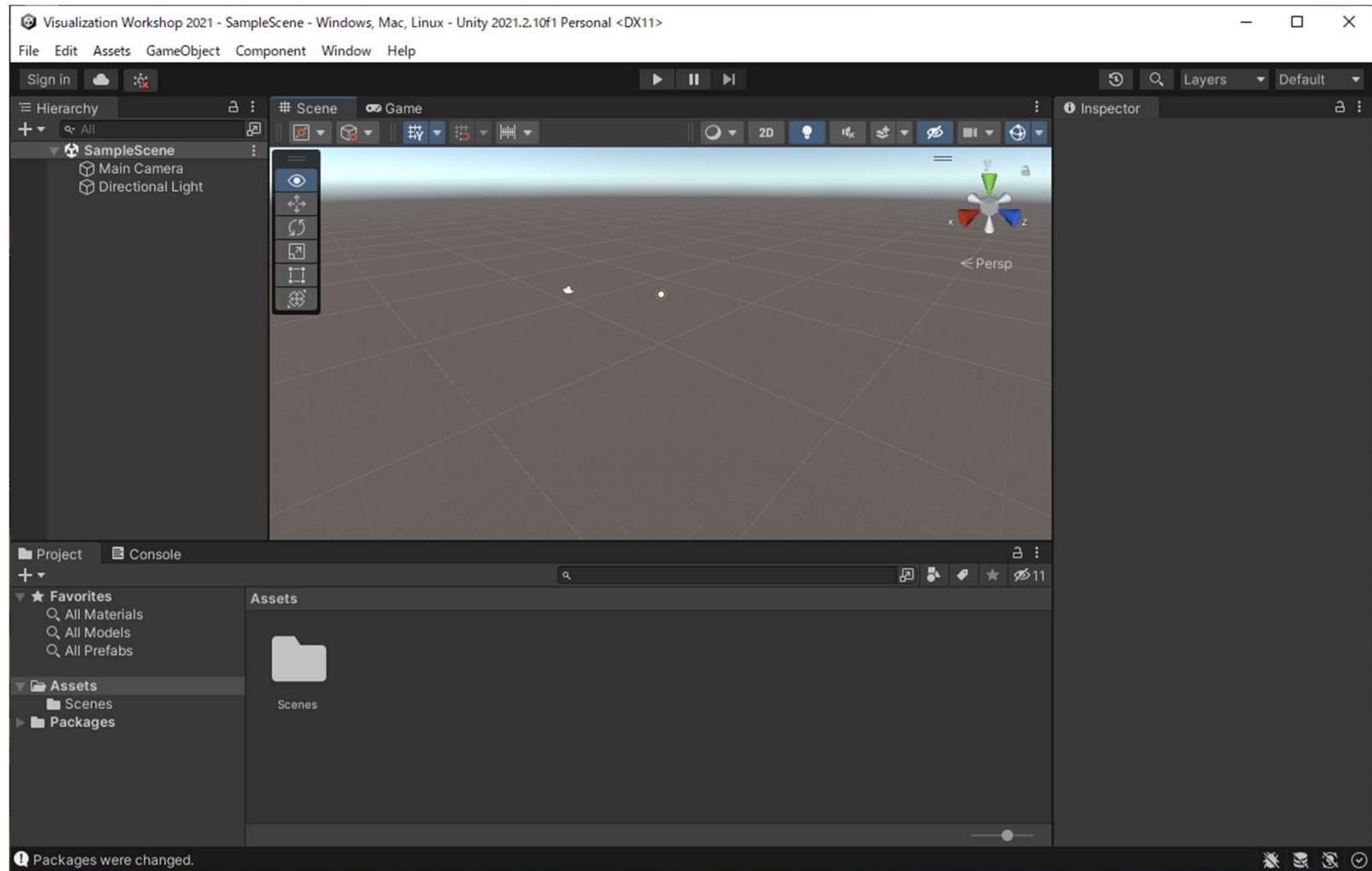
# 新規プロジェクトの作成(2)

エディターバージョンを「2021.2.XX」とし、テンプレートで「3D」を選択、プロジェクト名を設定し「プロジェクトを作成」ボタンを押します。ここではプロジェクト名を「Visualization Workshop 2021」としています。作成したプロジェクトファイルのデフォルトの保存先は、「C:¥Users¥ユーザ名¥Documents¥Unity Projects」となります。

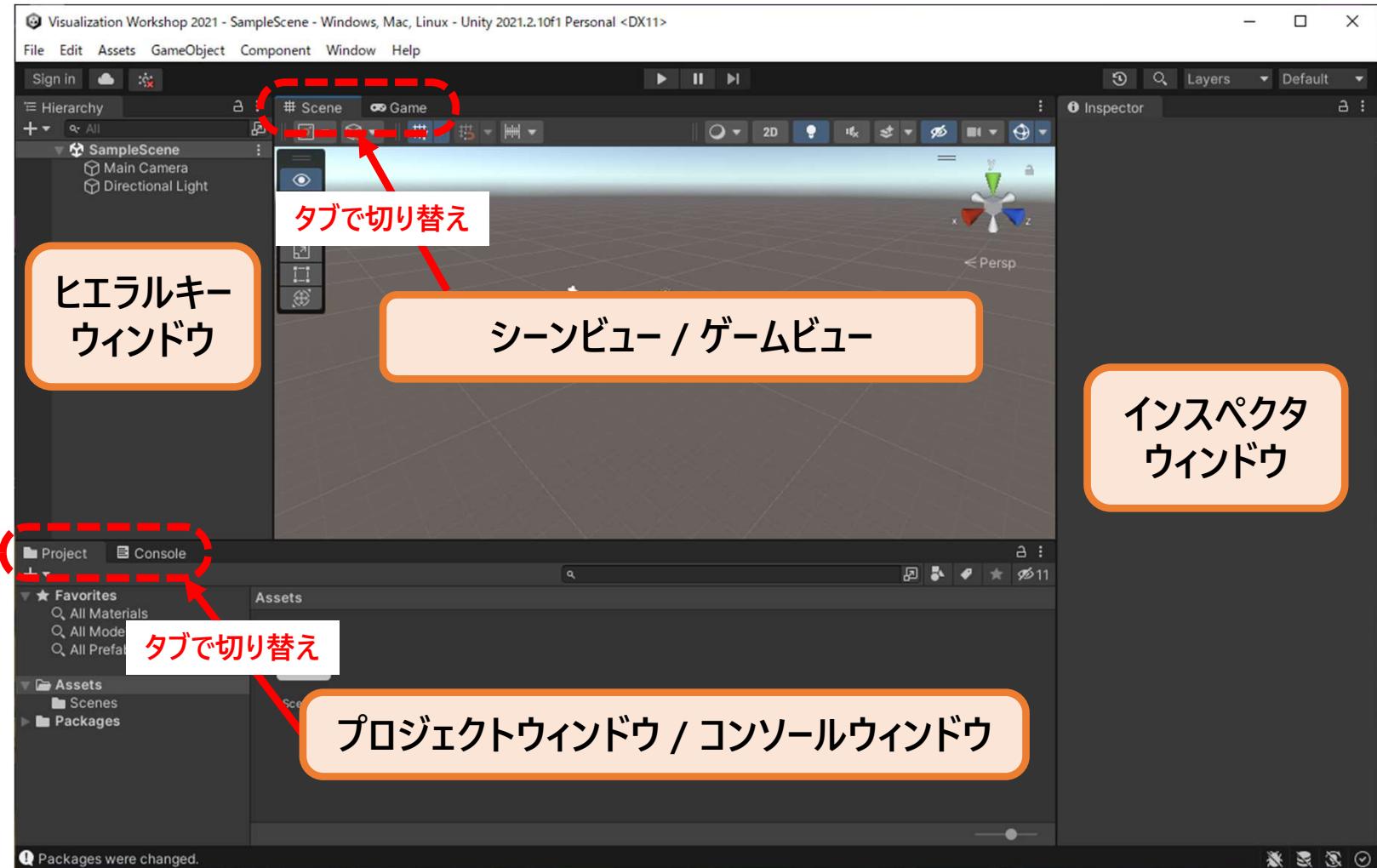


# Unityエディタの起動

作成した新規プロジェクトでUnityエディタが起動します。

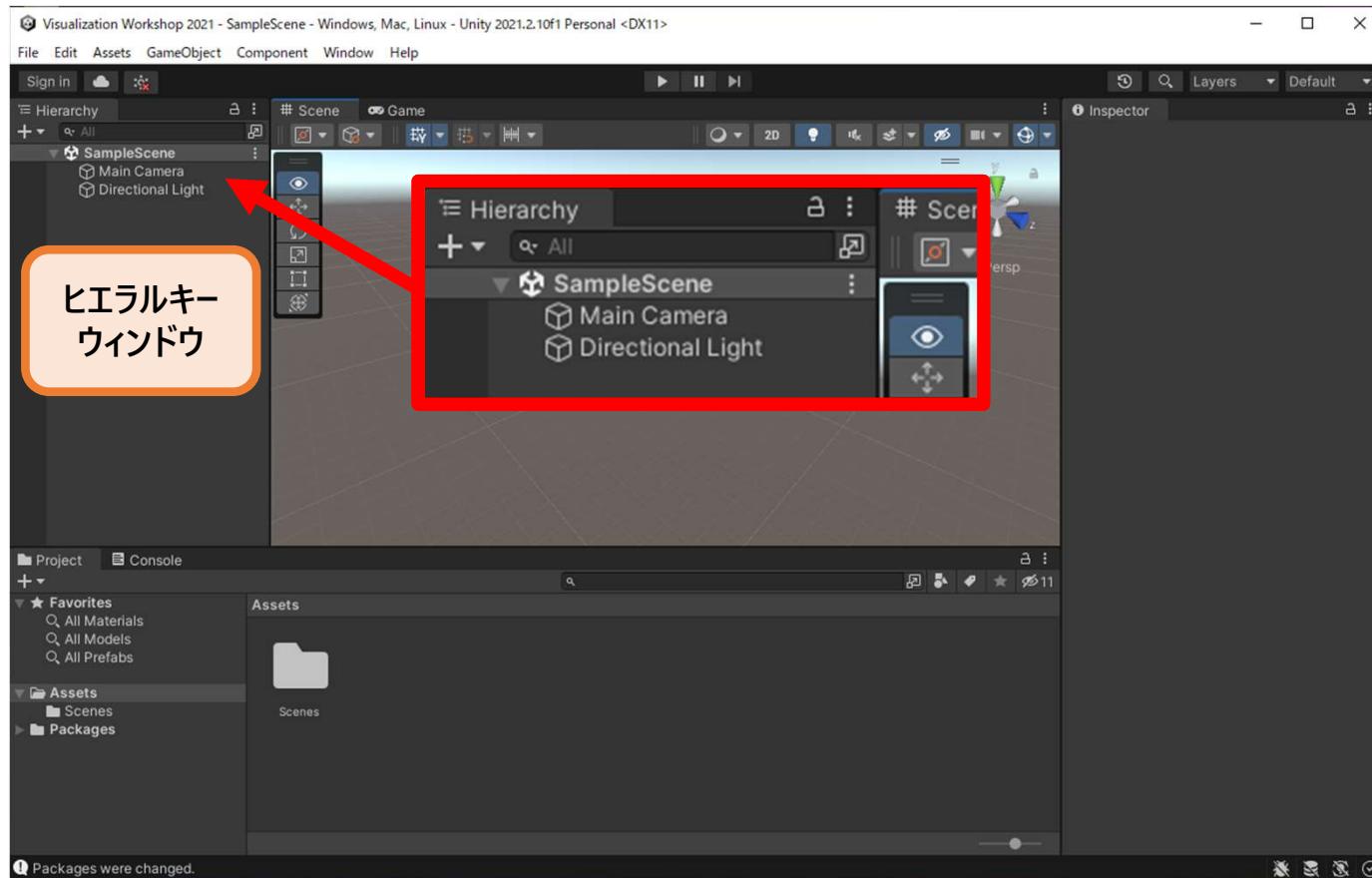


# Unityエディタの画面構成(1)



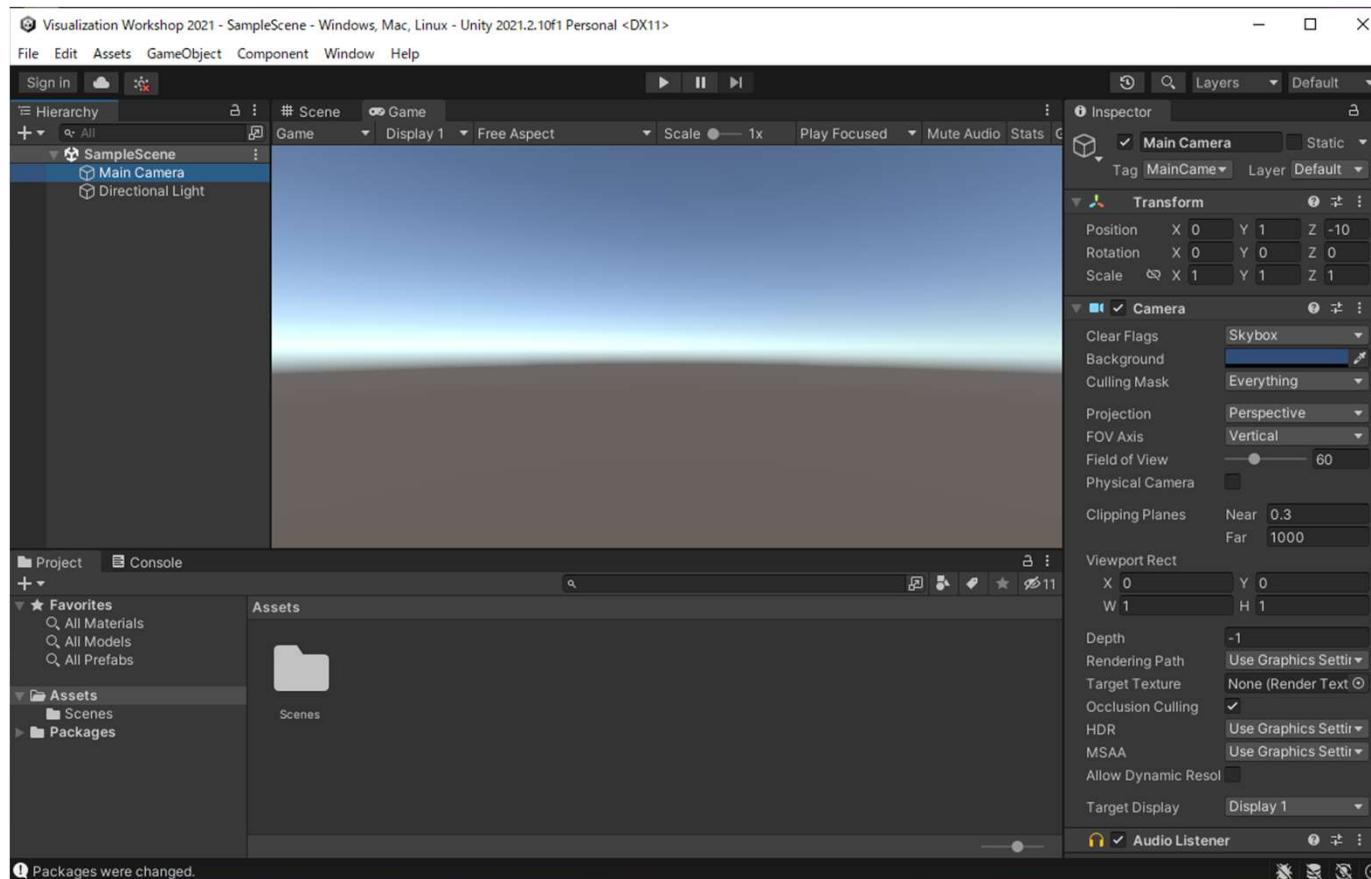
# Unityエディタの画面構成(2)

現在のシーンに配置されているオブジェクト(Unityではゲームオブジェクトと呼ぶ)は、ヒエラルキーで確認することができます。現時点でシーン上にあるゲームオブジェクトは、カメラ「Main Camera」と光源「Directional Light」の二つとなります



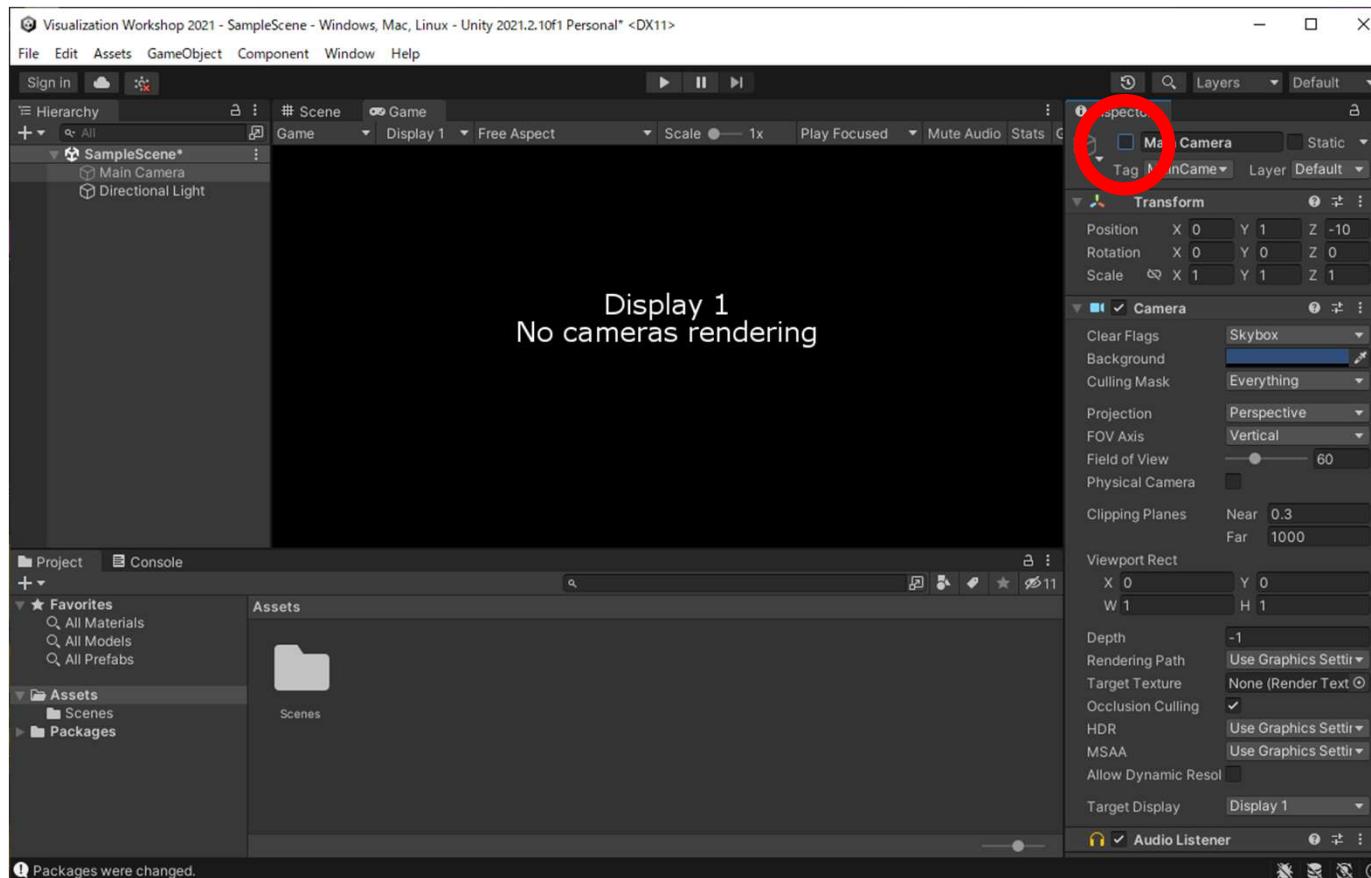
# Unityエディタの画面構成(2)

ヒエラルキーwindowで「Main Camera」を選択すると、インスペクタwindowに「Main Camera」を構成するコンポーネントと、そのパラメータが表示されます。ここからカメラの位置や回転、カメラの設定を変更することができます。



# Unityエディタの画面構成(3)

赤丸部分のチェックボックスで、ゲームオブジェクトの有効/無効の切り替えができます。下図の例では「Main Camera」をオフにしているため、カメラが無効となり、ゲームビューには何も表示されなくなります。確認ができたらチェックボックスをONに戻しておいてください。



# VisAssetsを導入するための準備(1)

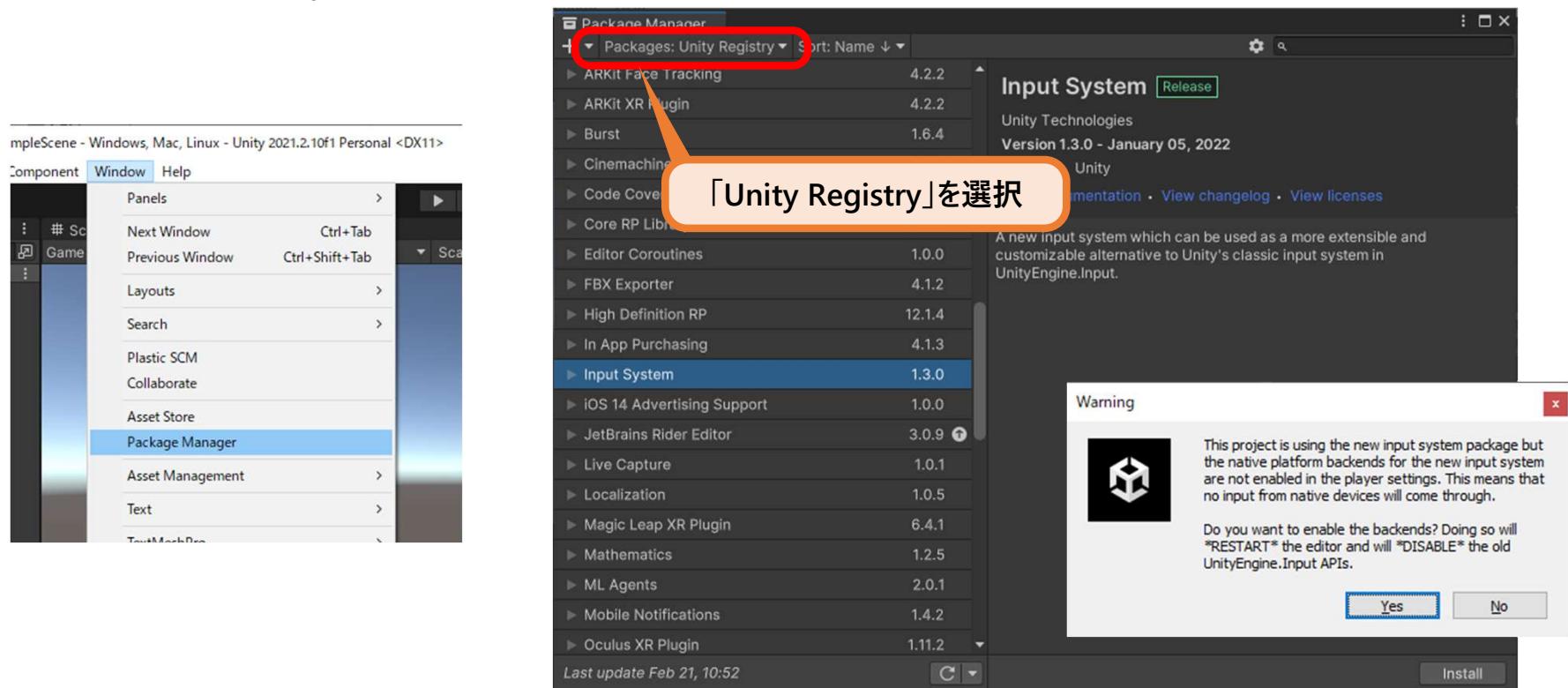
VisAssetsを導入する前に、VisAssetsの動作に必要なパッケージ群をプロジェクトに追加します。VisAssetsを含めた導入順は前後しても構いませんが、続くスライドの順番で導入した方が「必要なファイルが見つからない」等の余計なエラーメッセージを見なくて済みます。

パッケージ名	機能
Input System (New Input System)	ゲームパッド・キーボード・マウス・センサー等の各種入力デバイスからの入力を汎用的・統合的に取り扱うための追加パッケージ。VisAssetsでは、旧来のInput Systemと、本パッケージが提供する新しいInput Systemの両方を使用しているため導入が必要（Unity社提供パッケージ）
XR Interaction Toolkit	VR/ARインタラクションシステムを開発するためのツールキット (Unity社提供パッケージ、プレリリース版)
Simple File Browser	ファイル選択のためのダイアログを使用するためのパッケージ (Süleyman Yasir KULA氏開発、無償公開)

# VisAssetsを導入するための準備(2)

## Input System のインストール(1)

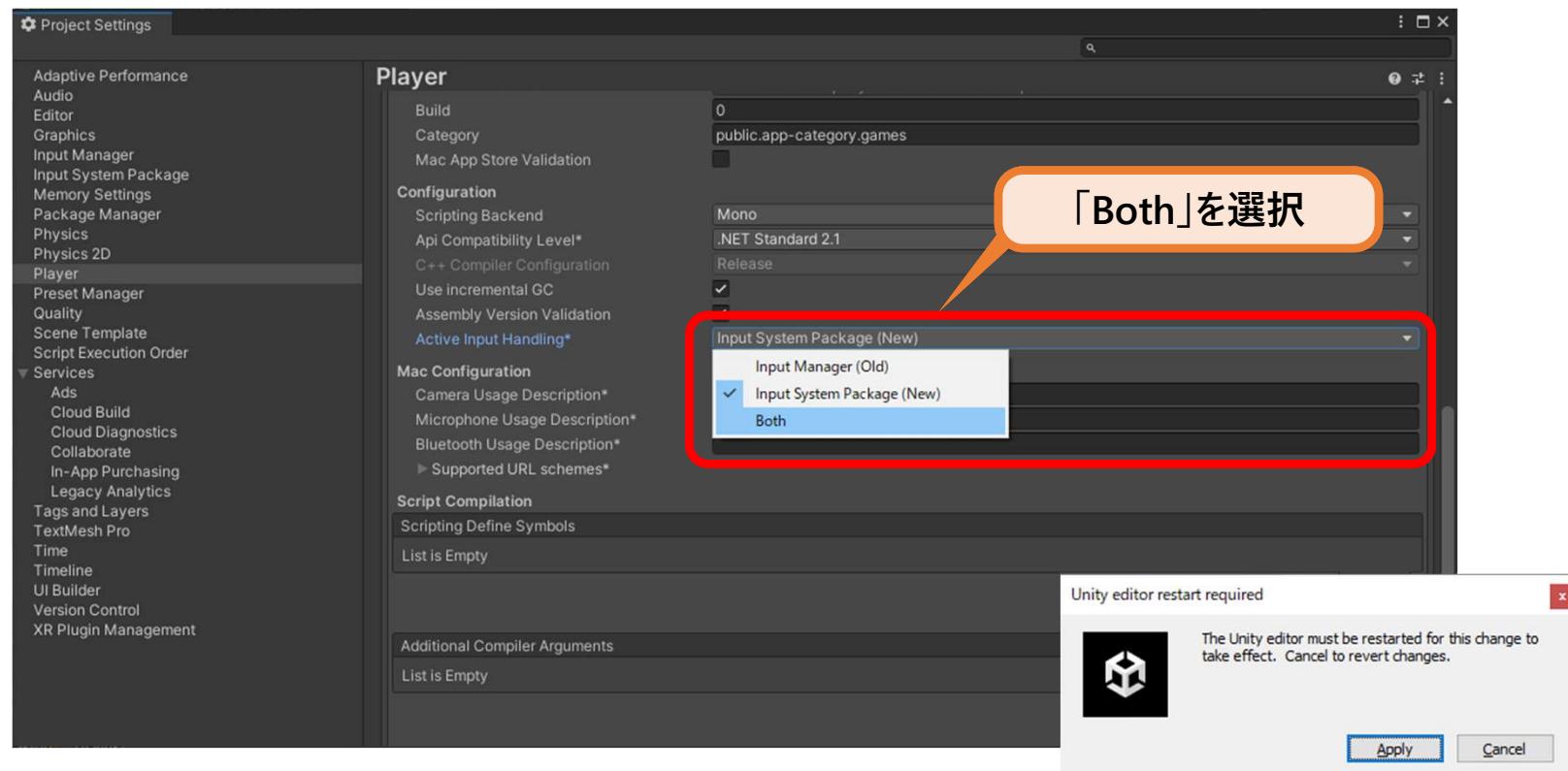
メインメニューの[Window]–[Package Manager]でパッケージマネージャを開き、Unity Registry カテゴリの中にある「Input System」パッケージをインストールします。インストールの最後に Warningダイアログが表示されますが、「Yes」ボタンを押してそのまま進めてください。Unityエディタが再起動されます。



# VisAssetsを導入するための準備(3)

## Input System のインストール(2)

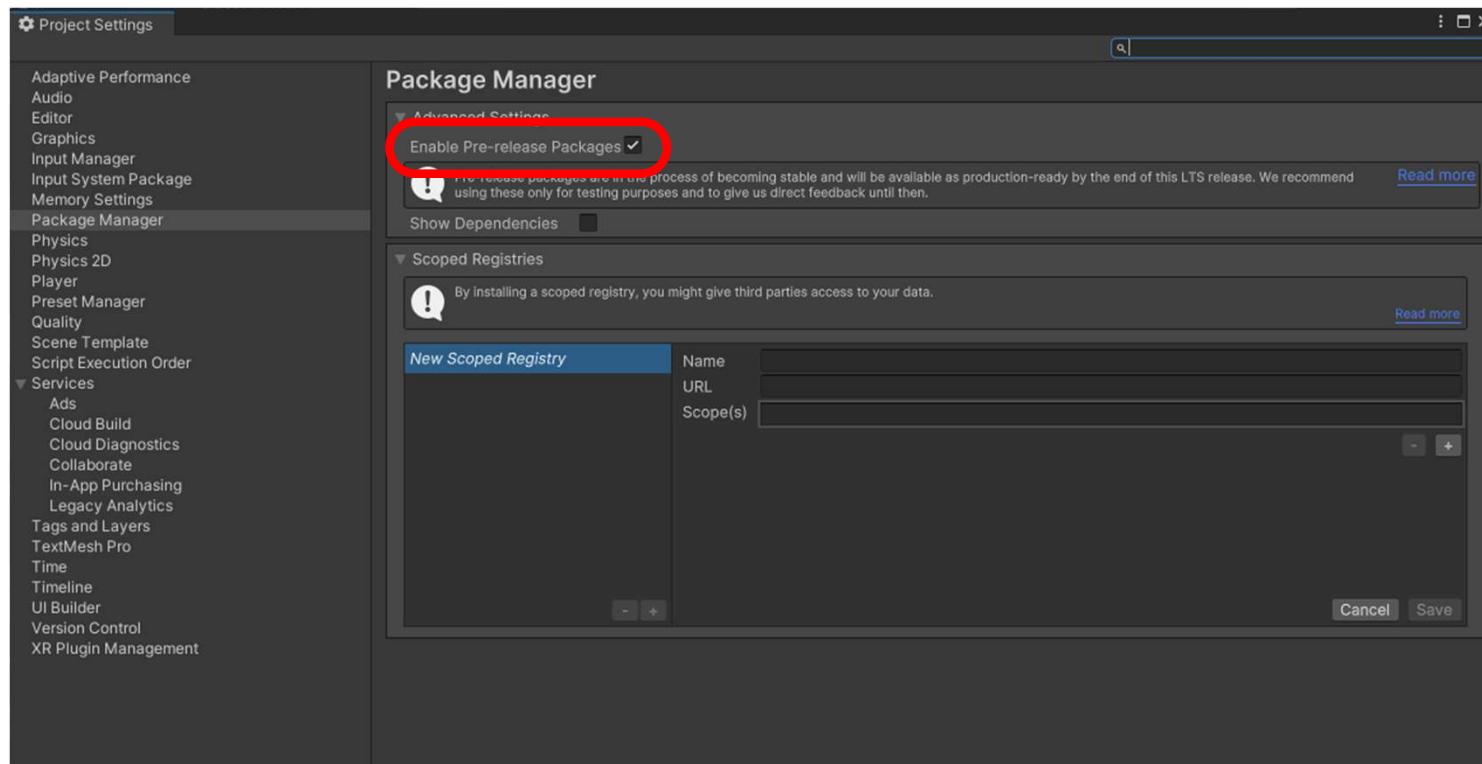
Unityエディタの再起動後、メインメニューの[Edit]–[Project Settings]でプロジェクト設定を開き、[Player]–[Other Settings]内にある「Active Input Handling」の項目で「Both」を選択します。選択後、Unityエディタの再起動を求められますのでそれに従い再起動します。



