

# Topic XX 3D都市モデルを使った位置情報共有ゲームを作る

---

近年位置情報を基盤としたサービスが多く使われています。Google MapsやUberなどはその典型でしょう。自分の位置情報をスマートフォン端末のGPSで取得し、サービスのサーバーに送信することで、さまざまな便利なサービスを受けることができます。本トピックでは、このような位置情報をオンラインでやり取りしたり、サーバーで処理してサービスを提供するなどの参考になるように、例としてPLATEAUの3D都市モデルを活用した位置情報ゲームを作りながら説明します。

## まずは基本のゲームを作る

本トピックではUnityでスマートフォンAR位置情報ゲームを作ります。ゲームを起動すると、プレイヤーは最初に赤緑青の三色から選び自分が属する陣営を決めます。AR画面になると、端末カメラが映す実際の街並みが表示されます。PLATEAUの3D都市モデルを使い、タップするとその場所の建物に「ペンキ」が塗られ、町中の色々なところを塗って最初の点までつないで閉じると、そこが自陣営の陣地になります。ゲームは獲得した陣地の面積を競います。

サーバーサイドにはPostGIS拡張をインストールしたPostgreSQLデータベースを使います。面積は、ゲームプレイ時はとった陣地の合計をサーバーで計算しますが、ゲーム終了後にバッチ処理で重ね合わせを考慮した結果の面積を計算する仕様にします。これは、時間順に重ね合わせて面積を計算する処理が、リアルタイムに計算にするには重いためです。

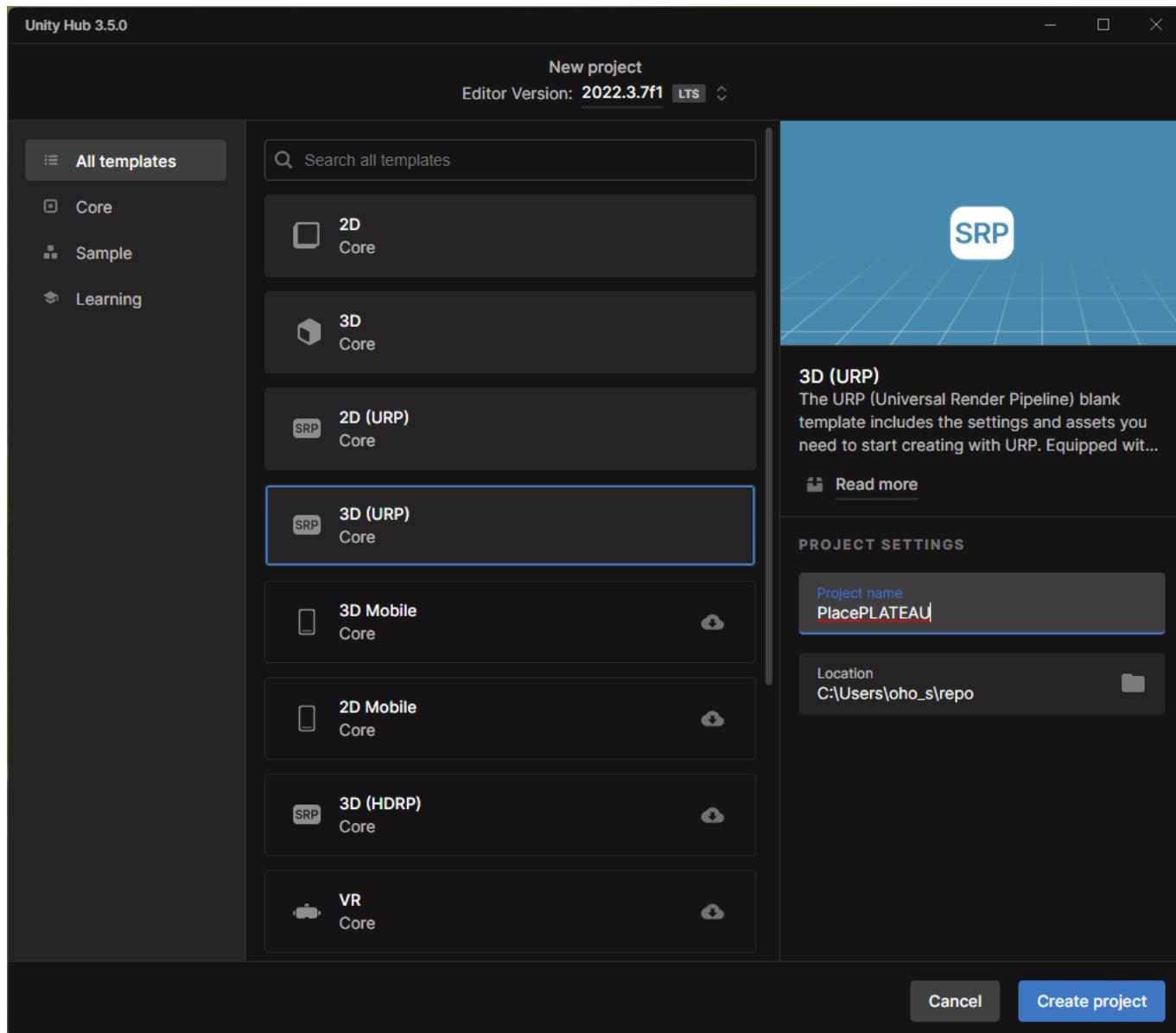
## ARの設定をする

最初にプロジェクトを作成し、基本のAR部分を設定します。本プロジェクトでは、iOS・Androidを対象として、ARFoundationおよびARCoreExtentionでGeospatial APIを使います。

まずはUnityで新規プロジェクトを作成します。今回は、Unity 2022.3.7f1を使います。

ARの設定は、PLATEAU公式サイトのチュートリアルコンテンツの[TOPIC14-3 「Google Geospatial APIで位置情報による3D都市モデルのARを作成する」](#)も参考にしてください。また、UnityでのAR開発の[公式ドキュメント](#)や、ARFoundationの[公式ドキュメント](#)なども必要に応じて参照してください。ここでは、最低限の説明をします。

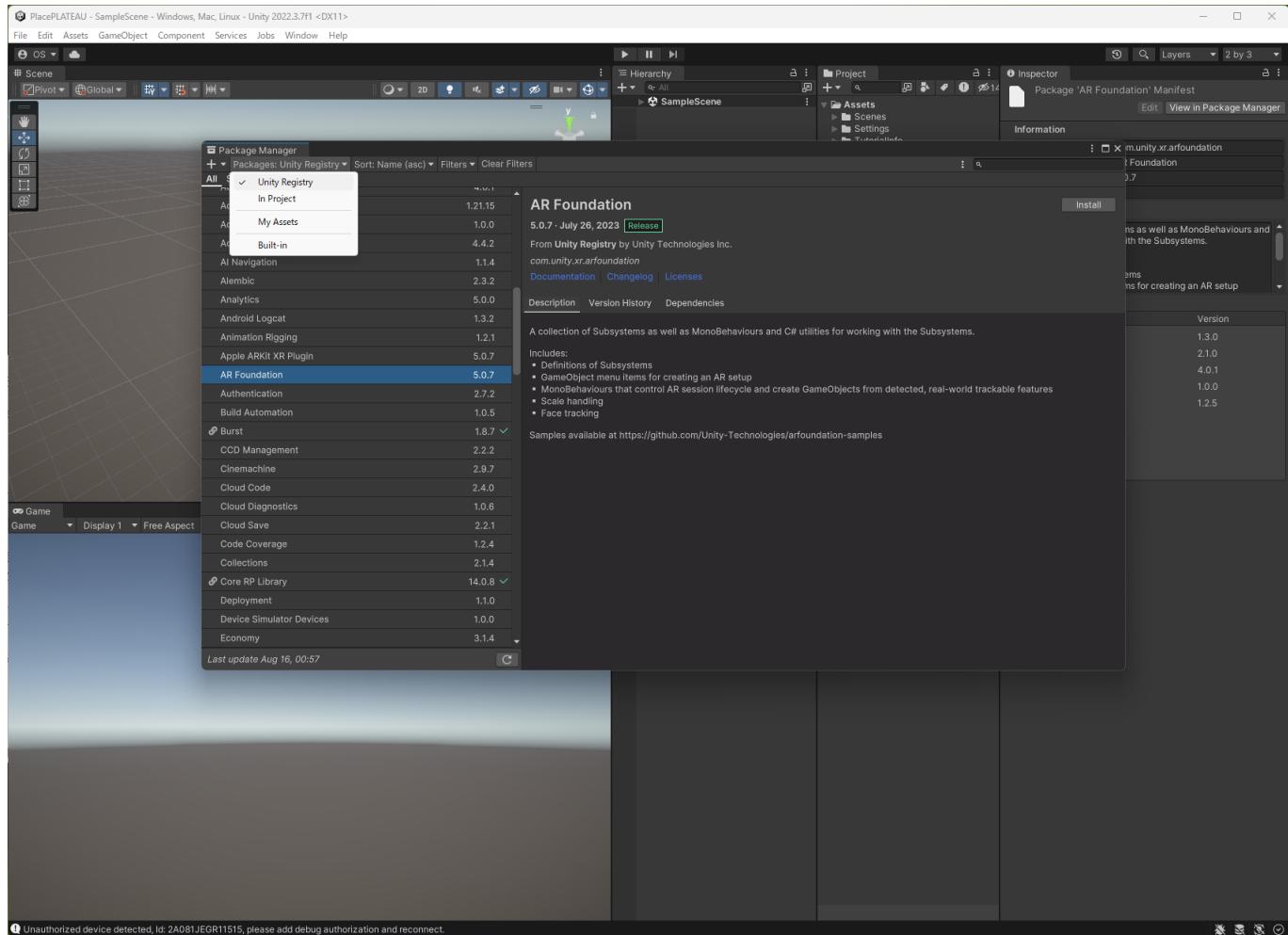
まず、Unity Hubからプロジェクトを新規作成します。今回は、「3D(URP)」を選択します。必要に応じてダウンロードボタンを押してテンプレートをダウンロードしてください。プロジェクト名を「PlacePLATEAU」にします。PLATEAUを使った陣取りゲームからの命名です。



プロジェクトが立ち上がったら、ARFoundationとGepspatialAPIを導入します。

[Window] メニューから [Package Manager] を開きます。

Package Managerが開いたら、「Unity Registry」を選択し、「ARFoundation」をインストールします。何らかのダイアログが開いたら適切に対処します。「New Input System」の有効化が必要であればダイアログの指示にしたがい、再起動などを起こさないでください。

