

Topic XX 3D都市モデルと位置情報をUnityで扱う

PLATEAUは他のさまざまな位置情報と組み合わせて使うことで、さらに多くの活用が考えられます。本トピックでは22年度チュートリアルのTopic14の発展として、スマートフォンのGPSによる位置情報など、座標系の違う位置情報をPLATEAUと重ね合わせて表示するARアプリの開発を通して、PLATEAUを他の位置情報と合わせて使う方法について説明します。

3D都市モデルを読み込む

最初に、PLATEAU SDK for Unityを使ってPLATEAUの3D都市モデルをUnityに読み込みます。この時、ファイルの扱い方、ツールの利用方法、どのような座標系で読み込まれるかなど、重ね合わせるためのポイントを確認します。

本チュートリアルでは、なるべく実際の処理の内容を詳細に記載するようにしています。

ですが、便利なライブラリなどもありますので、自分が読み込みたいデータに対応したライブラリなどを探してみてください。

PLATEAU SDK for Unityを使ってデータを読み込む

[PLATEAU SDK for Unity](#)を使って、PLATEAUの3D都市モデルをUnityに読み込みます。SDKについての詳細は、PLATEAU公式サイトのチュートリアルでも解説されています。詳細はそちらも参照してください。[TOPIC 17 | PLATEAU SDKでの活用\[1/2\] | PLATEAU SDK for Unityを活用する](#)また、GitHubのリポジトリのReadmeやマニュアルも参考にしてください。

今回は、Unity 2021.3.16f1を使用して進めていきます。.16f1などのマイナーバージョンはあまり気にしなくてよいと思いますが、PLATEAU SDK for Unityが現在Unityバージョン2021.3を想定しているので、2021.3系列を使用するとよいでしょう。

また、プロジェクトテンプレートとして、3D(URP)を使い、Universal Render Pipelineのプロジェクトとして作成していきます。

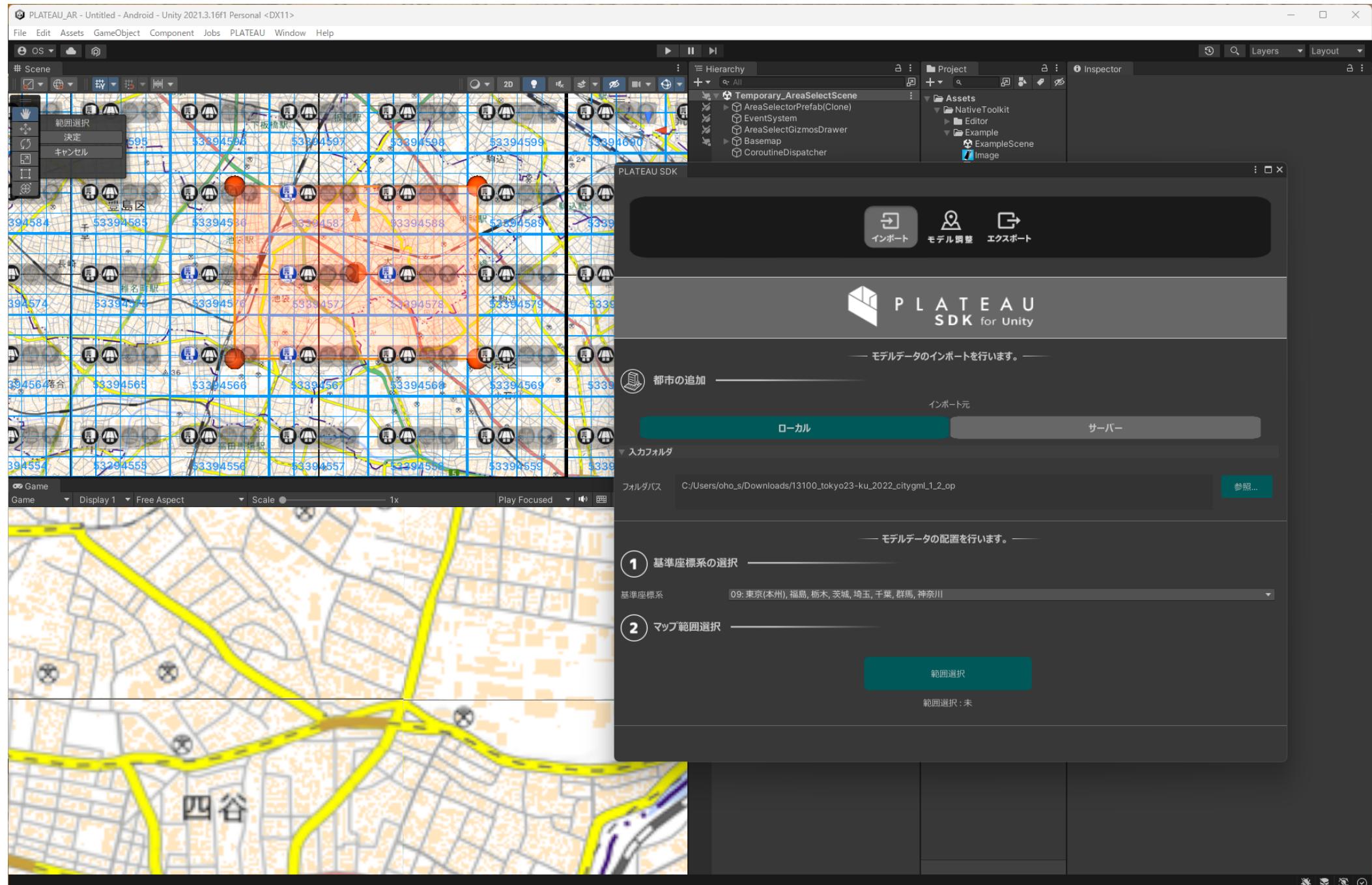
SDKのマニュアルなどにしたがって、新規プロジェクトを作成しSDKを導入します。

今回は、筆者の自宅周辺の東京都豊島区の大塚駅周辺でアプリを作っていきましょう。

SDKのダイアログを開き、あらかじめダウンロードし展開しておいた東京23区の3D都市モデルデータ(CityGML v2)のフォルダを指定します。また基準座標系の選択で「09:東京(本州),福島,栃木,茨木,埼玉,千葉,群馬,神奈川」を選びます。これらは対象とする3D都市モデルに合わせて読み替えてください。

PLATEAU SDK for Unityでは、インポート時に平面直角座標系に変換するようになっています。平面直角座標系は日本固有の投影座標系で、日本全国を19のゾーンに分け、ガウスの等角投影法を適用した座標系です。そのため、これから作るアプリでは最終的に平面直角座標系に変換することで、PLATEAUの3D都市モデルと合わせていきます。