# VisAssetsを導入するための準備(4)

## XR Interaction Toolkit のインストール(1)

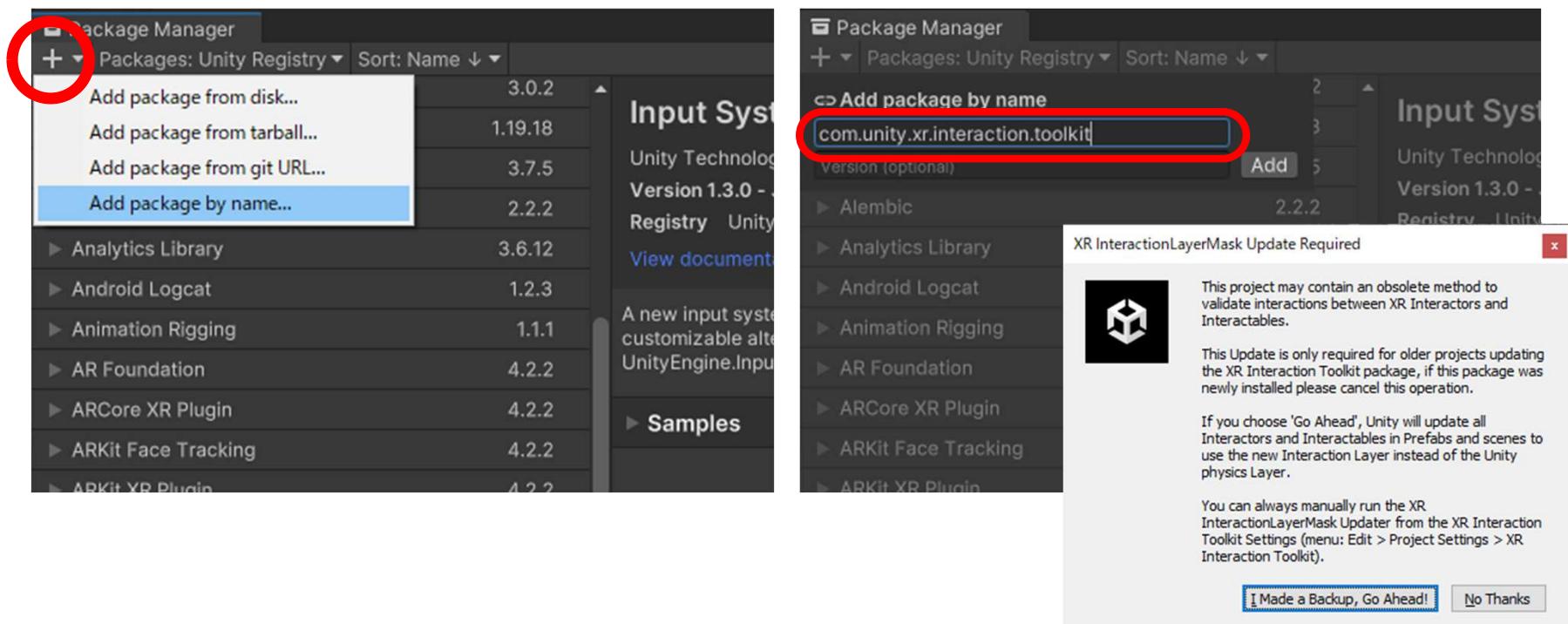
Unityエディタの再起動後、メインメニューの[Edit]–[Project Settings]でプロジェクト設定を開き、[Package Manager]–[Advanced Settings]内にある「Enable Pre-release Packages」のチェックボックスをONにします。



# VisAssetsを導入するための準備(5)

## XR Interaction Toolkit のインストール(2)

Package Managerを開き、左上にある「+」ボタンを押して「Add package by name」を選択、「com.unity.xr.interaction.toolkit」と入力し「Add」ボタンを押します。インストールの最後に表示されるダイアログでは「I Made a Backup, Go Ahead!」ボタンを押してください。



# VisAssetsを導入するための準備(6)

## Simple File Browser のインストール(1)

Webブラウザで <https://github.com/yasirkula/UnitySimpleFileBrowser> にアクセスし、Releasesページ内の「SimpleFileBrowser.unitypackage」をダウンロードします。

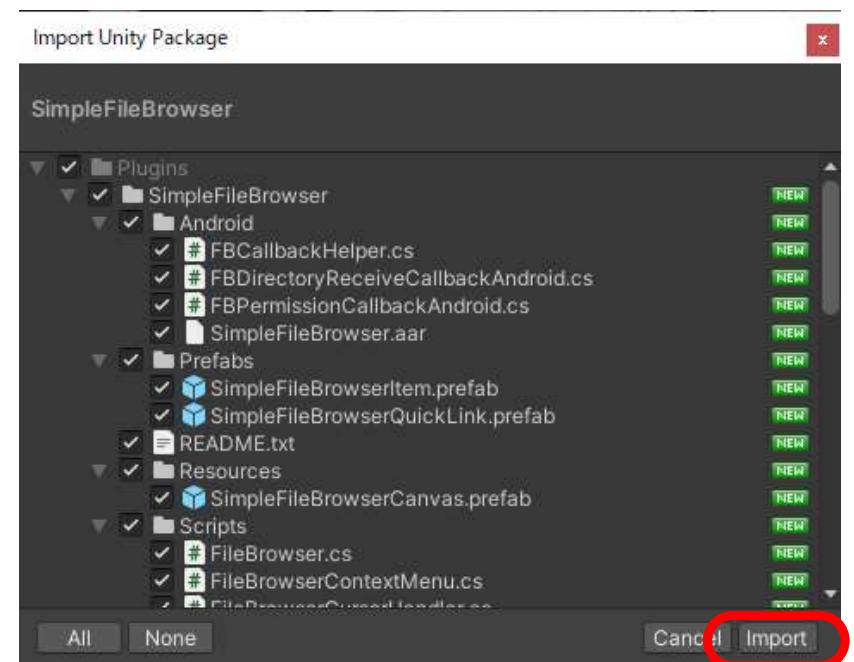
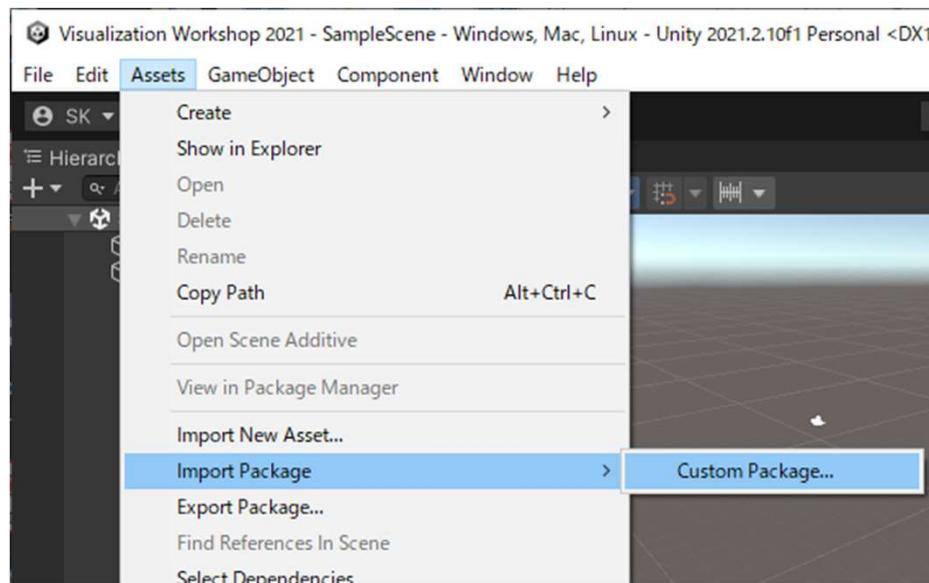
This screenshot shows the main GitHub repository page for 'UnitySimpleFileBrowser'. It includes a list of pull requests, issues, and discussions. On the right side, there's an 'About' section with a brief description of the project: 'A uGUI based runtime file browser for Unity 3D (draggable and resizable)'. Below this is a 'Releases' section with 23 entries. The first release, 'v1.5.3 (Latest)', is highlighted with a red circle. This specific release was made on 19 Jan and includes a note about adding a timeout for Windows network drives.

This screenshot shows the 'Releases' page for the same repository. It lists all releases from v1.5.3 down to v1.5.2. The 'Assets' section for each release is visible, showing the 'SimpleFileBrowser.unitypackage' file. The 'v1.5.3' entry is circled in red, indicating it is the target for download. The package file size is listed as 235 KB.

# VisAssetsを導入するための準備(7)

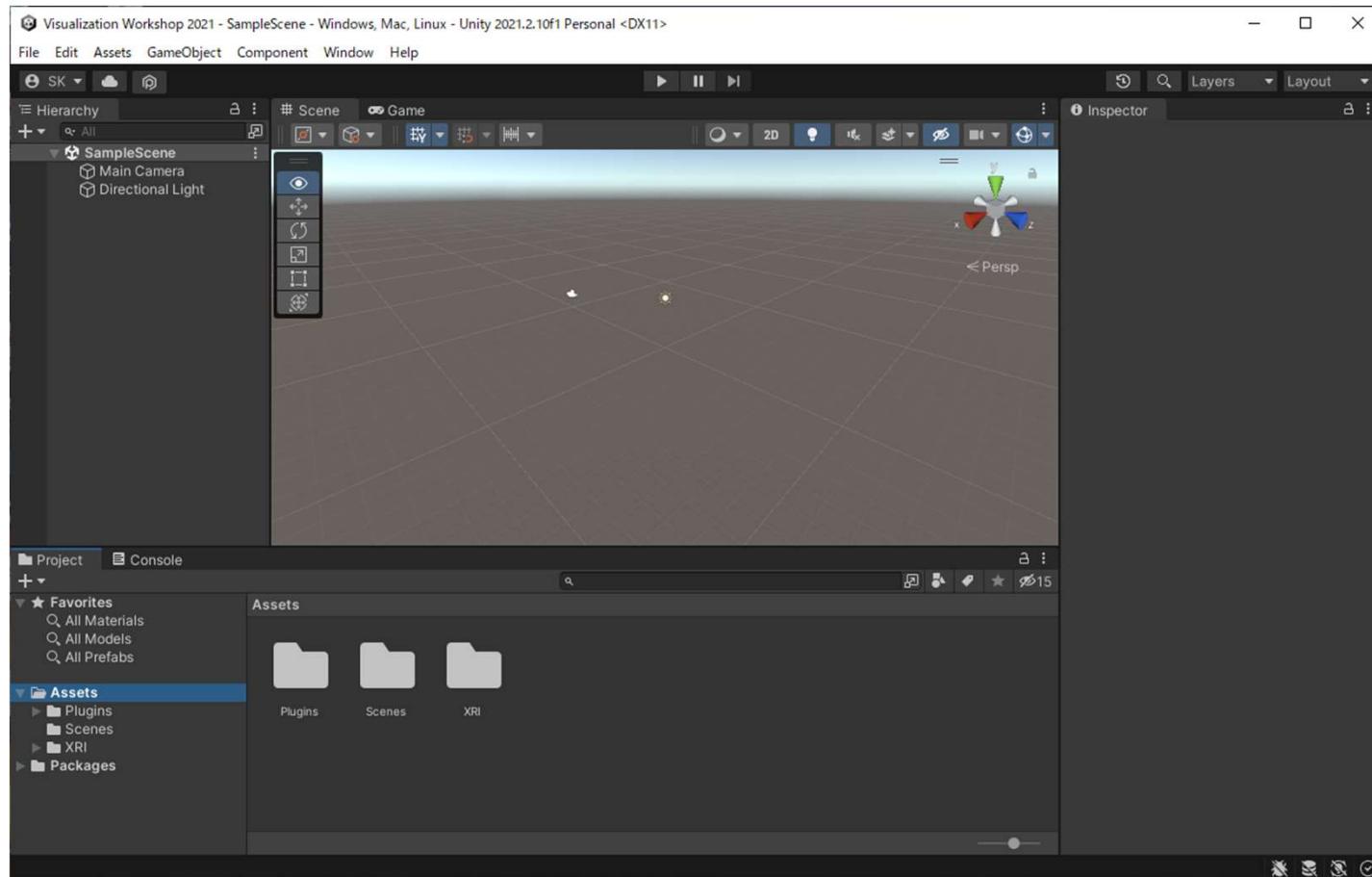
## Simple File Browser のインストール(2)

メインメニューから[Assets]–[Import Package]–[Custom Package]を選択し、ダウンロードした SimpleFileBrowser.unitypackage をプロジェクトにインポートします。



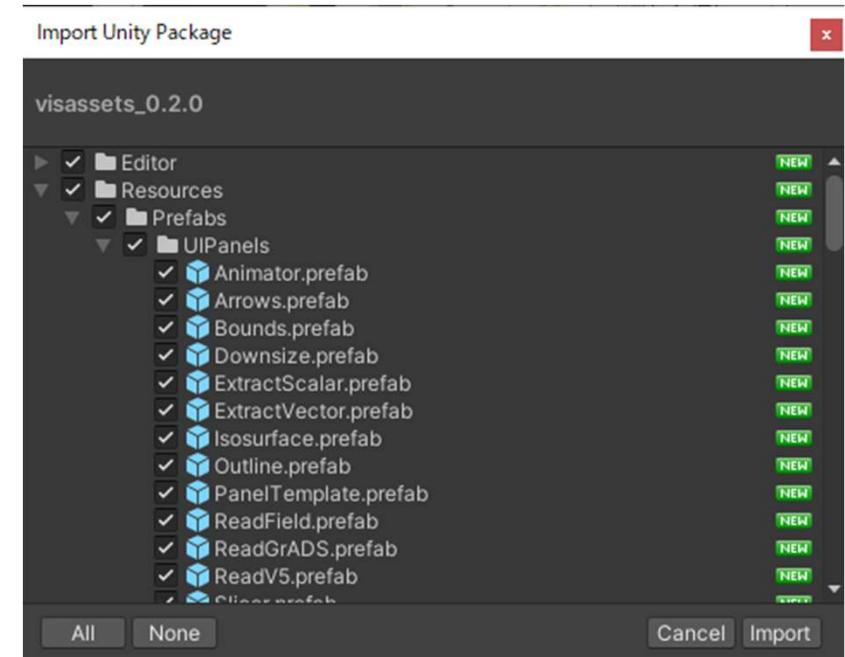
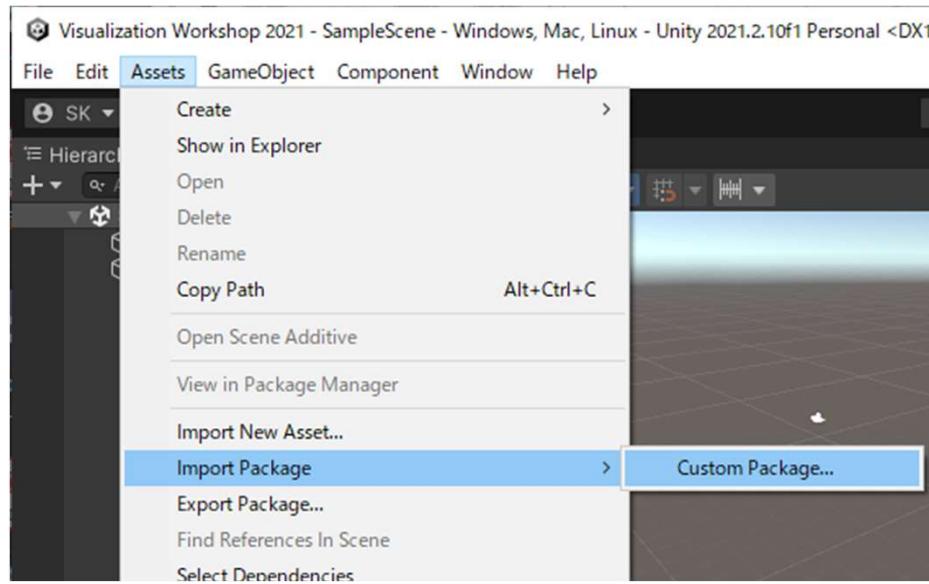
# VisAssetsを導入するための準備(8)

ここまで状態は下図の通りです。Assetsフォルダに「Plugins」と「XRI」の二つのフォルダが追加されています。



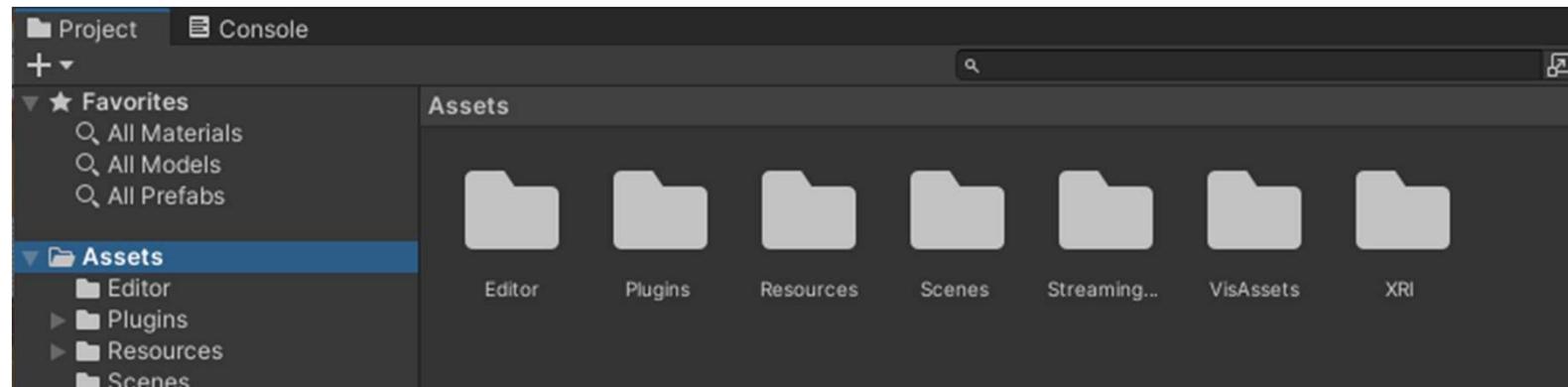
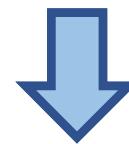
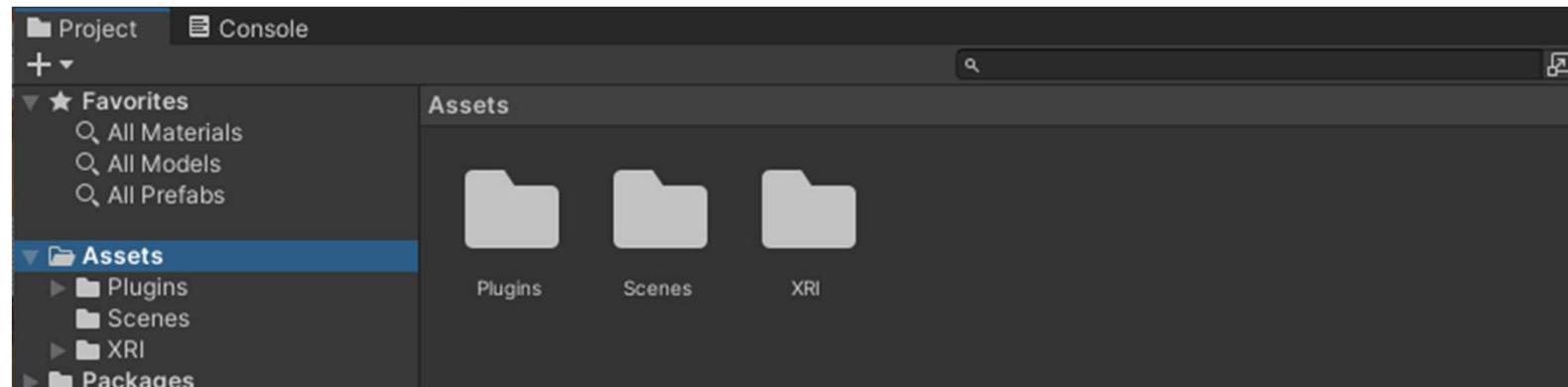
# VisAssetsの導入(1)

メインメニューから[Assets]–[Import Package]–[Custom Package]を選択し、ダウンロードしたVisAssetsのUnityパッケージをプロジェクトにインポートします。



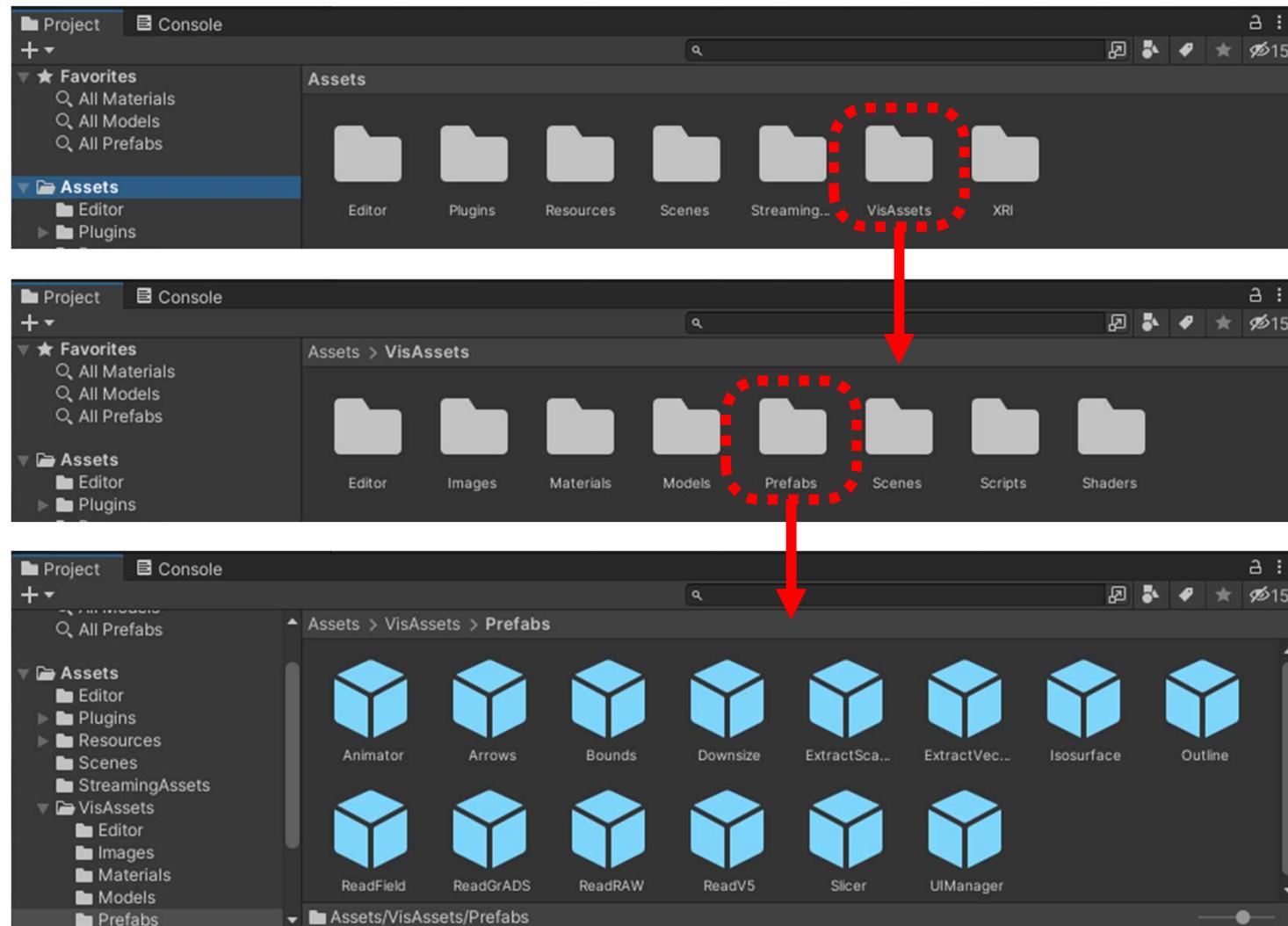
# VisAssetsの導入(2)

VisAssets導入前後で、プロジェクトウィンドウの表示が下図のように変化します。



# 可視化モジュールの場所

可視化モジュールは VisAssets/Prefabs 以下にあります。



# データを読み込む(1)

VisAssetsでは  $I \times J \times K (\times \text{時刻})$  の構造格子データを扱うことができます。

サンプルモジュールとして、4つのデータ読み込み用モジュールを用意しています。  
まずはテキストファイルからデータを読み込む「ReadField」モジュールを使ってみます。



モジュール名	対象データ
ReadField	テキストファイル
ReadRAW	バイナリファイル(单精度または倍精度実数)
ReadV5	<a href="#">CAVE型VR装置用可視化ソフトウェア「VFIVE」用データ (テキスト + バイナリ)</a>
ReadGrADS	<a href="#">大気海洋データ用可視化ソフトウェア「GrADS」用データ (テキスト + バイナリ)</a>

# データを読み込む(2)

ReadFieldモジュールで読み込み可能なテキストデータの例

11, 11, 8 格子数 ( $I \times J \times K$ )

```
-5, -5, -3.5, 7.889866919, -5, -5
-4, -5, -3.5, 7.297259760, -4, -5
-3, -5, -3.5, 6.800735254, -3, -5
-2, -5, -3.5, 6.422616289, -2, -5
-1, -5, -3.5, 6.184658438, -1, -5
0, -5, -3.5, 6.103277808, 0, -5
1, -5, -3.5, 6.184658438, 1, -5
2, -5, -3.5, 6.422616289, 2, -5
3, -5, -3.5, 6.800735254, 3, -5
4, -5, -3.5, 7.297259760, 4, -5
5, -5, -3.5, 7.889866919, 5, -5
```

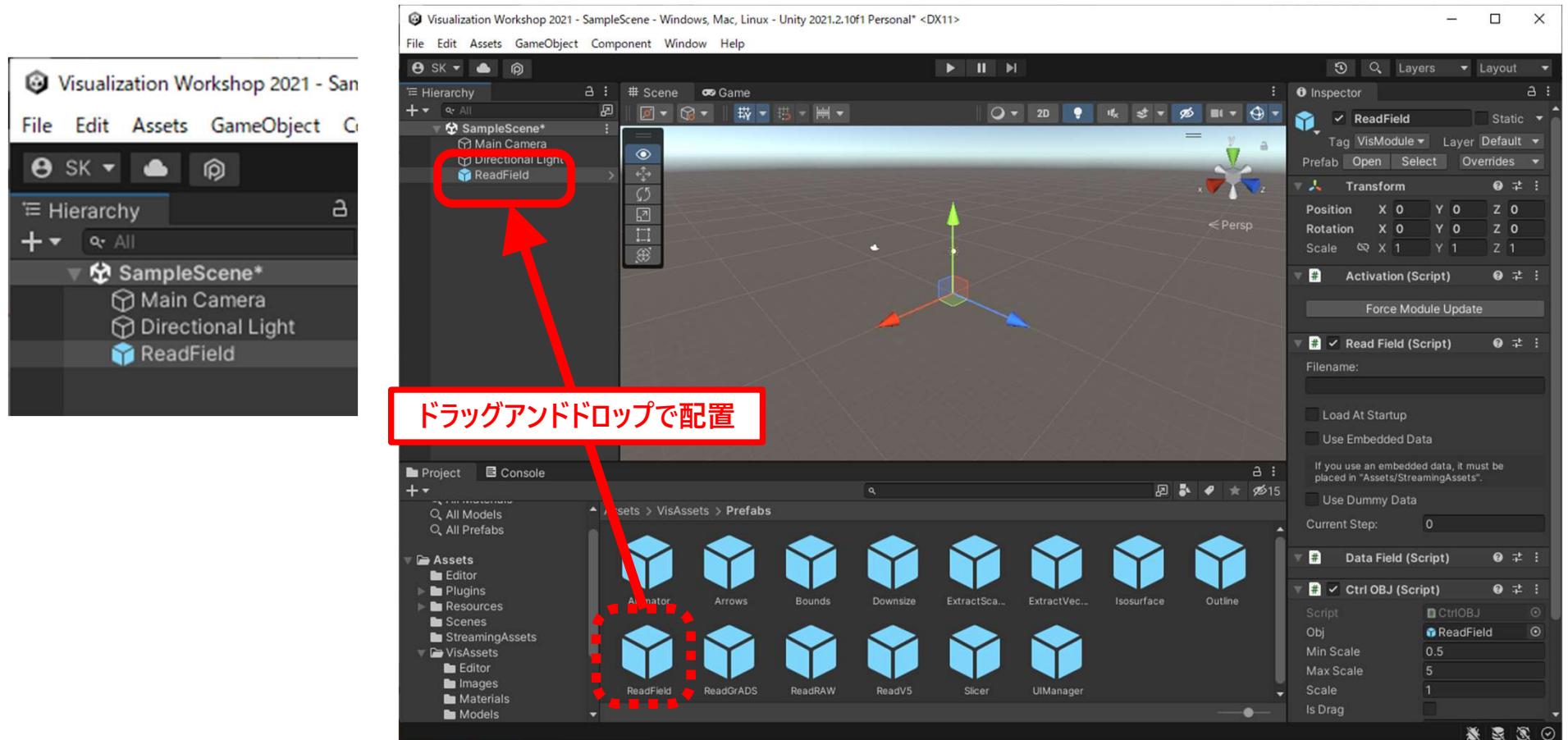
～(後略)～

格子点座標

格子点上の値  
(3成分)