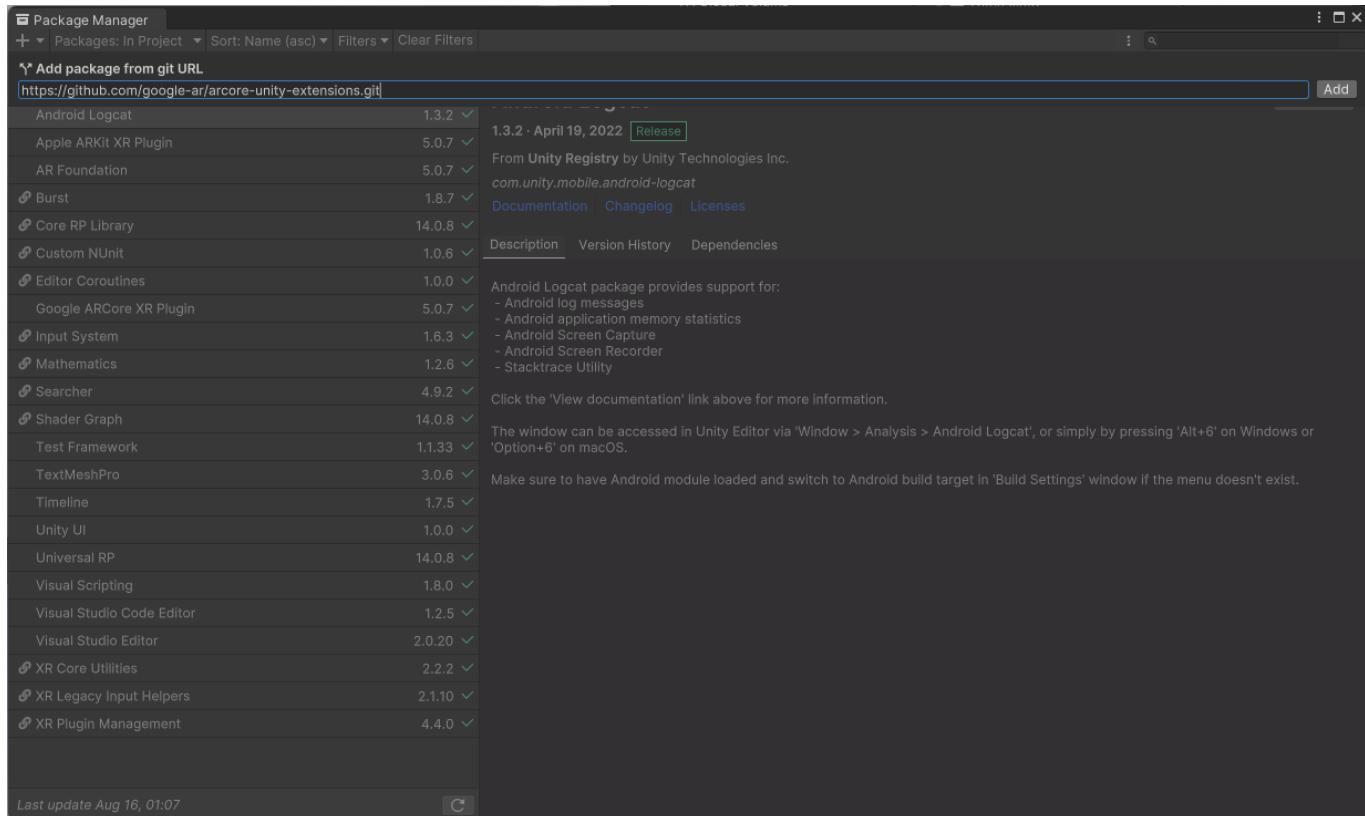


同様にして、「Apple ARKit XR Plugin」、「Google ARCore XR Plugin」をインストールします。

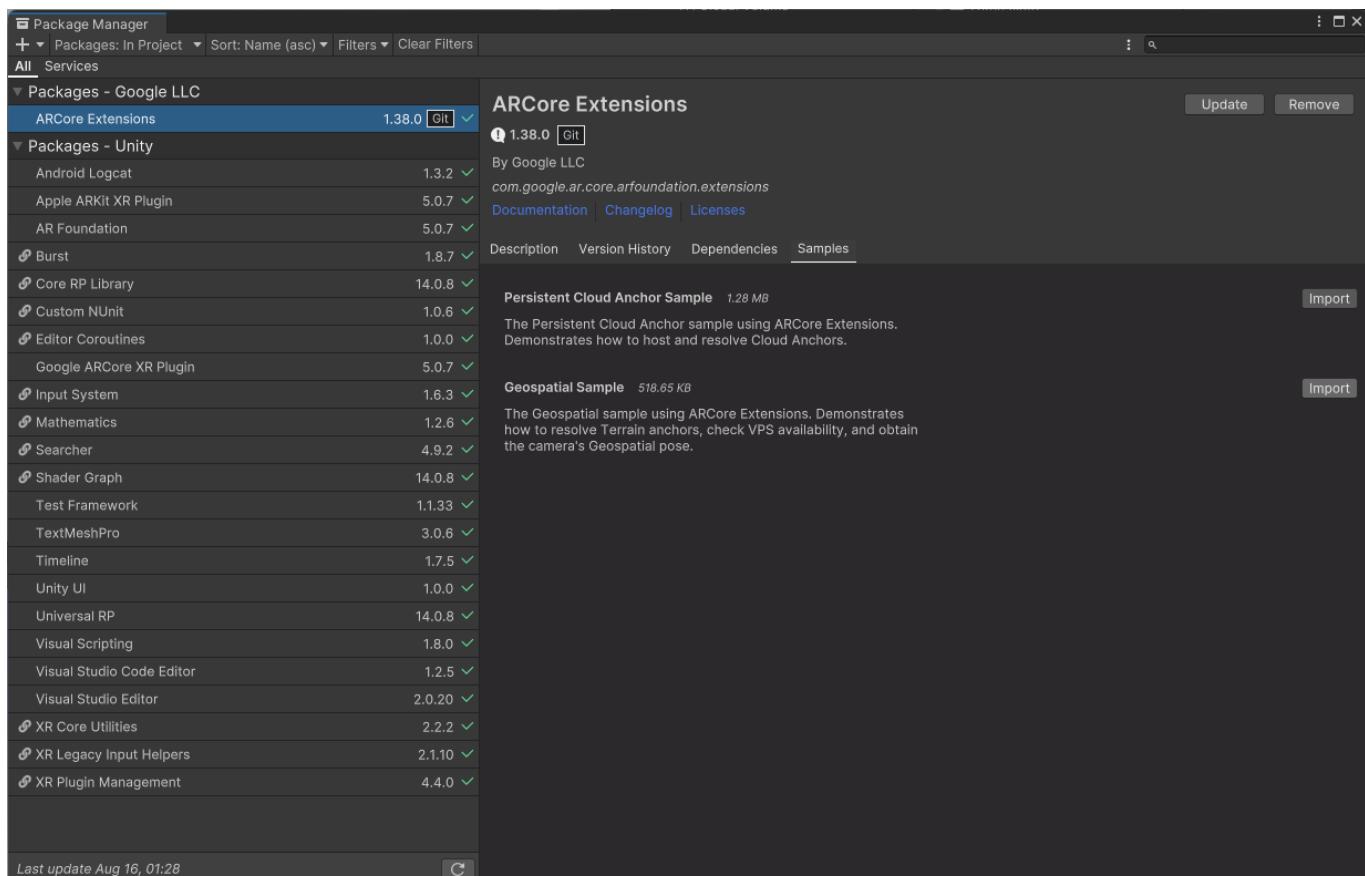
この段階で必要に応じてインストール済みのパッケージのアップデートも行っておくとよいでしょう。

次に、「ARCore Extentions for ARFoundation」をインストールします。このパッケージにGeospatialAPIの機能が含まれます。なお、ARCore Extentionsのドキュメントと、GeospatialAPIのドキュメントも参考にしてください。Package Managerの左上の [+] をクリックし、[Add package from git URL] を選択します。そして次のURLを貼り付け、[Add] をクリックします。

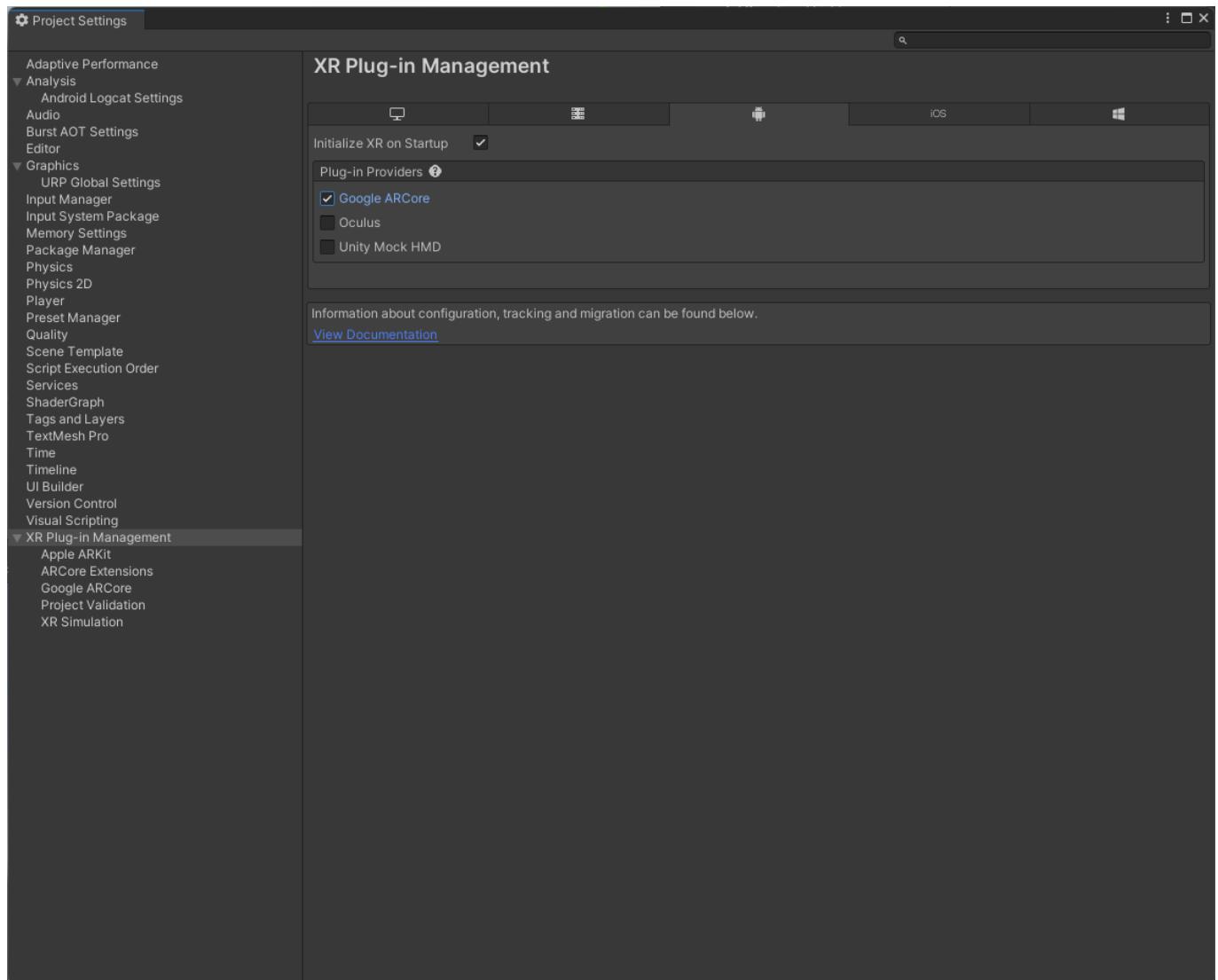
```
https://github.com/google-ar/arcore-unity-extensions.git
```

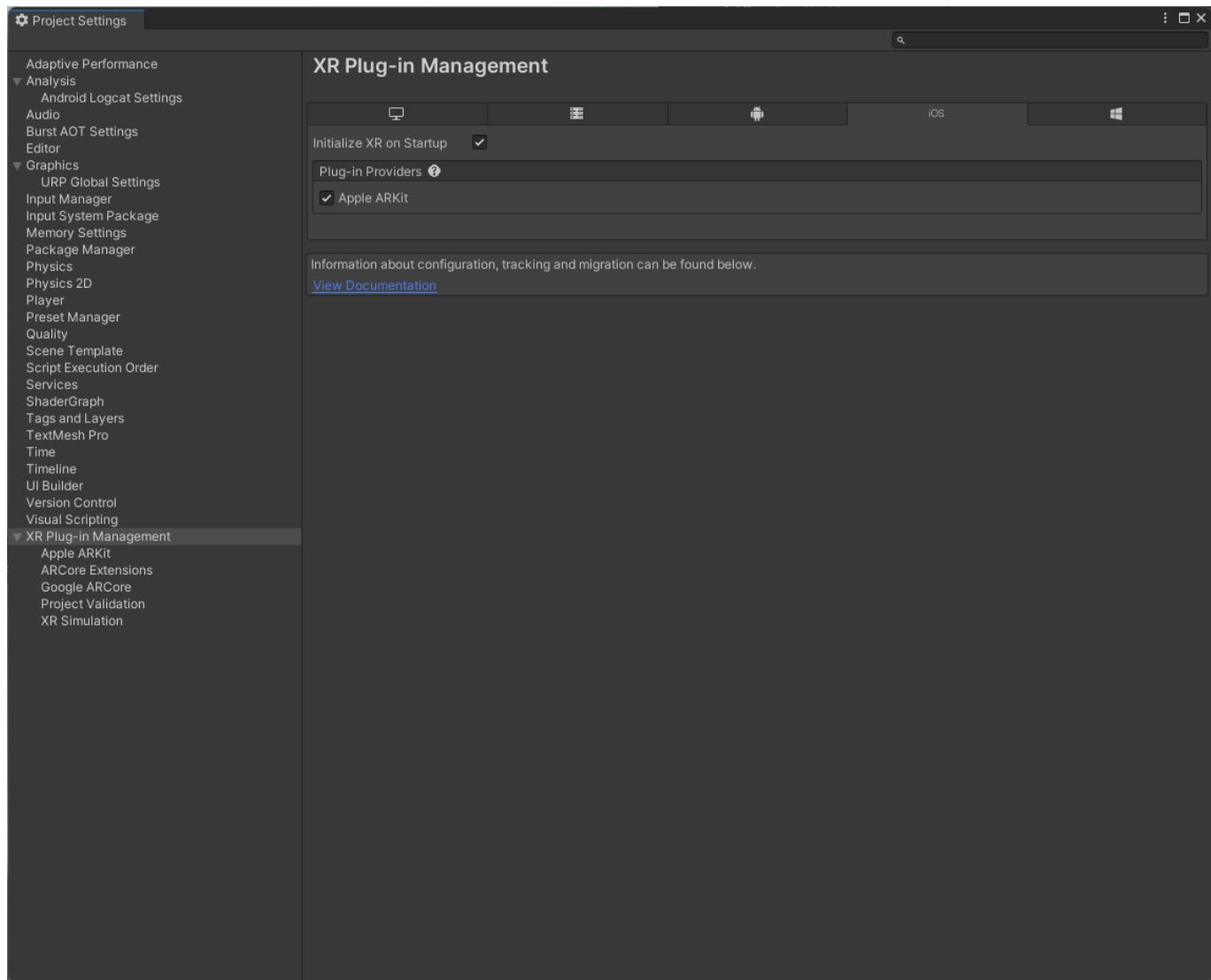


ARCore Extensionsの画面が表示されたら、[Samples] の項目にある [Geospatial Sample] の [Import] をクリックして、このサンプルもインポートしておいてください。