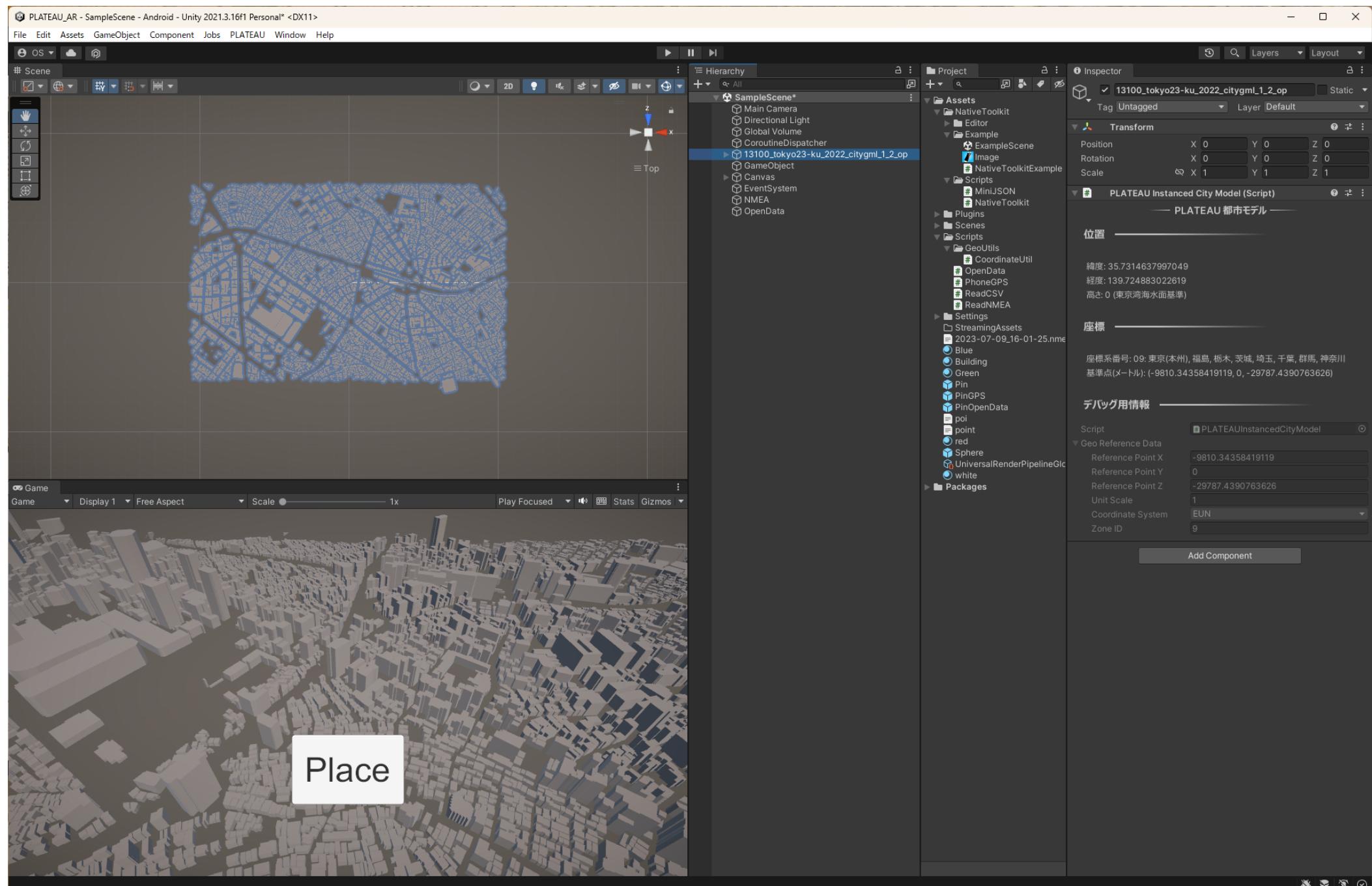
次に必要範囲を選択します。



地物別選択では、建築物のLOD1のみを選択します。

今回は3D都市モデルをアタリを付けるためとオクルージョンのために使用するので、あまり詳細なモデルではなく、軽量なLOD1を使いましたが、アプリケーションによって使い分けてください。

基準座標系からのオフセット値の設定では、自動的に範囲の中心が設定されるのでデフォルトで設定されている値で進めます。この値は、インポートされた3D都市モデルのデータのインスペクタで確認できます。これ以降の座標変換でも使いますので、確認しておきます。



最後に、「モデルをインポート」ボタンを押して、3D都市モデルのインポートを実行します。

さまざまな地理情報を重ねる

次に、座標変換の方法を説明します。実際に座標変換の式をプログラムにして、変換を確認します。また、GPSのデータやオープンデータの変換を行います。

座標計算の一般的な注意

緯度経度の座標から、平面直角座標系の座標に変換するプログラムを作成します。変換式は国土地理院が公開しているので、それをプログラムに落とし込みます。

一般的な注意点ですが、地理座標を扱うプログラムでは数値計算の誤差に注意します。とくに、地球規模の座標計算をする場合、たとえば地球の周囲長は約4万kmなので、メートルで記述すると約40,000,000mとなります。一方で、コンピューターが扱う浮動小数点規格の標準であるIEEE754を参照すると、一般的にfloatとして知られる32ビットの単精度浮動小数点の形式では実際の数字を現す仮数部が23ビットとなり、10進数でおよそ7桁の数字を表現できます。つまり、地球の周囲長と比べてみると上から数えて10mの位までしか表現できません。地球規模の座標の演算を行う際にfloatを使うと、10m以下の数字は計算されないことになり、大きく正確性が落ちます。

一方で、Unityなどの3Dコンピュータグラフィックスを描画するプログラムは、一般的に高速化のためfloatで座標を表します。

こうした違いのため、座標変換のプログラム内では倍精度浮動小数であるdoubleで計算し、扱いやすい局所的な座標に計算してからfloatにキャストしてUnityなどに渡す、という方法をとるのがよいでしょう。

平面直角座標系への変換

平面直角座標系への変換は、ガウス＝クリューゲル変換として知られる計算を行います。国土地理院のWebに、[測量計算サイト](#)というページがあり、ここでよく使う計算についてWebページ上で計算できる仕組みが用意されています。また、それぞれの計算式やアルゴリズムが解説されています。

[地理院ホーム](#) > [位置の基準・測量情報](#) > [測量計算サイト](#)[お知らせ](#) ↓ · [「測量計算サイト」について](#) ↓ · [注意事項](#) ↓

お知らせ

2023年6月2日 No.10「ジオイド高計算」について

計算に使用するジオイド・モデルを「日本のジオイド2011」（Ver.2.2）に更新しました。

なお、今回のジオイド・モデルの改定では、東京都硫黄島周辺の値を追加しており、その他の地域は「日本のジオイド2011」（Ver.2.1）から数値の変更はありません。

2023年3月24日 「基準点・測地観測データ」のページのリニューアルのお知らせ

3月28日（火）に、国土地理院サイト「基準点・測地観測データ」は「位置の基準・測量情報」に名称を変更し、ページのリニューアルを実施いたします。

このリニューアルに伴い、「便利なプログラム・データ」（<https://www.gsi.go.jp/keikaku/program.html>）にありました「重力値推定計算サービス」、「地磁気値」については、当「測量計算サイト」のページに移動いたします。