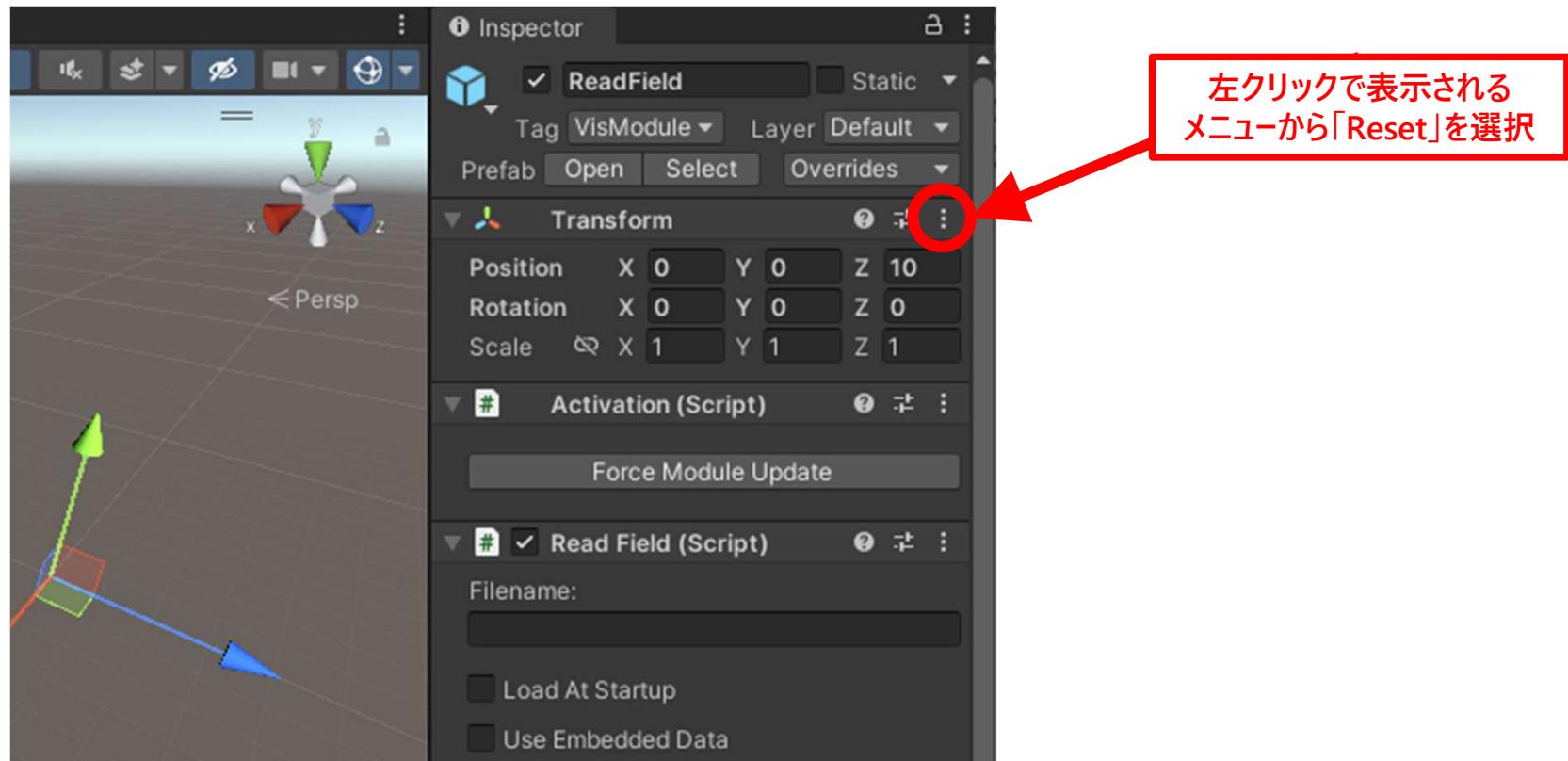
# データを読み込む(3)

プロジェクトウィンドウから、「ReadField」モジュールをヒエラルキーインドウにドラッグアンドドロップします。下図のように、Main CameraやDirectional Lightと同じ階層に配置してください。



# データを読み込む(4)

ヒエラルキーインペクタウインドウで「ReadField」モジュールを選択し、インスペクタウインドウのTransformのPositionが原点(X=0, Y=0, Z=0)でない場合はリセットし、Positionの「Z」を「10」としておきます。



# データを読み込む(5)

「ReadField」モジュールでのデータ読み込み方法には以下の三種類があります。

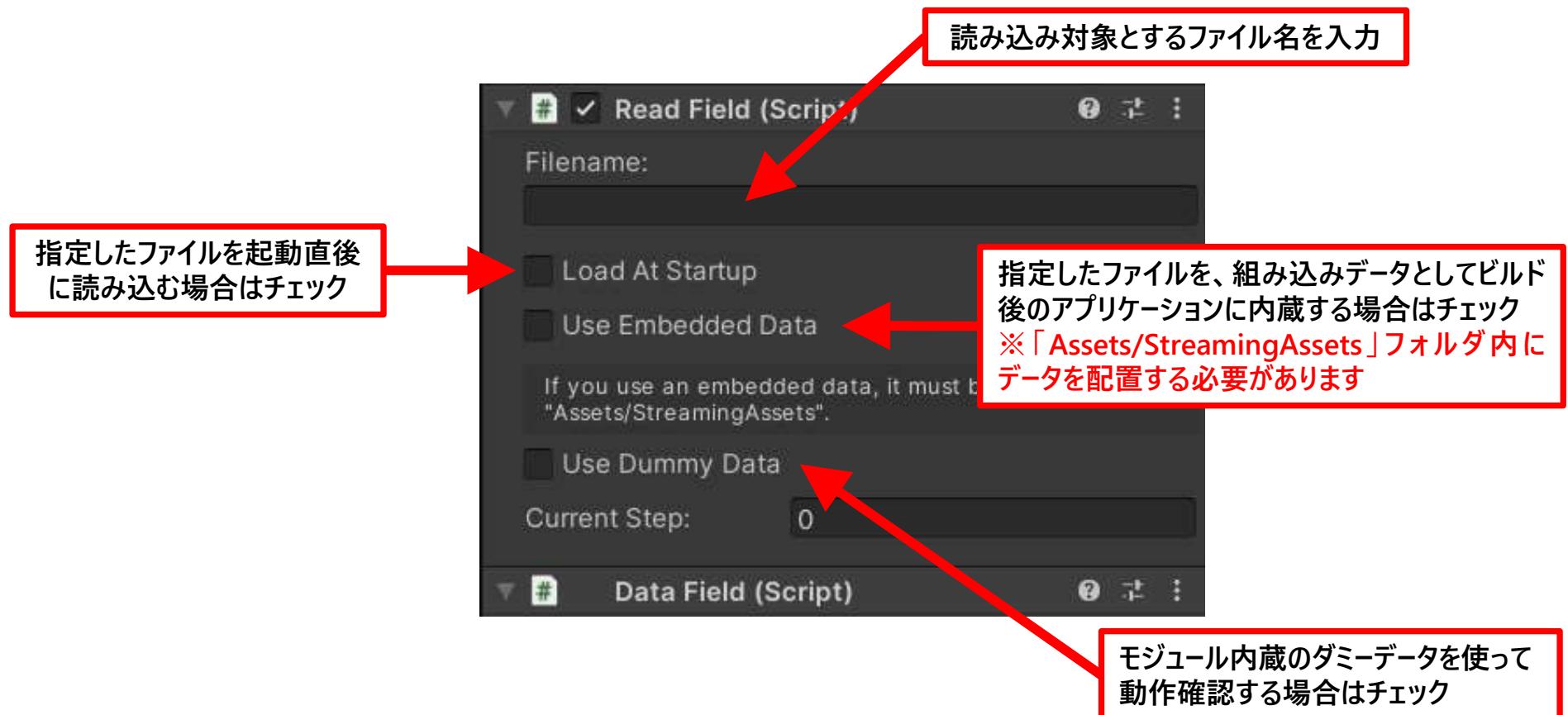
1. モジュール内蔵のテスト用データを使う
2. 指定したデータファイルを読み込む
3. ビルド後のアプリケーション実行中に、ファイルダイアログ※からデータファイルを指定して読み込む

本チュートリアルでは 1. の「モジュール内蔵のテスト用データを使う方法」と、2. の「指定したデータファイルをビルド後のアプリケーションに組み込む方法」の二つを紹介します。

※ 「Simple File Browser」はここで使っています

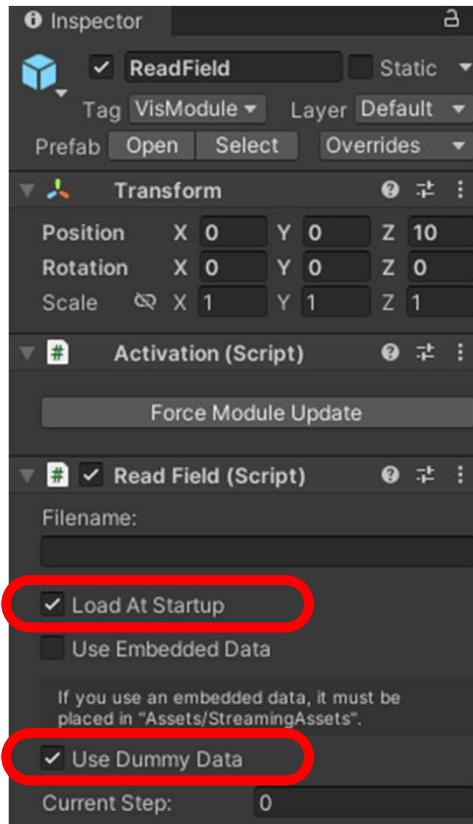
# データを読み込む(6)

「ReadField」モジュールのインスペクタ表示と設定項目は下記の通りです。多くの設定は、他のデータ読み込みモジュールでも共通の項目となります。

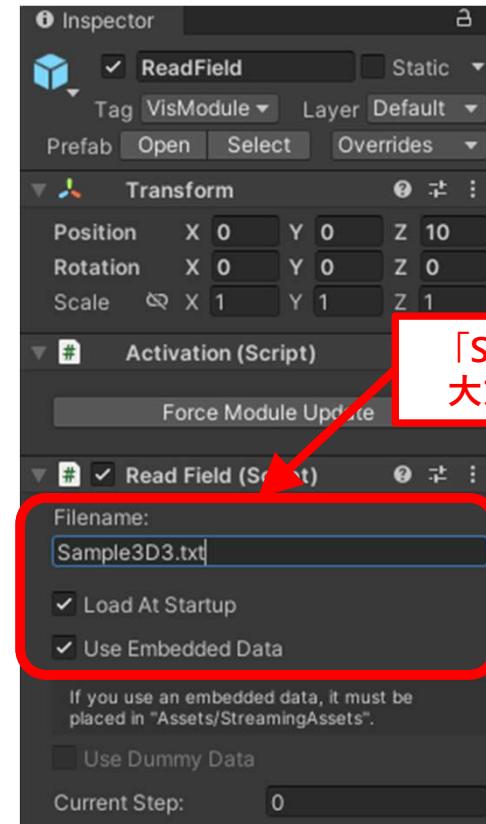


# データを読み込む(7)

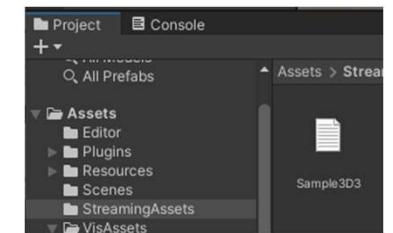
ヒエラルキーインドウで「ReadField」を選択し、インスペクタインドウ内の「Read Field (Script)」のオプションを下記のいずれかに設定します。



モジュール内蔵のテスト用データを使用する場合  
11×11×11、要素数2

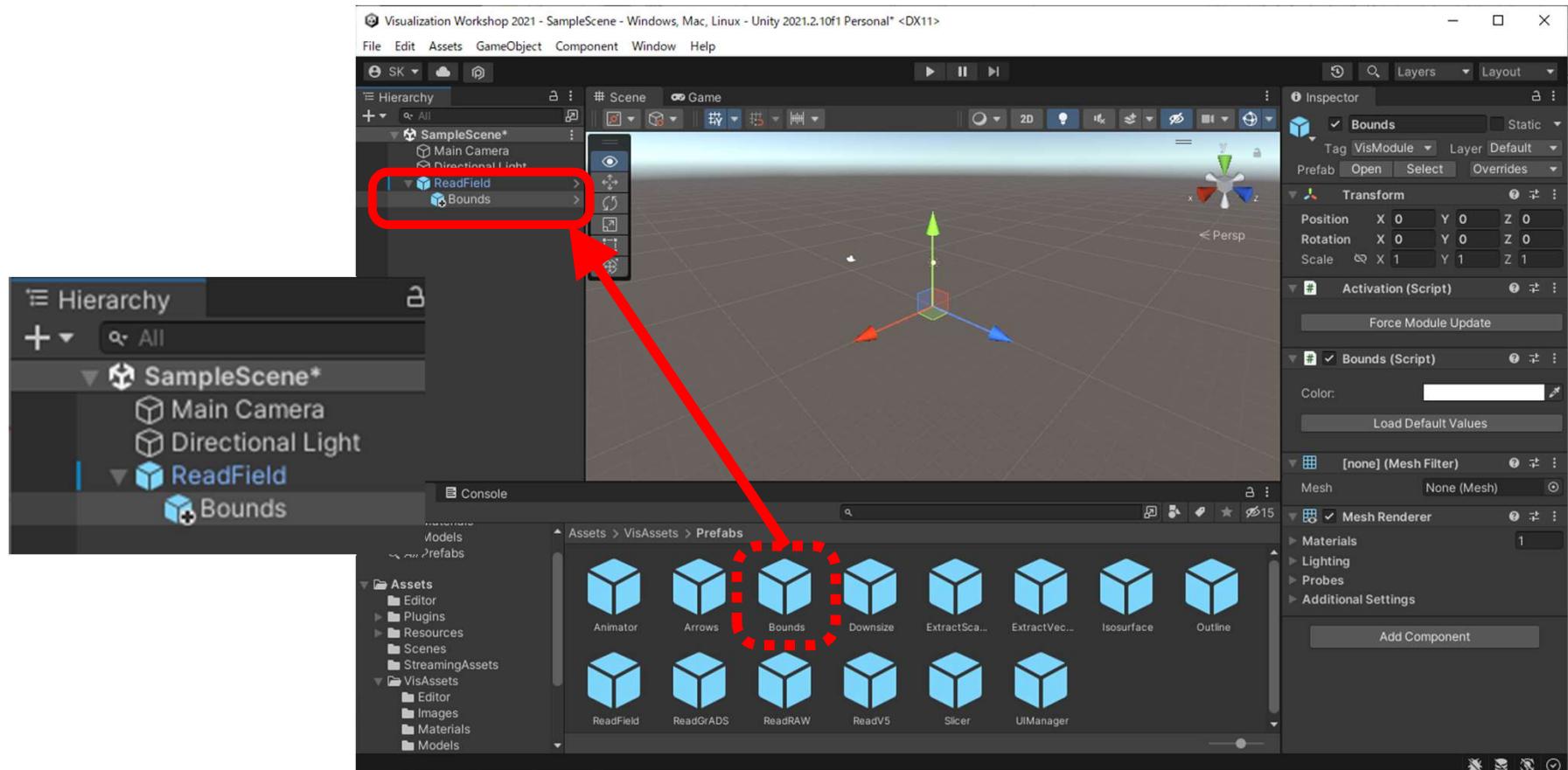


データファイルをアプリケーションに組み込む場合  
11×11×8、要素数3  
(Assets/StreamingAssetsフォルダに配置済み)



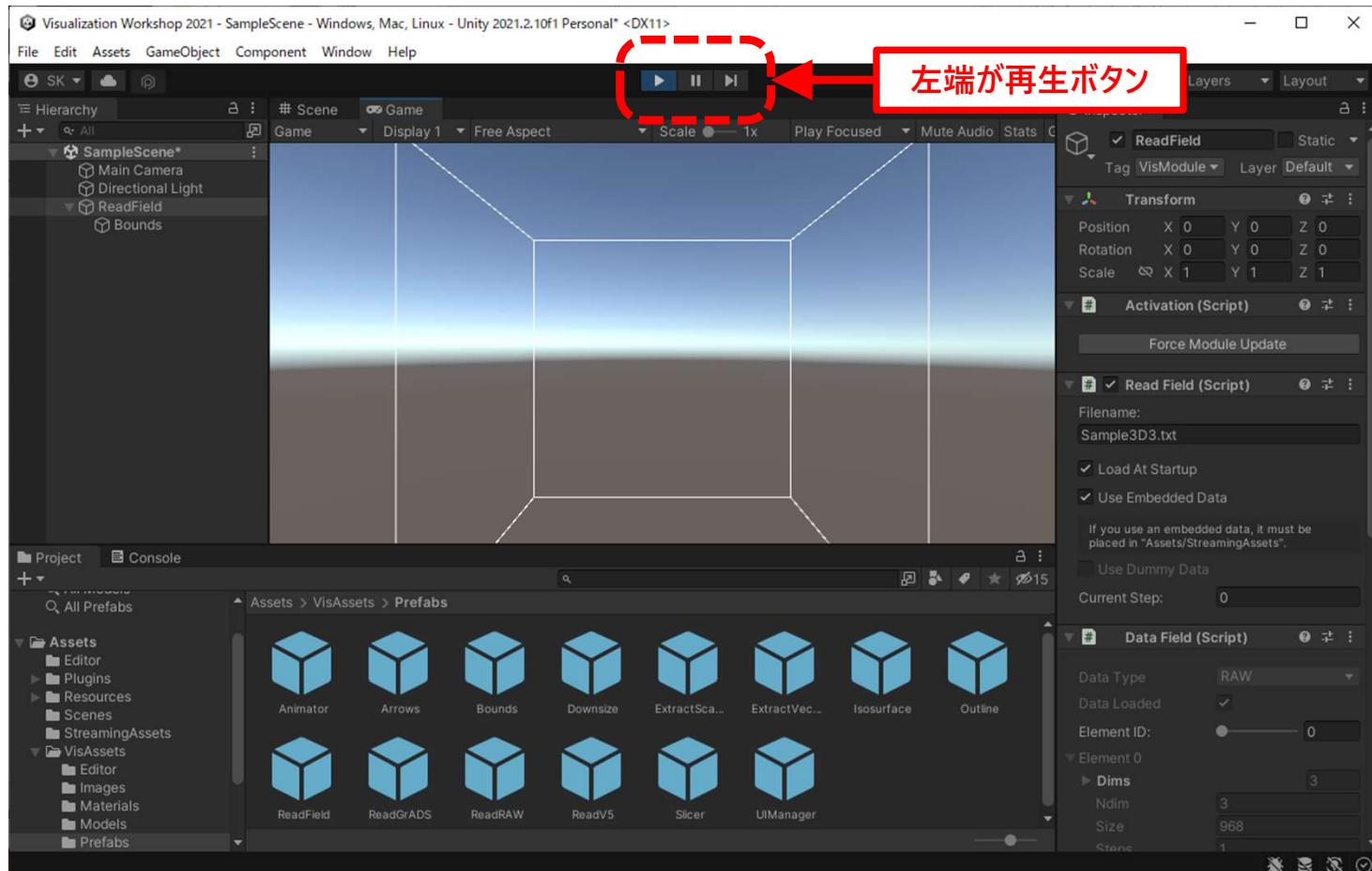
# データを読み込む(8)

データが正しく読み込まれたかどうかの確認のため、データ境界線を表示するBoundsモジュールを使ってみます。プロジェクトウィンドウから「Bounds」モジュールをヒエラルキーイングの「ReadField」モジュールの子となるようにドラッグアンドドロップで配置します。



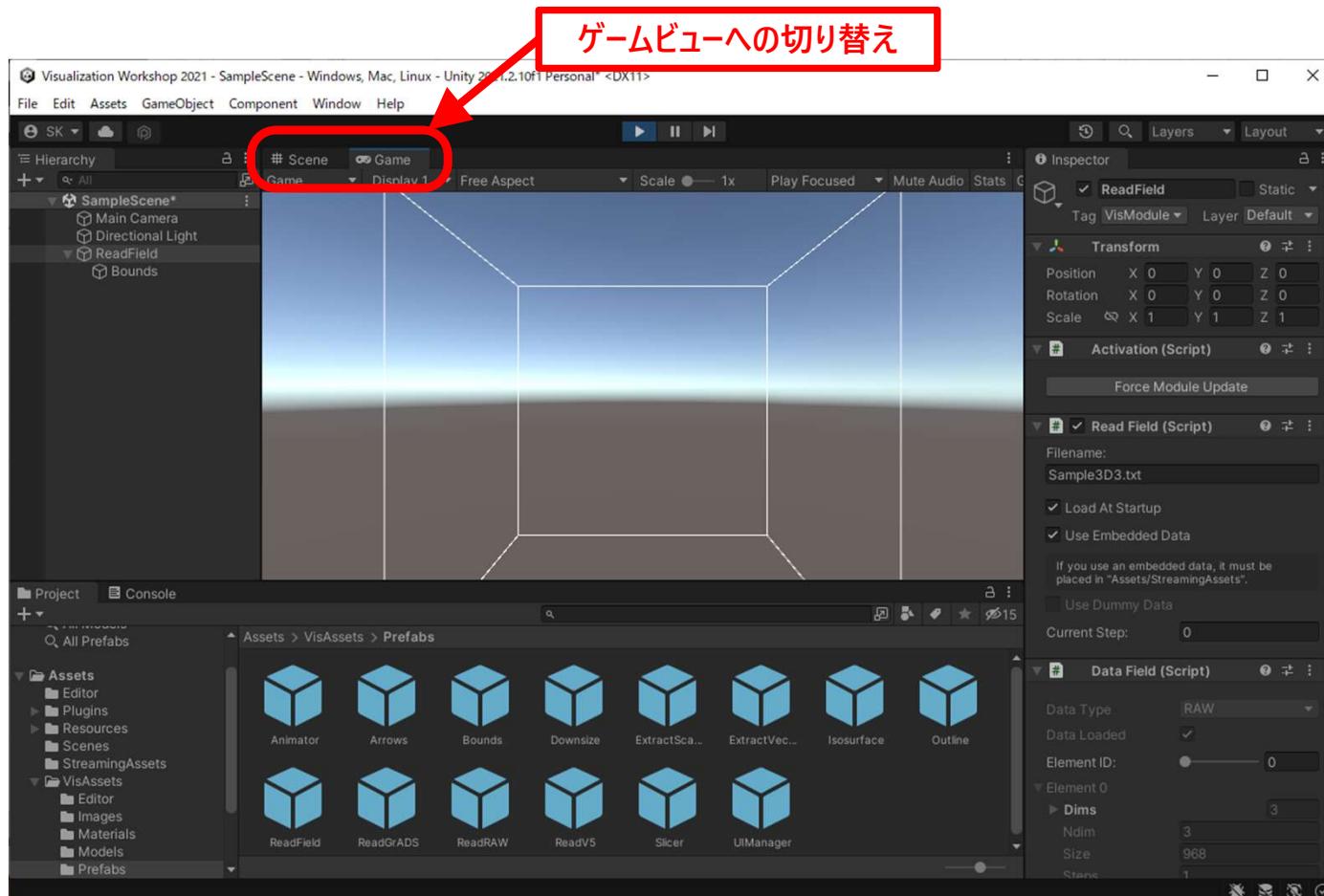
# ここまでの中の状態の確認(1)

ツールバーの再生ボタンを押すと現在のシーンがプレビュー実行され、ゲームビューに実行結果が表示されます(この段階ではデータ領域を示す線のみが表示されます)。



# ここまで状態の確認(2)

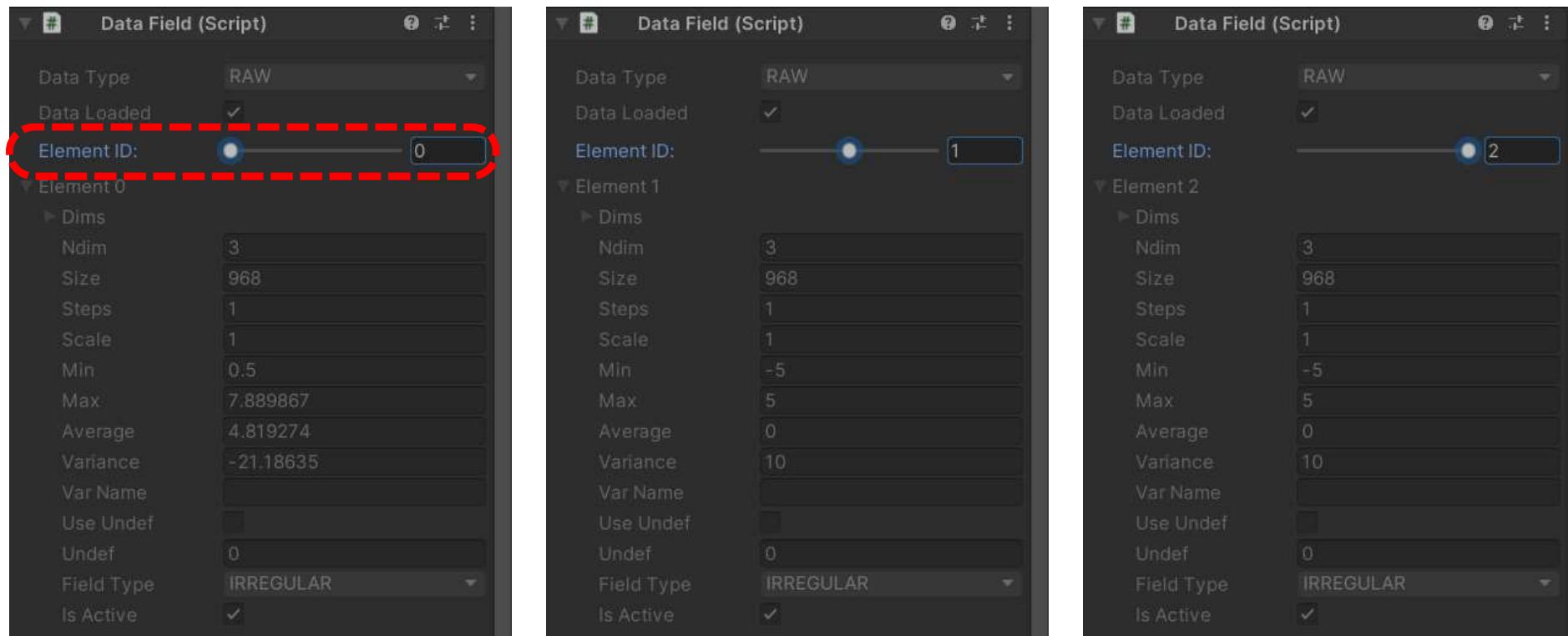
ゲームビューに切り替えることで、ゲームビューウィンドウ内でマウスの左ドラッグで回転、ホイールの前後で拡大縮小表示ができます。



# ここまで状態の確認(3)

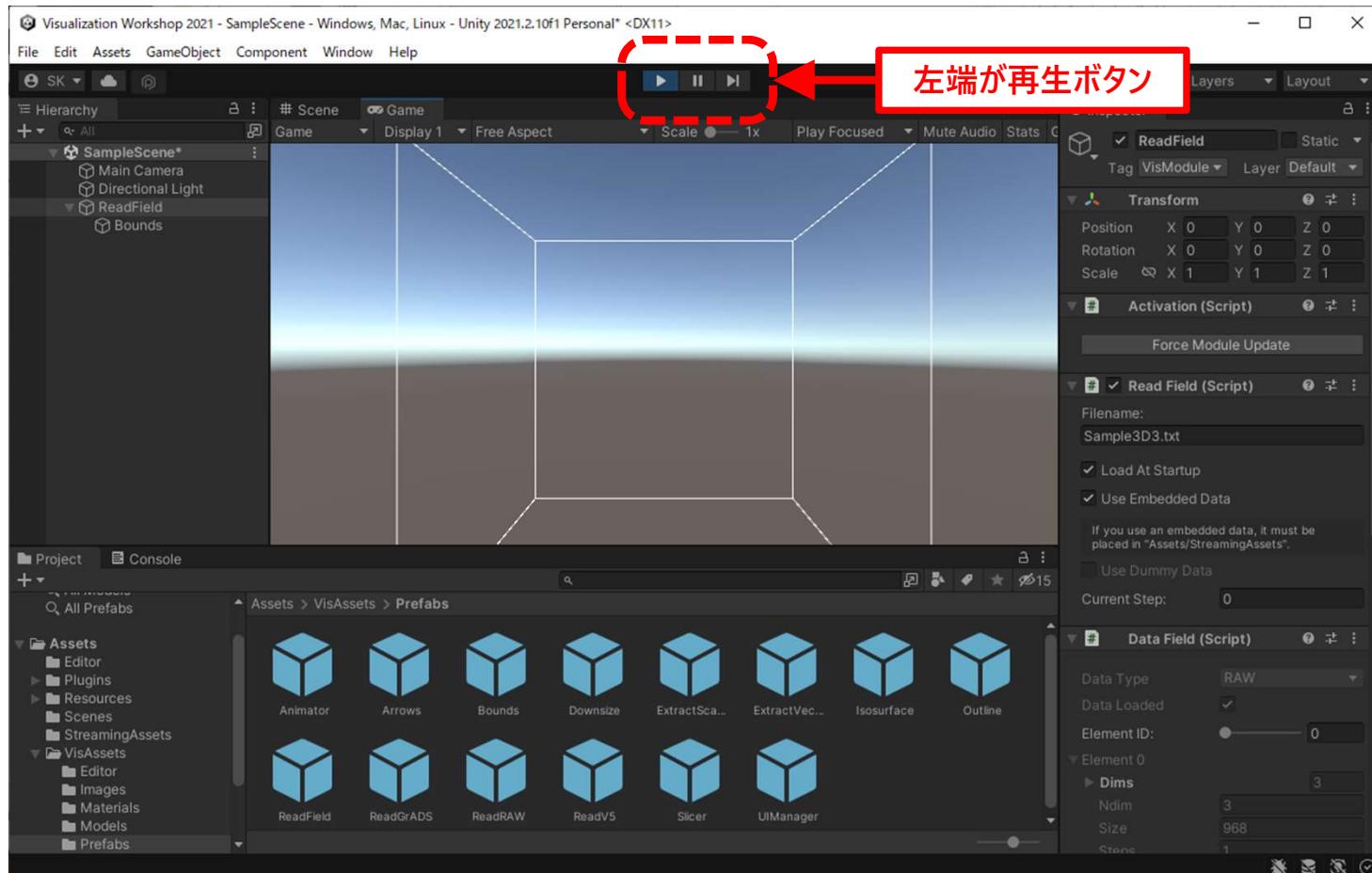
DataField のインスペクタで読み込まれたデータを確認することができます。

Element IDのスライダーを左右に動かすと各要素の詳細が表示されます。



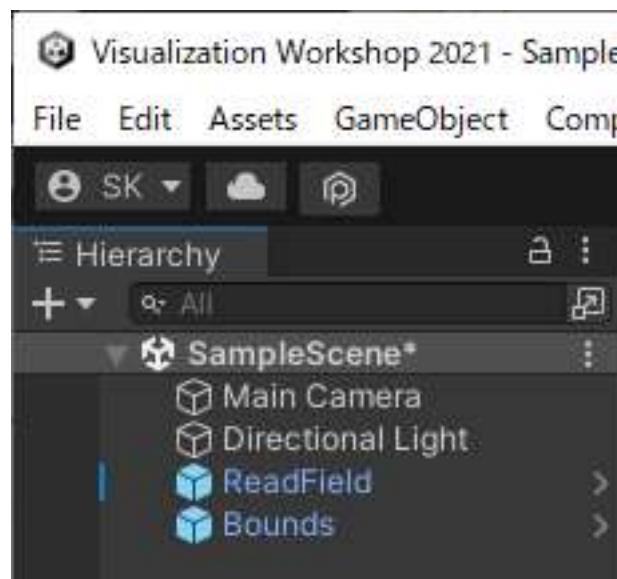
# ここまで状態の確認(4)

再生ボタンをもう一度押すとゲームビューでの実行が終了します(実行中の回転・拡大縮小の操作は保存されず、実行前の状態に戻ります)。

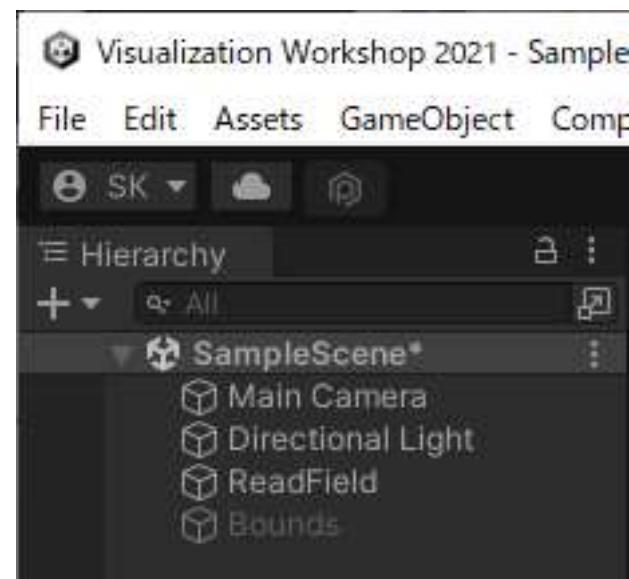
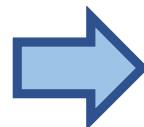


# 接続を間違えた場合

下図の例では、BoundsがReadFieldの下に接続されておらず、境界線の描画に必要なデータを受け取ることができないため無効な接続となります。VisAssetsでは、無効な接続と判定されたモジュールは非アクティブな状態となり、アプリケーションの動作には影響を与えません。