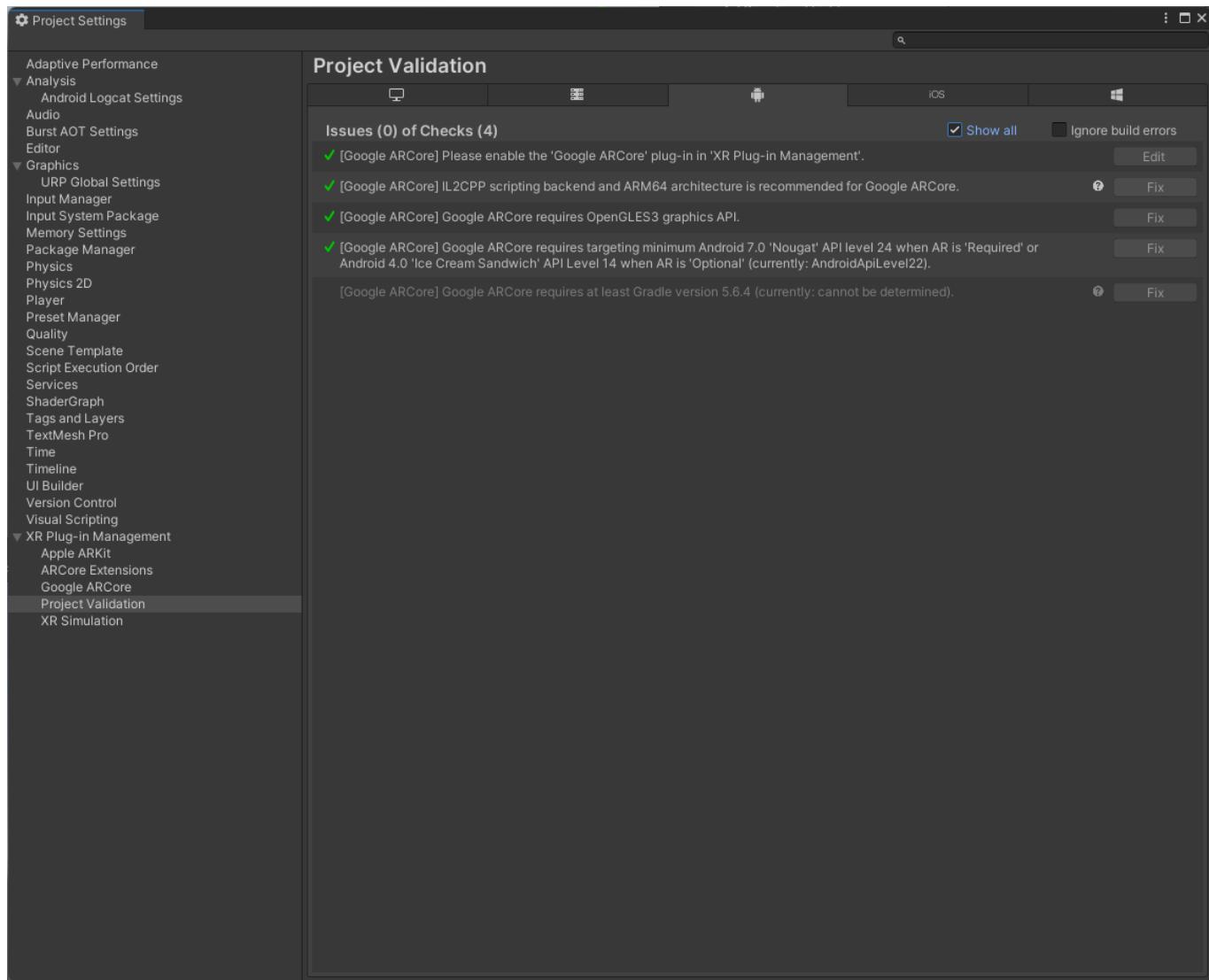


さらに、Unity Editorの[Edit]メニューから[Project Settings]を開き、いくつかの設定を加えます。

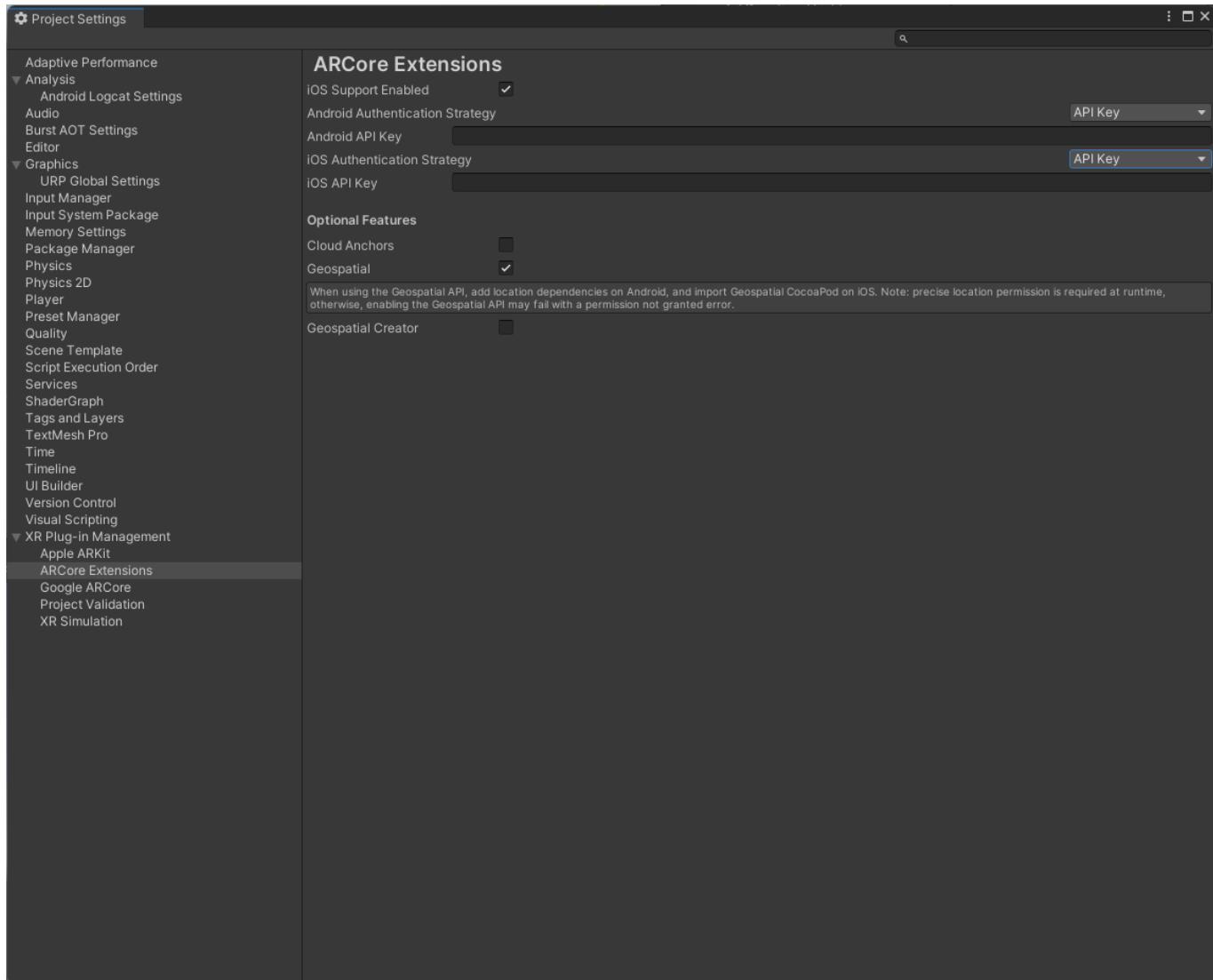




「XR Plug-in Management」の「Project Validation」では、プロジェクトの設定を自動的にARに最適な形に修正してくれます。もし、ここでエラーなどが出ていたら、「Fix All」ボタンを押して修正します。



Geospatial API の API キーは Google Cloud のコンソールで作成し、「XR Plug-in Management」の「ARCore Extentions」で設定してください。詳細な手順は [公式ドキュメント](#) や、PLATEAU 公式サイトのチュートリアルコンテンツの [TOPIC14-3 「Google Geospatial API で位置情報による3D都市モデルのARを作成する」](#) の「■ API キーの作成」の項目を参照してください。また、「iOS Support Enabled」「Geospatial」のチェックもつけておきます。

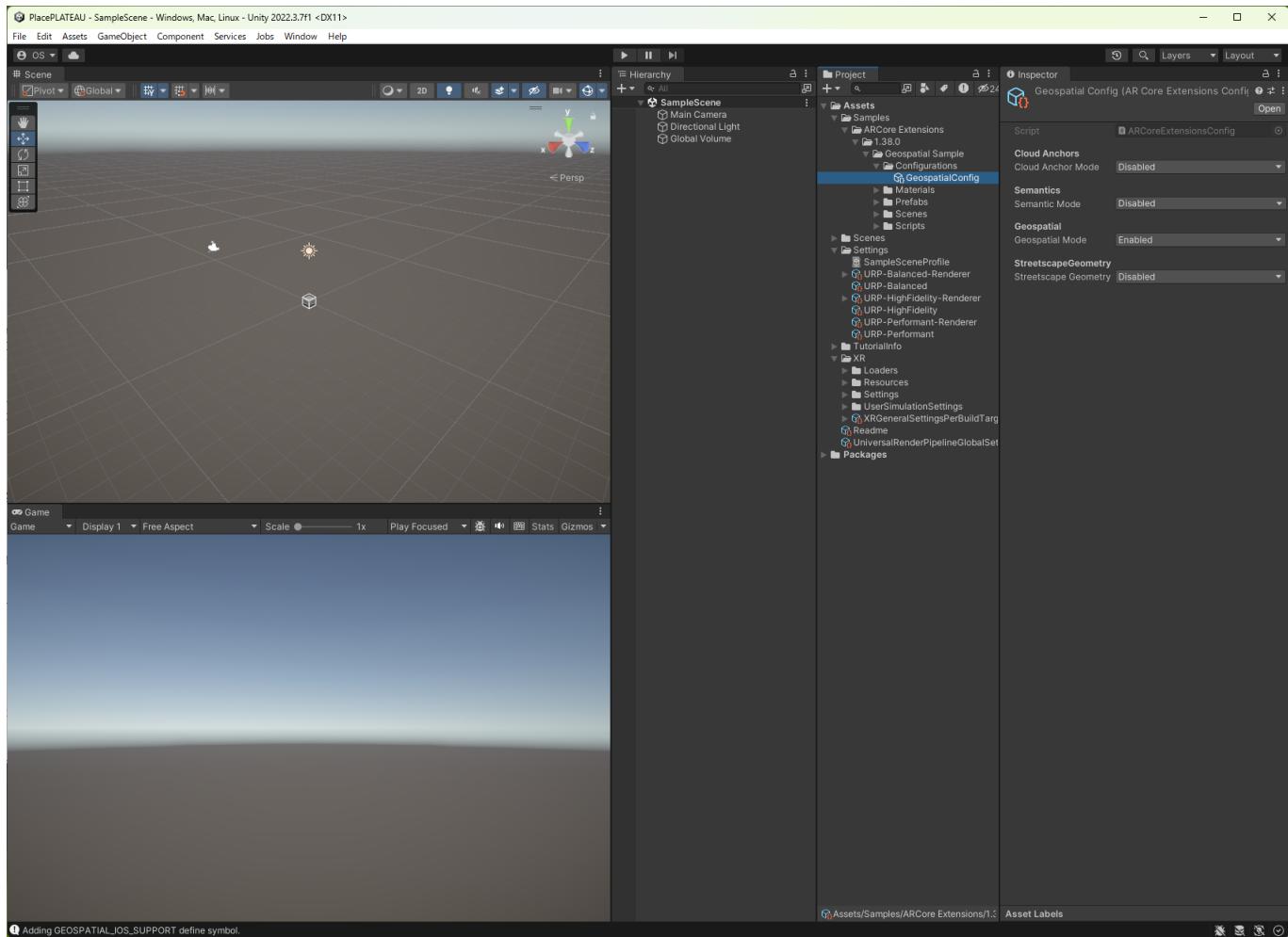


※※※ 注意 ※※※

ここではAPIキーを設定していますが、秘密情報をアプリに組み込むことになるためセキュリティ的には推奨されません。  
後半のアプリをビルドする際により適切な方法の説明をします。  
APIキーのままでアプリを公開しないように気を付けてください。