「便利なプログラム・データ」のページは削除予定です。

2023年2月15日 障害に伴うサービス断について

本日障害が発生し、測量計算サイトが一時的にご利用いただけませんでしたが、現在復旧しております。

利用者の皆様にはご不便をおかけし申し訳ございませんでした。

2023年1月25日 メンテナンスに伴うサービス断について

ネットワークメンテナンスのため、以下の時間について測量計算サイトをご利用いただくことができません。

1月25日（水） 13:00～16:00

利用者の皆様にはご不便をおかけいたしますが、ご理解のほど、どうぞよろしくお願ひいたします。

[過去の更新情報](#)

「測量計算サイト」について

以下の1～13の項目名（計算種類）のリンクをクリックすることで各計算サイトに移動できます

1～11及び13の項目では「1点毎の計算」あるいは「一括計算」することができます。

1～10の項目ではREST APIでの計算が可能です。詳細については[「API使用方法」のページ](#)をご覧下さい。

No	計算種類	説明
1	緯度・経度と地心直交座標の相互換算	緯度、経度、標高、ジオイド高と地心直交座標（X, Y, Z値）との相互換算を行います。
2	距離と方位角の計算	緯度、経度から2点間の距離と方位角を求めます。

3	距離と方向角の計算	平面直角座標から2点間の距離と方向角を求める。
4	平面直角座標への換算	緯度、経度から平面直角座標へ換算します。
5	緯度、経度への換算	平面直角座標から緯度、経度へ換算します。
6	世界測地系座標変換 (TKY2JGD)	日本測地系に基づく経緯度および平面直角座標を世界測地系に基づく経緯度および平面直角座標へ変換します。
7	PatchJGD	地殻変動前の座標値から変動後の座標値へ補正します。
8	PatchJGD (標高版)	地殻変動前の標高値から変動後の標高値へ補正します。
9	SemiDynaEXE	定常的な地殻変動による歪みの影響を補正するセミ・ダイナミック補正を行います。
10	ジオイド高計算	ジオイドの計算には、日本の測地基準系に適合するジオイド・モデル「日本のジオイド 2011」(Ver.2.2)を使用しています。(海域部や一部の離島等では、ジオイド高データがない無効領域があります。)
11	補正パラメータによる標高成果計算サイト	測量の地域や時期等を選択することで自動的に必要なパラメータを選択し、標高成果の補正計算を実施することができます。 ※本サイトは一部の離島の標高改定には対応しておりません。補正を必要とする場合は、No.8「PatchJGD(標高版)」を別途ご利用ください。また、適応すべき補正パラメータについては 平成26、28年三角点標高成果改定(パラメータ補正計算方法) を参照ください。
12	重力値推定計算サービス	緯度、経度、標高から重力の推定値を計算します。
13	地磁気値	緯度、経度から地磁気の値を計算します。

こここの計算式を使って、緯度経度から平面直角座標系に変換するプログラムを作成してみましょう。

[平面直角座標への換算](#)のページの上段に、計算式というリンクがあり、

平面直角座標への換算

[トップページ](#)[操作方法](#)[計算式](#)[お問い合わせ](#)

入力値

[1点毎の計算](#)[一括計算](#)

座標値の入力方法 数値入力 地図上で選択

測地系

平面直角座標系

座標値の入力

緯度

経度

入力単位 度分秒 十進法度単位

【緯度・経度の値の入力例(度分秒)】

緯度 $36^{\circ} 6' 13.58925'' \rightarrow 360613.58925$

経度 $140^{\circ} 5' 16.27815'' \rightarrow 1400516.27815$

ddd mm ss.s \rightarrow dddmmss.s

計算結果





地図は世界測地系であり、日本測地系に対応しておりません。日本測地系（bessel橢円体）を選択した場合、「地図上で選択」して入力したり、「地図上で確認」機能を使用しても正しい結果となりませんのでご注意ください。

国土地理院 (C) 2013- Geospatial Information Authority of Japan. All rights reserved.

そこから計算式を確認できます。

経緯度を換算して平面直角座標、子午線収差角及び縮尺係数を求める計算

x 座標及び y 座標

$$x = \bar{A} \left(\xi' + \sum_{j=1}^5 \alpha_j \sin 2j\xi' \cosh 2j\eta' \right) - \bar{S}_{\varphi_0}, \quad y = \bar{A} \left(\eta' + \sum_{j=1}^5 \alpha_j \cos 2j\xi' \sinh 2j\eta' \right)$$

子午線収差角 γ 及び縮尺係数 m

$$\gamma = \tan^{-1} \left(\frac{\tau \bar{t} \lambda_c + \sigma t \lambda_s}{\sigma \bar{t} \lambda_c - \tau t \lambda_s} \right), \quad m = \frac{\bar{A}}{a} \sqrt{\frac{\sigma^2 + \tau^2}{t^2 + \lambda_c^2} \left\{ 1 + \left(\frac{1-n}{1+n} \tan \varphi \right)^2 \right\}}$$

ただし、

φ, λ ：新点の緯度及び経度

φ_0, λ_0 ：平面直角座標系原点の緯度及び経度

a, F ：橢円体の長半径及び逆扁平率

m_0 ：平面直角座標系の X 軸上における縮尺係数 (0.9999)

$$n = \frac{1}{2F - 1}$$

$$t = \sinh \left(\tanh^{-1} \sin \varphi - \frac{2\sqrt{n}}{1+n} \tanh^{-1} \left[\frac{2\sqrt{n}}{1+n} \sin \varphi \right] \right), \quad \bar{t} = \sqrt{1+t^2}$$

$$\lambda_c = \cos(\lambda - \lambda_0), \quad \lambda_s = \sin(\lambda - \lambda_0), \quad \xi' = \tan^{-1} \left(\frac{t}{\lambda_c} \right), \quad \eta' = \tanh^{-1} \left(\frac{\lambda_s}{\bar{t}} \right)$$

$$\sigma = 1 + \sum_{j=1}^5 2j\alpha_j \cos 2j\xi' \cosh 2j\eta', \quad \tau = \sum_{j=1}^5 2j\alpha_j \sin 2j\xi' \sinh 2j\eta'$$

$$\alpha_1 = \frac{1}{2}n - \frac{2}{3}n^2 + \frac{5}{16}n^3 + \frac{41}{180}n^4 - \frac{127}{288}n^5, \quad \alpha_2 = \frac{13}{48}n^2 - \frac{3}{5}n^3 + \frac{557}{1440}n^4 + \frac{281}{630}n^5,$$

$$\alpha_3 = \frac{61}{240}n^3 - \frac{103}{140}n^4 + \frac{15061}{26880}n^5, \quad \alpha_4 = \frac{49561}{161280}n^4 - \frac{179}{168}n^5, \quad \alpha_5 = \frac{34729}{80640}n^5$$

$$\bar{S}_{\varphi_0} = \frac{m_0 a}{1+n} \left(A_0 \frac{\varphi_0}{\rho''} + \sum_{j=1}^5 A_j \sin 2j\varphi_0 \right), \quad \bar{A} = \frac{m_0 a}{1+n} A_0$$

$$A_0 = 1 + \frac{n^2}{4} + \frac{n^4}{64}, \quad A_1 = -\frac{3}{2} \left(n - \frac{n^3}{8} - \frac{n^5}{64} \right), \quad A_2 = \frac{15}{16} \left(n^2 - \frac{n^4}{4} \right)$$

$$A_3 = -\frac{35}{48} \left(n^3 - \frac{5}{16} n^5 \right), \quad A_4 = \frac{315}{512} n^4, \quad A_5 = -\frac{693}{1280} n^5$$