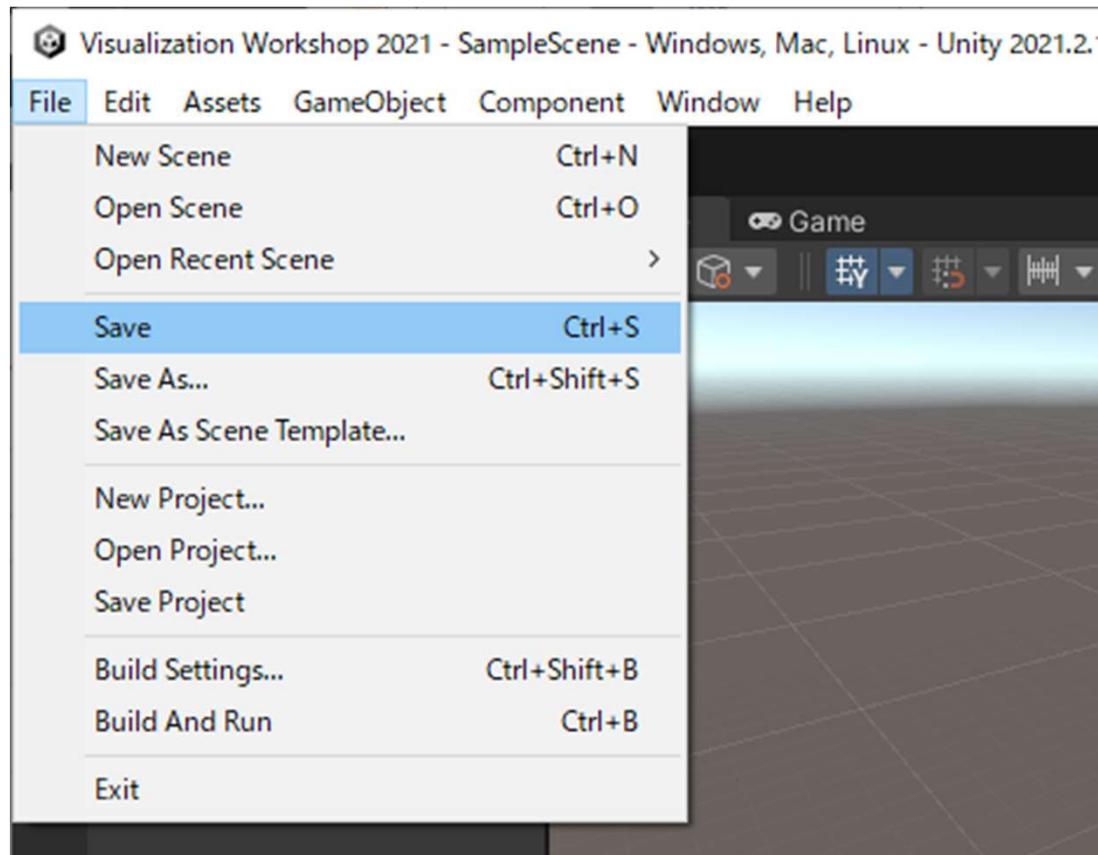
実行前



実行中  
(Boundsが非アクティブな状態に)

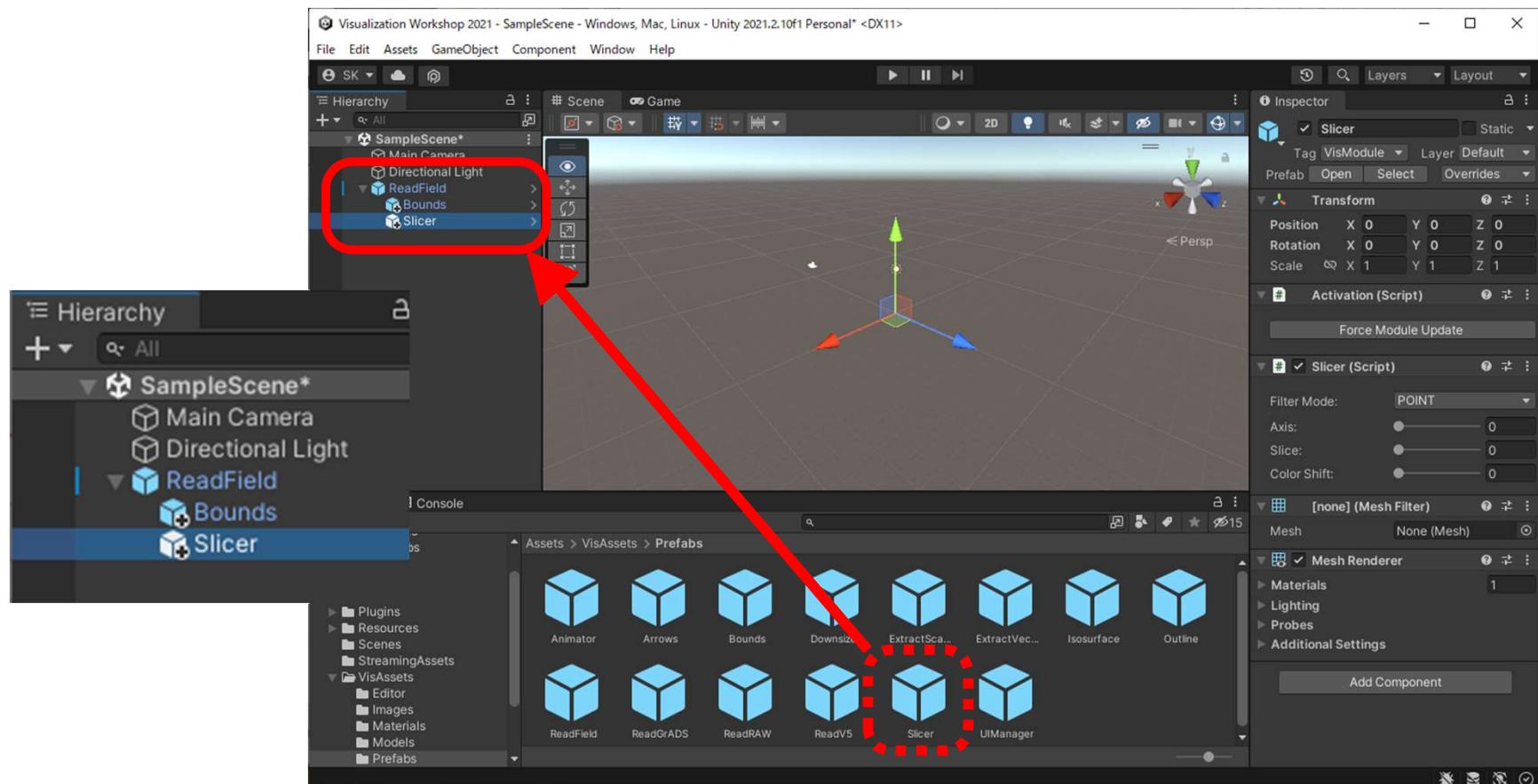
# シーンの保存

再生を止めた状態で、メニューの「File」-「Save」（または「Save As」）で現在のシーンを保存します。以下、進捗確認後には適宜シーンを保存することをおすすめします。



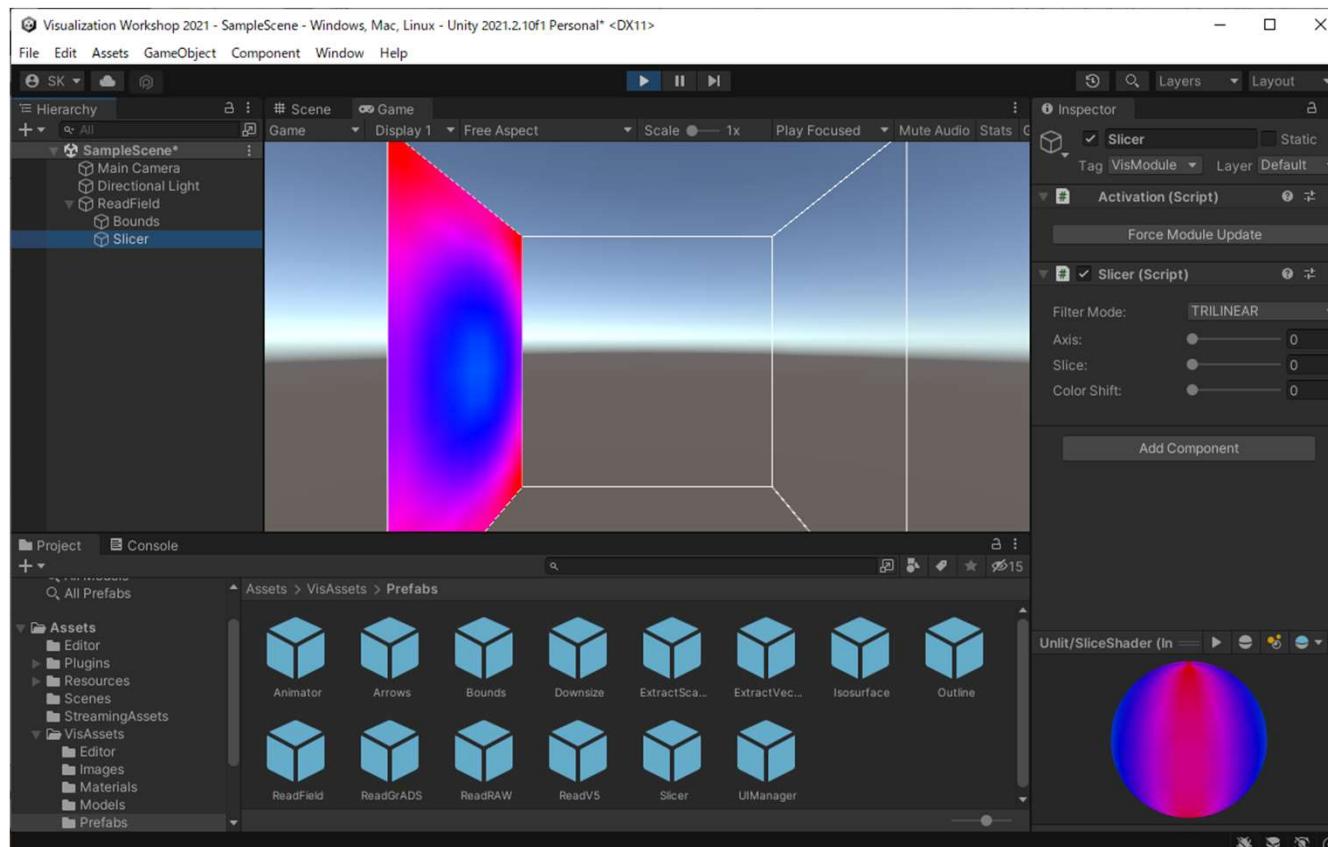
# カラースライスを表示する(1)

プロジェクトウィンドウから、「Slicer」モジュールをヒエラルキーイングの「ReadField」モジュールの子となるようドラッグアンドドロップして接続します。



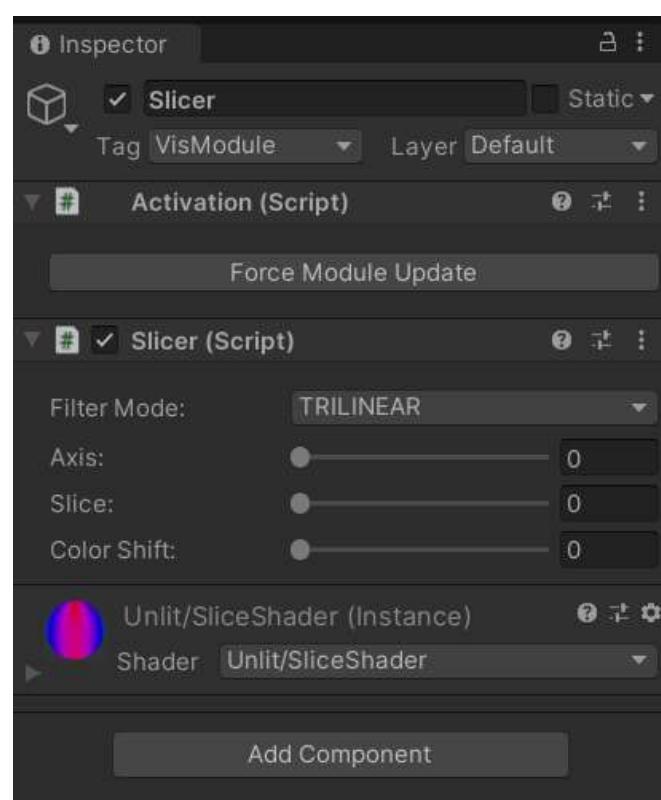
# カラースライスを表示する(2)

ツールバーの再生ボタンを押すと、ゲームビューにBoundsによるデータ領域に加えてカラースライスが表示されます。間違えてBoundsの下にSlicerを接続してしまうとカラースライスは表示されませんので注意してください。

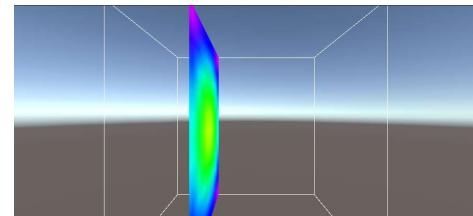


# カラースライスを表示する(3)

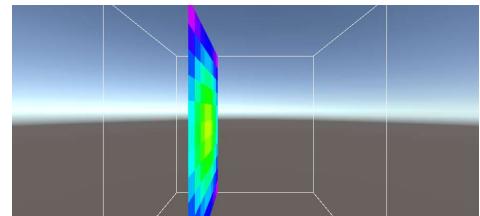
実行中、ヒエラルキーインドウで「Slicer」を選択し、インスペクタインドウ内のオプションを変更することで、軸、スライスレベル(0~1の範囲)等を設定できます。



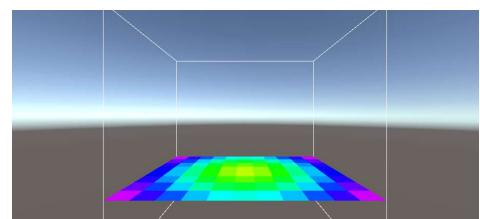
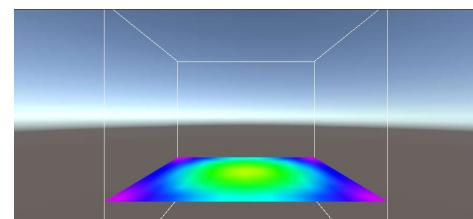
Filter Mode = TRILINEAR



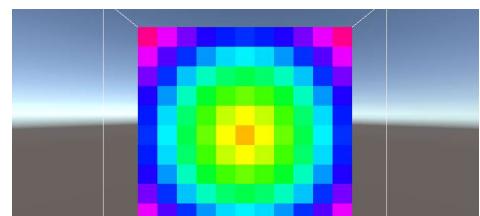
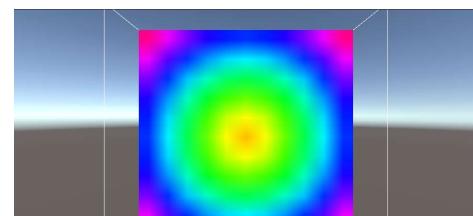
Filter Mode = POINT



Axis = 0, Slice = 0.3



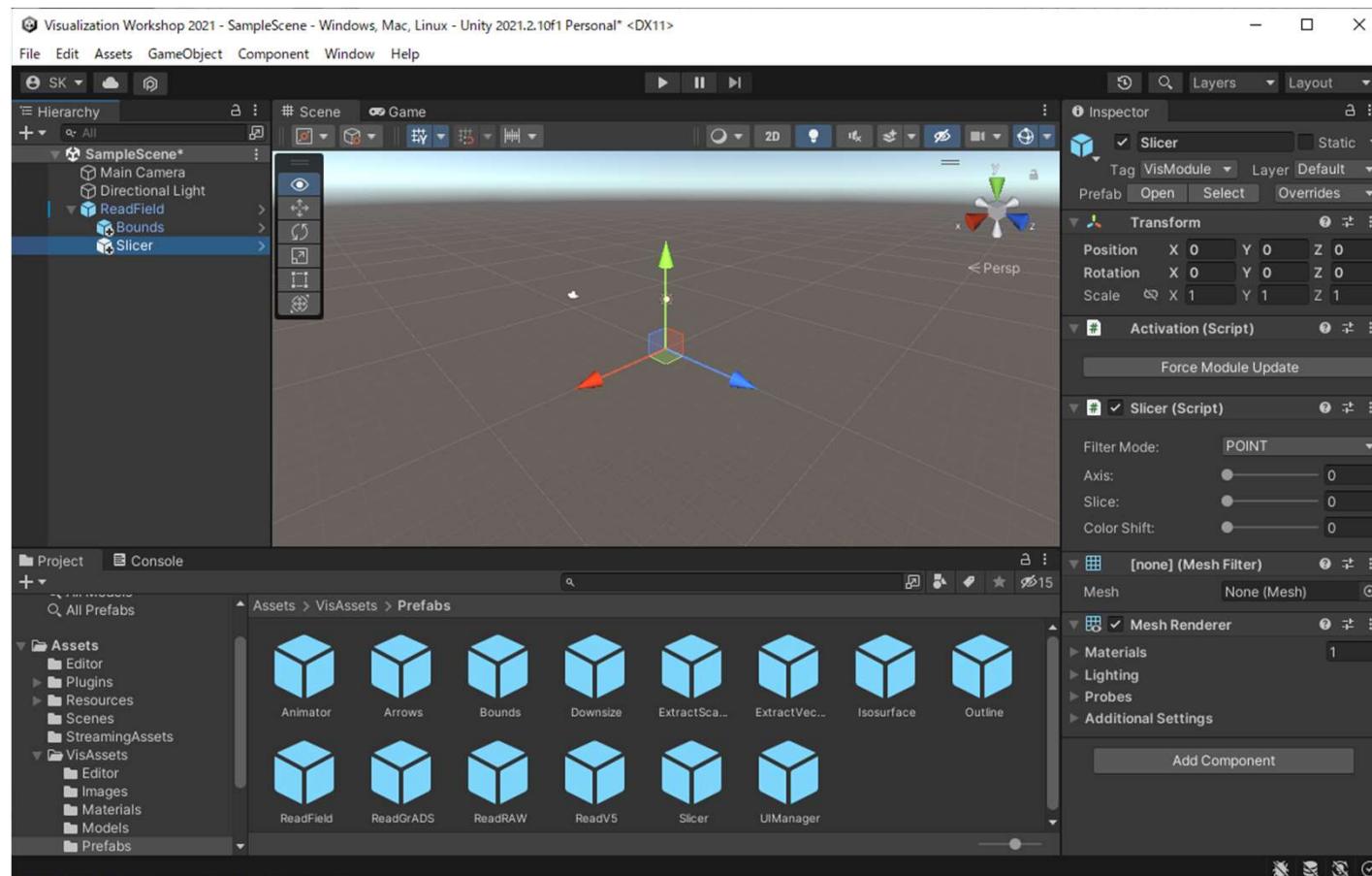
Axis = 1, Slice = 0.3



Axis = 2, Slice = 0.3

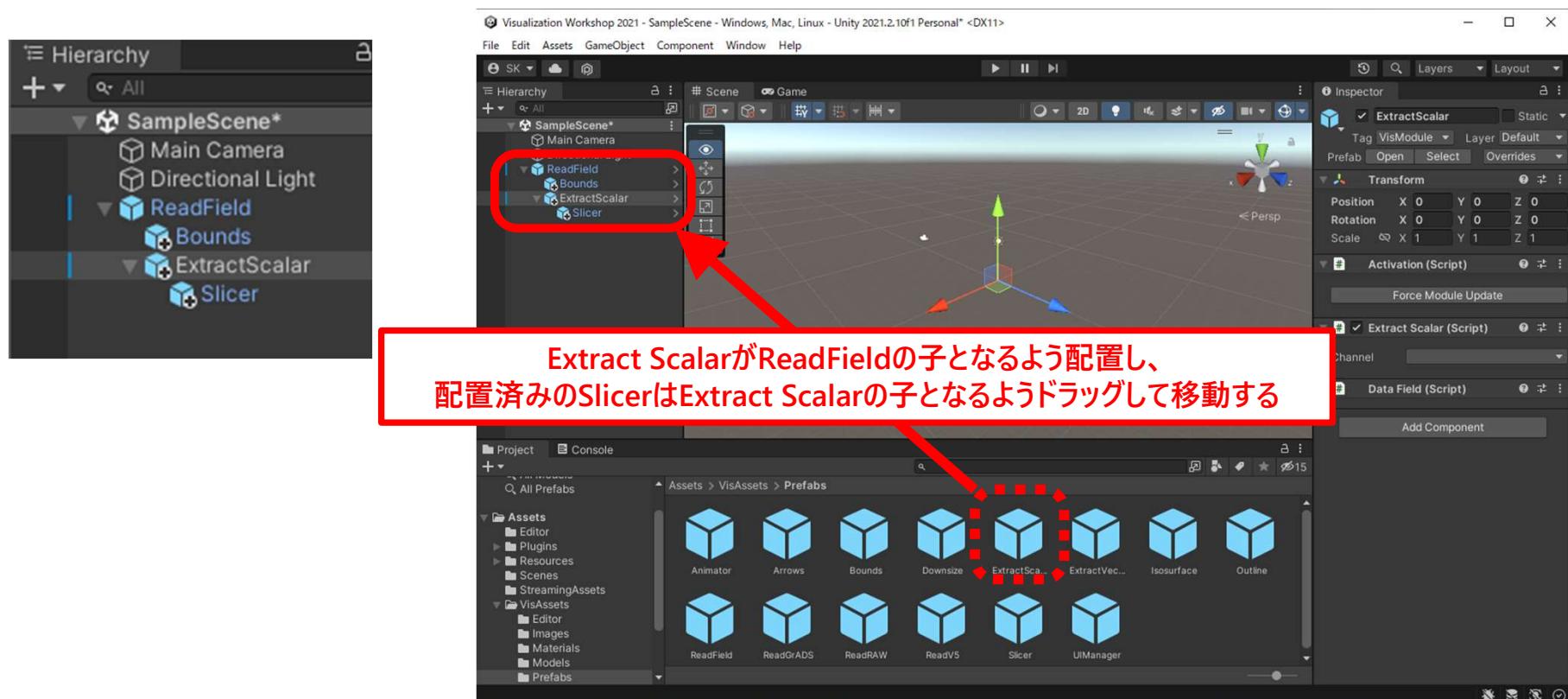
# カラースライスを表示する(4)

次の作業に移る前に、再生ボタンを押してプレビュー実行を終了します。



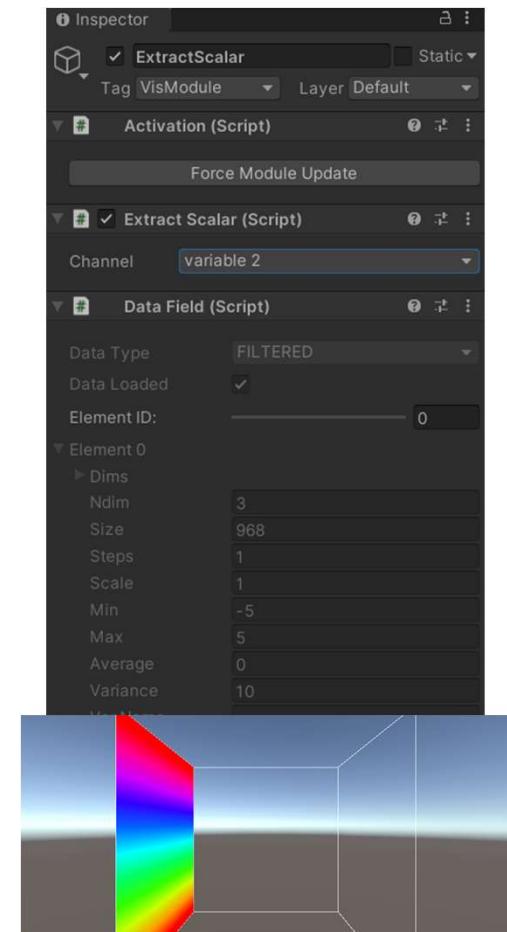
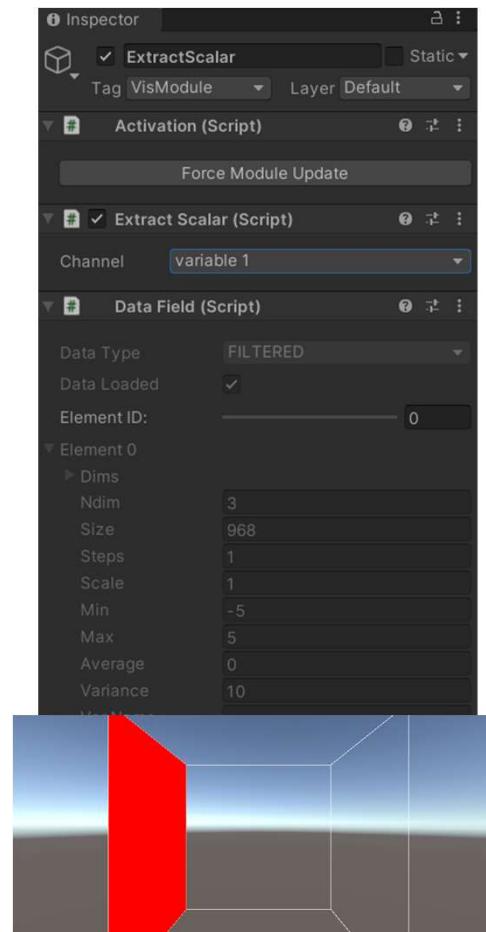
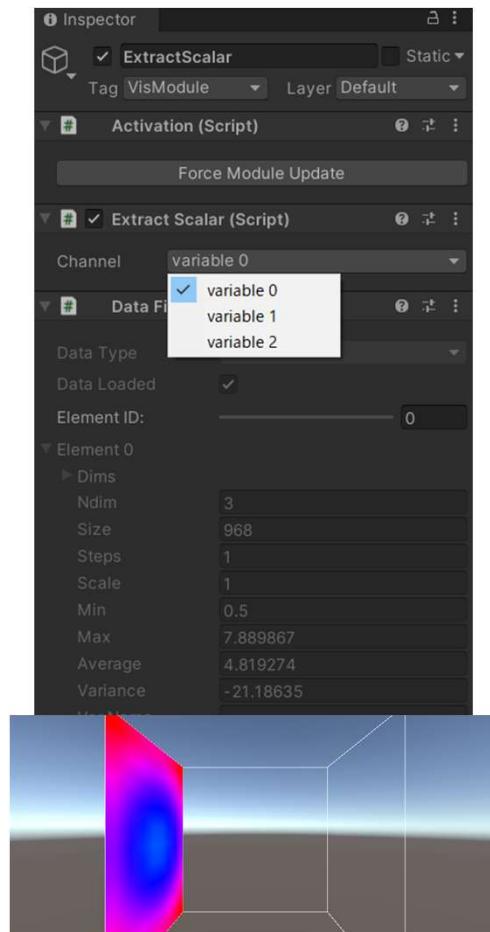
# 任意の要素のカラースライスを表示する(1)

ReadFieldの直下に接続したSlicerへの入力データは、入力データに含まれる要素の内、1番目に格納されている要素となります。入力データに複数の要素が含まれる場合、2番目以降の要素をSlicerへの入力したい場合には、「Extract Scalar」モジュールを挿入し、任意の要素を抽出できるようにする必要があります。



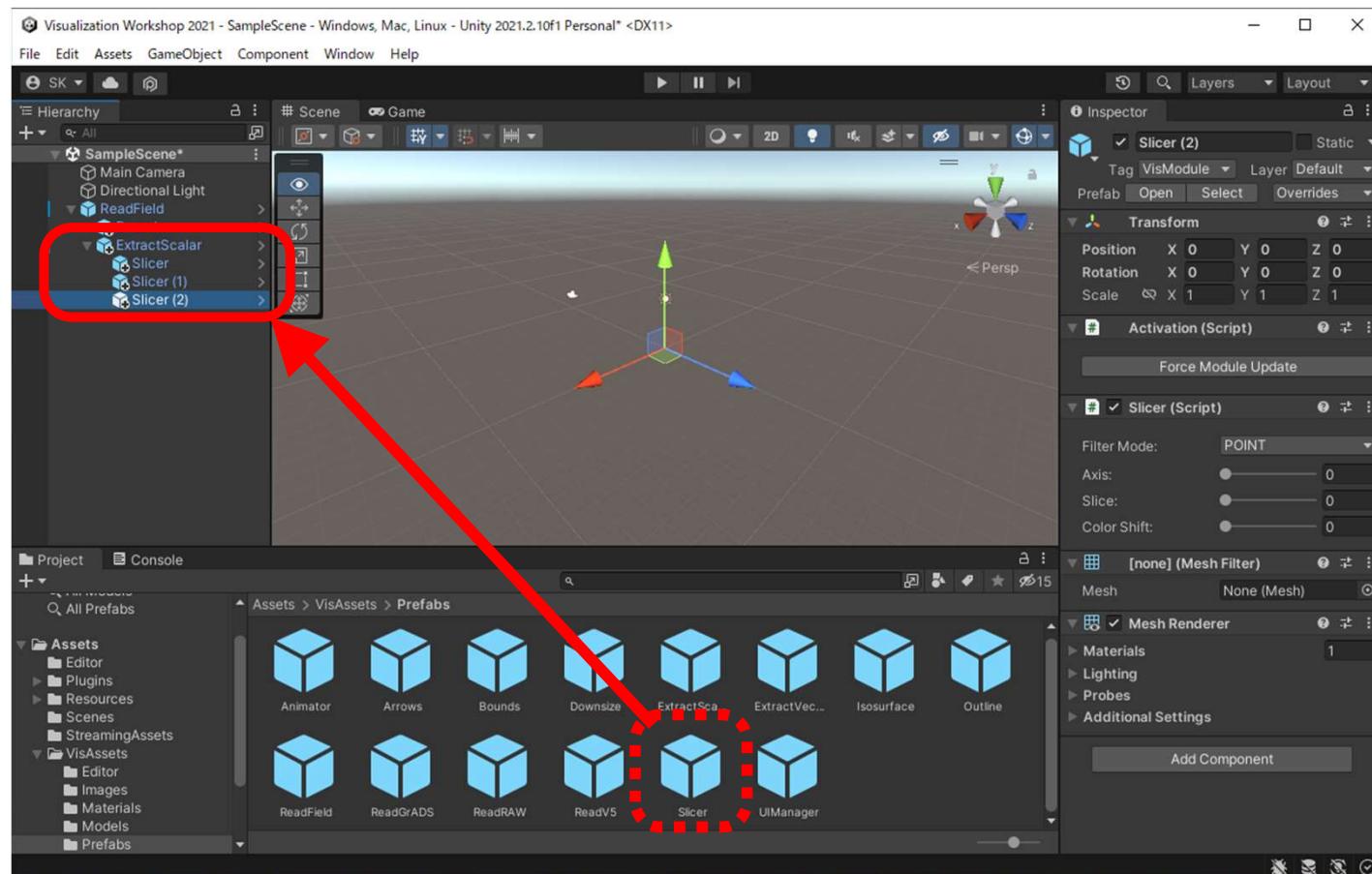
# 任意の要素のカラースライスを表示する(2)

実行中、「Extract Scalar」のインスペクタ設定で、入力データからSlicerに渡す要素(Channel)を一つ選択可能となります。サンプルデータには各要素の名称が設定されていません。この場合、Extract Scalarは仮の要素名「variable 要素番号」を自動的に付与します。