次に、Unity Editorの「Project」ビューにうつり、「Assets/Samples/ARCore Extentions/1.38.0/Geospatial Samples/Configurations/GeospatialConfig」をクリックします。「Inspector」ビューの「Geospatial」を

「Enabled」にします。



Playerタブで、Android,iOSの各プラットフォームを以下のように設定します。

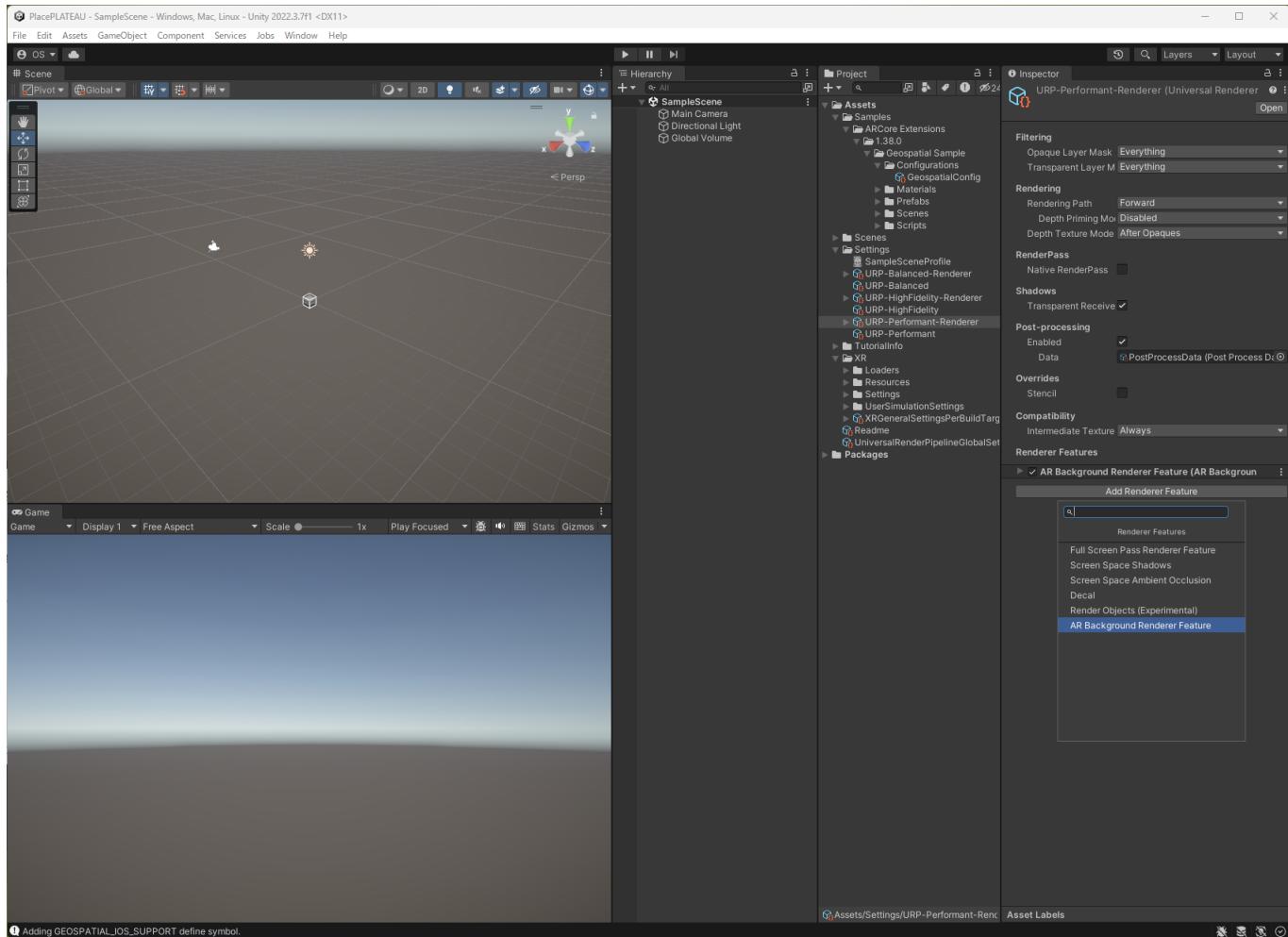
#### 【Androidの場合】

- Minimum API Levelは、28以上が妥当です。
- IL2CPPでビルドします。
- Graphics APIは、OpenGL ES3以上が必要です。（Vulkanは外しておいた方が無難です）
- Target ArchitectureのARMv7のチェックを外す

#### 【iOSの場合】

- サポートするOSは、iOS11以上です。
- ARM64でビルドします。
- 「Require ARKit Support」にチェックを入れます。
- 「Location Usage Description」に、位置情報を使う際にユーザーに通知するメッセージの文字列を入れます。

URPの設定で、Renderer FeaturesにAR Background Renderer Featureが設定されていない場合、追加します。使用するURP設定が分からなければ、すべての設定に追加します。



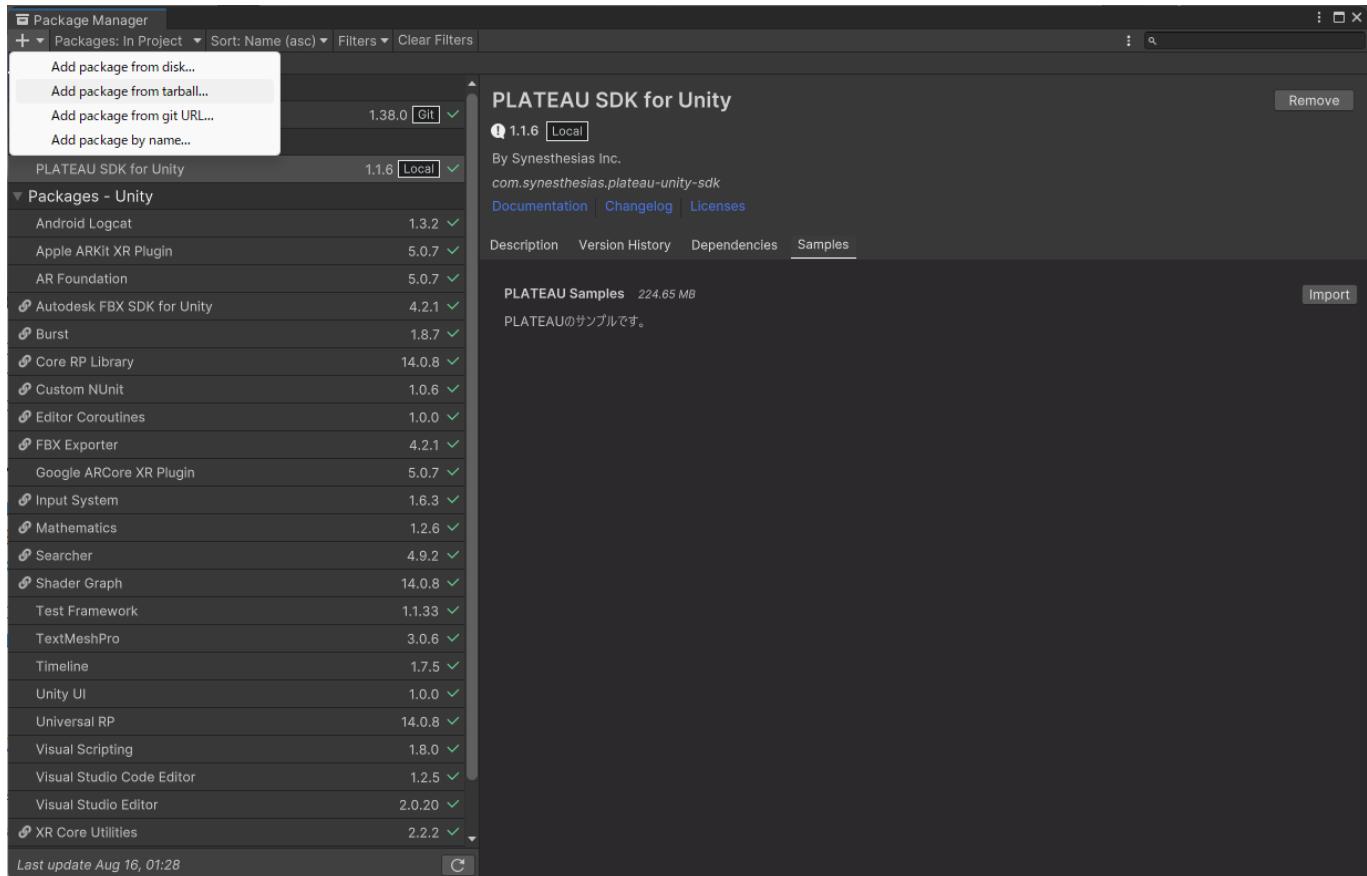
ここまで設定で、Geospatial APIを使ったARアプリの基本の設定が終わりました。一度各プラットフォーム向けに「Geospatial Samples」の「Geospatial」シーンをビルド対象にして動作確認のためにビルドしておくとよいでしょう。

## PLATEAUを読み込む

「PLATEAU SDK for Unity」をインストールし、ゲーム対象地域の3D都市モデルを読み込みます。詳細は、公式の[ドキュメント](#)や、PLATEAU公式サイトのチュートリアルコンテンツの[TOPIC 17 | PLATEAU SDKでの活用\[1/2\] | PLATEAU SDK for Unityを活用する](#)なども参考にしてください。