ここで ρ'' は緯度 φ_0 をラジアン単位に変換する量であり、緯度を表現する単位によって与え方が異なる。例えば緯度が秒単位の場合には、 $\rho'' = 3600 \times \frac{180}{\pi}$ であり、分単位の場合には、 $\rho'' = 60 \times \frac{180}{\pi}$ であり、度単位の場合には、 $\rho'' = \frac{180}{\pi}$ である。

引用文献

河瀬和重 (2011): [Gauss-Krüger投影における経緯度座標及び平面直角座標相互間の座標換算についてのより簡明な計算方法](#), 国土地理院時報, 121, 109-124.

今回は、Unityなので、C#でこの計算式をプログラムに落とし込みます。

ここでは、座標変換のクラスは、UnityのMonoBehaviorを継承する必要がないので、素のクラスとして記述します。また、状態を持つ必要がないので、変換メソッドはstaticとして実装します。サンプルでは座標の2つの値を返すのにタプルを使っていますが、ここも必要に応じて設計を変えるとよいでしょう。（ここはどのように使うかを想定して各自で設計を考えてみてください。）

以下が緯度経度と平面直角座標系の変換プログラムのサンプルです。

```

using System;

public class CoordinateUtil
{
    // GRS80 Ellipsoid
    private const double a = 6378137d;
    private const double F = 298.257222101d;

    // 平面直角座標系のX軸上における縮尺係数
    private const double m0 = 0.9999d;

    private const double n = 1d / (2d * F - 1d);

    // Geographic -> Plane Rectangular
    private const double a1 = 1d * n / 2d - 2d * n * n / 3d + 5d * n * n * n / 16d + 41d * n * n * n * n / 180d - 127d * n * n * n * n / 288d;
    private const double a2 = 13d * n * n / 48d - 3d * n * n * n / 5d + 557d * n * n * n * n / 1440d + 281d * n * n * n * n / 630d;
    private const double a3 = 61d * n * n * n / 240d - 103d * n * n * n * n / 140d + 15061d * n * n * n * n * n / 26880d;
    private const double a4 = 49561d * n * n * n * n / 161280d - 179d * n * n * n * n * n / 168d;
    private const double a5 = 34729d * n * n * n * n * n / 80640d;

    private const double A0 = 1d + n * n / 4d + n * n * n * n / 64d;
    private const double A1 = -3d / 2d * (n - n * n * n / 8d - n * n * n * n * n / 64d);
    private const double A2 = 15d / 16d * (n * n - n * n * n * n / 4d);
    private const double A3 = -35d / 48d * (n * n * n - 5d * n * n * n * n * n / 16d);
    private const double A4 = 315d * n * n * n * n / 512d;
    private const double A5 = -693d * n * n * n * n * n / 1280d;

    // Plane Rectangular -> Geographic
    private const double b1 = n / 2d - 2d * n * n / 3d + 37d * n * n * n / 96d - n * n * n * n / 360d - 81d * n * n * n * n * n / 512d;
    private const double b2 = n * n / 48d + n * n * n / 15d - 437d * n * n * n * n / 1440d + 46d * n * n * n * n * n / 105d;
    private const double b3 = 17d * n * n * n / 480d - 37d * n * n * n * n / 840d - 209d * n * n * n * n * n / 4480d;
    private const double b4 = 4397d * n * n * n * n / 161280 - 11d * n * n * n * n * n / 504d;
    private const double b5 = 4583d * n * n * n * n * n / 161280d;

    private const double d1 = 2d * n - 2d * n * n / 3d - 2d * n * n * n + 116d * n * n * n * n / 45d + 26d * n * n * n * n * n / 45d - 2854d * n * n * n * n * n * n / 675;
    private const double d2 = 7d * n * n / 3d - 8d * n * n * n / 5d - 227d * n * n * n * n / 45d + 2704d * n * n * n * n * n / 315d + 2323d * n * n * n * n * n * n / 945d;
    private const double d3 = 56d * n * n * n / 15d - 136d * n * n * n * n / 35d - 1262d * n * n * n * n * n / 105d + 73814d * n * n * n * n * n * n / 2835d;
    private const double d4 = 4279d * n * n * n * n / 640d - 332d * n * n * n * n * n / 35d - 399572d * n * n * n * n * n * n / 14175d;
    private const double d5 = 4174d * n * n * n * n * n / 315d - 144838d * n * n * n * n * n * n / 6237d;
    private const double d6 = 601676d * n * n * n * n * n * n / 22275d;

    public static (double x, double y) JGD2011ToPlaneRectCoord(double lat, double lon, double o_lat, double o_lon)

```

```

{
    double latr = lat * Math.PI / 180d; // TO Radian
    double lonr = lon * Math.PI / 180d;
    double o_latr = o_lat * Math.PI / 180d;
    double o_lonr = o_lon * Math.PI / 180d;

    double t = Math.Sinh(Math.Atanh(Math.Sin(latr))
        - 2d * Math.Sqrt(n) / (1d + n) * Math.Atanh(2d * Math.Sqrt(n) / (1d + n) * Math.Sin(latr)));
    double _t = Math.Sqrt(1d + t * t);

    double Lc = Math.Cos(lonr - o_lonr);
    double Ls = Math.Sin(lonr - o_lonr);

    double Xi_ = Math.Atan(t / Lc);
    double Eta_ = Math.Atanh(Ls / _t);

    double _S = m0 * a / (1d + n) * (A0 * o_latr +
        A1 * Math.Sin(2d * o_latr) +
        A2 * Math.Sin(2d * 2d * o_latr) +
        A3 * Math.Sin(2d * 3d * o_latr) +
        A4 * Math.Sin(2d * 4d * o_latr) +
        A5 * Math.Sin(2d * 5d * o_latr));

    double _A = m0 * a / (1d + n) * A0;

    double x = _A * (Xi_ +
        a1 * Math.Sin(2d * 1d * Xi_) * Math.Cosh(2d * 1d * Eta_) +
        a2 * Math.Sin(2d * 2d * Xi_) * Math.Cosh(2d * 2d * Eta_) +
        a3 * Math.Sin(2d * 3d * Xi_) * Math.Cosh(2d * 3d * Eta_) +
        a4 * Math.Sin(2d * 4d * Xi_) * Math.Cosh(2d * 4d * Eta_) +
        a5 * Math.Sin(2d * 5d * Xi_) * Math.Cosh(2d * 5d * Eta_)) - _S;
    double y = _A * (Eta_ +
        a1 * Math.Cos(2d * 1d * Xi_) * Math.Sinh(2d * 1d * Eta_) +
        a2 * Math.Cos(2d * 2d * Xi_) * Math.Sinh(2d * 2d * Eta_) +
        a3 * Math.Cos(2d * 3d * Xi_) * Math.Sinh(2d * 3d * Eta_) +
        a4 * Math.Cos(2d * 4d * Xi_) * Math.Sinh(2d * 4d * Eta_) +
        a5 * Math.Cos(2d * 5d * Xi_) * Math.Sinh(2d * 5d * Eta_));

    return (x, y);
}

public static (double lat, double lon) PlaneRectCoordToJGD2011(double x, double y, double o_lat, double o_lon)