# 複数のカラースライスを表示する(1)

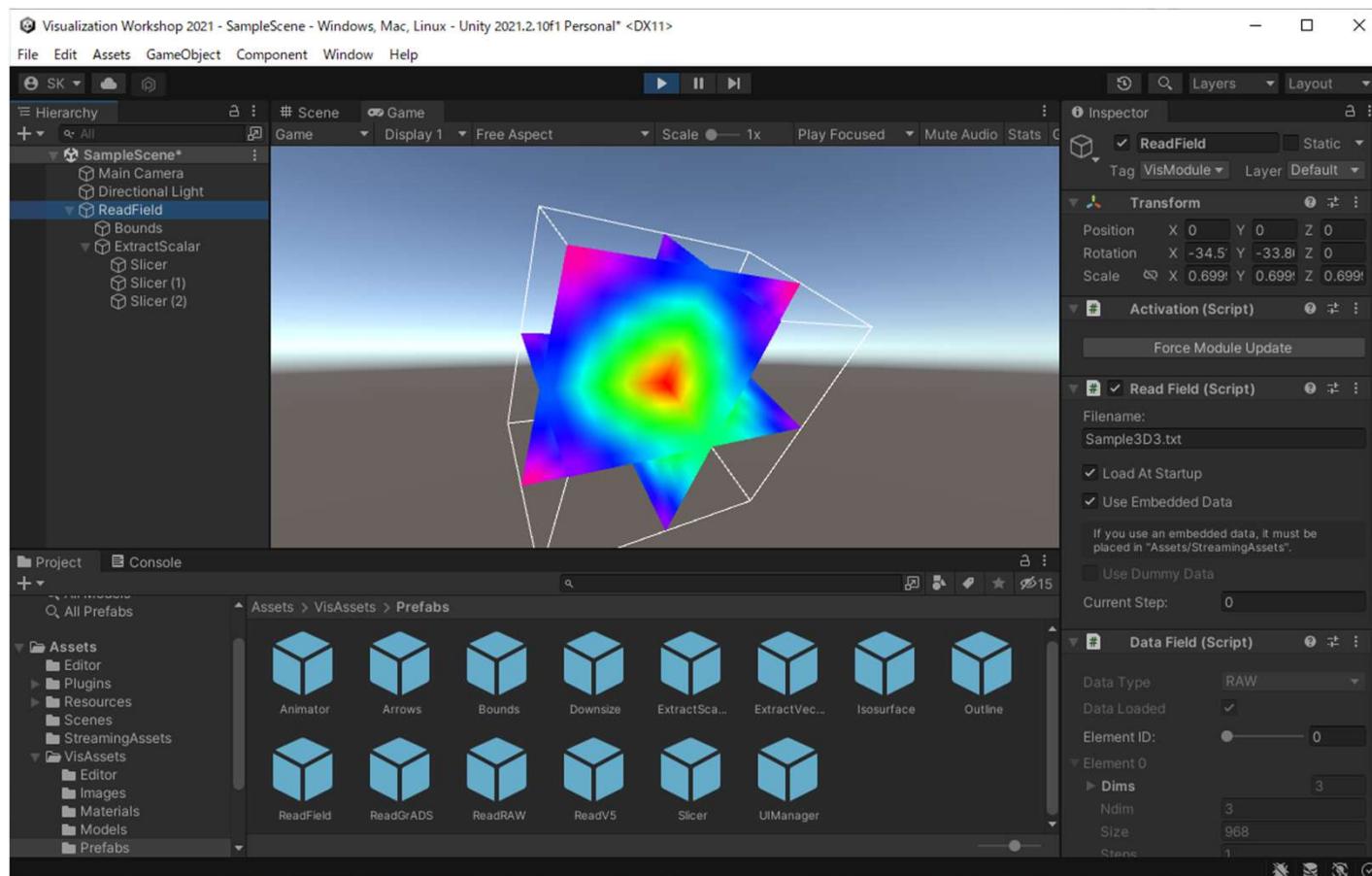
実行を停止し、「Extract Scalar」の子として「Slicer」を二つ追加します（同一階層に同名モジュールが配置された場合、Unityエディタがモジュール名の末尾に識別子を自動で付与します）。



# 複数のカラースライスを表示する(2)

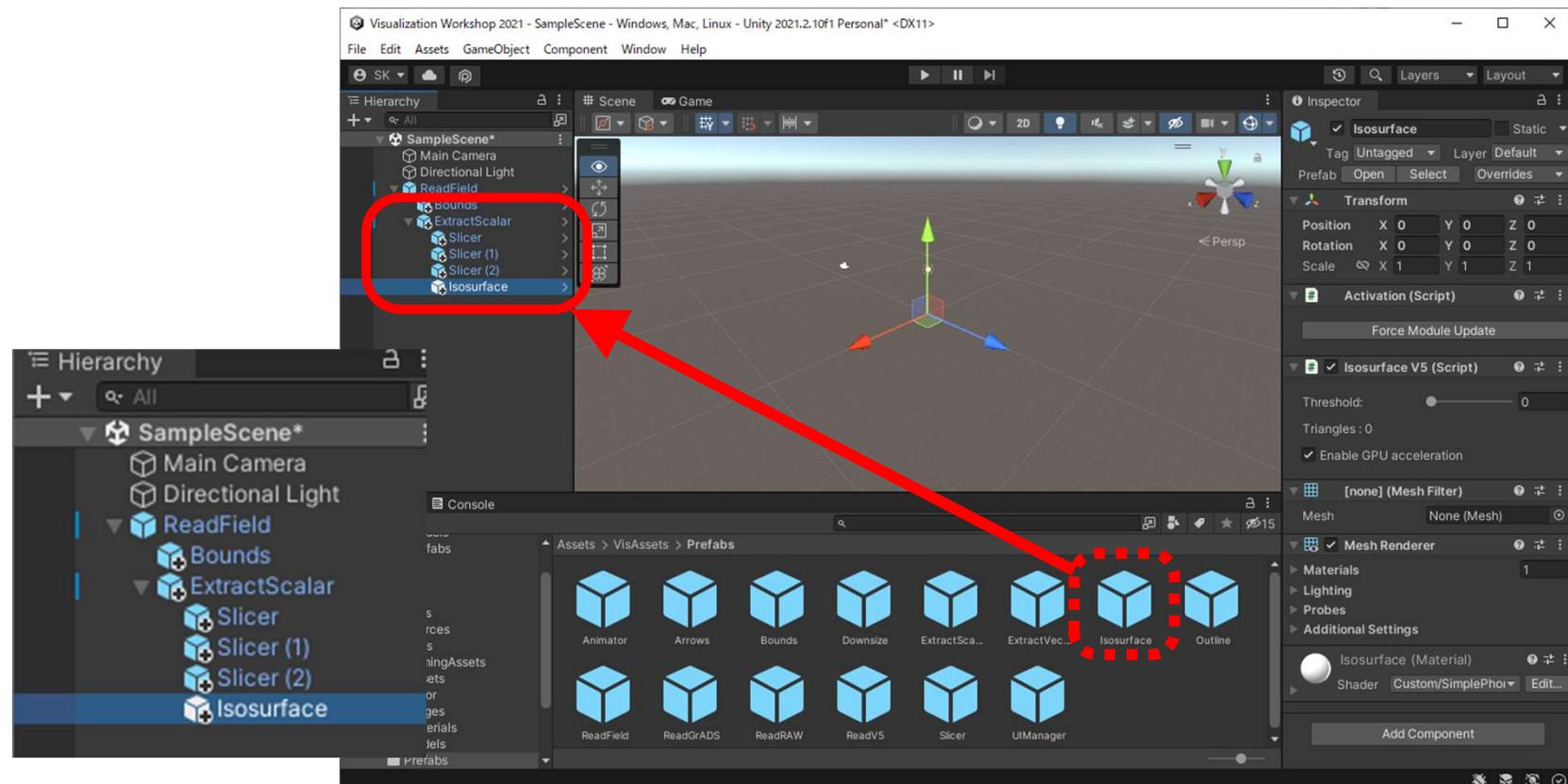
再生ボタンで実行し、各「Slicer」モジュールの設定をインスペクタから

Slicer:Axis=0、Slicer (1):Axis=1、Slicer (2):Axis=2、いずれもSlice=0.5  
とすると下図のような結果が得られます。



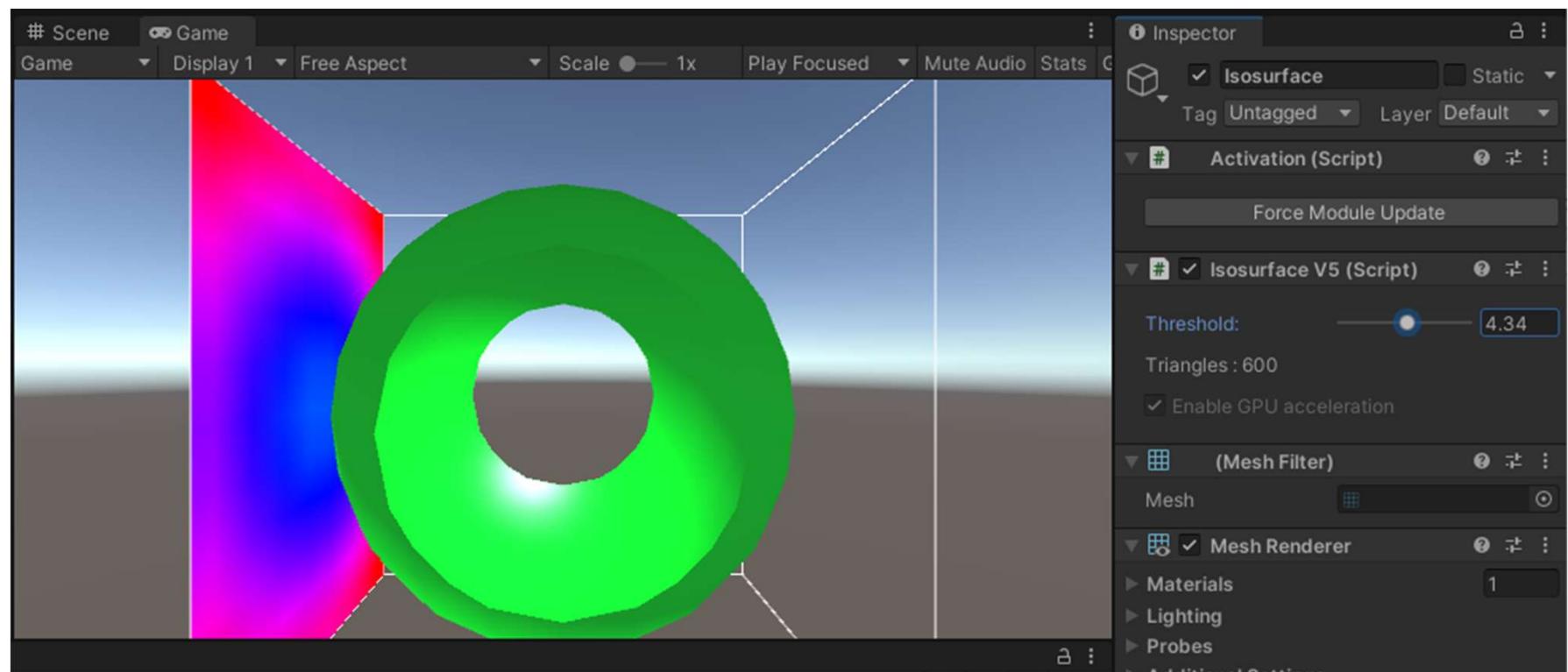
# 等値面を表示する(1)

実行を停止し、「Extract Scalar」の子として「Isosurface」を追加します。「Slicer」とは別の要素の等値面を表示したい場合は、「ReadField」の下に「Extract Scalar」を追加し、その下に「Isosurface」を接続します。



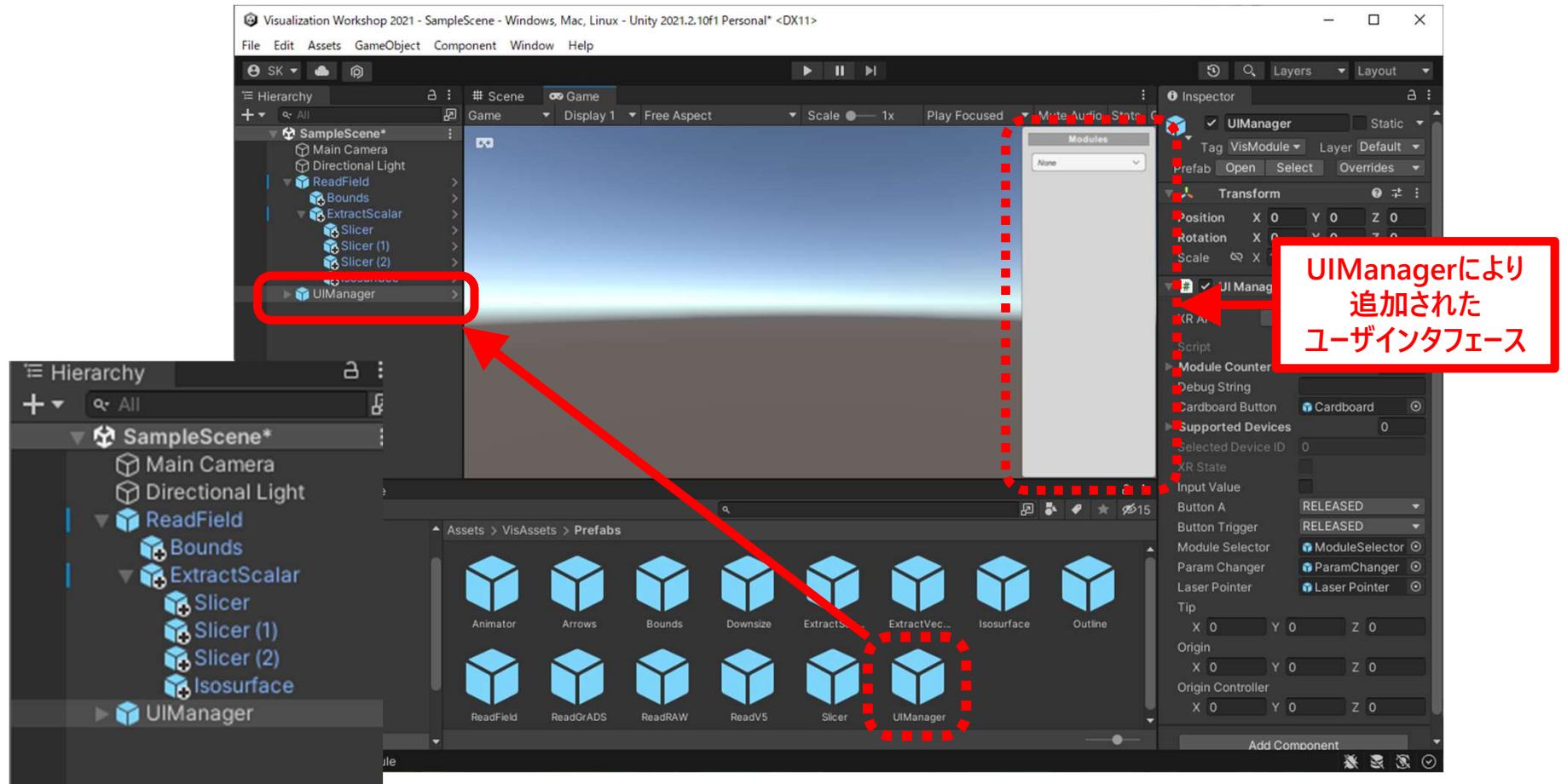
# 等値面を表示する(2)

実行時に、「Isosurface」モジュールのインスペクタから閾値(Threshold)を変更することで等値面が生成されます。デフォルトではGPUを使用する設定になっていますが、使用するPCのスペックによっては正常に動作しない可能性がありますので、その場合は「Enable GPU acceleration」をOFFにしてください。



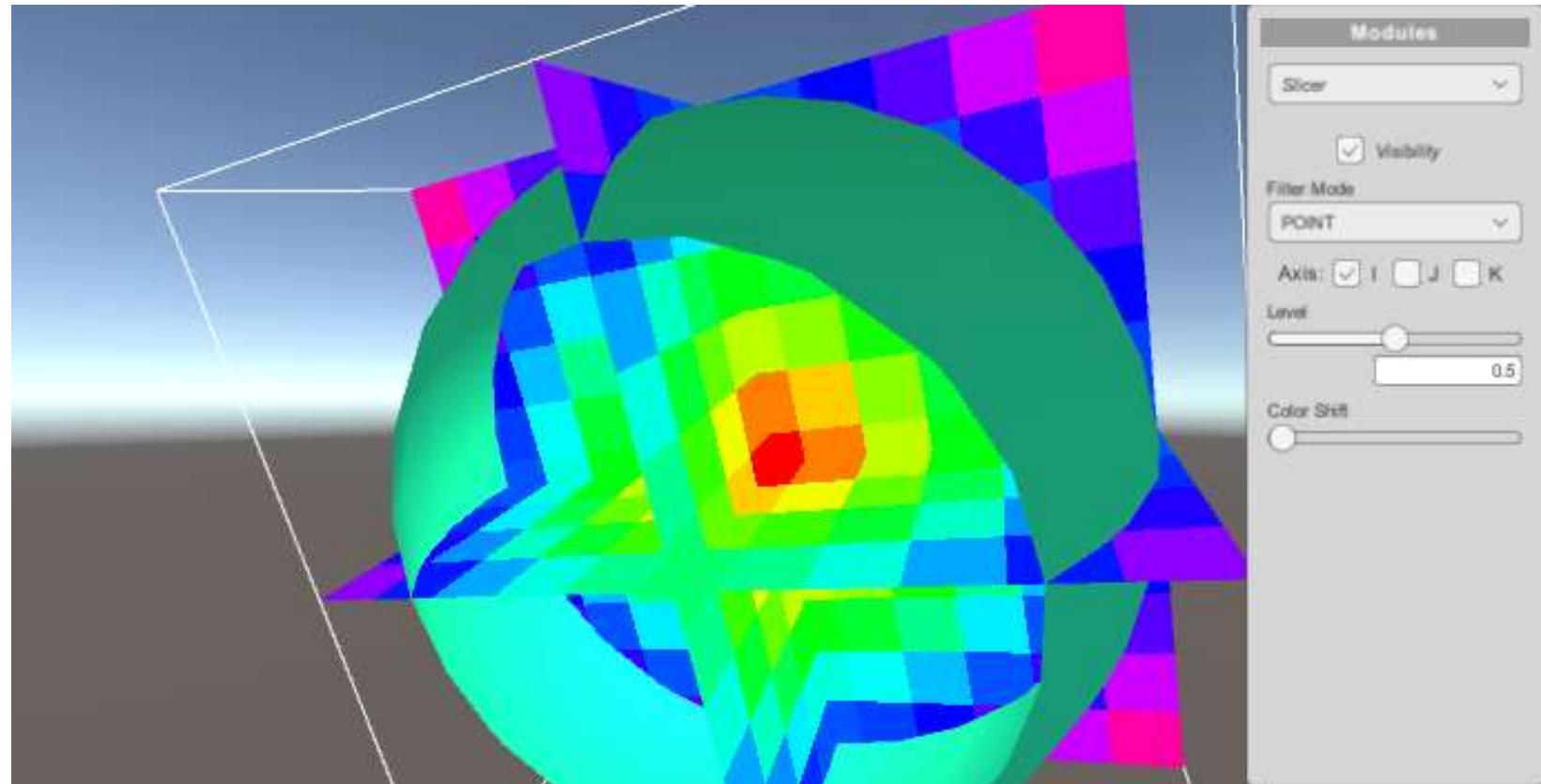
# ユーザインターフェースの追加(1)

実行を停止し、「UIManager」モジュールをヒエラルキーに「Main Camera」と同じ階層にドラッグアンドドロップして追加することにより、ゲームビューの右端にUIが追加されます。



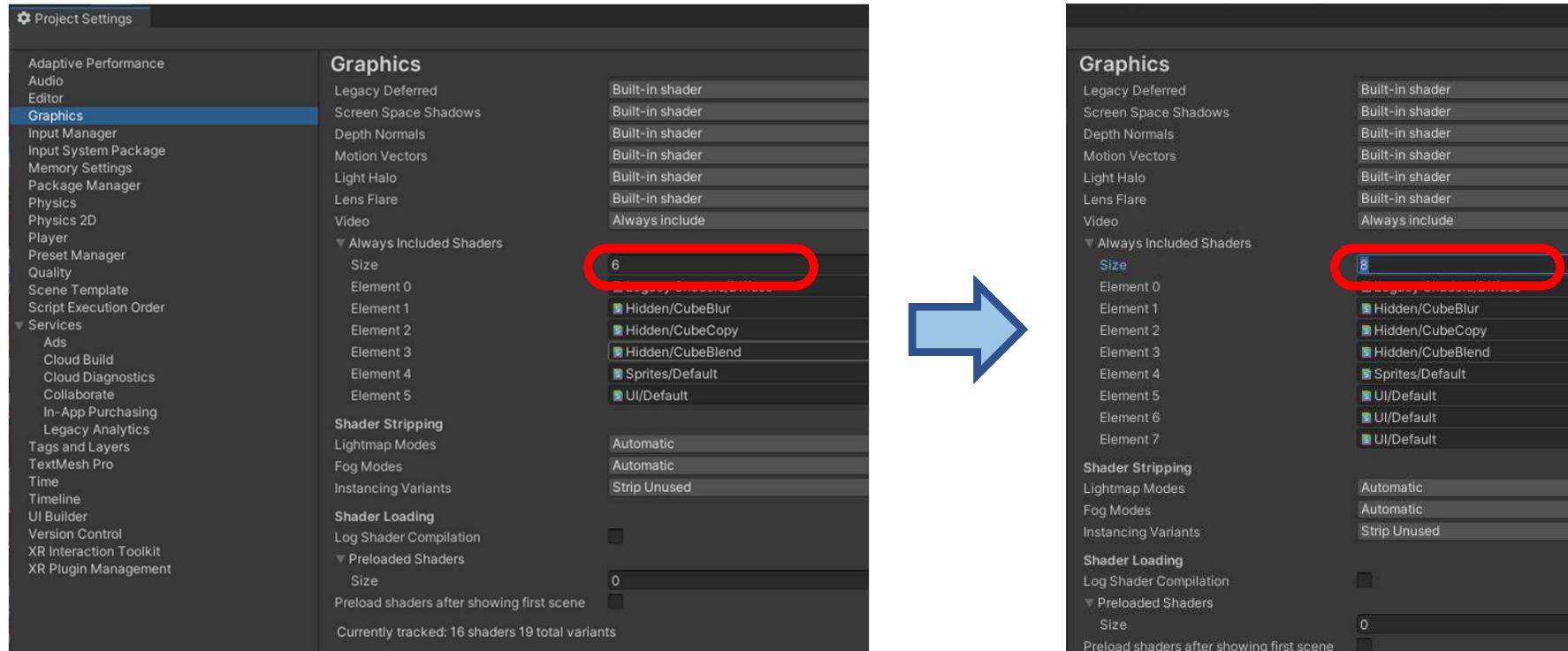
# ユーザインターフェースの追加(2)

再生ボタンで実行すると、ゲームビュー内に表示されたUIから、操作対象となるモジュールの選択と、選択したモジュールのパラメータの変更ができるようになります。インスペクタからのパラメータ変更はUnityエディタでの実行時のみ可能で、ビルド後にはできないため、このようなUIが必要となります。



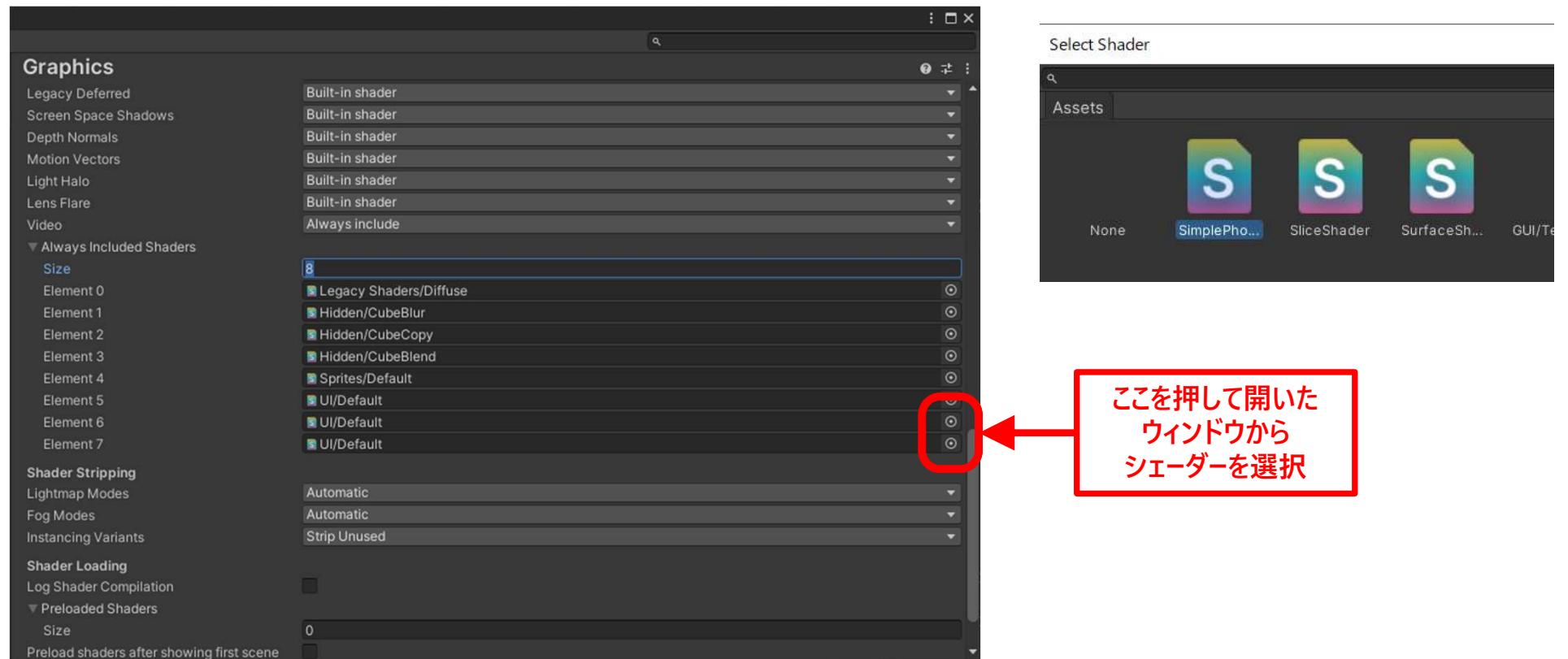
# ビルド前の準備(1) ~組み込みシェーダーの登録~

メインメニューから [Edit] – [Project Settings] で Project Settings ウィンドウを表示し、「Graphics」内の[Built-in Shader Settings]-[Always Included Shaders]のSizeを現在の値(下図では6)から2つ増やします(6→8に変更し、Enterキーで確定)。



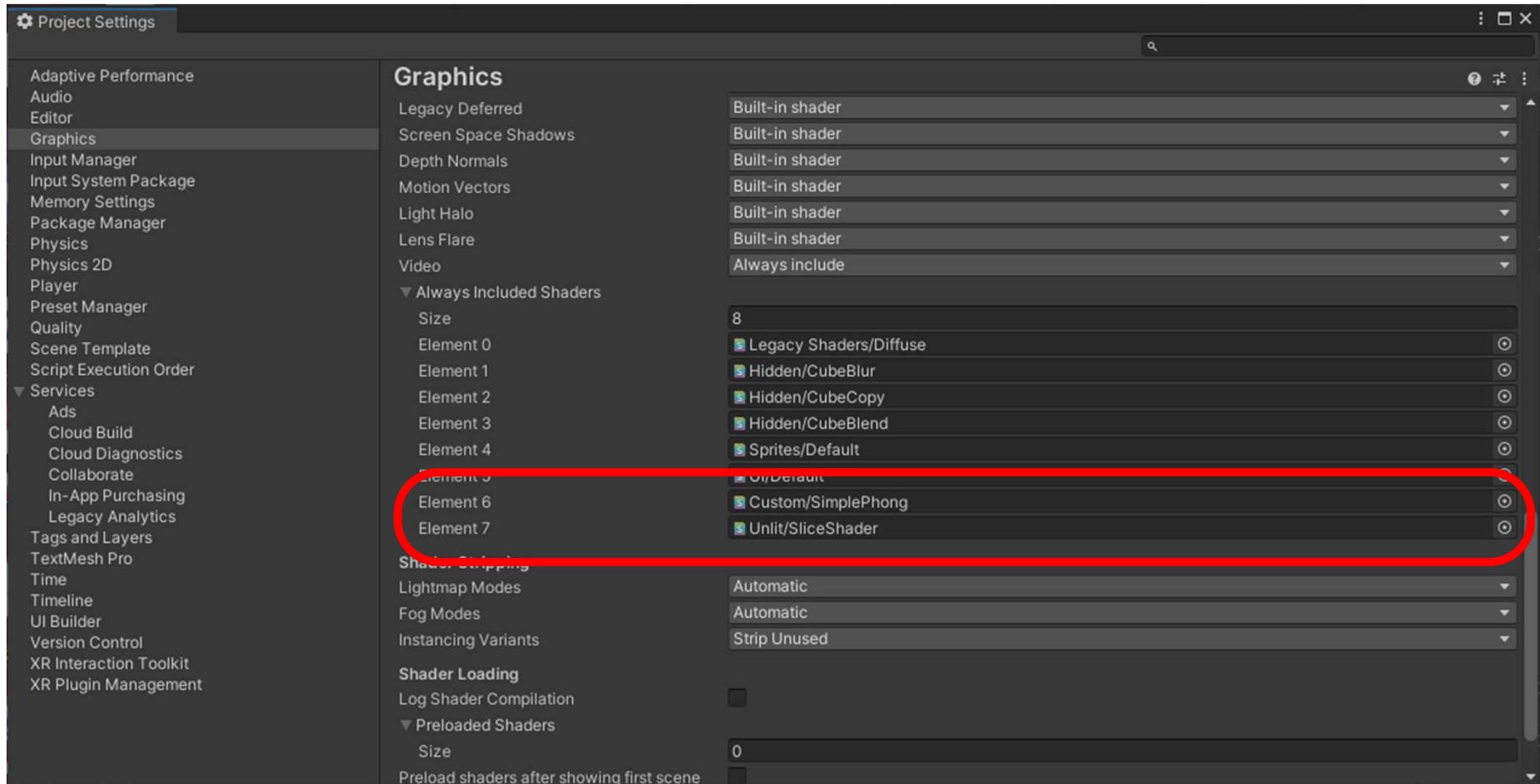
# ビルド前の準備(2) ~組み込みシェーダーの登録~

前のスライドの操作でシェーダーリストの末尾に追加された「Element 7」と「Element 8」に SimplePhongとSliceShaderを登録します(順不同)。