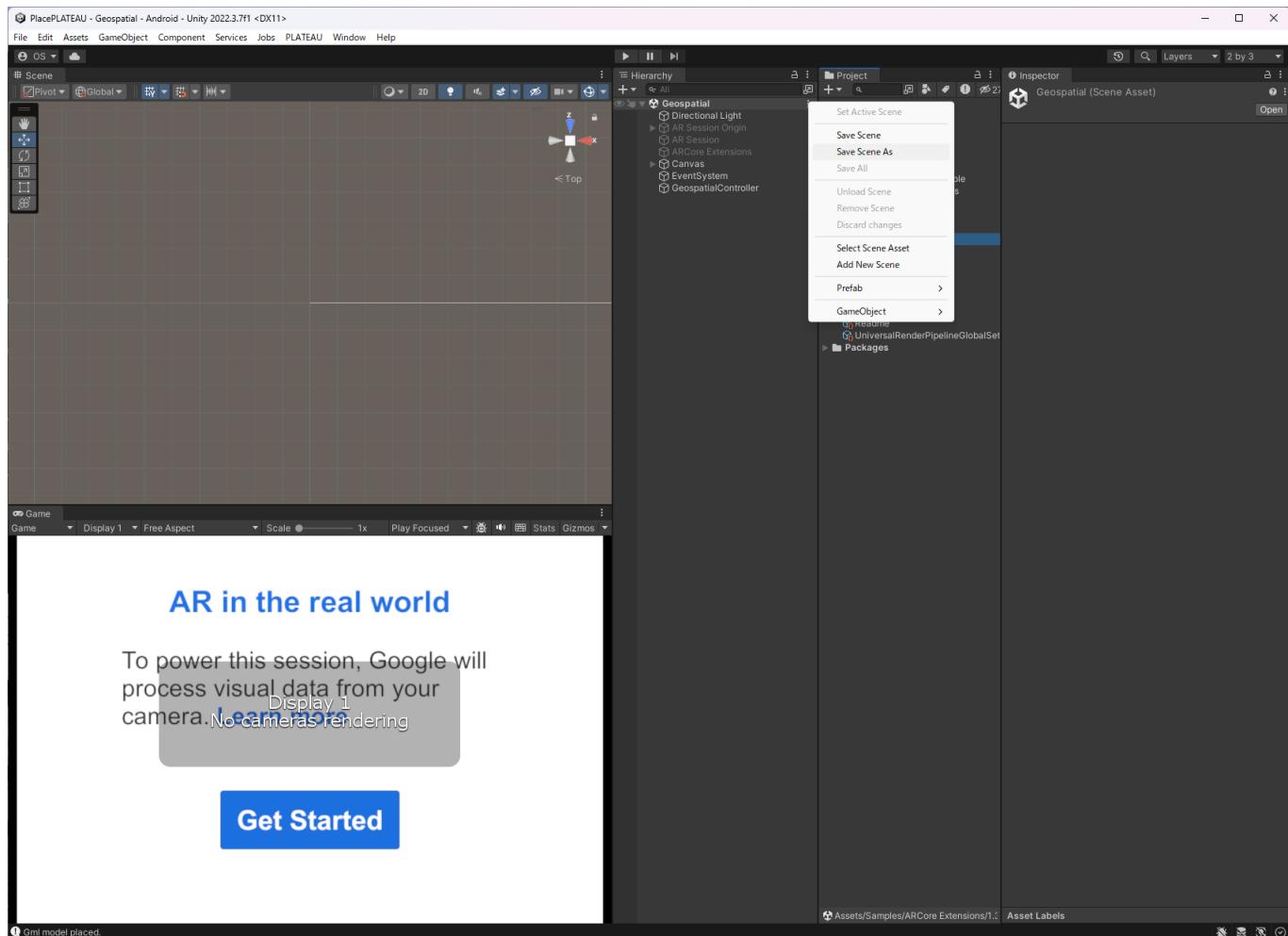
ここでは、GitHubからtarボールでインストールします。「PLATEAU SDK for Unity」のリリースページからtgzファイルをダウンロードします。

Unity Package Managerを開き、左上の+ボタンから「Add Package from tarball...」を選択し、ダウンロードしたtgzファイルを指定してインストールします。

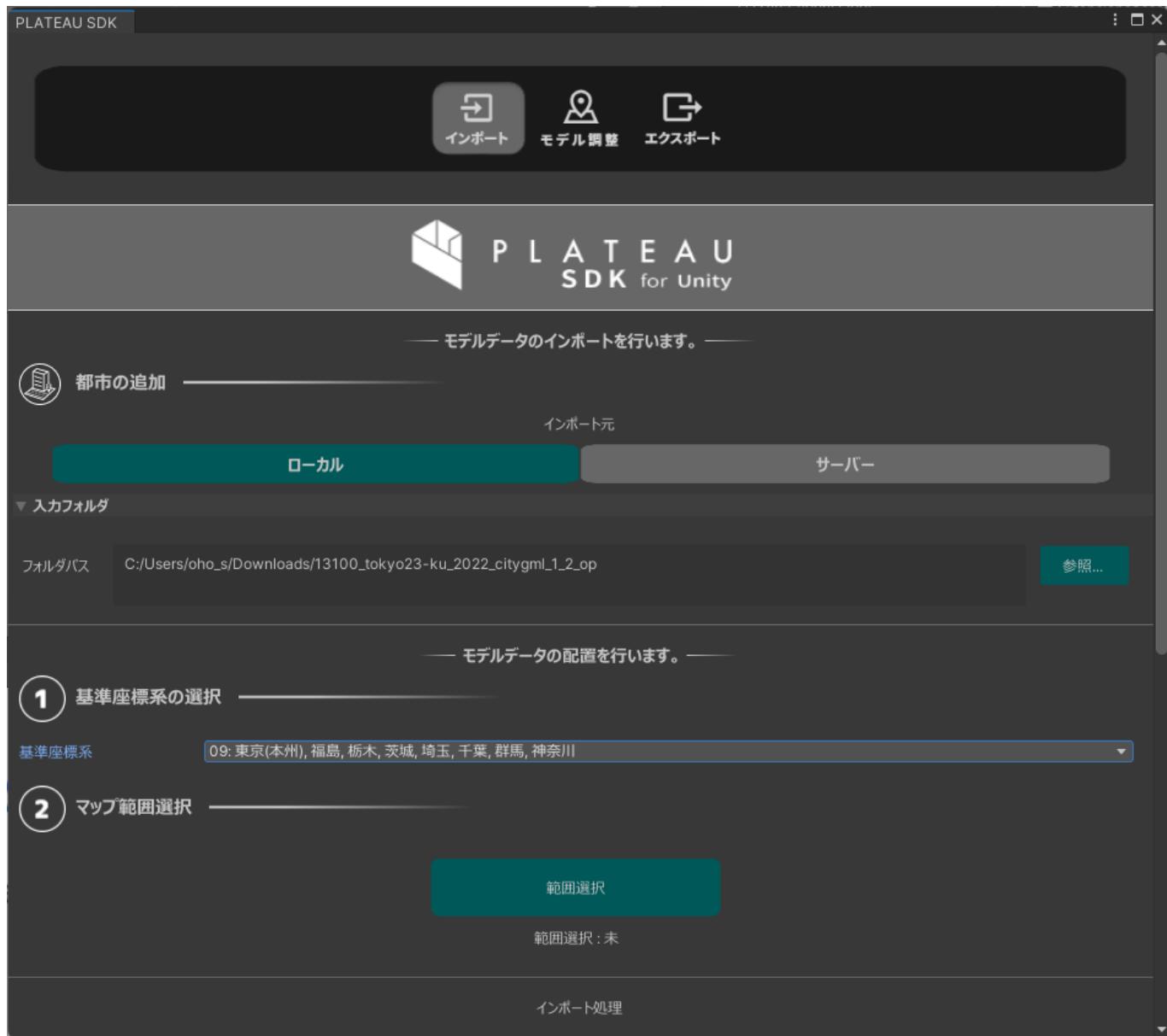


インストールが正常に行われると、Unity Editorのメニューに「PLATEAU」の項目が増えていきます。早速3D都市モデルを読み込んでみます。

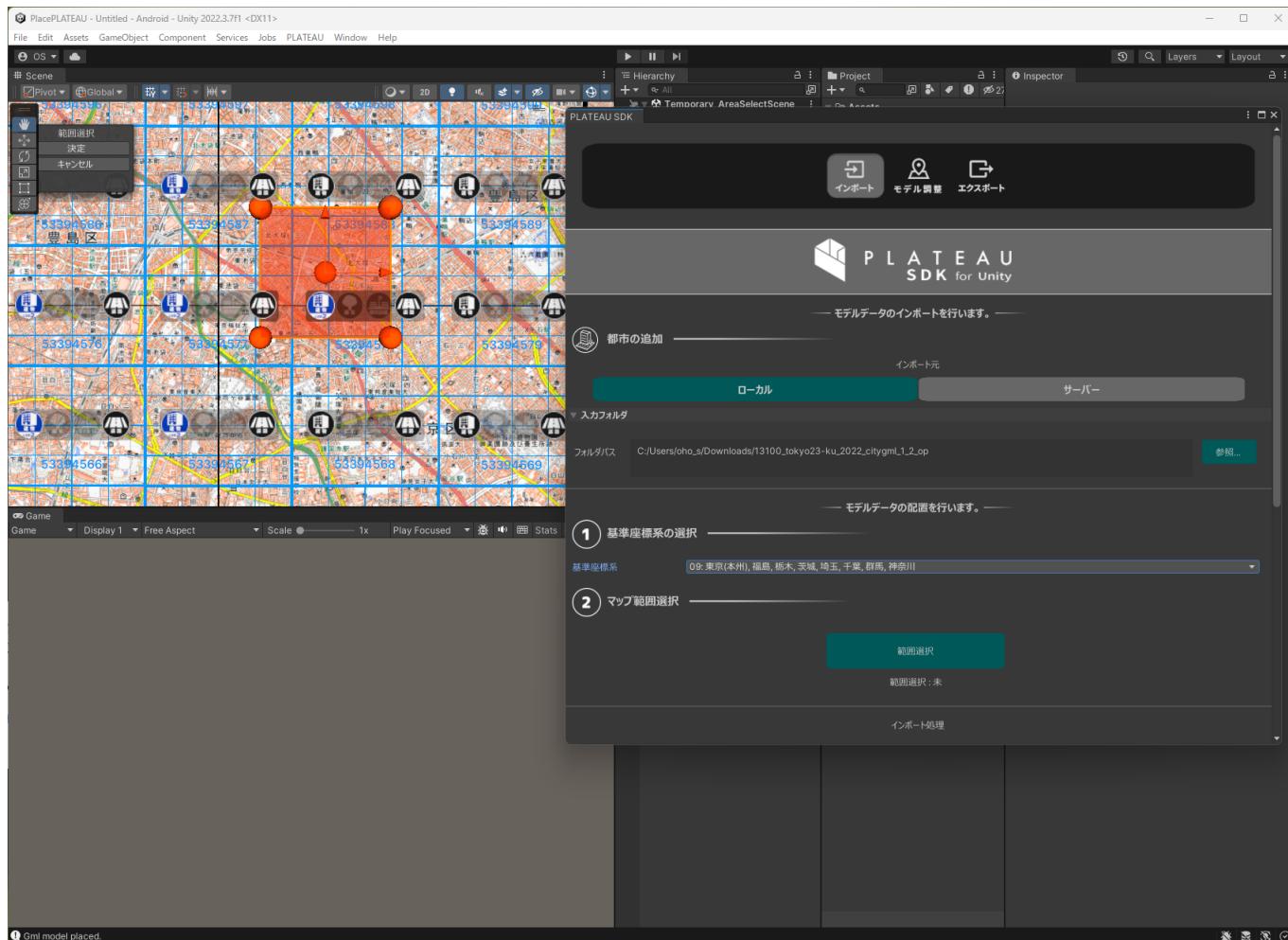
まず、作業用のシーンを用意します。[Assets/Samples/ARCore Extensions/1.38.0/Geospatial Samples/Scenes/Geospatial](#)シーンを開きます。「ヒエラルキー」のシーン名の右からメニューを開くと、シーンを別名で保存する選択肢があるので、[Assets/Scenes](#)などに保存します。（Geospatial Samplesのサンプルシーンをコピーして作業用のシーンとしています。）