```

```

{
    double o_latr = o_lat * Math.PI / 180d;
    double o_lonr = o_lon * Math.PI / 180d;

    double _S = m0 * a / (1d + n) * (A0 * o_latr +
        A1 * Math.Sin(2d * o_latr) +
        A2 * Math.Sin(2d * 2d * o_latr) +
        A3 * Math.Sin(2d * 3d * o_latr) +
        A4 * Math.Sin(2d * 4d * o_latr) +
        A5 * Math.Sin(2d * 5d * o_latr));

    double _A = m0 * a / (1d + n) * A0;

    double Xi = (x + _S) / _A;
    double Eta = y / _A;

    double Xi_ = Xi - (
        b1 * Math.Sin(2d * Xi) * Math.Cosh(2d * Eta) +
        b2 * Math.Sin(2d * 2d * Xi) * Math.Cosh(2d * 2d * Eta) +
        b3 * Math.Sin(2d * 3d * Xi) * Math.Cosh(2d * 3d * Eta) +
        b4 * Math.Sin(2d * 4d * Xi) * Math.Cosh(2d * 4d * Eta) +
        b5 * Math.Sin(2d * 5d * Xi) * Math.Cosh(2d * 5d * Eta));
    double Eta_ = Eta - (
        b1 * Math.Cos(2d * Xi) * Math.Sinh(2d * Eta) +
        b2 * Math.Cos(2d * 2d * Xi) * Math.Sinh(2d * 2d * Eta) +
        b3 * Math.Cos(2d * 3d * Xi) * Math.Sinh(2d * 3d * Eta) +
        b4 * Math.Cos(2d * 4d * Xi) * Math.Sinh(2d * 4d * Eta) +
        b5 * Math.Cos(2d * 5d * Xi) * Math.Sinh(2d * 5d * Eta));

    double Kai = Math.Asin(Math.Sin(Xi_) / Math.Cosh(Eta_));

    double lat = 180d / Math.PI * (Kai +
        d1 * Math.Sin(2d * Kai) +
        d2 * Math.Sin(2d * 2d * Kai) +
        d3 * Math.Sin(2d * 3d * Kai) +
        d4 * Math.Sin(2d * 4d * Kai) +
        d5 * Math.Sin(2d * 5d * Kai) +
        d6 * Math.Sin(2d * 6d * Kai)
    );
    double lon = o_lon + 180d / Math.PI * Math.Atan2(
        Math.Sinh(Eta_), Math.Cos(Xi_));
}

```

```
    return (lat, lon);
}
}
```

変換メソッドを呼ぶときに、変換したい座標と平面直角座標系の原点座標と一緒に渡します。

平面直角座標系の原点座標は、[国土地理院のページ](#)で確認してください。

次に、この変換プログラムを使って緯度経度を持った点をPLATEAUの3D都市モデルと重ねて表示してみましょう。

位置情報の読み込み

まず初めに、座標を羅列したCSVファイルを読み込み、その場所にオブジェクトを置いてみます。

CSVファイルはカンマでデータを区切ったテキストファイルです。テキストエディタで作成でき、扱いが容易なのでデータの保存形式としてよく使われます。

ここでは、緯度,経度,高さのように3つの数値を羅列したファイルを以下のように作ります。

```
35.729550,139.730014,20.8
35.730756,139.725266,31.8
35.730020,139.727659,24.2
35.730020,139.723019,30.8
```