# ビルド前の準備(3) ~組み込みシェーダーの登録~

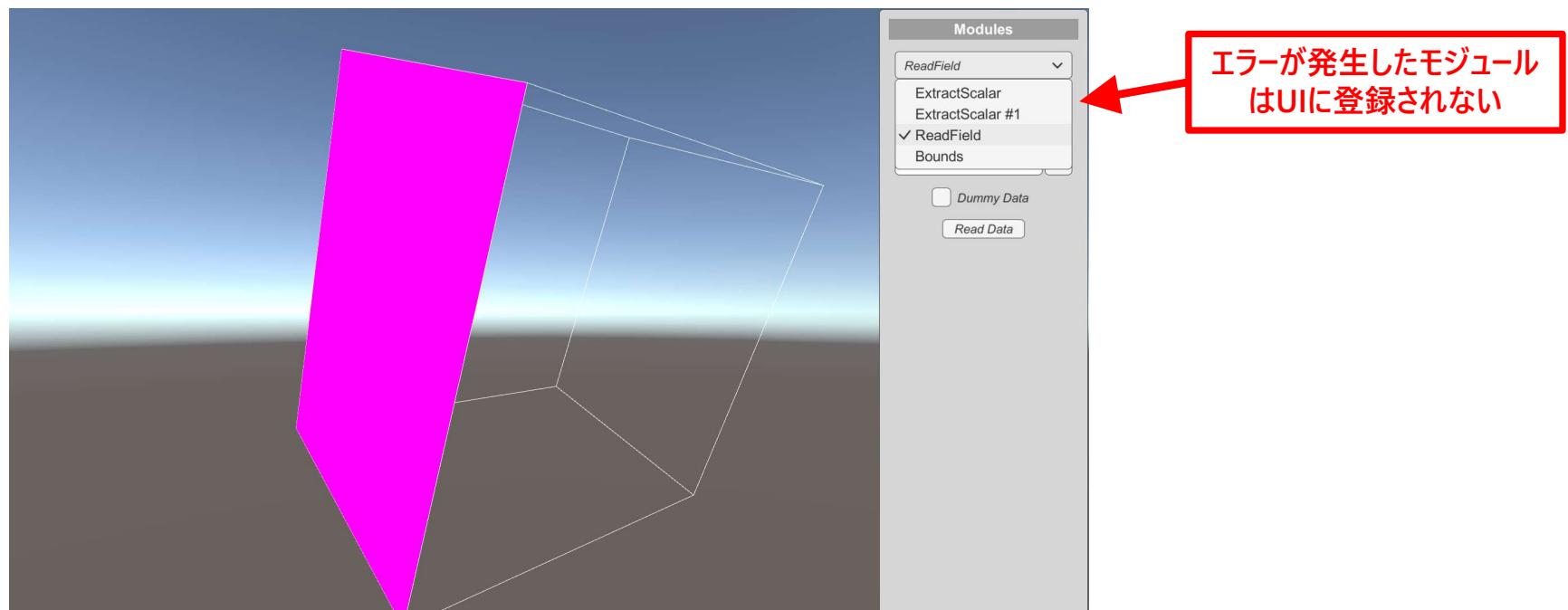
## 組み込みシェーダー登録後のGraphics設定画面



# シェーダーを登録せずにビルドした場合

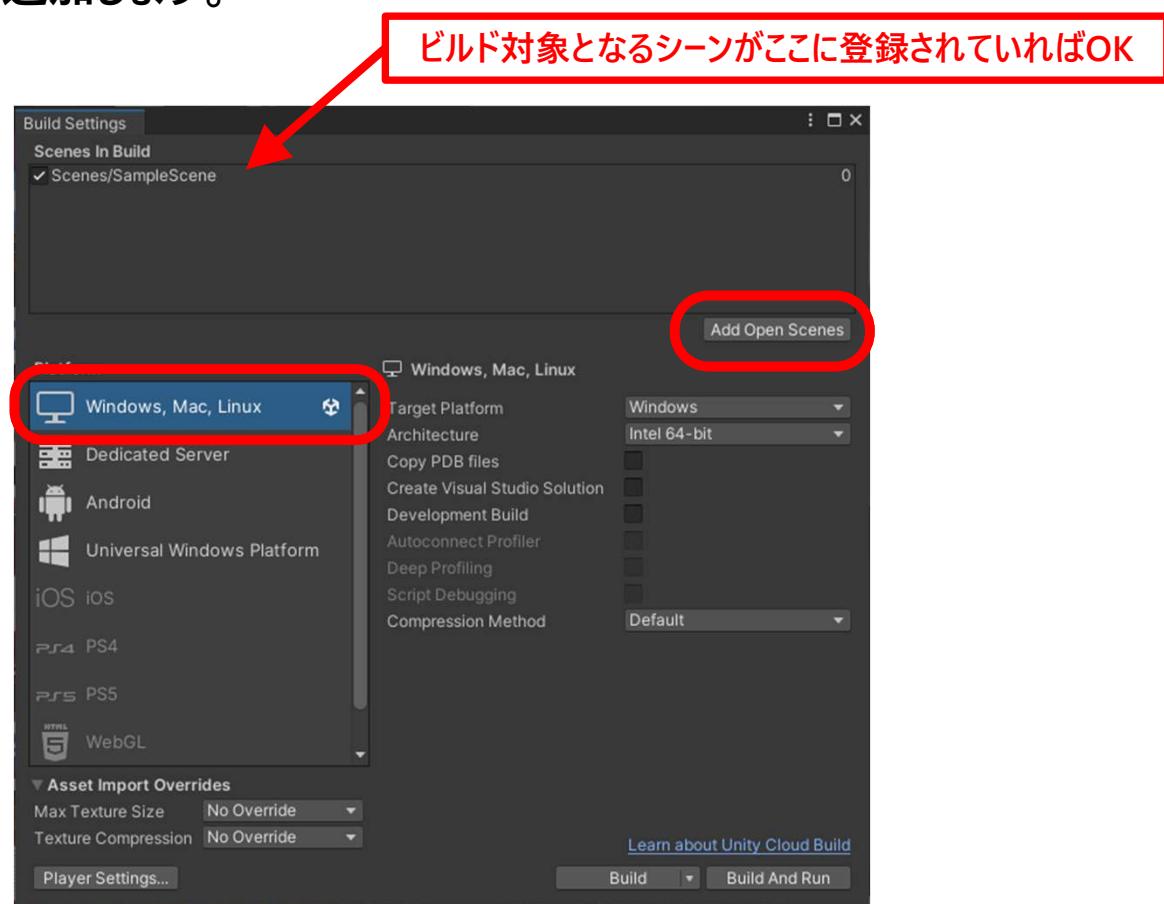
Unityエディタでのプレビュー実行時には正常に動いていたアプリケーションですが、この後のステップでビルドが正常に完了しても、ビルド後の実行バイナリにカスタムシェーダーが登録されていないため、Slicerが図のような表示になります。

シェーダが見つからず、Slicerモジュールの実行途中に内部エラーとなるため、UIのモジュール一覧からSlicerが選択できない状態になります(Isosurfaceも同様)。



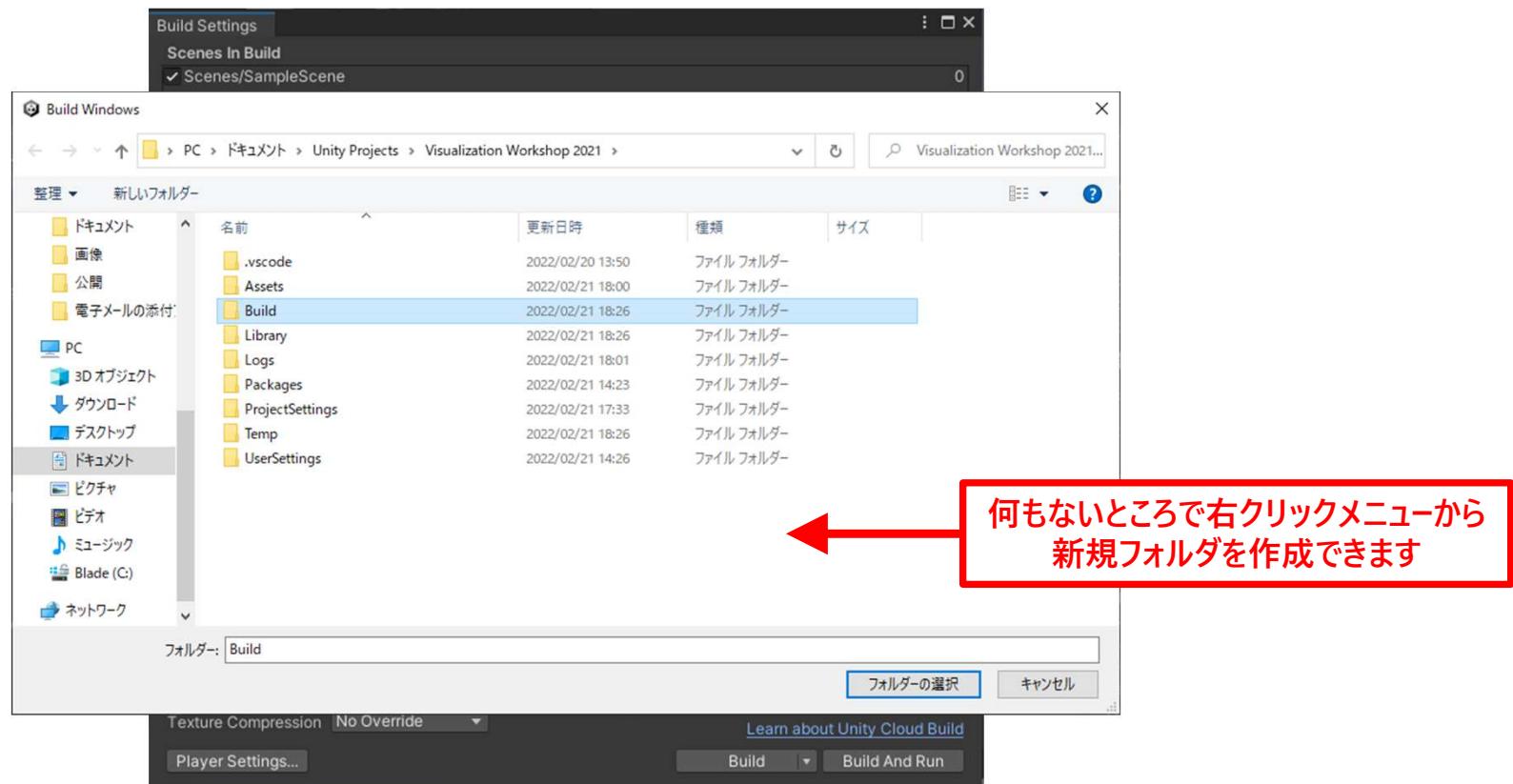
# Windows用実行ファイル作成(1)

メインメニューから[File]–[Build Settings]でビルド設定ウィンドウを開き、Platformとして「PC, Mac & Linux」が選択されていることを確認し、「Add Open Scenes」ボタンで現在開いているシーンをビルド対象として追加します。



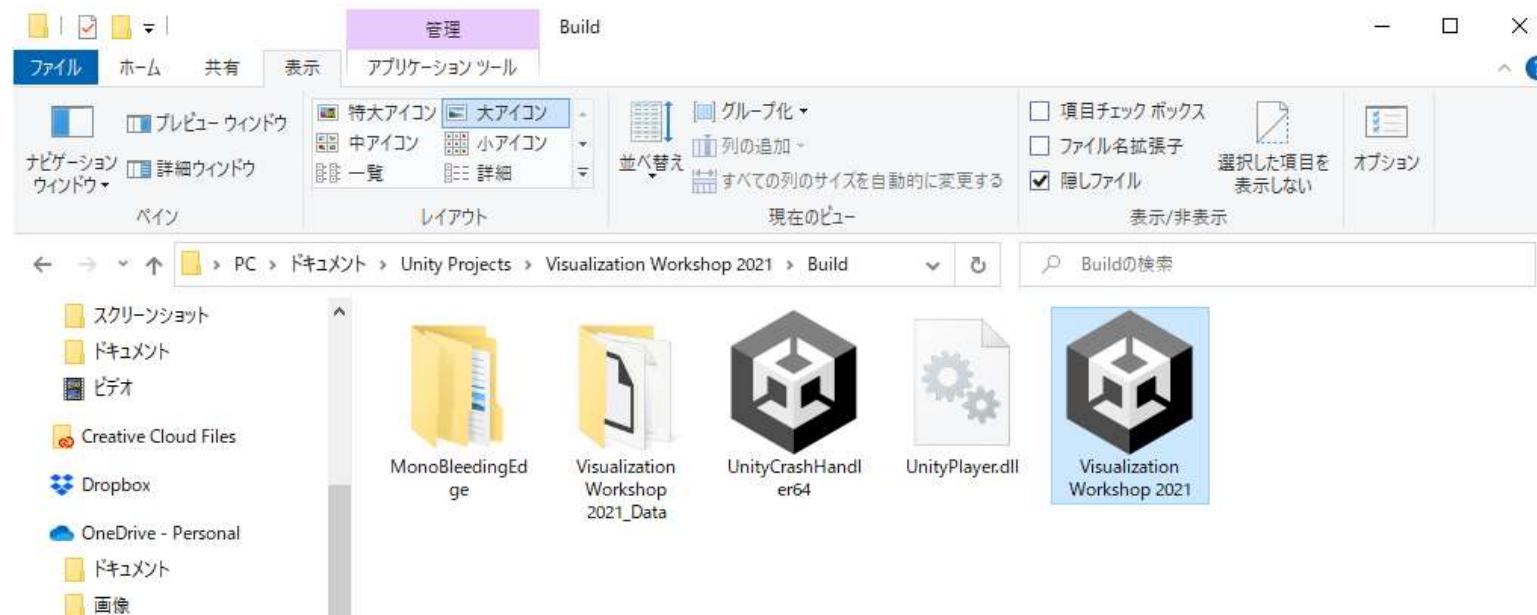
# Windows用実行ファイル作成 (2)

「Build」ボタンを押してファイルの出力先を指定します。ビルドの際には複数のファイルやディレクトリが作成されるため、ビルドファイルを格納するための専用のディレクトリを用意するとよいでしょう。下の図ではプロジェクトディレクトリ内に「Build」という名前のフォルダを新規に作成し、出力先として指定しています。



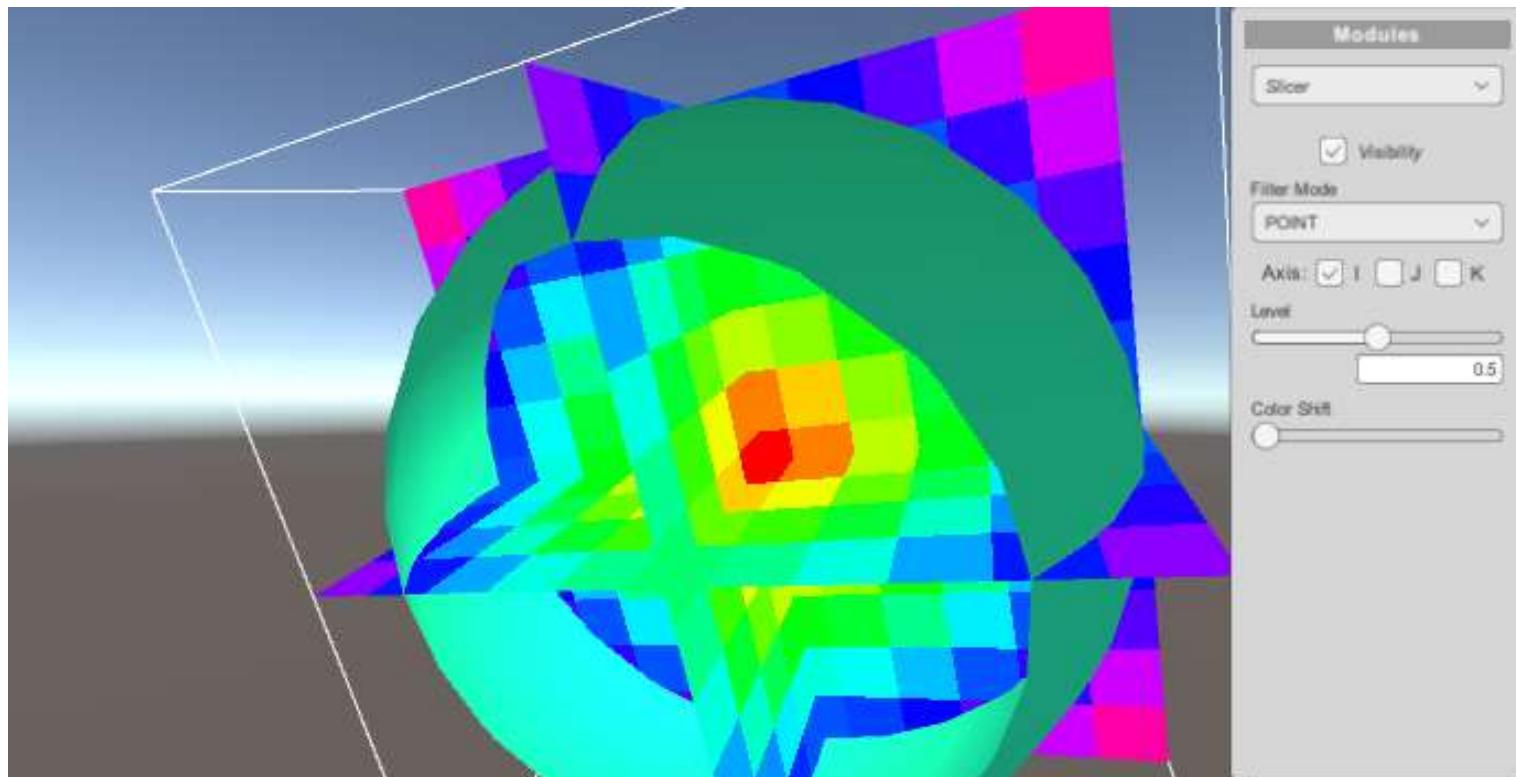
# Windows用実行ファイル作成(3)

ビルドが完了するとビルドフォルダ以下に実行ファイルが生成されます。プロジェクトフォルダはUnityエディタのプロジェクトウィンドウでの右クリックメニューから「Show in Explorer」を選択することでも開くことができます。実行ファイルは下の図でハイライトされている右端のファイルです。実行ファイル名はプロジェクト名と同じものとなります。



# Windows用実行ファイル作成 (4)

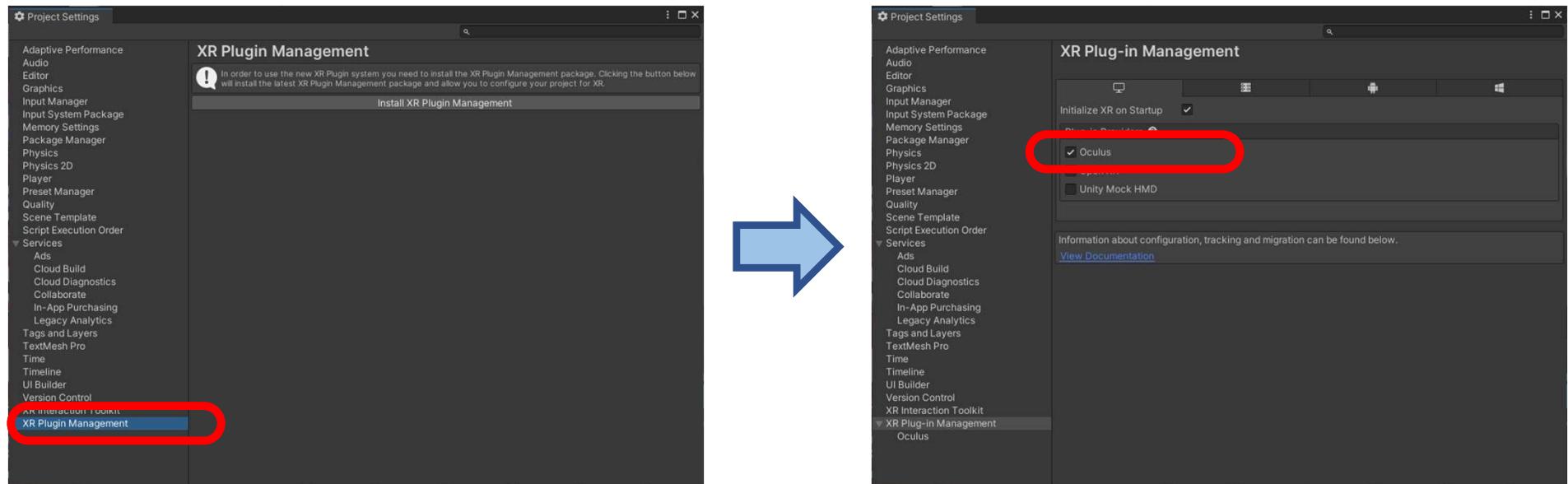
生成された実行ファイルをダブルクリックすると、構築したアプリケーションが全画面表示で実行されます。UIからのモジュール選択やパラメータ設定ができるることを確認してください。アプリケーションの終了は Alt + F4 キーです。



参考

# PC-HMDでの立体視表示(1)

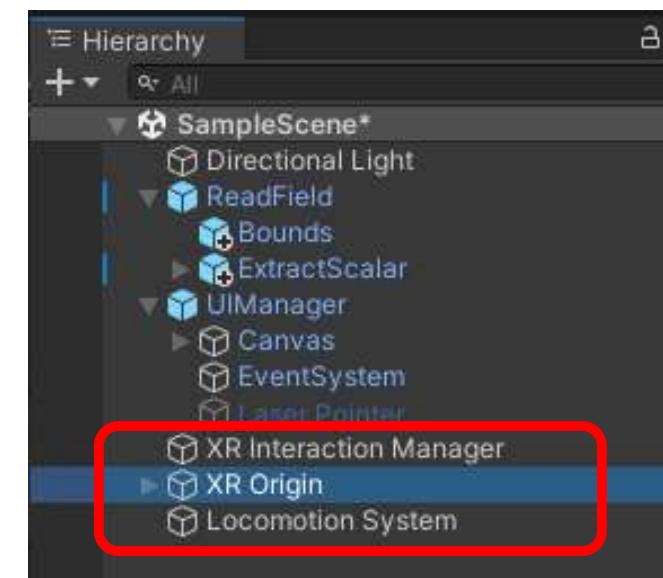
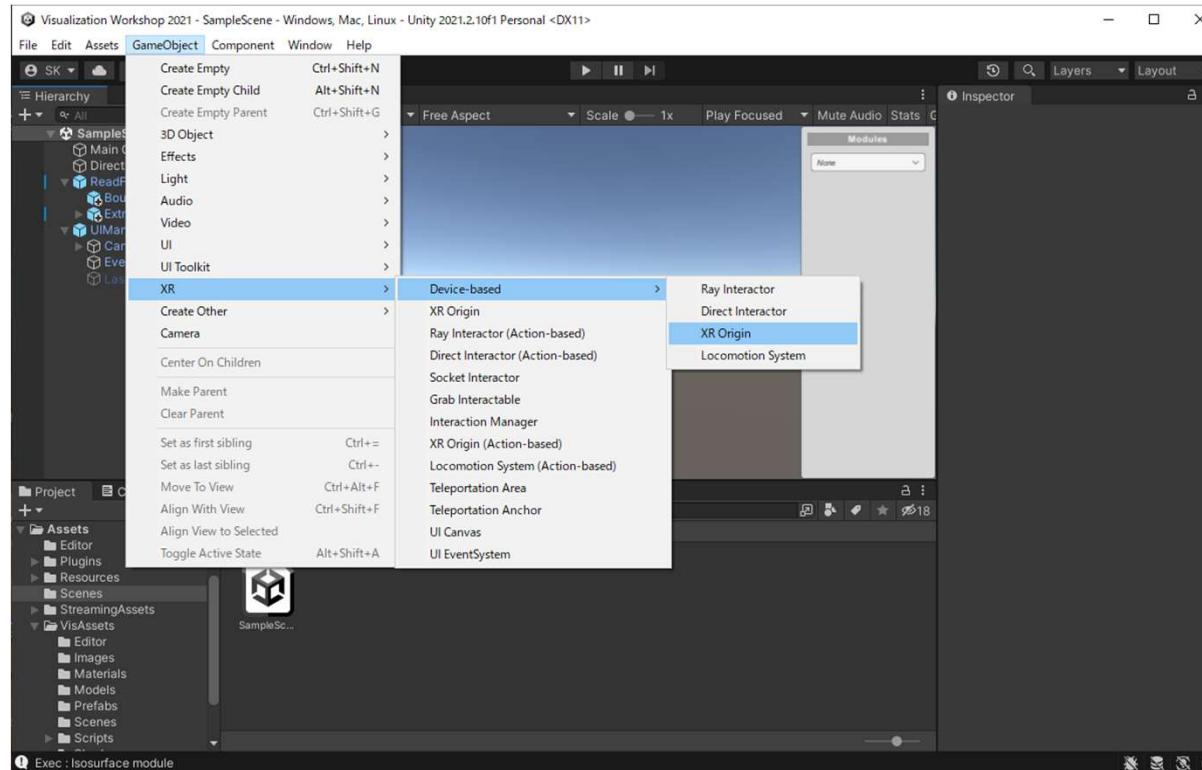
メインメニューから[Edit]–[Project Settings]でProject Settingsウィンドウを表示し、カテゴリの最下部にある[XR Plugin Management]を選択します。[Install XR Plugin Management]ボタンを押してXR Plugin Managementをインストールし、有効にするHMDプラグインを選択します。ここではOculusプラグインを有効にしており、以下の例ではOculus LinkまたはAirLinkを使ってPC-HMDとして接続したOculus Quest2で動作するまでの手順を解説します。



参考

# PC-HMDでの立体視表示(2)

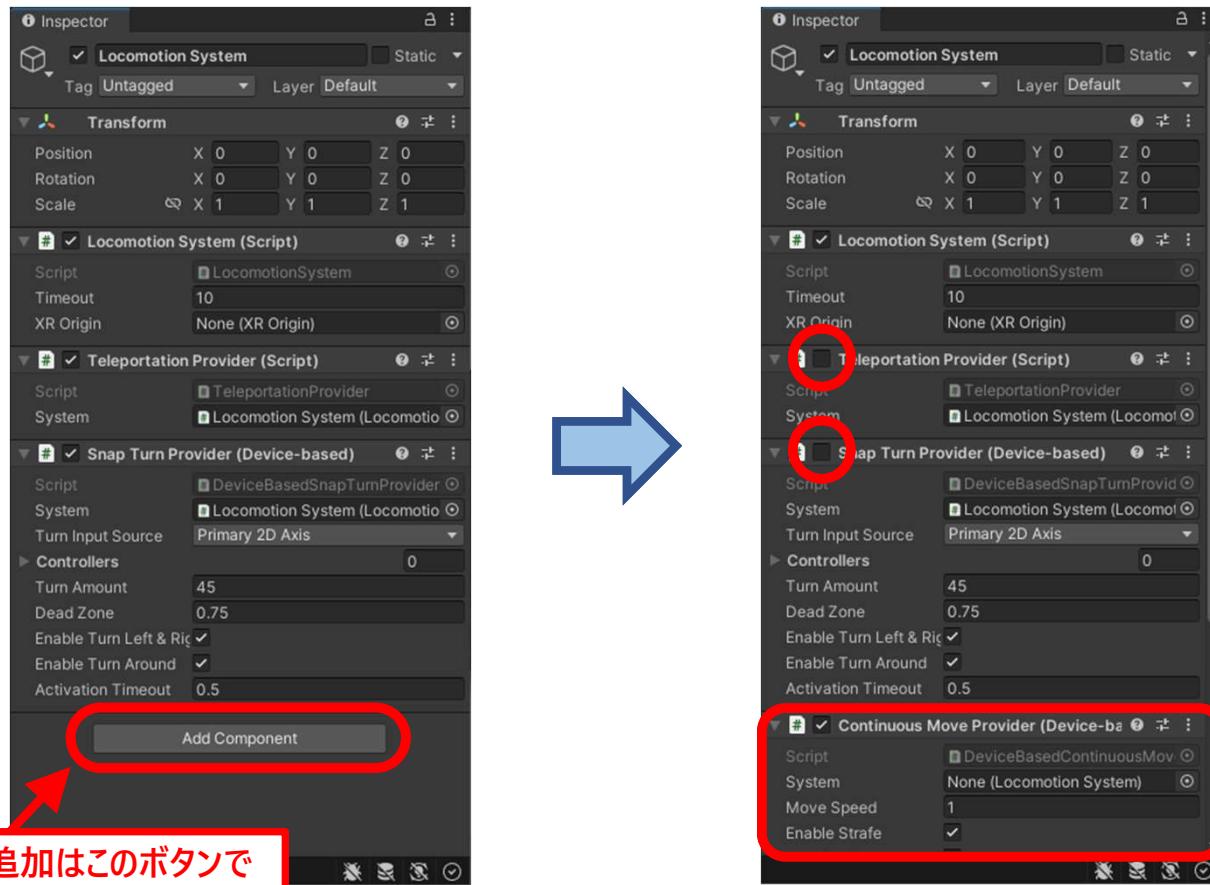
メインメニューから[GameObject]–[XR]–[Device-based]にある「XR Origin」と「Locomotion System」をシーンに追加します。「XR Origin」を追加すると、元々あった「Main Camera」はシーンから削除されます。



参考

# PC-HMDでの立体視表示(3)

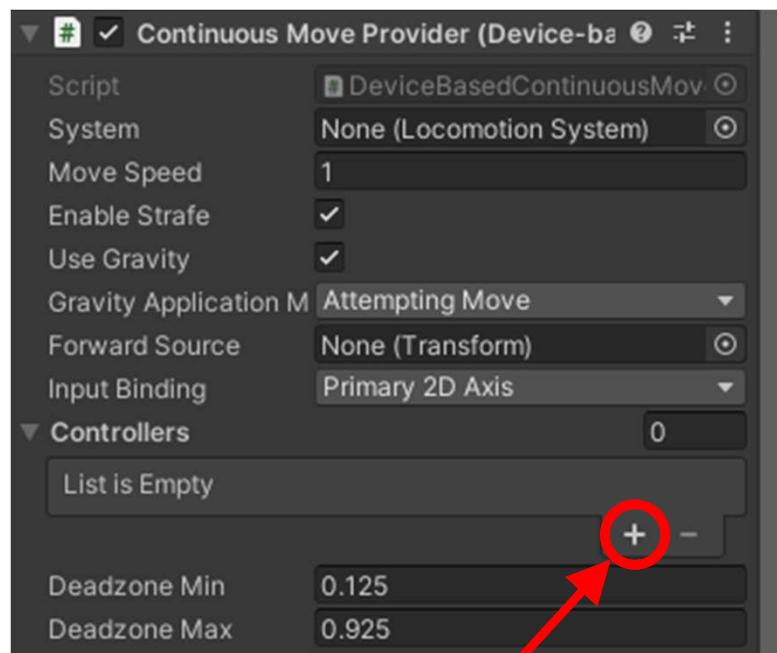
「Locomotion System」のインスペクタから、「Teleportation Provider (Script)」と「Snap Turn Provider (Device-based)」の二つをOFFにし、「Continuous Move Provider (Device-based)」を追加します。



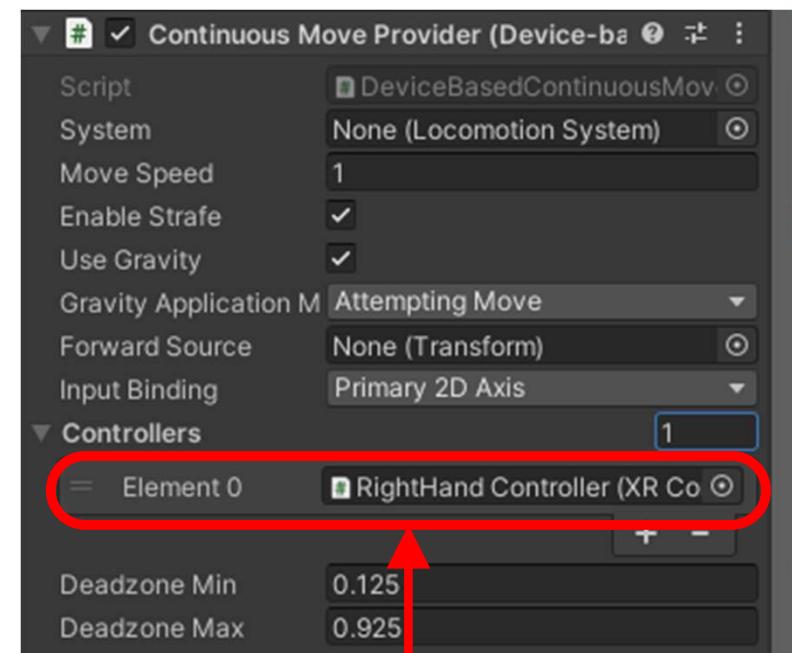
参考

# PC-HMDでの立体視表示(4)

「Continuous Move Provider (Device-based)」の「Controllers」の項目に、操作用コントローラとして「RightHand Controller」を登録します。