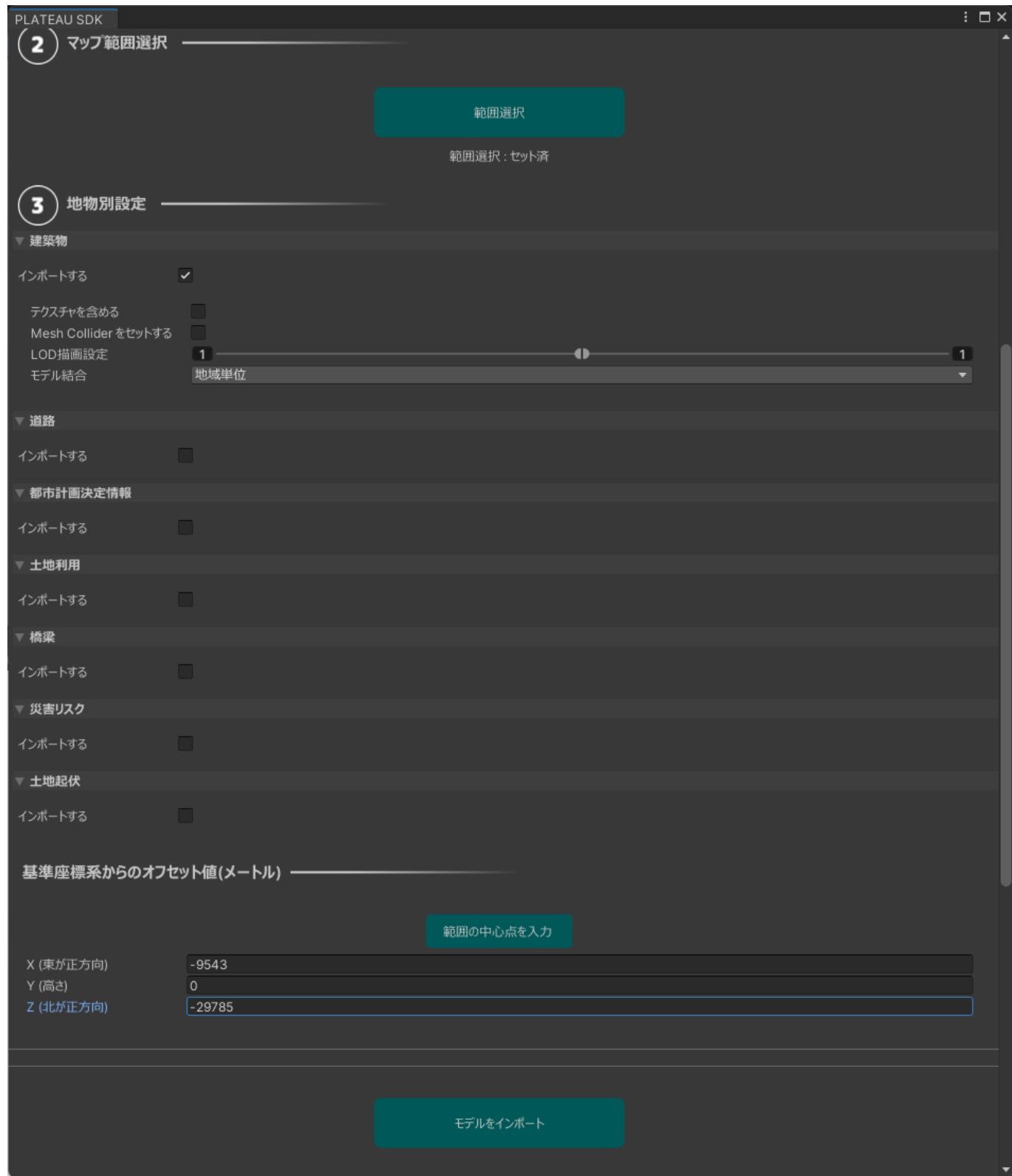
作業用シーンが準備できたら、「PLATEAU」メニューから「PLATEAU SDK」を選択し、SDKのダイアログを表示します。インポートの画面で、ローカルの方であらかじめダウンロードしておいた使いたい地域のPLATEAUの3D都市モデルを展開したフォルダーを選択するか、サーバーの方で使いたい地域を選択します。



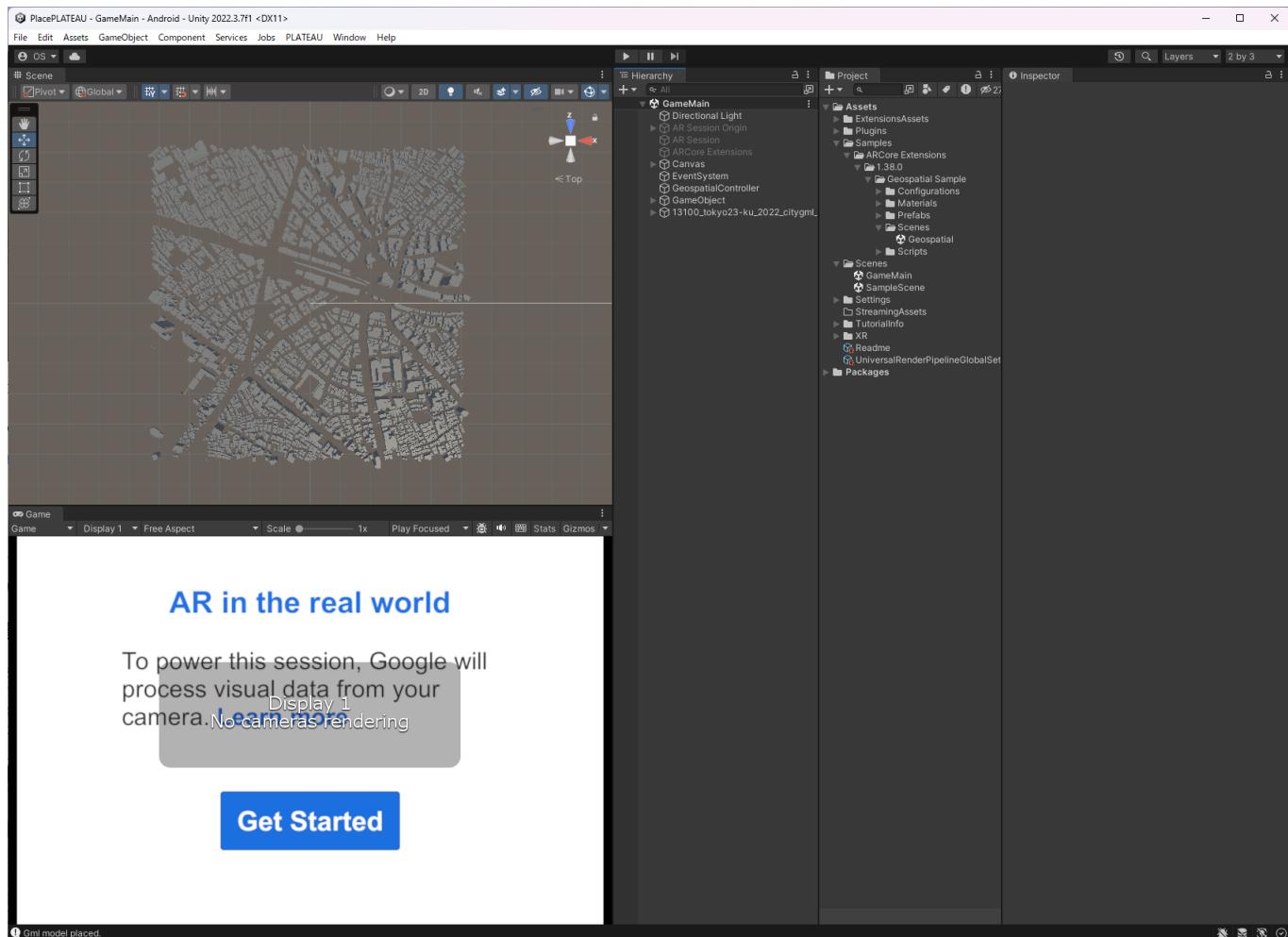
続いて範囲選択を行います。範囲選択ボタンを押すとEditor画面の方で選択UIが起動するので、必要な範囲を指定します。ここでは、あまり広い範囲を指定してしまうと処理負荷がかかるので、気を付けてください。



次に地物別設定をします。今回は、建築物のLOD1のみを「地域単位」で結合してインポートします。建築物以外の「インポートする」のチェックを外します。また、テクスチャは必要ないので「テクスチャを含める」のチェックは外します。これで「モデルをインポート」を押し、処理を開始します。範囲やPCの性能にもよりますが、数分の時間がかかる場合もあります。

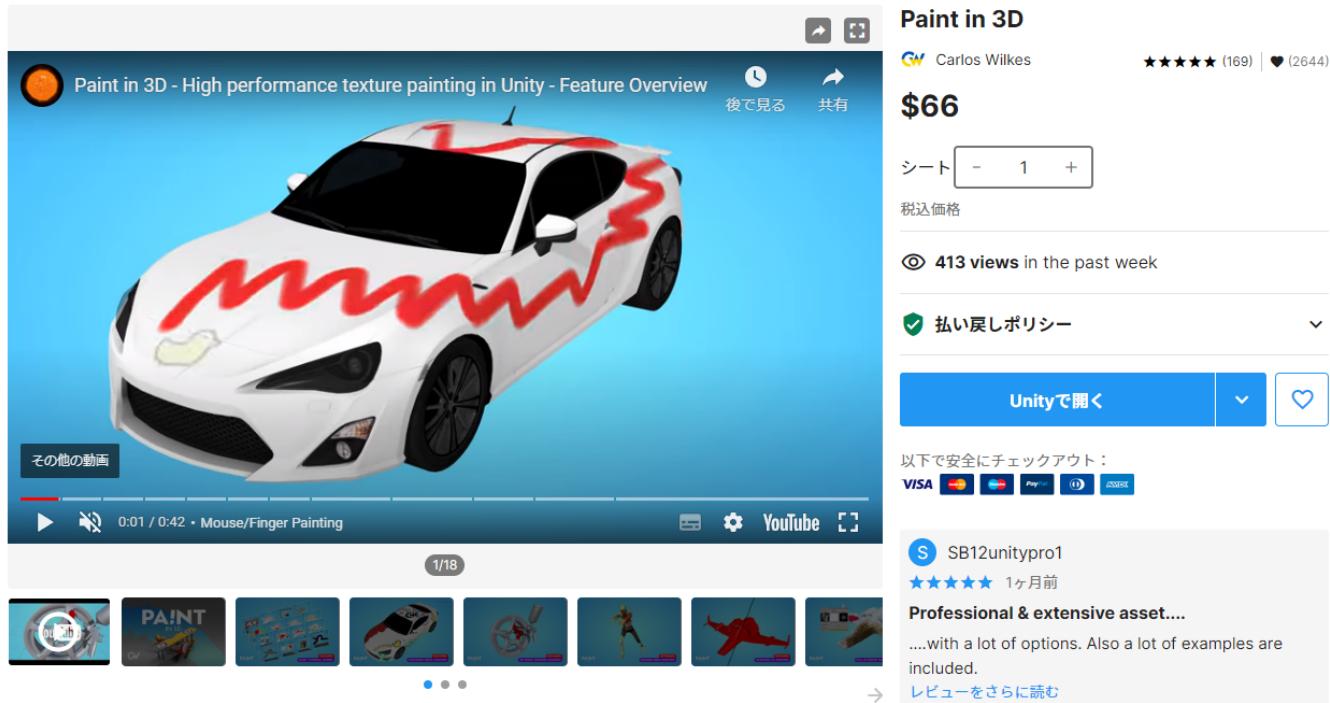


最後に、3D都市モデルがシーンに読み込まれたことを確認します。



## Paint in 3Dを導入し、PLATEAUに印を塗れるようにする

ここまでがゲーム開発の基本となる準備となります。ここからPLATEAUを使ったゲームロジックを実装します。最初に、ARでPLATEAUの3D都市モデルに弾を発射すると、ビルに印が塗られる仕組みを作ってみます。Unity Asset Storeで「Paint in 3D」というアセットを購入(\$66)し使用します。このアセットは、3Dのオブジェクトにお絵描きができるものです。購入したら通常のアセットの導入と同様にUnity Package Managerから導入してください。