手作業で場所のデータを作成するときは、国土地理院の公開している[地理院地図](#)が便利です。地図中心の地理座標や標高などを調べることができます。



地図の種類

トップ

令和5年石川県能登地方の地震

年代別の写真



標高・土地の凹凸



土地の成り立ち・土地利用



基準点・地磁気・地殻変動



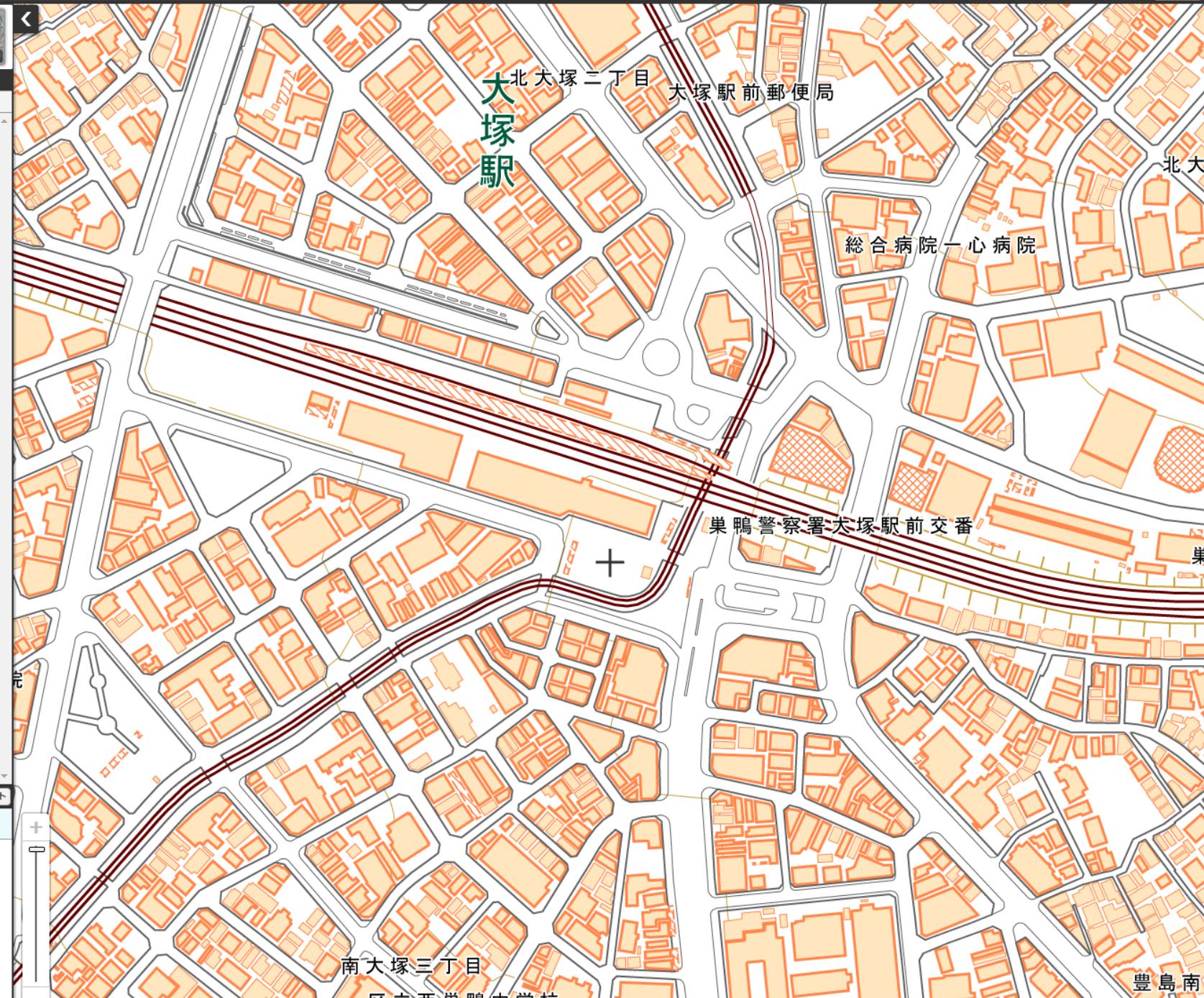
災害伝承・避難場所



近年の災害



その他





次に、用意したCSVファイルを読み込み、Prefabを配置するC#スクリプトを書きます。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ReadCSV : MonoBehaviour
{
    public TextAsset csv;
    public GameObject go;

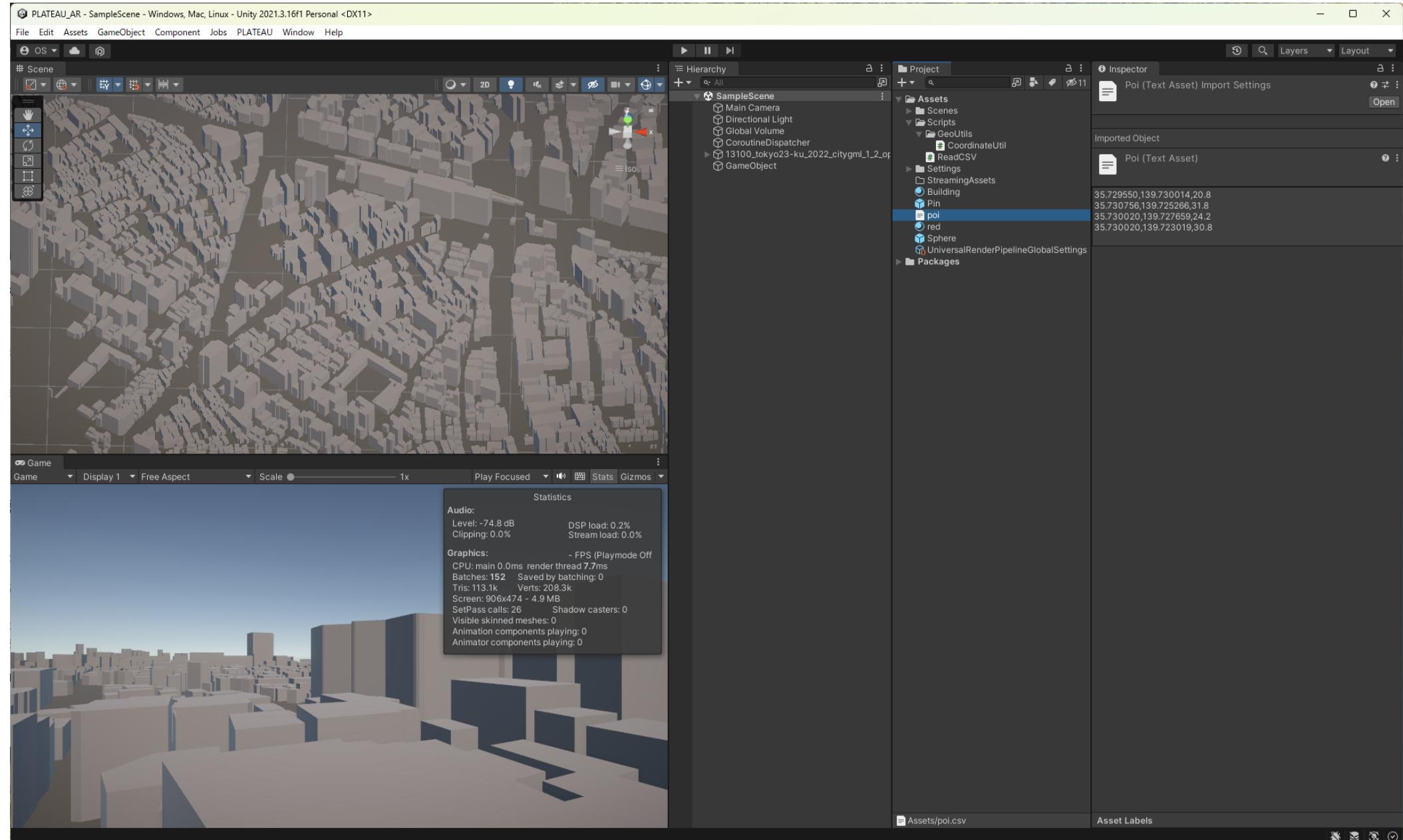
    // Start is called before the first frame update
    void Start()
    {
        var lines = csv.text.Split("\n");
        foreach (var line in lines)
        {
            if (line.Contains(","))
            {
                var tokens = line.Split(",");
                double lat = double.Parse(tokens[0]);
                double lon = double.Parse(tokens[1]);
                double height = double.Parse(tokens[2]);

                (var x, var y) = CoordinateUtil.JGD2011ToPlaneRectCoord(lat, lon, 36d, 139.83333333333d);