「+」ボタンでコントローラを追加

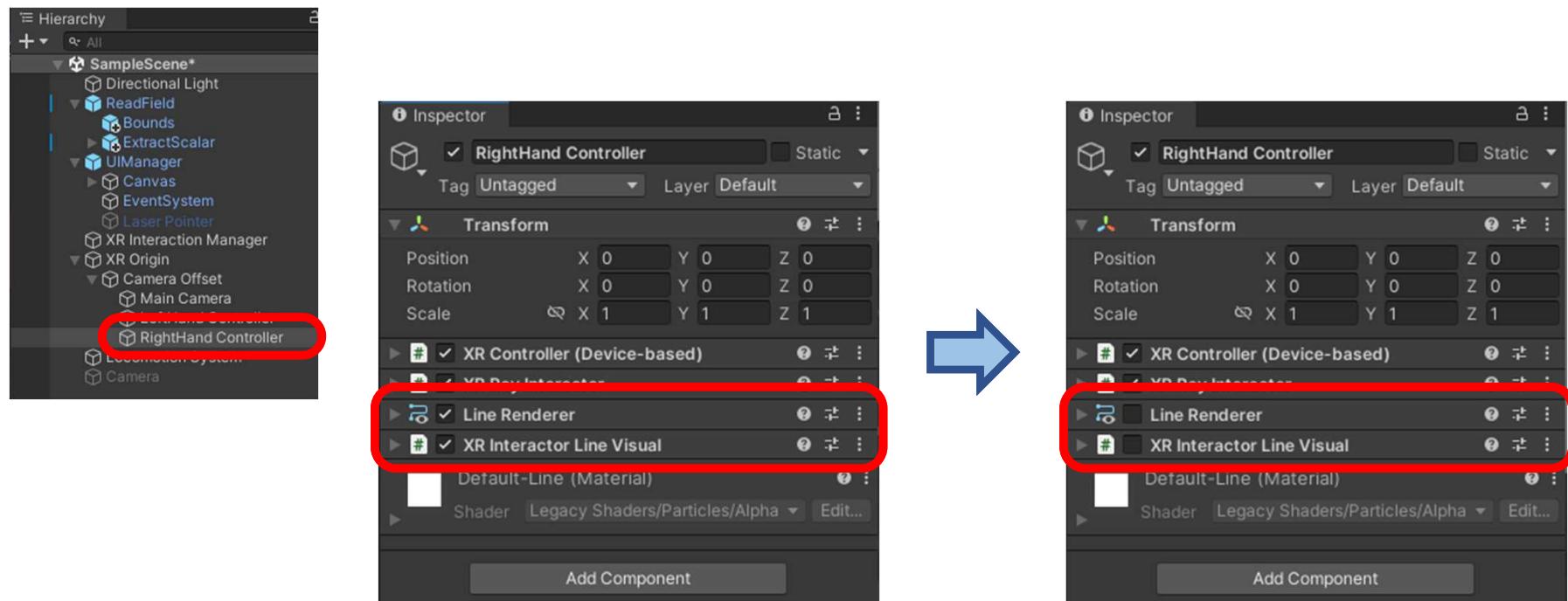


「RightHand Controller」を追加

参考

# PC-HMDでの立体視表示(5)

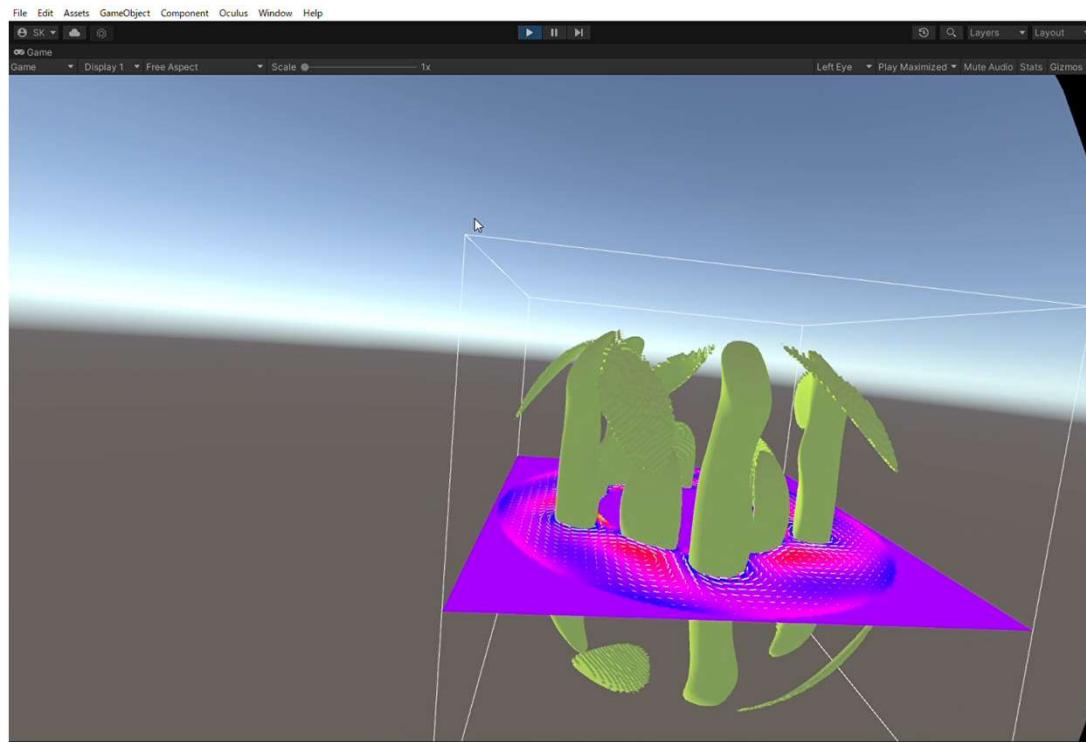
「RightHand Controller」のインスペクタから、「Line Renderer」と「XR Interactor Line Visual」の2つのコンポーネントをOFFにします。この手順は行わなくてもよいですが、コントローラから伸びたレーザポインタが実行中に常に表示された状態になります(VisAssetsではボタン押下によるVRUI表示時ののみレーザポインタを表示するようになっています)。



参考

# PC-HMDでの立体視表示(6)

HMDをPCに接続し、プレビュー実行するとHMDでの立体視表示と、コントローラによるVRUIの操作が可能となります（コントローラの表示には3-Dモデルデータが別途必要です）。Quest2の場合、左コントローラのAボタンを押している間VRUIとレーザポインタが表示され、その状態でトリガー・ボタンを使ってUIパネル上のスライダ等を操作できます。



# Android用ビルドの準備(1)

## Unity側の設定

Androidビルドに必要なモジュールとして、Unityのインストール時のオプション、またはモジュールの追加で「Android SDK & NDK Tools」と「OpenJDK」をインストールします。



# Android用ビルドの準備(2)

## スマートフォン側の設定

### [開発者オプションの有効化]

- ・ [設定]→[デバイス情報]にあるビルド番号を7回タップして開発者オプションを有効にする(PIN/パターン/パスワードの入力が必要になります)

### [USBデバッグの有効化]

- ・ [設定]→[システム]→[開発者オプション]でUSBデバッグを有効にする
- ・ USBケーブルでPCに接続した際、「USBデバッグを許可しますか」のダイアログ出たら「許可」を押す

※機種によって設定項目の場所や表示されるメッセージが異なる可能性があります

※WS終了後、不要であれば開発者オプション、USBデバッグは共に無効にしてください。

**特にUSBデバッグを有効のままにした場合、一部のアプリが動作しない場合があります**

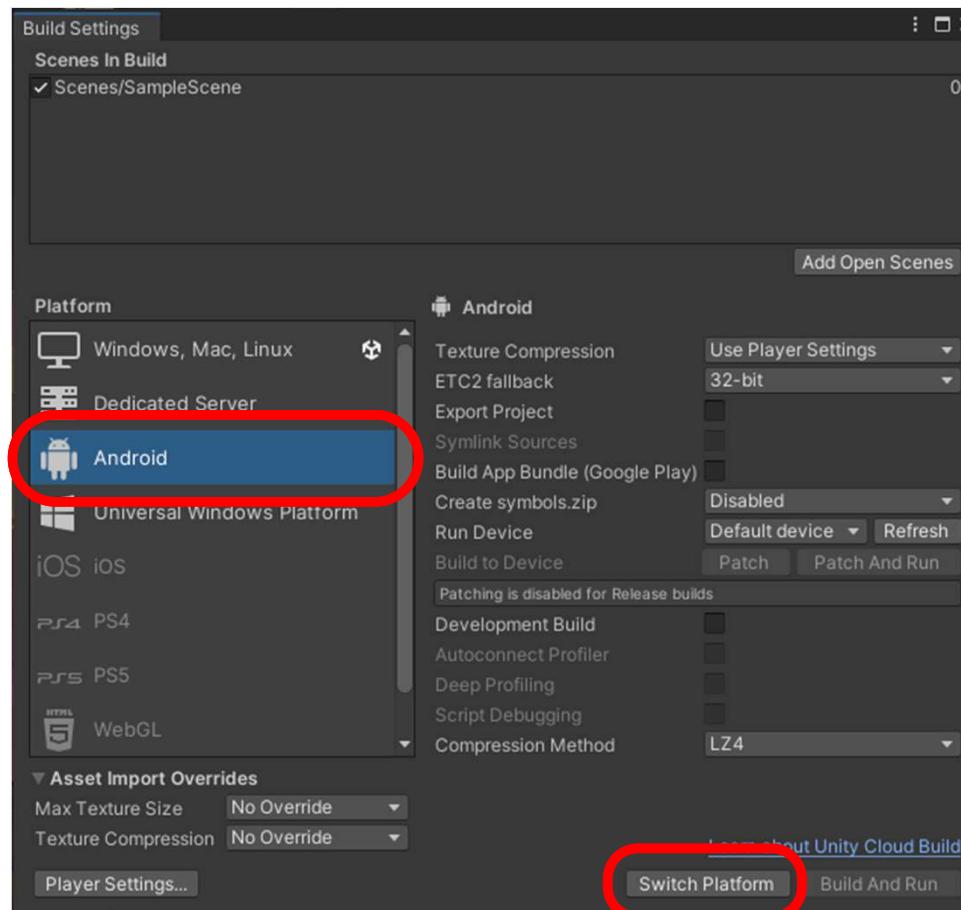
### [開発者オプションの無効化]

- ・ [設定]→[システム]→[開発者オプション]で最上部のスイッチをONからOFFにする

参考

# Android用ビルド(1)

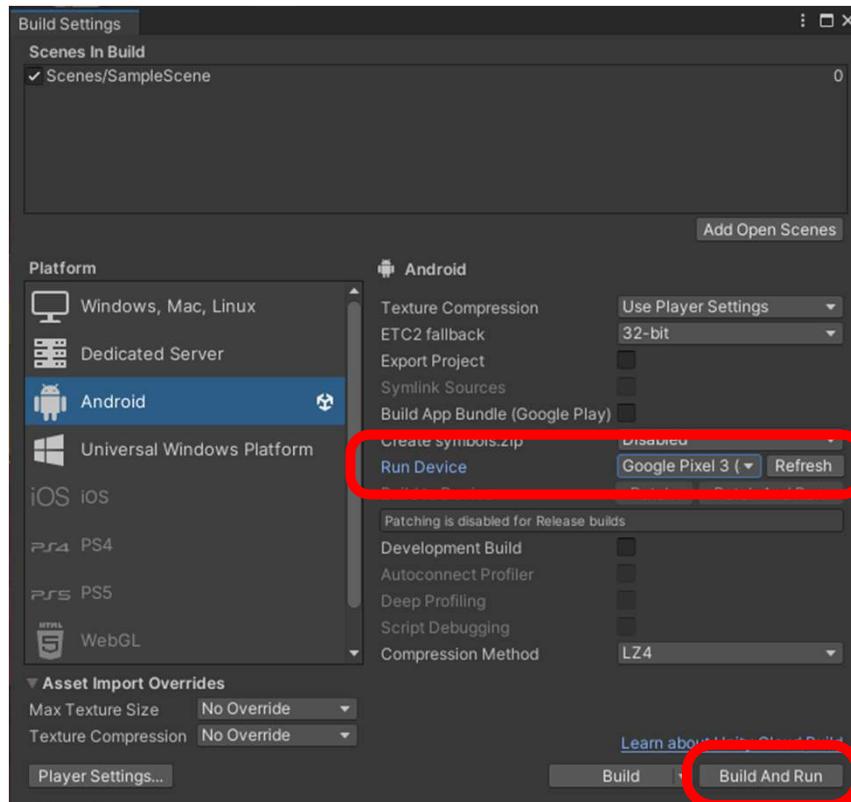
メインメニューから[File]–[Build Settings]でビルト設定を開き、ターゲットプラットフォームを「Windows, Mac, Linux」から「Android」に変更し、「Switch Platform」ボタンを押して確定します。



参考

# Android用ビルド(2)

USBデバッグを有効にしたAndroid機を、データ通信可能なUSBケーブルでPCに接続し、「Build And Run」ボタンを押します。Windowsビルドの際と同様、ビルドの際に複数のファイルやディレクトリが作成されるため、ビルド関連のファイルを格納するディレクトリを指定します。ビルドが成功するとスマートフォンでVisAssetsアプリケーションが動作します。実機にインストールされるため、USBケーブルを抜いた後も実行可能です(動作確認はGoogle Pixel3で行っています)。



# スタンドアロンHMDでの立体視表示

Android用にビルドする際、「PC-HMDでの立体視表示」の項で紹介したXRデバイス用設定をしておくことで、VisAssetsを使って構築したアプリケーションをスタンドアロンHMDでも実行することができます(動作検証はQuest2で行っています)。

その他、Android用ビルト設定のいくつかの項目をQuest2用に変更する必要がありますが、Quest2向けのビルト方法については本チュートリアルでは詳述しません。下記のURLなどを参考にビルトしてみてください。

[参考] Unity + Oculus Quest (Meta Quest) 開発メモ

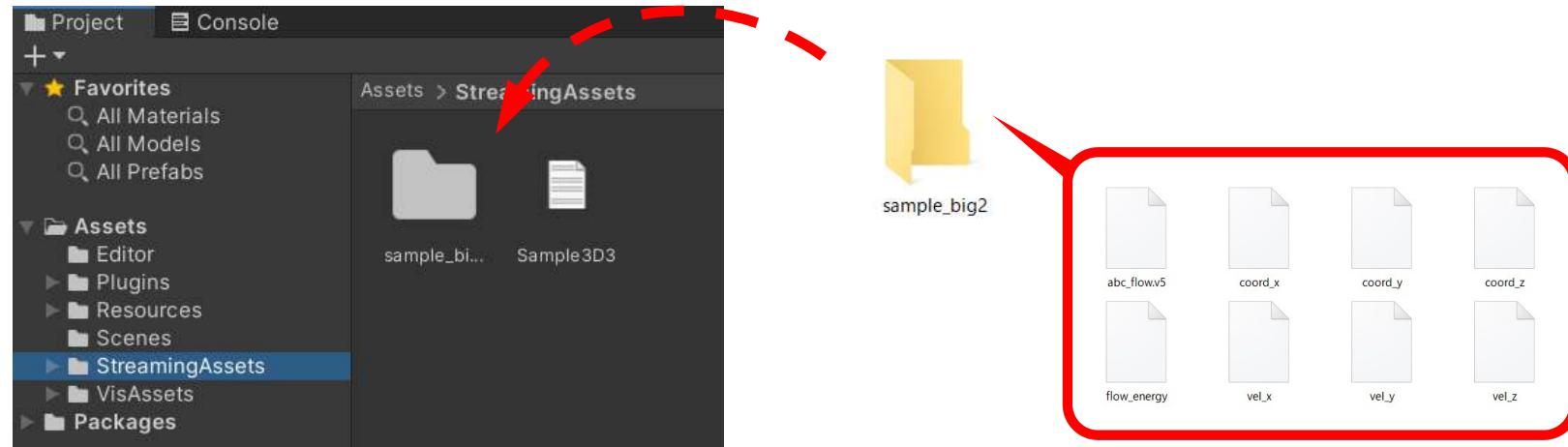
<https://framesynthesis.jp/tech/unity/oculusquest/>

# ベクトル場の可視化(1)

ReadField用データにはベクトル場のデータが含まれていませんので、「ReadV5」モジュールを使ってCAVE型VR装置用可視化ソフトウェアVFIVEのサンプルデータ(ABC Flow)に含まれるベクトル場データを可視化してみましょう。

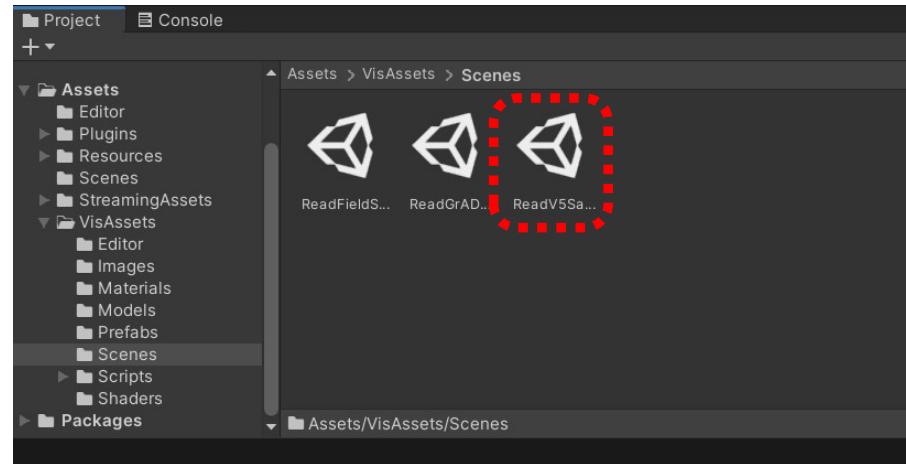
ダウンロードしたデータ(sample\_big2.tar.gz)を展開し、アーカイブに含まれる8つのファイルを「sample\_big2」という名前のフォルダに入れます(フォルダの作成場所はどこでも構いません)。

「sample\_big2」フォルダをUnityエディタのプロジェクトウィンドウの Assets/StreamingAssets にドラッグアンドドロップし、アセットとして登録します。



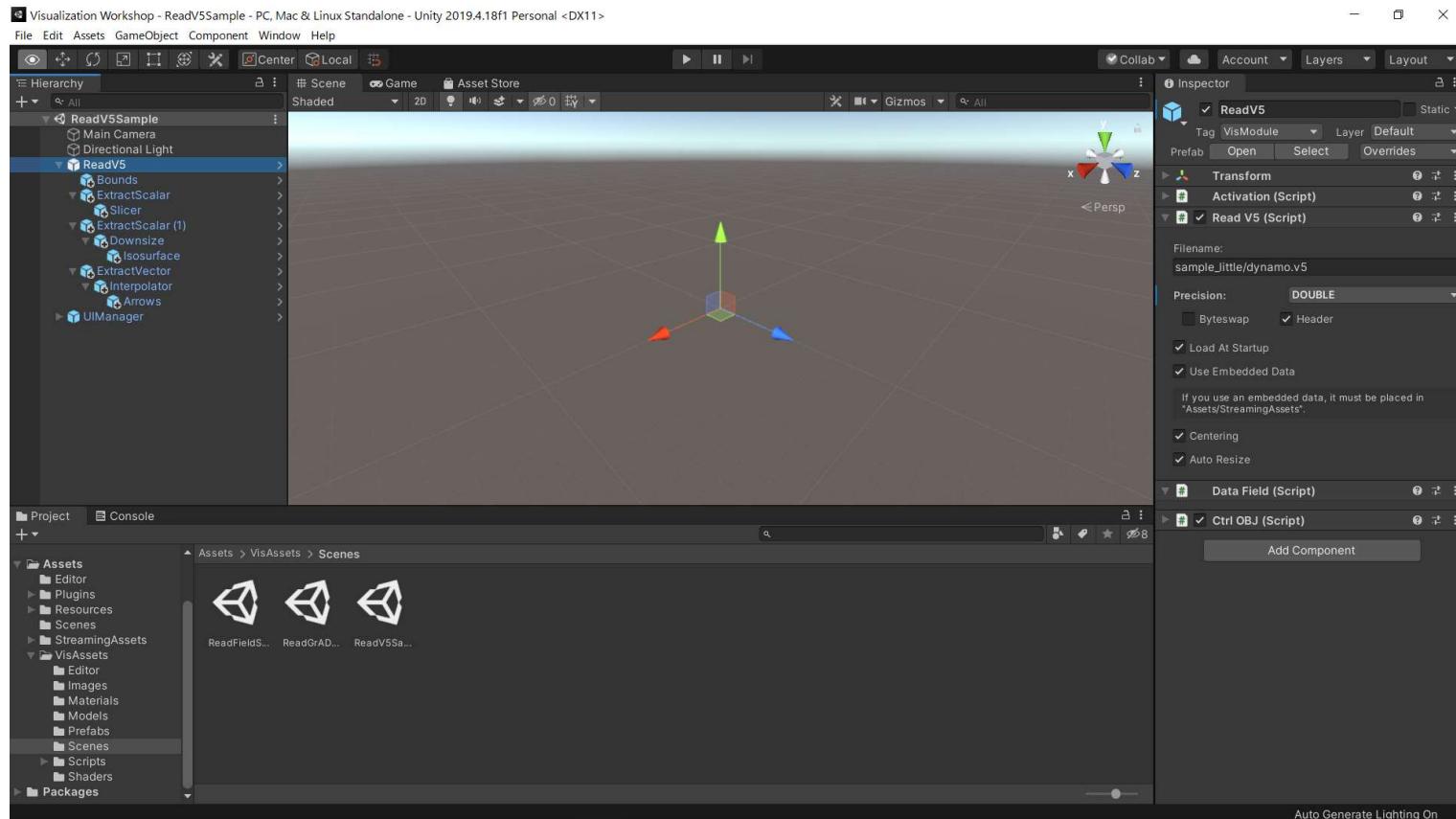
# ベクトル場の可視化(2)

現在開いているシーンを保存し、プロジェクトウィンドウから  
Assets/VisAssets/Scenes にある ReadV5Sample.scene をダブルクリックします。



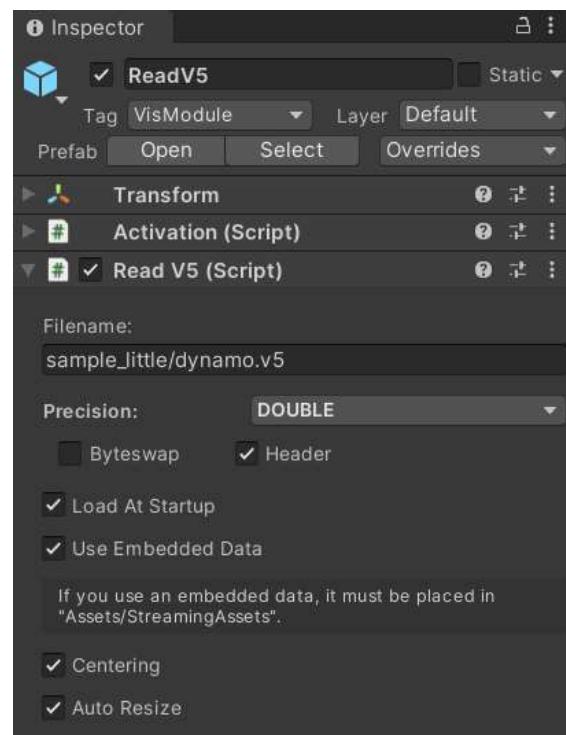
# ベクトル場の可視化(3)

このサンプルシーンでは既に可視化モジュールが接続されています。次のスライドでABC Flowデータを読み込むための設定をします。

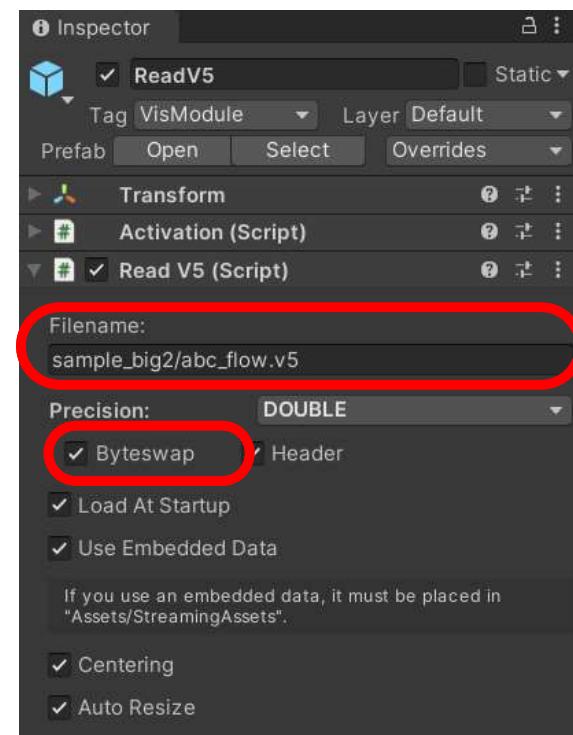


# ベクトル場の可視化(4)

サンプルシーンはABC Flowとは別のVFIVE用データ(ダイナモシミュレーション)向けの設定がされていますので、ABC Flowのデータ用に一部設定を変更します。ファイル名の変更の他、バイトオーダーの変換が必要なデータのためByteswapをONにします。TransformコンポーネントのPositionの値は「Z=10」にしておきましょう。



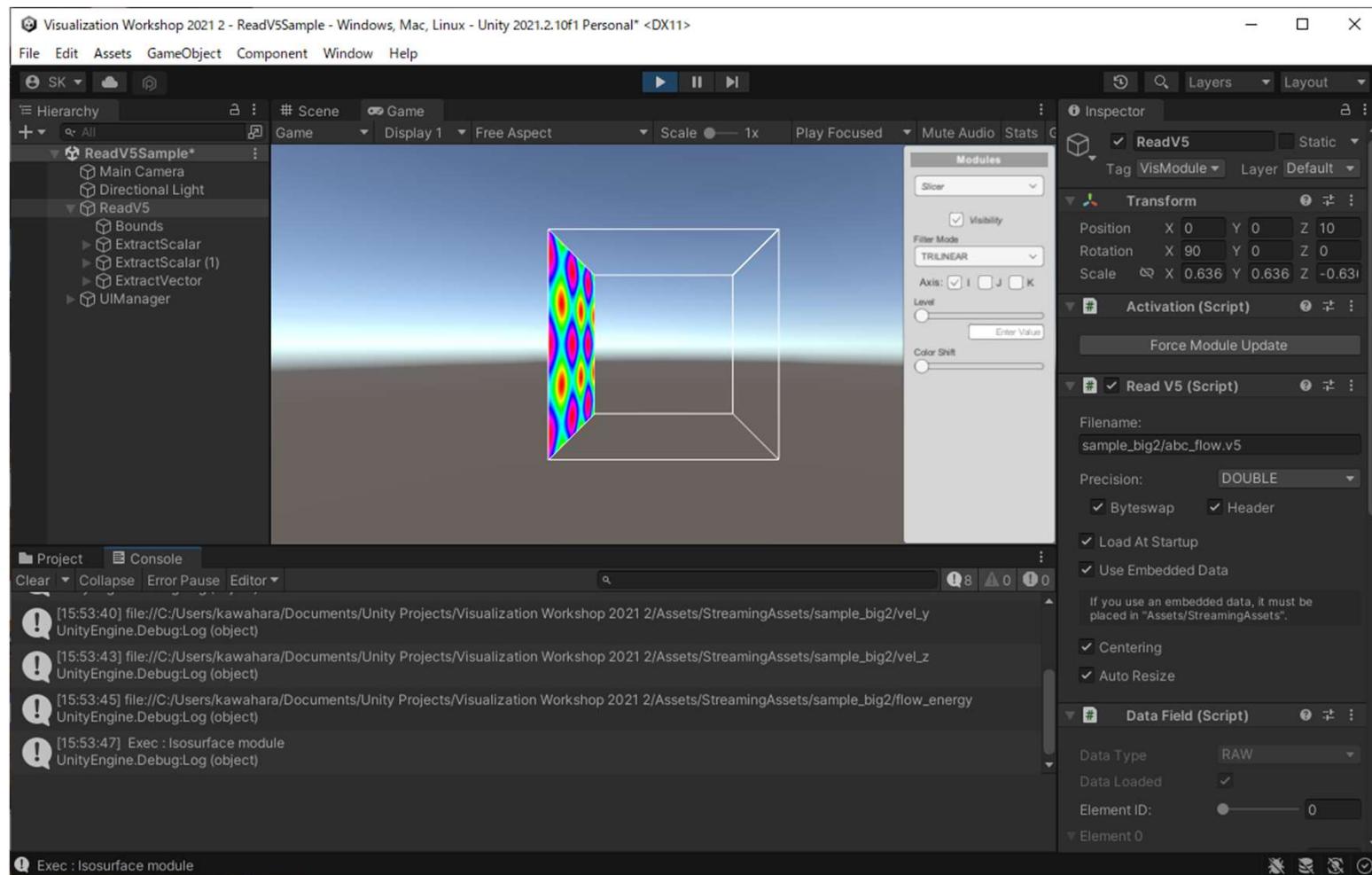
変更前



変更後

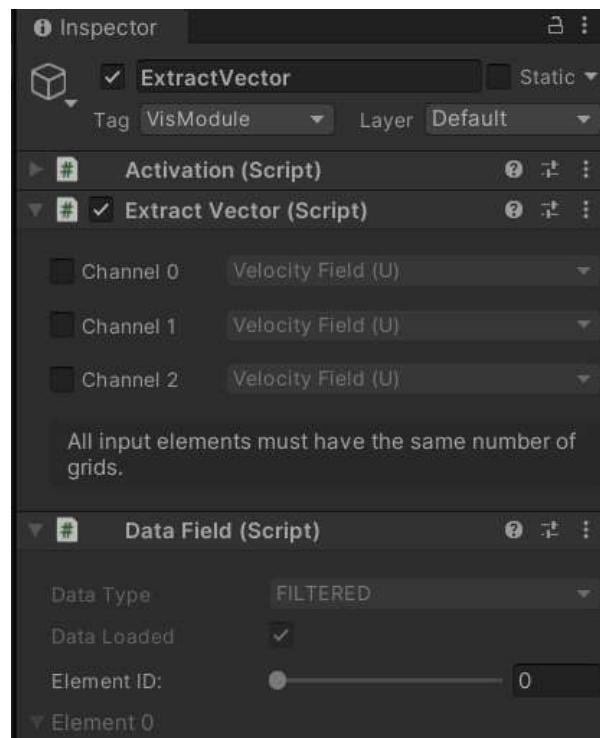
# ベクトル場の可視化(5)

再生ボタンを押し、読み込みが完了すると下記のような表示になります。

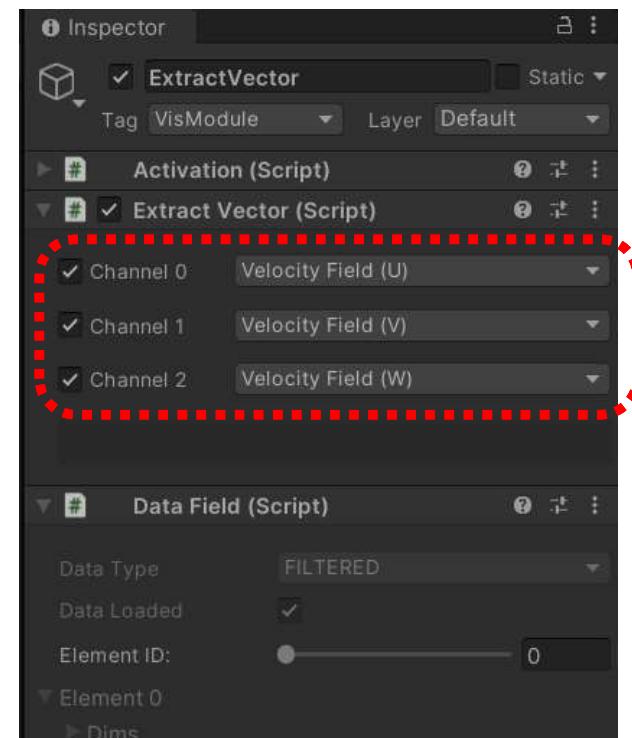


# ベクトル場の可視化(6)

ヒエラルキーインドウで「Extract Vector」モジュールを選択し、インスペクタからベクトル場の各軸成分(Channel 0~2)を下図のように設定します。各軸の成分は、各ChannelのチェックボックスをONにすると選抲可能になります。VFIVE用データでは、データセット内の各要素に名称が付けられており、その名称が表示されます。