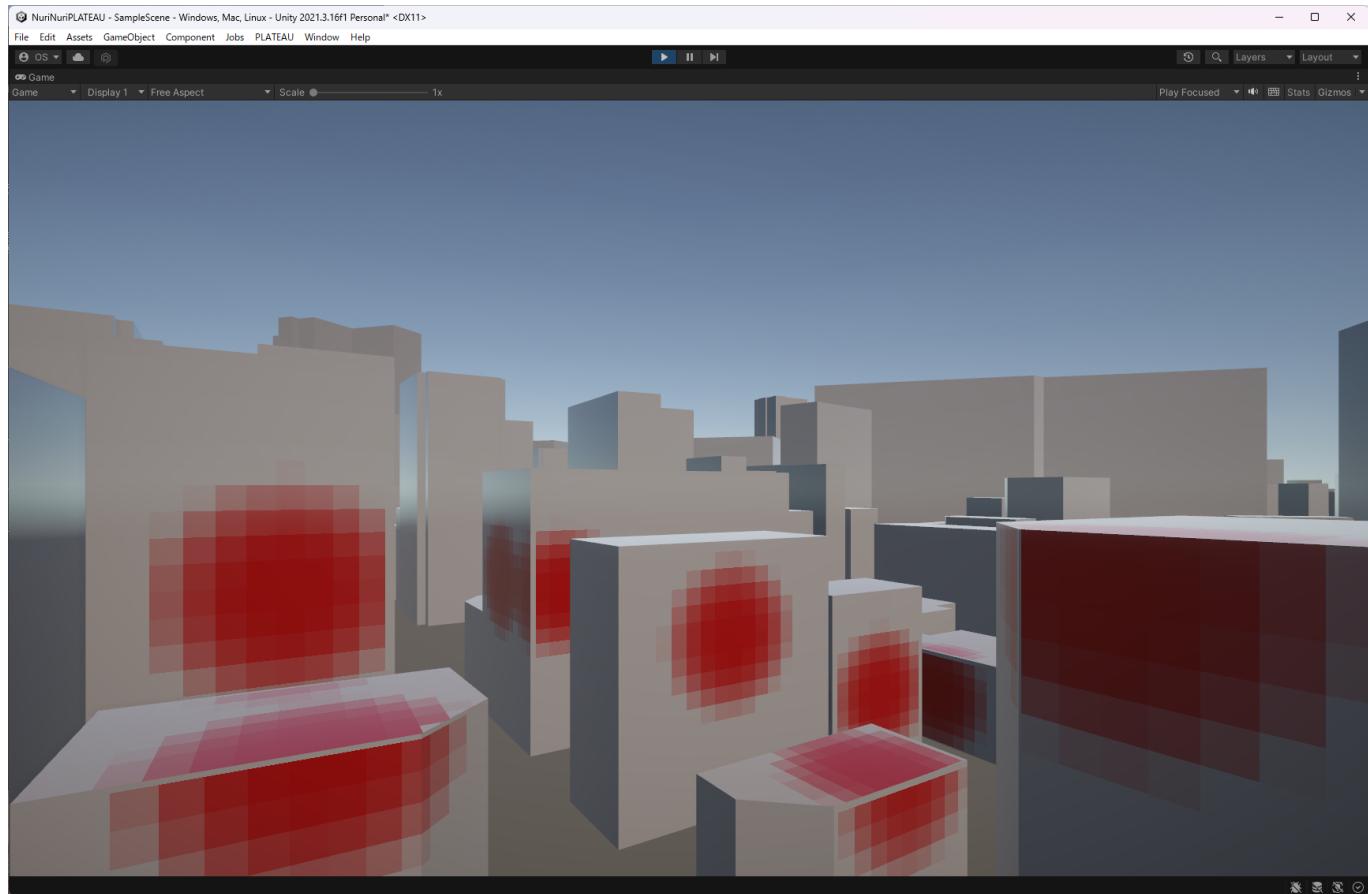


※※※ ヒント ※※※

ここでは、有料アセットを使用しましたが、例えばコライダーとレイキャストの機能を使って、タップ位置からの当たり判定を計算し、そこにオブジェクトを表示するなどでも近いことはできると思います。

興味のある人はチャレンジしてみてください。

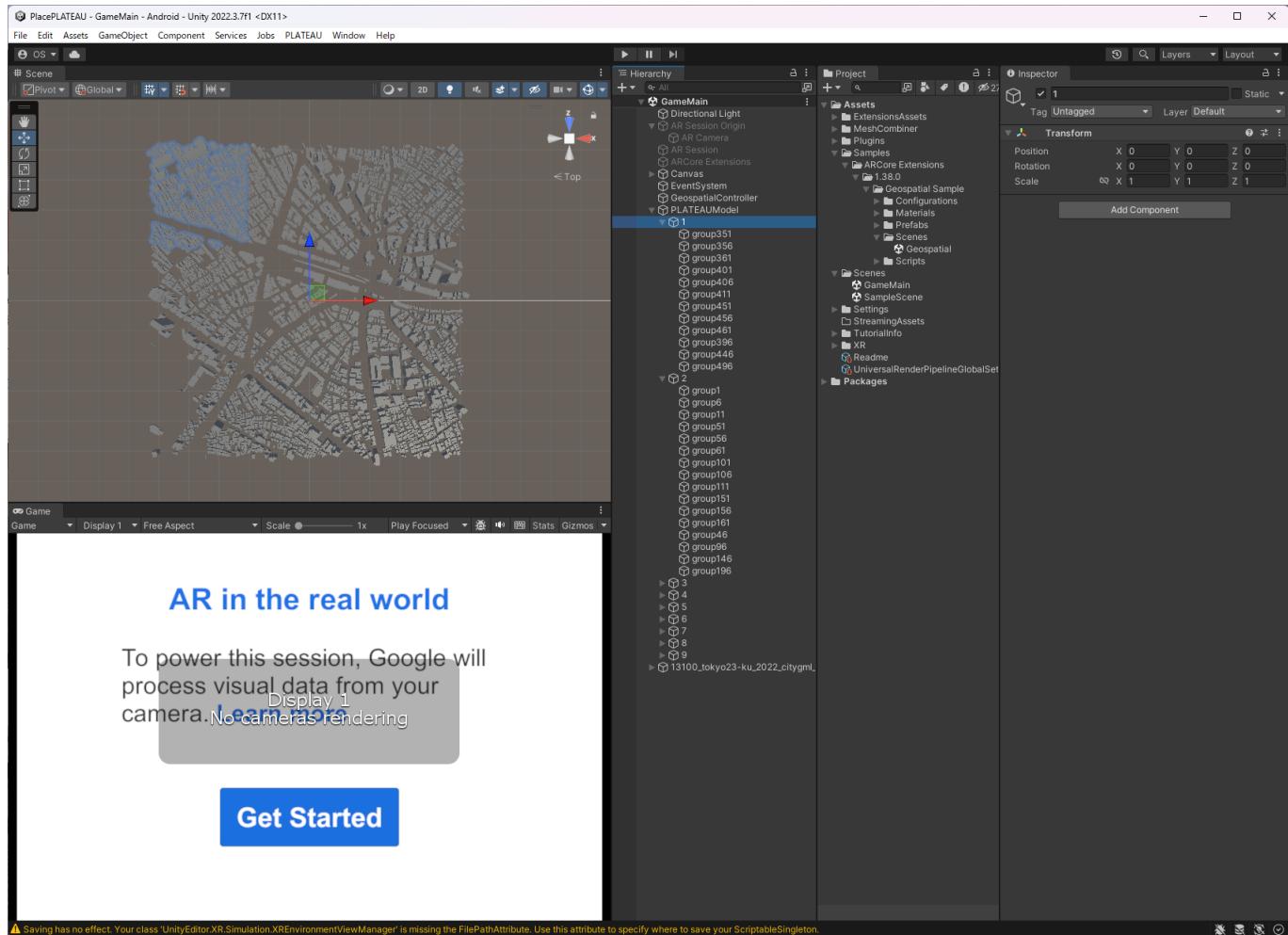
さて、読み込んだままのPLATEAUの3D都市モデルだと、Paint in 3Dで使うためには不便です。Paint in 3Dは、設定したテクスチャ画像に描画することで3Dモデルに描画しているように見せる仕組みです。そのため、広範囲の3Dモデルを結合した巨大なモデルを対象とした場合、テクスチャの解像度が足りずジヤギーが発生したりモザイクのようになってしまいます。一方で、3Dモデルを細かく分割しそれぞれに高解像度のテクスチャを貼ると、描画が非常に高負荷になってしまいます。



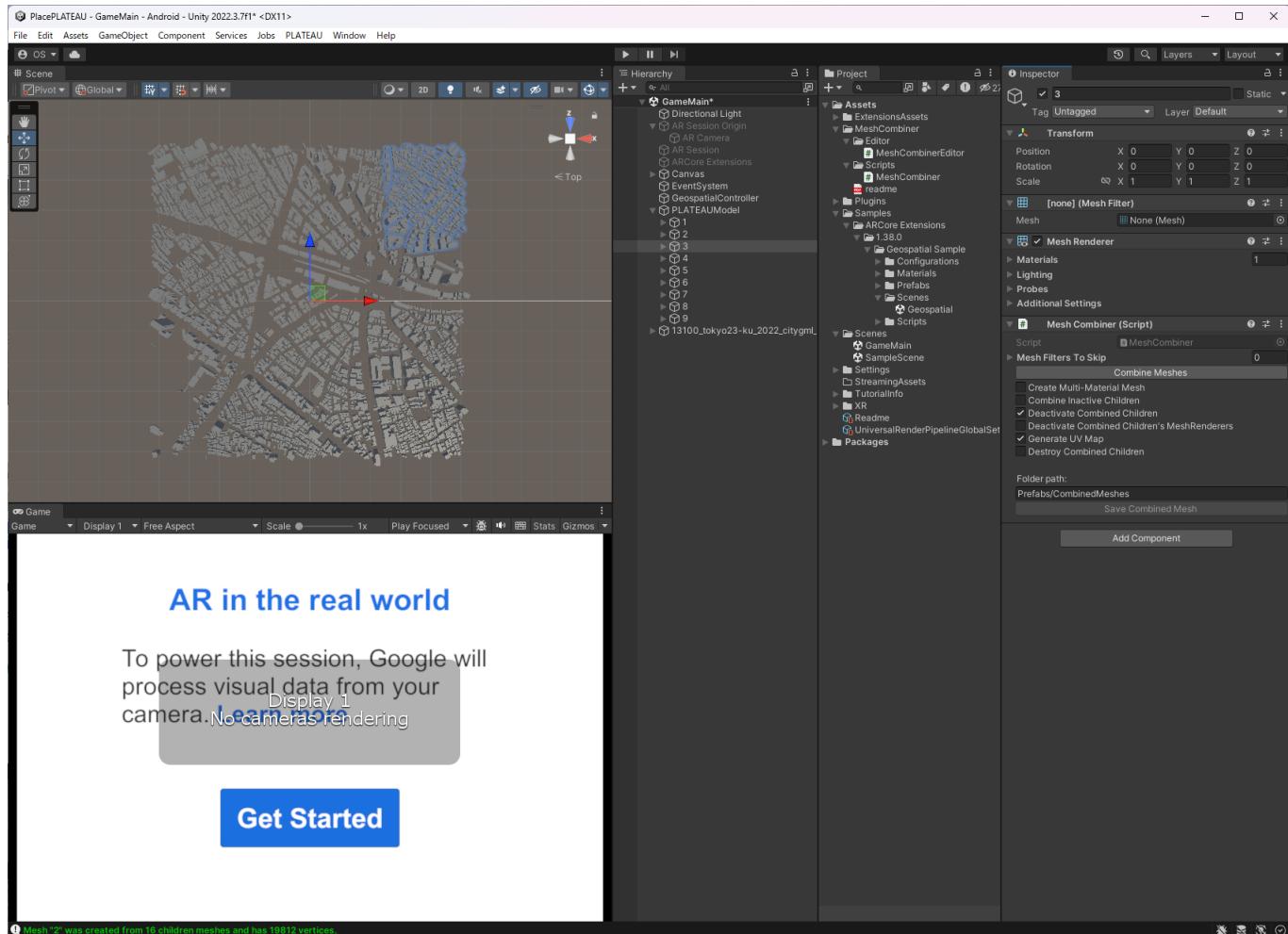
ここでは、読み込んだPLATEAUの3Dモデルを9分割して [「Mesh Combiner」](#) というアセットで結合することで、テクスチャの品質と描画負荷のバランスのよい設定にします。

Mesh CombinerはAsset Storeで入手し、プロジェクトにインストールしておきます。

最初に、Editor上で読み込んだ3D都市モデルをマウスで選択しながらまとめるための空のGameObjectの子にしていき、縦横それぞれ3分割した9つのGameObjectに分けた階層構造にします。



Assets/MeshCombiner/Scripts/MeshCombiner.csを、9分割したそれぞれのGameObjectにD&Dでアタッチします。「Deactivate Combined Children」と「Generate UV Map」にチェックを入れて、「Combine Meshes」を押すと。メッシュが結合されます。正しく結合されていることが分かったら、子オブジェクトは消しても問題ありません。



ただ、PLATEAUの3D都市モデルは巨大なので、どうしてもテクスチャの解像度を上げるのが難しい場合があります。そこで、表現を工夫することで、

PLATEAUモデルの準備 Mesh Combiner UV1の設定 マテリアル Decalで塗るのを作成

塗った場所の位置座標を計算する

GeospatialでWorld座標から緯度経度にする

サーバーに位置情報を送る

位置情報を送る先のサーバーを作ります。機能としては、HTTPでJSONにした位置情報をクライアントとやり取りするシンプルなAPIサーバーです。サーバーでは位置情報を受信すると、それをPostGISの空間情報データとしてデータベースに格納し、空間クエリで面積を計算してクライアントに結果を返します。

サーバーにはさまざまな技術がありますが、ここではフレームワークにはPythonのFlaskを使い、PostgreSQL+PostGISをバックエンドデータベースとして使います。また、動作させるインフラとして、AWSを使うこととします。