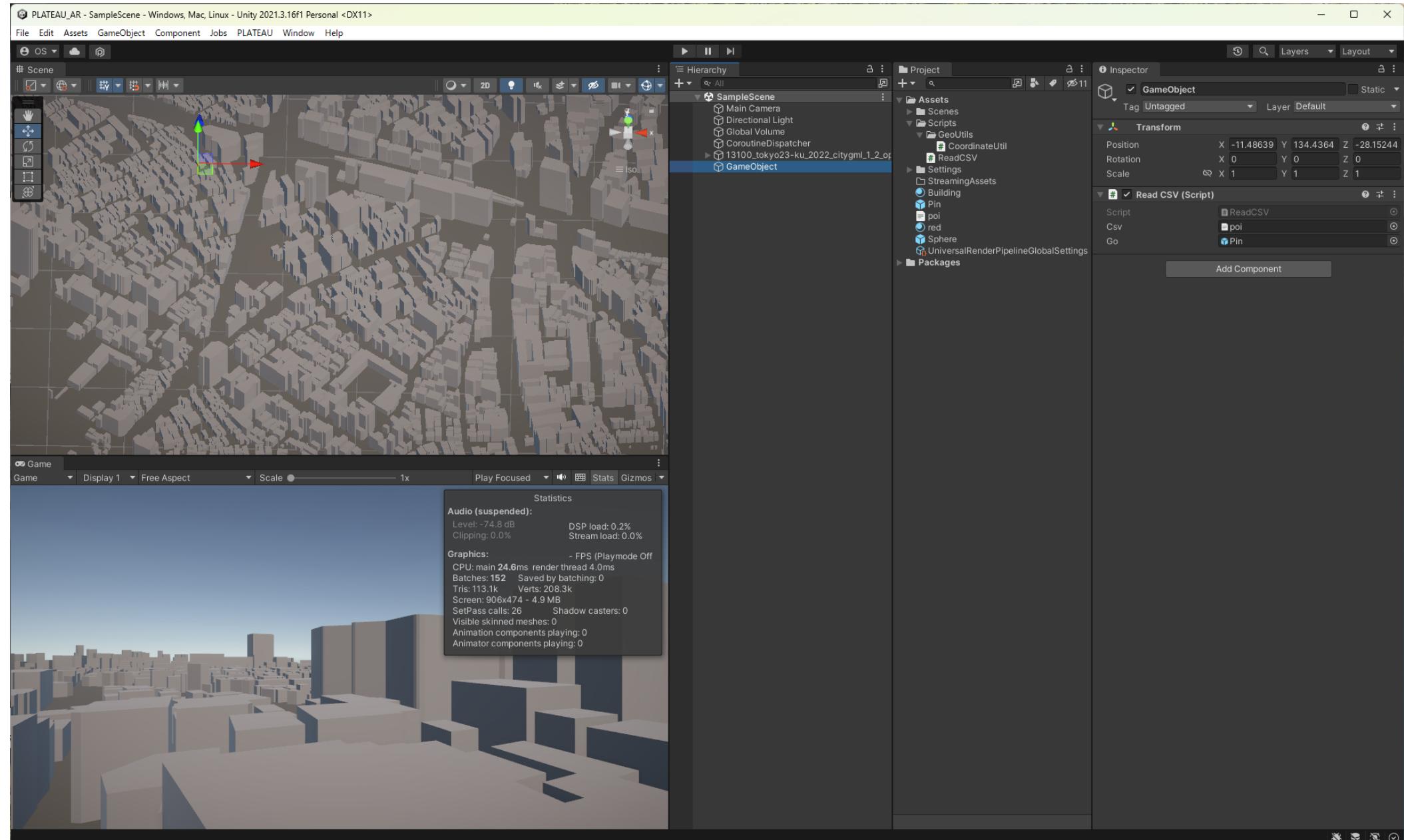
                x = x + 29787.4390;
                y = y + 9810.3435;