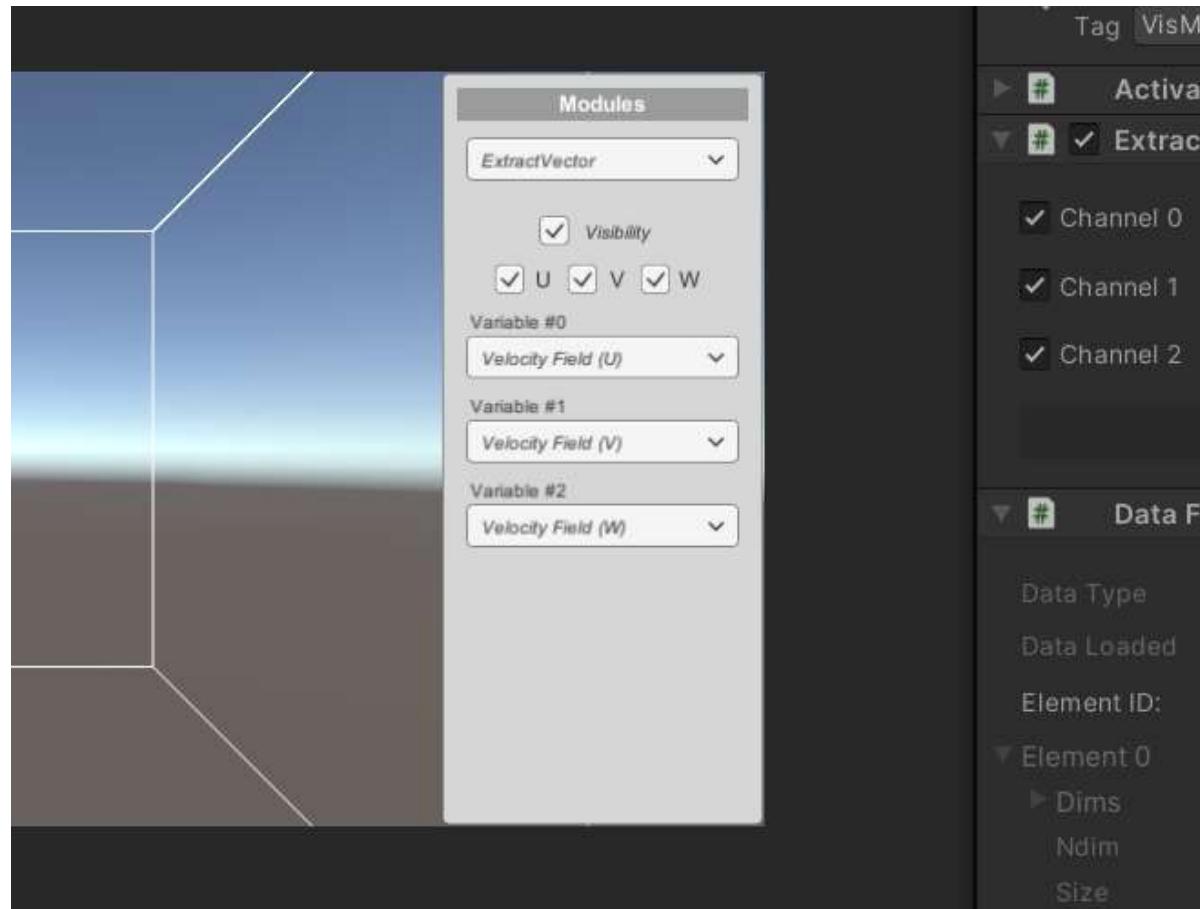
変更前



変更後

# ベクトル場の可視化(7)

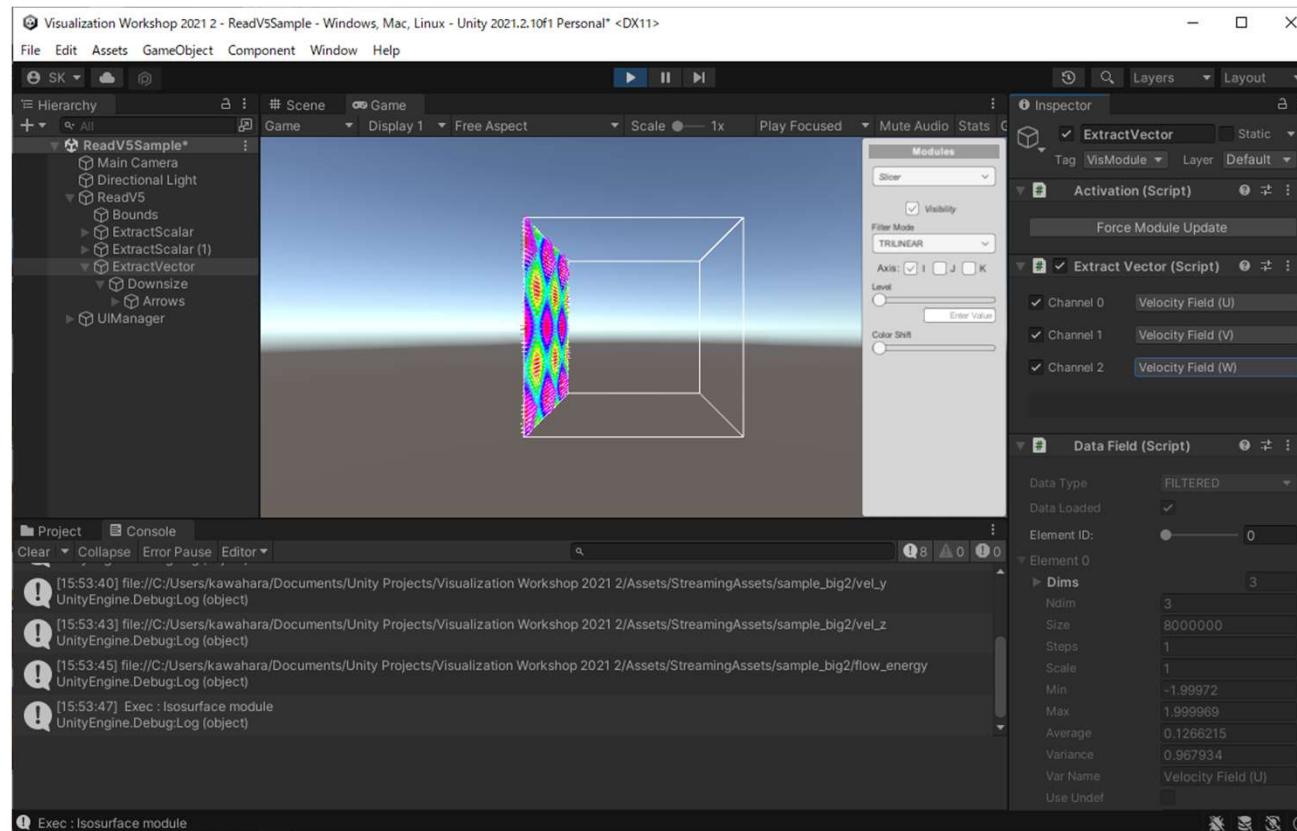
UIManagerをシーンヒエラルキーに配置している場合、実行画面上に表示されたUIからベクトル場の各軸成分を設定することもできます。



# ベクトル場の可視化(8)

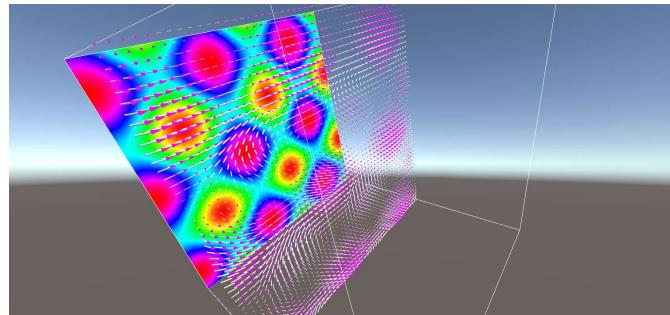
設定が完了すると下記のような表示になります。

左端のカラースライスと重なってわかりにくいですが、「Extract Vector」モジュールで設定したベクトル場が、子として接続された「Arrows」モジュールにより多数の矢印として描画されます。

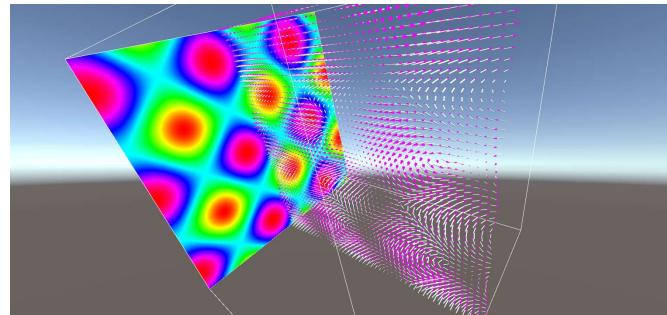


# ベクトル場の可視化(9)

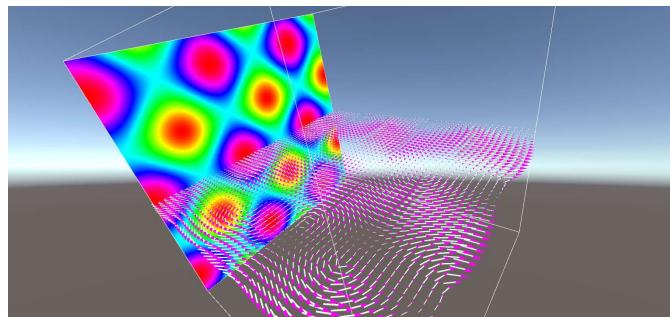
「Arrows」モジュールは、ベクトル場データを入力とし、指定した断面上のベクトル場を矢印で表現するためのモジュールです。インスペクタから、またはシーン上のGUIから軸と断面位置の設定ができます。



Axis=0, Slice=0.5



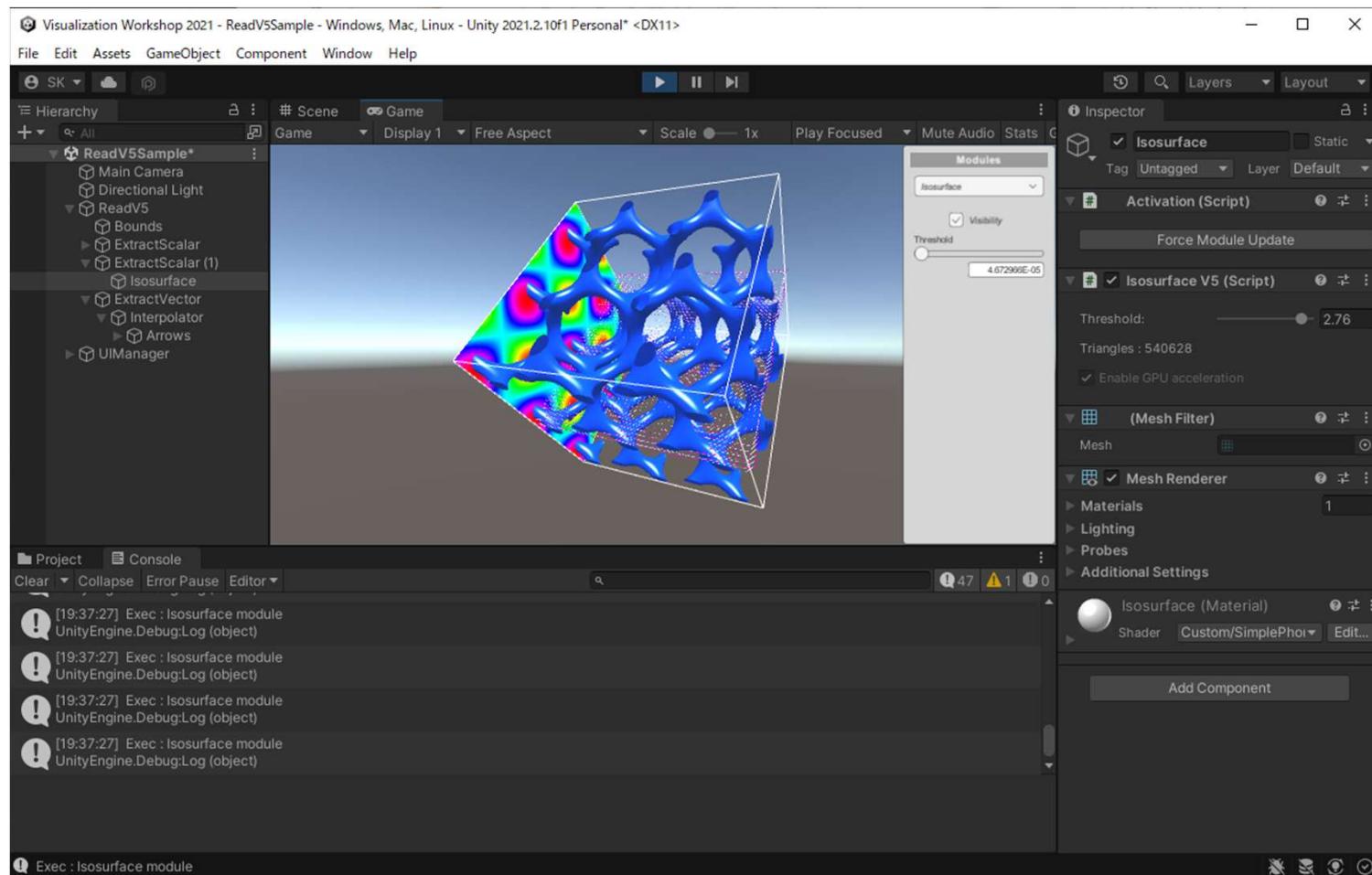
Axis=1, Slice=0.5



Axis=2, Slice=0.5

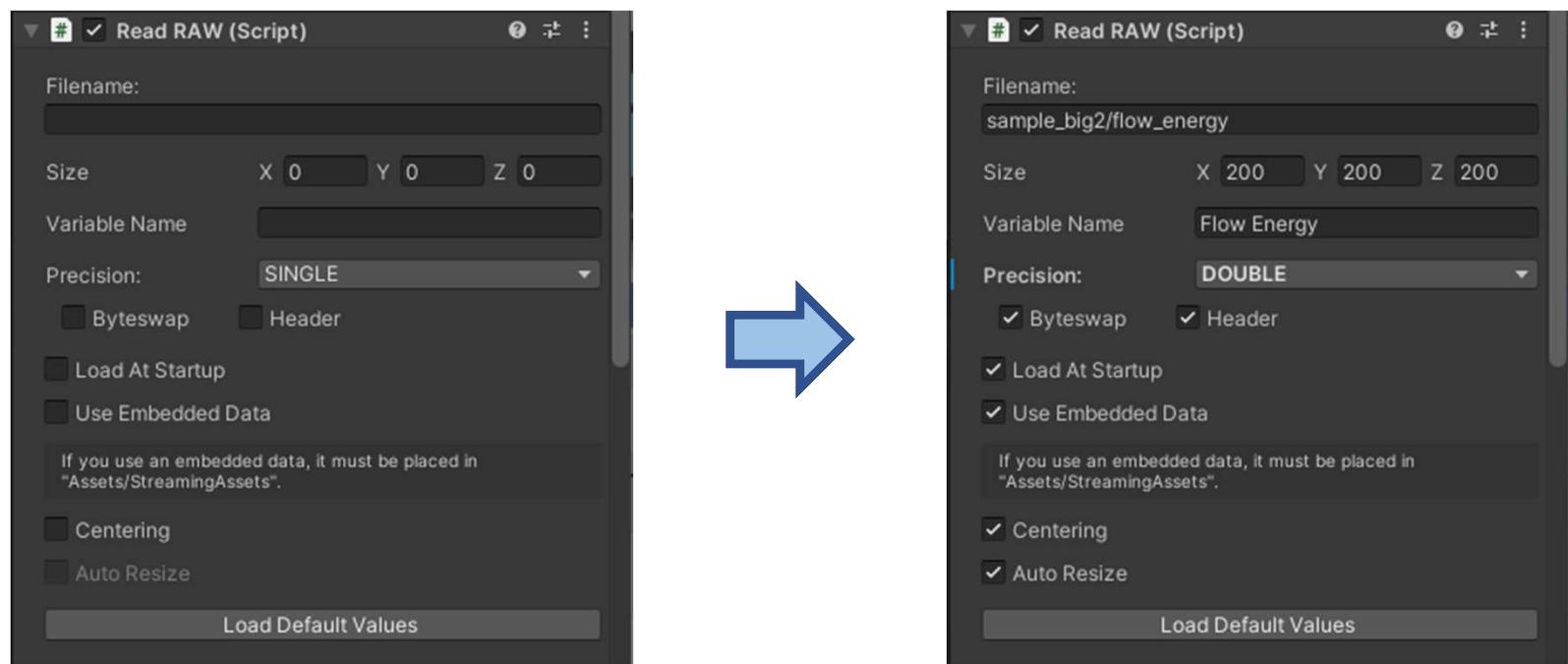
# ベクトル場の可視化(10)

等値面についても併せて設定したものが下図となります。



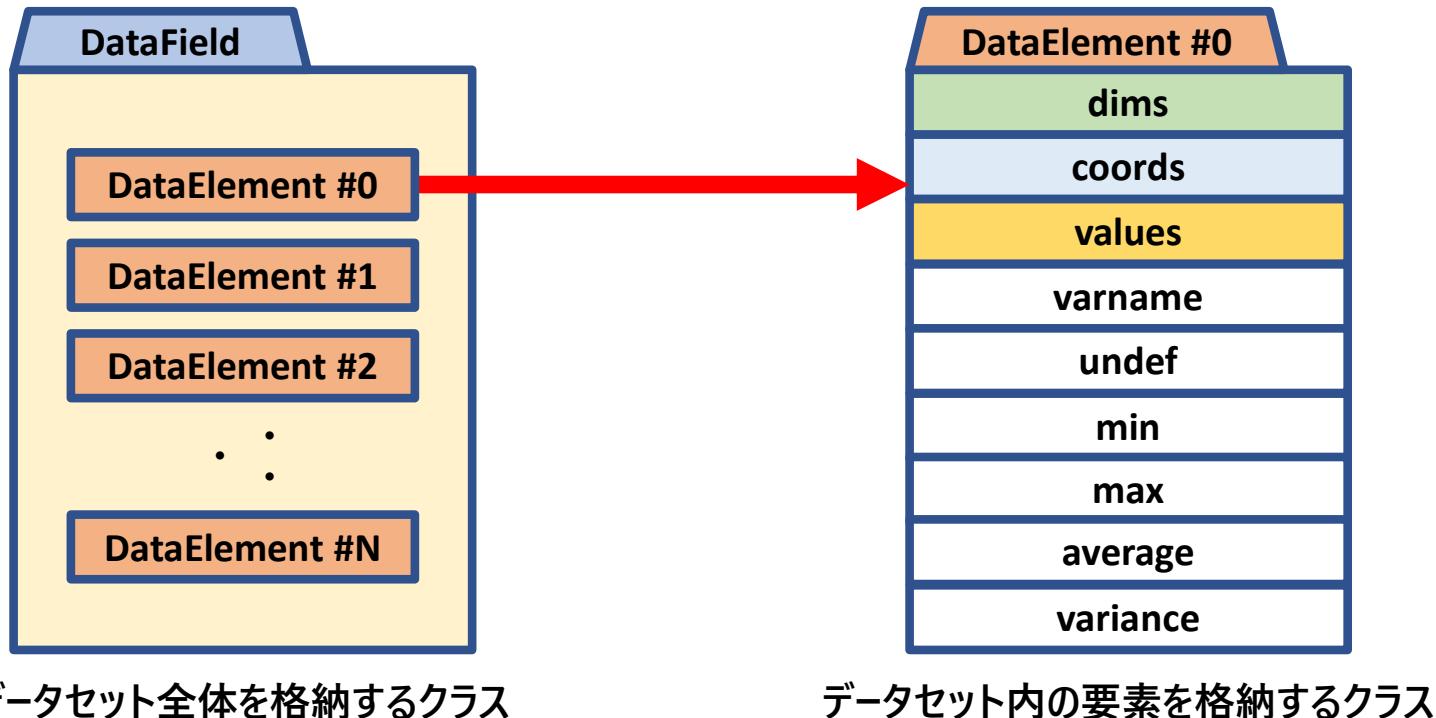
# バイナリデータの読み込み

ReadRAWモジュールを使うことで、素のバイナリデータを読み込むこともできます。「ベクトル場の可視化」の項で使用したVFIELD用データに含まれるスカラー場データ (sample\_big2/flow\_energy、グリッドサイズ $200 \times 200 \times 200$ 、倍精度、ビッグエンディアン、FORTRANのレコード長データ付き) を読み込む場合の設定は下記の通りです。



# 内部データ構造

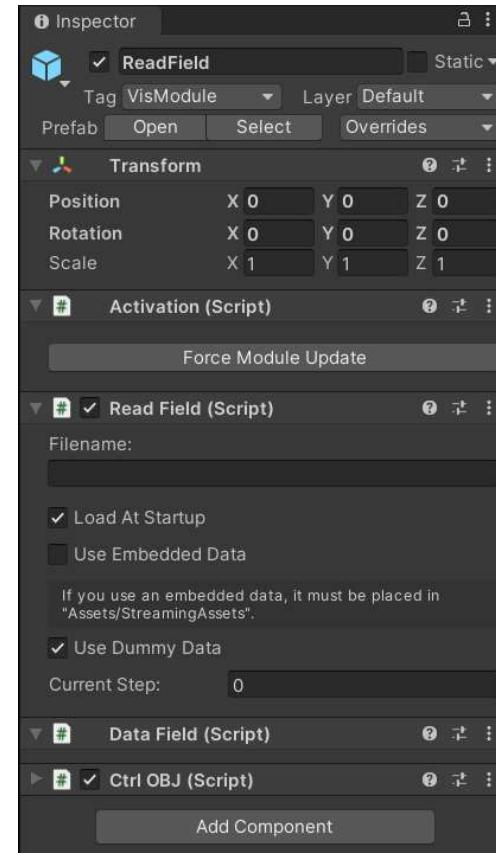
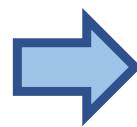
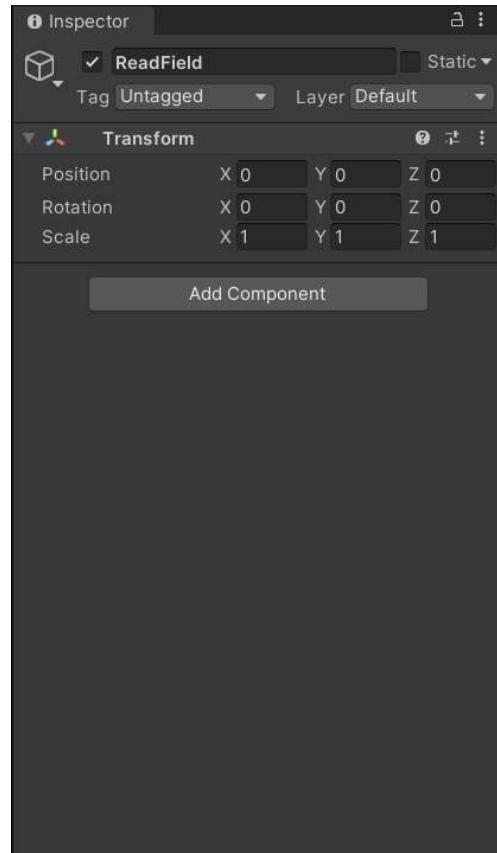
構造格子のデータセットであれば、下図に示すデータ構造に格納することで他の可視化モジュールについては共用することができます。テキストデータ用のReadFieldや、バイナリデータ用のReadRAW、まずテキストデータを読み込み、そこに書かれたバイナリデータを読み込むReadV5やReadGrADSにアタッチされたC#スクリプトを参考に、各自のデータを読み込むための新たなモジュールを開発してみてください。



# 新たなモジュールの開発(1)

## VisAssetsにおける可視化モジュール

- 必要な機能を実装したC#スクリプトなどをアタッチしたゲームオブジェクト(を再利用しやすい「プレハブ」と呼ばれる形式にしたもの)



ReadFieldモジュール

C#スクリプト  
(モジュール間の状態遷移の制御用)

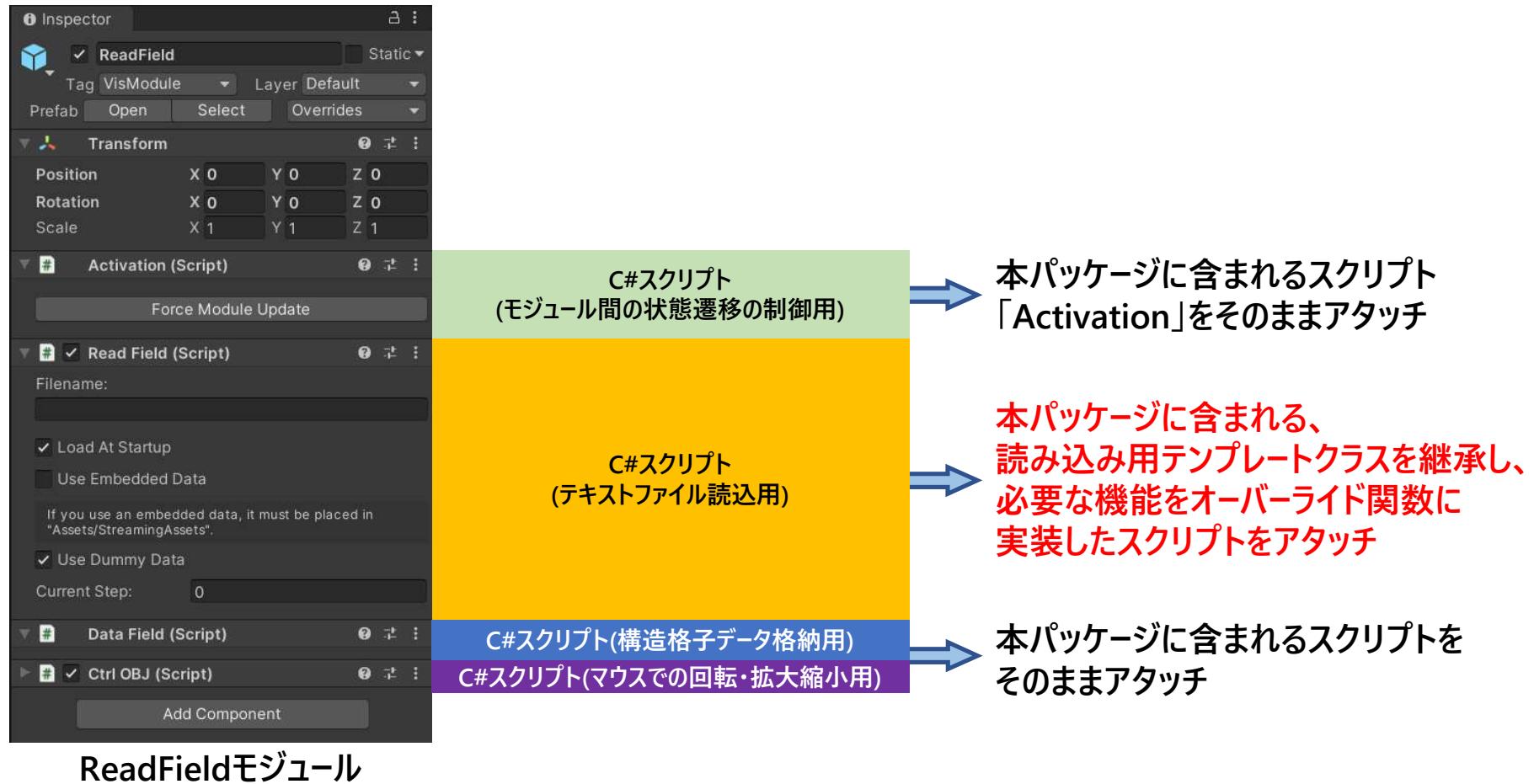
C#スクリプト  
(テキストファイル読込用)

C#スクリプト(構造格子データ格納用)

C#スクリプト(マウスでの回転・拡大縮小用)

# 新たなモジュールの開発(2)

モジュール間の接続チェックや、パラメータ変更時の他のモジュールへの伝搬についてはVisAssetsに含まれるテンプレートクラスが処理します。このため、テンプレートクラスを継承した新たなクラスを作成し、そのオーバーライド関数内に必要な機能を実装するだけで新たなモジュールが開発できます。



# 新たなモジュールの開発(3)

## C#スクリプトの例(1)：データ読み込みモジュール用

```
// 読み込みモジュール用テンプレートクラスを継承したクラス

public class ReadField : ReadModuleTemplate
{
    public override void InitModule()
    {
        // モジュールの初期化処理を記述
    }

    public override int BodyFunc()
    {
        // Unityにおけるフレーム毎のイベントループで親モジュールや自身のパラメータを監視し、
        // それらに変更があった場合に実行される処理を記述する
        // このスクリプトの場合はデータ読み込み用なので、
        // ファイルダイアログから指定されたファイルや組み込みデータの読み込み処理を記述する
    }
}
```

# 新たなモジュールの開発(4)

## C#スクリプトの例(2): フィルターモジュール用

```
// フィルターモジュール用テンプレートクラスを継承したクラス
public class ExtractScalar : FilterModuleTemplate
{
    // 下の二つのメンバ変数はテンプレートクラスで宣言しているため
    // 継承したクラスではそれぞれ pdf, df の変数名でアクセス可能
    // public DataField pdf; // 親モジュールのDataField
    // public DataField df; // 自身のDataField
    public int selected_channel;

    public override void InitModule()
    {
        // モジュールの初期化処理を記述
        selected_channel = 0;
    }

    public override int BodyFunc()
    {
        // Unityにおけるフレーム毎のイベントループで親モジュールや自身のパラメータを監視し、
        // それらに変更があった場合に実行される処理を記述する
        df.elements[0] = pdf.elements[selected_channel].Clone();
    }
}
```

# まとめ

本チュートリアルでは、Unity用可視化フレームワーク「VisAssets」をハンズオン形式で体験して頂きました。本フレームワークは現在も開発中のものであり、同梱した各モジュールもサンプル実装のため、高速化など改良の余地は多々あります。新たなモジュール開発もしやすい設計となっていますので、よろしければ是非皆さんも開発にご参加ください。

# 謝辞

本研究の一部はJSPS科研費19K11995および21K11916の助成を受けたものです。

# VFIVE用データの読み込み

CAVE型VR装置用可視化ソフトウェア VFIVE

- <https://www.jamstec.go.jp/ceist/aeird/avcrg/vfive.ja.html>

ダイナモシミュレーションデータ (sample\_little.tar.gz)

- アプリ組み込みデータとする際は Assets/StreamingAssets/sample\_little の下にデータを配置し、下記のようにReadV5モジュールの設定をする
- ReadV5の設定 : Filename = sample\_little/dynamo.v5  
Load At Startup = ON, Use Embedded Data = ON  
Precision=DOUBLE, byteswap = OFF, header = ON

ABC Flow (sample\_big2.tar.gz)

- アプリ組み込みデータとする際は Assets/StreamingAssets/sample\_big2 の下にデータを配置し、下記のようにReadV5モジュールの設定をする
- ReadV5の設定 : Filename = sample\_big2/abc\_flow.v5  
Load At Startup = ON, Use Embedded Data = ON  
Precision=DOUBLE, byteswap = ON, header = ON

# GrADS用データの読み込み

GrADS (The Grid Analysis and Display System)

- <http://cola.gmu.edu/grads/>
- テキストファイル(読み用) + バイナリファイル(データ本体)で構成
- サンプルモジュール(**ReadGrADS**)では非圧縮のバイナリデータのみに対応  
(本家GrADSはNetCDFやGRIBにも対応)

サンプルデータ

- ダウンロードページにある example.tar.gz