これらの構成はあくまでもこのサンプルでの説明用で、同じようなことをやる場合のさまざまな選択肢があります。今回はなるべく汎用的かつ基本的で、読者が自分の環境に読み替えて考えやすい作り方を目指しました。言語もフレームワークもクラウドもデータベース多くの選択肢があるので、自分の慣れているものに読み替えてください。

また、HTTPでやり取りするAPIは、リアルタイムでの同期には向いていません。FirebaseなどのMBaaSや、Photonなどのリアルタイム通信基盤など、まったく別の考え方の選択肢もあります。ここでは詳細を説明しませんが、作りたいサービスに合わせて選択してください。

## サーバーの準備

AWSでEC2インスタンスを立ち上げます。AWSを使うにはアカウントの登録が必要です。今回は無料枠の範囲内でできるように構成していますが、設定のミスなどで予想外の金額が請求されることもあります。アカウントの2段階認証を設定するなど、セキュリティにも十分気を付けてください。また、本項はある程度AWSを使い慣れている前提で進めます。分からぬことがある場合はAWSのドキュメントなどを参照してください。

また、内容の中にセキュリティの側面などで本番環境で動作させるには適さない部分があります。本項のアプリはあくまでも解説用のサンプルということで、実際のサービスなどを構築していく際には十分考慮してください。

EC2インスタンスはいわゆるクラウドの仮想マシンです。クラウドの使い方としては旧式な物ですが、なるべく基本的なところから説明して分かりやすくという方向で説明します。サーバーレスなど最近のモダンな構成を試したい方は是非チャレンジしてみてください。

OSは、著者が使い慣れていることもあります、Ubuntu 22.04を使います。インスタンスタイプは現時点ではt3a.nanoが単価が安いのでこちらを使っていきます。（※注：無料枠の範囲でやりたい人はt2.microを選択してください）キーペアなどは適切に設定してください。セキュリティグループは、SSHとHTTPSが通るように設定してください。テスト用にHTTPが通ってもよいと思いますが、スマホアプリからのアクセスではHTTPSが必須である場合が大半なので基本はHTTPSを使います。（※注：本番環境ではHTTPは外にさらさない方がいいと思います。）

EC2 > インスタンス > インスタンスを起動

## インスタンスを起動 情報

Amazon EC2 では、AWS クラウドで実行される仮想マシン（インスタンス）を作成できます。以下の簡単なステップに従ってすばやく開始できます。

**名前とタグ** 情報

名前  [さらにタグを追加](#)

**▼ アプリケーションおよび OS イメージ (Amazon マシンイメージ)** 情報

AMI は、インスタンスの起動に必要なソフトウェア設定（オペレーティングシステム、アプリケーションサーバー、アプリケーション）を含むテンプレートです。お探しのものが以下に表示されない場合は、AMI を検索または参照してください。

最新 | 自分の AMI | **クリックスタート**

Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Linux | [その他 AMI を閲覧する](#)

**Amazon マシンイメージ (AMI)**

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type  
ami-0d52744d6551d851e (64 ビット (x86)) / ami-0342c9aa06b2a6488 (64 ビット (Arm))  
仮想化: hvm ENA 有効: true ルートデバイスタイプ: ebs

説明  
Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2023-05-16

**概要**

インスタンス数 情報  
1

ソフトウェアイメージ (AMI)  
Canonical, Ubuntu, 22.04 LTS, ...[続きを読む](#)  
ami-0d52744d6551d851e

垂直サーバータイプ (インスタンスタイプ)  
t3a.nano

ファイアウォール (セキュリティグループ)  
SSH&Web

ストレージ (ボリューム)  
1ボリューム - 8 GiB

**① 無料利用枠:** 最初の 1 年には、1 か月あたりの無料利用枠による AMI での t2.micro (または t2.micro が利用できないリージョンでは t3.micro) インスタンスの 750 時間の使用、30 GiB の EBS ストレージ、200 万の IOs、1 GiB のスナップショット、インターネットへの 100 GiB の帯域幅が含まれます。

[キャンセル](#) **インスタンスを起動** [コマンドを確認](#)

SSHで接続し環境構築します。最初に、PostgreSQLとその地理空間拡張のPostGISをインストールします。ついでにシステムのアップデートも一緒にしておきましょう。

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install postgresql postgis
```

まずログインできるようにパスワードを設定します。

```
$ sudo -u postgres psql -c "ALTER USER postgres WITH PASSWORD 'passwd';"
```

このままだと、PostgreSQLのPeer認証になっているので、postgresユーザーしかアクセスできません。この後の作業に不便なので、Peer認証を変更します。以下のコマンドでPostgreSQLの設定ファイルを開きます。（エディターはvimでもEmacsでもお好みのものを使ってください。例ではnanoを使います。）

```
$ sudo nano /etc/postgresql/14/main/pg_hba.conf
```

```
# Database administrative login by Unix domain socket
local  all          postgres          peer
```

この行を

```
# Database administrative login by Unix domain socket
local  all          postgres          scram-sha-256
```

このように変えて保存します。

```
$ sudo service postgresql restart
```

PostgreSQLのサービスを再起動します。

次のように `psql` コマンドを `ubuntu` ユーザーで実行し、さきほど設定したパスワードで PostgreSQL のプロンプトに入れたら準備完了です。 (※注: 本来はデータベース操作用のユーザーを作成しますが、簡単のためこのような設定で進めます。)

```
$ psql -U postgres
Password for user postgres:
psql (14.9 (Ubuntu 14.9-0ubuntu0.22.04.1))
Type "help" for help.

postgres=#
```

データベースの準備

次にデータベースを作成します。データベースの名前は、 `placeplateau` にします。

```
postgres=# create database placeplateau;
postgres=# \c placeplateau
```

PostGISを有効にします。

```
placeplateau=# CREATE EXTENSION postgis; CREATE EXTENSION postgis_topology;
```

アプリ用のテーブルを作ります。

```
placeplateau=# create table placedata (
  id SERIAL PRIMARY KEY,
```

```
userid VARCHAR(255) NOT NULL,  
side INTEGER NOT NULL,  
created_at TIMESTAMP NOT NULL,  
geom GEOMETRY(POLYGON,4326) NOT NULL,  
area REAL NOT NULL);
```

## Webアプリケーションプログラムの作成

Python3とその周辺環境をインストールします。wsgi経由でnginxをWebサーバーに使います。

```
$ sudo apt install python3 python3-pip python3-venv nginx
```

Pythonのvenvを設定しFlaskと必要なライブラリをインストールします。

```
$ sudo apt install libpq-dev  
$ mkdir placeplateau_web  
$ cd placeplateau_web/  
$ pwd  
/home/ubuntu/placeplateau_web  
$ python3 -m venv venv  
$ source venv/bin/activate  
(venv) $ pip install Flask uwsgi geoalchemy2 flask_sqlalchemy psycopg2  
geoalchemy2[shapely]
```

次のPythonプログラムをエディタなどで作成します。

```
from flask import Flask  
from flask_sqlalchemy import SQLAlchemy  
from geoalchemy2 import Geometry  
from geoalchemy2.shape import to_shape  
from datetime import datetime, date  
import json  
  
app = Flask(__name__)  
app.config['SQLALCHEMY_DATABASE_URI'] =  
'postgresql://postgres:passwd@localhost/placeplateau'  
db = SQLAlchemy(app)  
  
class PlaceData(db.Model):  
    __tablename__ = 'placedata'  
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)  
    userid = db.Column(db.String)  
    side = db.Column(db.Integer)  
    created_at = db.Column(db.Time)  
    geom = db.Column(Geometry('POLYGON'))  
    area = db.Column(db.Float(5, False, 3))
```

```
def getdict(self):
    return {"id":self.id,
            "userid":self.userid,
            "side":self.side,
            "created_at":str(self.created_at),
            "geom":to_shape(self.geom).wkt,
            "area":self.area}

@app.route('/')
def hello():
    return 'PlacePLATEAU API SERVER'

@app.route('/getarea')
def getArea():
    today = date.today()
    placedatas = db.session.query(PlaceData).where(PlaceData.created_at >=
today).order_by(PlaceData.id)
    datas = []
    for placedata in placedatas:
        print(placedata.id)
        datas.append(placedata.getdict())
    return json.dumps(datas)

@app.route('/makearea', methods=['POST'])
def makeArea():
    data = request.json

    lastid=data['lastid']
    newarea=data['newarea']

    placedatas = db.session.query(PlaceData).where(PlaceData.created_at >=
today).order_by(PlaceData.id)
    datas = []
    for placedata in placedatas:
        print(placedata.id)
        datas.append(placedata.getdict())
    return json.dumps(datas)

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

メモ ドメインの確保どうしようか...例として進める分には自分が持ってるドメインを使えばいいけど... 無料ではできんよなあ。無料のDDNSとかでやれるかなあ？

```
sudo -u postgres psql
```

```
postgres=# create database nurinuriplateau; CREATE DATABASE
postgres=# \c nurinuriplateau
You are now connected to database "nurinuriplateau" as user "postgres".
nurinuriplateau=# CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION CREATE EXTENSION nurinuriplateau=
nurinuriplateau=# \q
```

```
postgres=# postres=# \c nurinuriplateau You are now connected to database "nurinuriplateau" as user "postres". nurinuriplateau=# create table main ( nurinuriplateau(# id SERIAL PRIMARY KEY, nurinuriplateau(# geom GEOMETRY(POLYGON,4326), nurinuriplateau(# area REAL); CREATE TABLE nurinuriplateau=# INSERT INTO main (geom, area) VALUES ( nurinuriplateau(# ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0)'), 4326), nurinuriplateau(# ST_Area(ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0)'), 4326)); INSERT 0 1 nurinuriplateau=# SELECT * FROM main; nurinuriplateau=# \q ubuntu@ip-172-31-17-190:~$ sudo -u postres psql could not change directory to "/home/ubuntu": Permission denied psql (14.8 (Ubuntu 14.8-0ubuntu0.22.04.1)) Type "help" for help.
```

```
postgres=# \c nurinuriplateau You are now connected to database "nurinuriplateau" as user "postres". nurinuriplateau=# SELECT * FROM main; nurinuriplateau=# INSERT INTO main (geom, area) VALUES ( ST_GeomFromText('POLYGON((0 0, 0 1.1, 1.1 1.1, 1.1 0, 0 0)'), 4326), ST_Area(ST_GeomFromText('POLYGON((0 0, 0 1.1, 1.1 1.1, 1.1 0, 0 0)'), 4326)); INSERT 0 1 nurinuriplateau=# SELECT * FROM main;
```

---

```
create table placedata ( id SERIAL PRIMARY KEY, userid VARCHAR(255) NOT NULL, side INTEGER NOT NULL, created_at TIMESTAMP NOT NULL, geom GEOMETRY(POLYGON,4326) NOT NULL, area REAL NOT NULL);
```

```
nurinuriplateau=> SELECT ST_Area(ST_GeomFromText('POLYGON((0 0, 0 1.1, 1.1 1.1, 1.1 0, 0 0)'),4326)::geography); st_area
```

14893547154.626783 (1 row)

Python・Flaskで作るかな 送信可能なAPIをまず作る デプロイ先はAWSにしよう スキーマの決定

## サーバーへの位置情報の送信

UnityWebRequest APIに合わせてリクエストループを閉じたときにまとめてリクエスト ユーザーが受け取っている最後のIDも送る DBに、ID (AutoInc) 、時刻、ユーザーのUUID、Geomを格納 ユーザーが受け取っていないIDの時刻内のGeomと面積の集計を返却する トランザクション忘れるな

## サーバーでの面積の計算

PostGISのクエリでSQLで計算してみる SELECT でArea計算

## サーバーで塗った場所の表示

これはQGISで直接DBにアクセスしてやろうかな それが一番楽そう

## 正確な面積計算のプログラム

データ分析の文脈でこの辺をQGIS ?あたりを使いながら説明

## コラム チート対策

地理情報を扱ううえでのチート対策など

## コラム プライバシーについて

地理情報は個人情報であることの説明

## アプリとしてリリースする

Google Playに出す

AppStoreに出す

///// 以下メモ

## 位置情報を複数人で共有する

サーバーと連携する仕組みの作成 位置情報を扱うときに気にすること 座標系 精度 データベース 格納方法  
単なるDoubleか、空間データ構造か インデックス

## 池袋塗りつぶしゲームを作る

チート対策 データ分析 どんな分析ができる？ 分析のヒント オクルージョン

## アプリとしてリリースする

AppStore・Google Playに出す方法