                Instantiate(go, new Vector3((float)y, (float)height, (float)x), Quaternion.identity);
            }
        }
    }
}
```

CSVファイルは `TextAsset` として読み込むようにしましたが、`FileStream`などで読み込んでもよいでしょう。



図のように、CSVファイルをドラッグ＆ドロップで、Unityプロジェクトに取り込み、次の図のように、ReadCSV.csをアタッチしたゲームオブジェクトのインスペクターでCSVファイルを設定します。

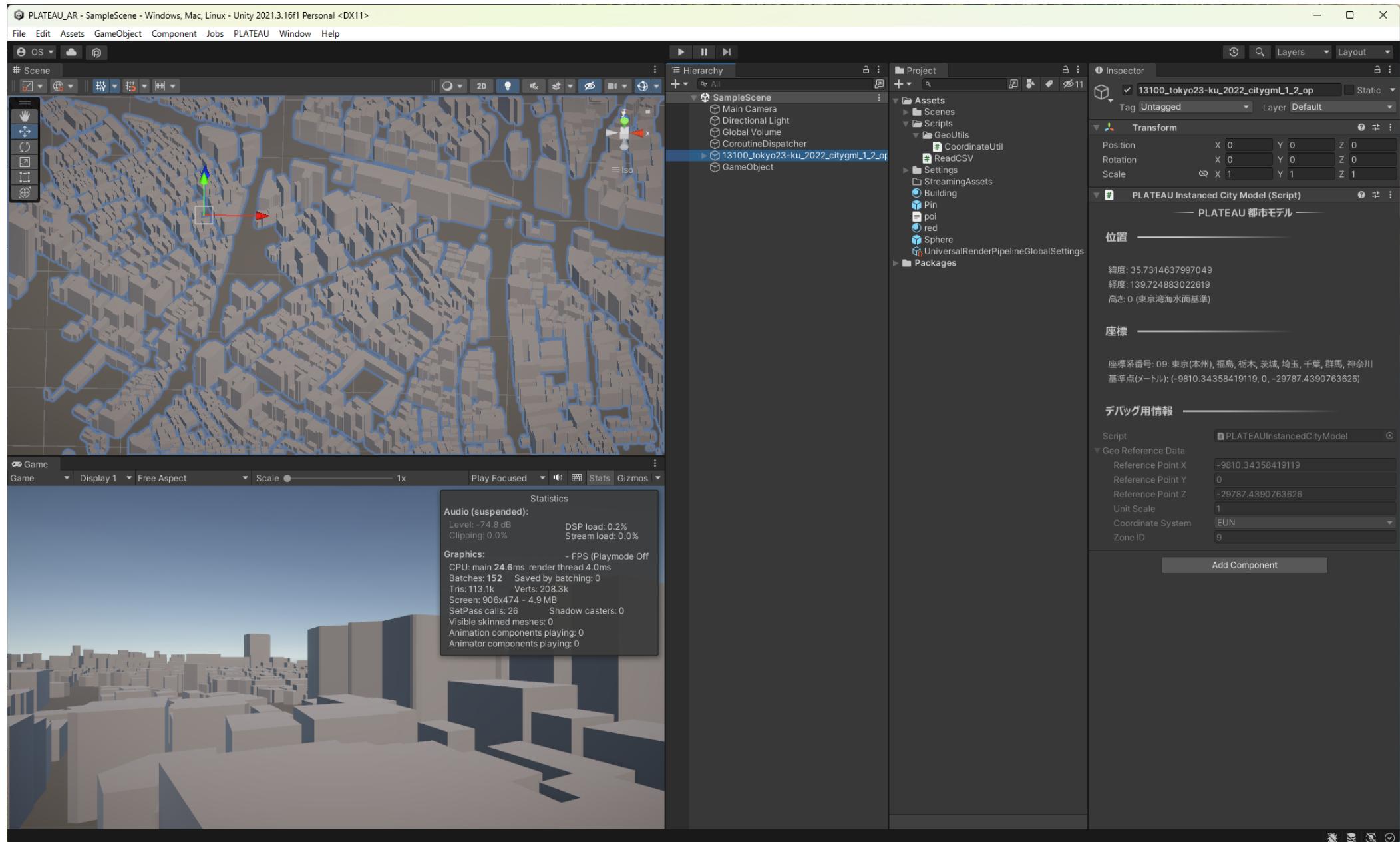


CSVファイルのパースは便利なライブラリなどもありますが、今回は簡単な読み込みロジックを書きました。座標変換には、作成した変換クラスを使っています。平面直角座標系の原点は9系の値を設定しています。

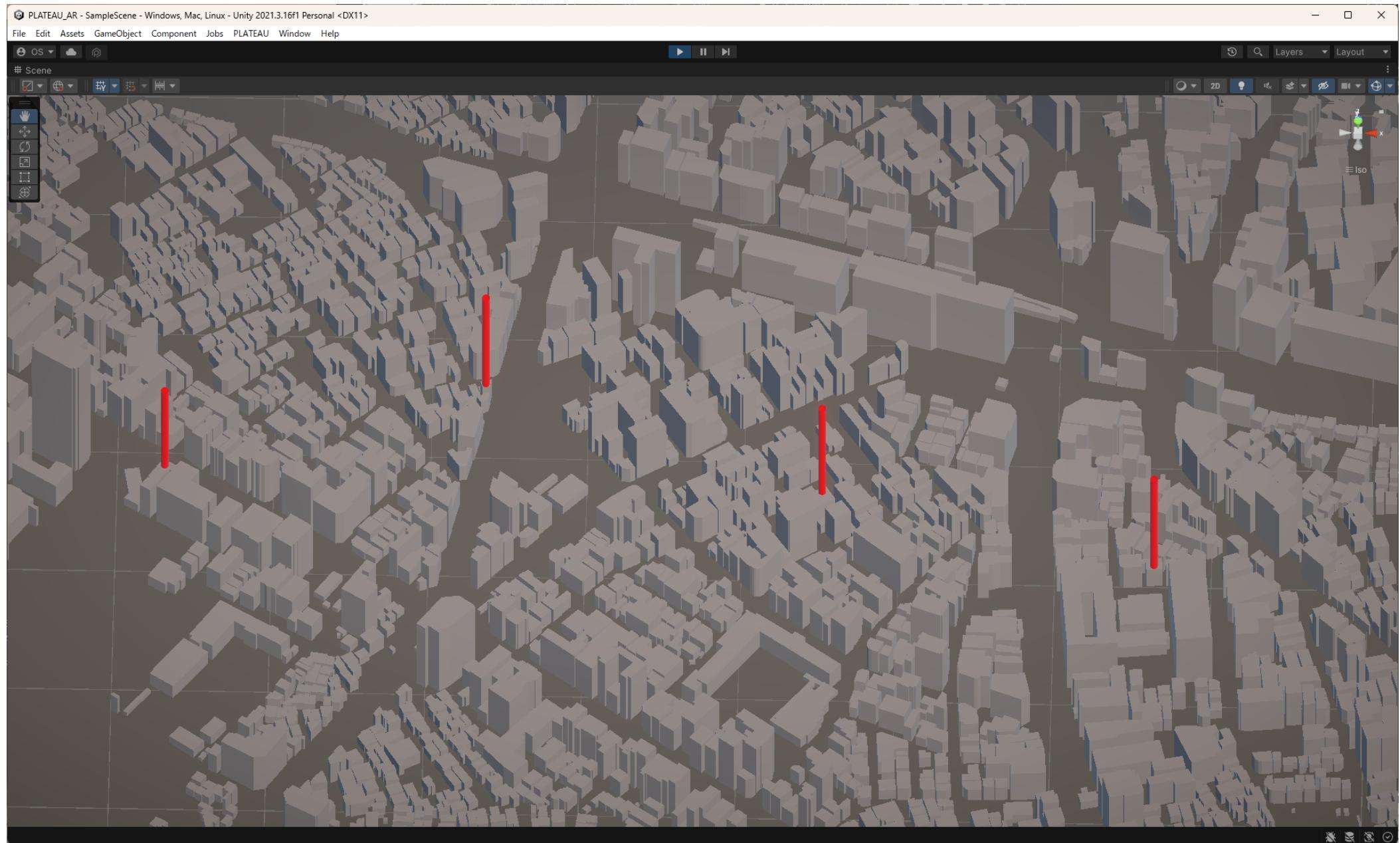
また、Unity SDK for PLATEAUで読み込む際に設定したオフセットの値を引き算しています。

平面直角座標では、Xが南北方向、Yが東西方向を表すので、そこも注意しましょう。

オフセットの座標は読み込んだ3D都市モデルのインスペクターで確認できます。



実行すると図のよう、PLATEAUの3D都市モデルと合わせて指定した位置に赤い柱状のオブジェクトを配置します。



スマホのGPS情報を重ねる

スマートフォンのGPS情報を取得して、その位置にオブジェクトを置いてみましょう。

端末のGPSの座標をUnityで取得するには、 `Input.Location` を使うことができますが、これまでの歴史的経緯のために、緯度経度の座標値がfloatの単精度浮動小数で返ってきます。これでは精度として足りないので、AssetsStoreに公開されている `NativeToolkit` を使います。 `NativeToolkit` は、iOSとAndroidのモバイル端末向けの複数の機能をまとめたネイティブライブラリです。

まず、AssetStoreから入手します。UnityのAssetStoreのNativeToolKitのページで「マイアセットに追加する」を押します。

ワークスペース Native Toolkit | 機能統合 | +

https://assetstore.unity.com/packages/tools/integration/native-toolkit-132452?locale=ja-JP

30 for \$30 Mega Bundle: save up to 95% on quality assets

Unity Asset Store アセットの検索

3D 2D アドオン オーディオ AI 分散化 Unity公式アセット テンプレート ツール ビジュアルエフェクト Sale Sell Assets

11,000 種類を超える 5 つ星アセット 8.5 万人以上の顧客による評価 10 万人を超えるフォーラムメンバーが支持 すべてのアセットを Unity が審査済み

ホーム > ツール > 機能統合 > Native Toolkit

Native Toolkit

Native functionality made easy on iOS and Android.

1/6

Native Toolkit Hello there! This is an email sent from my App!

Rate This App Please take a moment to rate this App.

Native Toolkit This is a one-off item!

Cookie 設定 すべての Cookie を受け入れる X

Native Toolkit

Second Fury ★★★☆☆ (22) | ❤ (241)

FREE

82 views in the past week

マイアセットに追加する

標準Unity Asset Store EULA

ライセンス シート

ファイルサイズ 755.5 KB

最新バージョン 1.2

最新リリース日 2019年5月22日

オリジナルの Unity バージョン ② 2018.2.20 以降

Frequently bought together

Cookie 設定 すべての Cookie を受け入れる X



100% 2:57

AssetStoreで入手したら、PackageManagerを開き、Download,Importします。「Packages:My Assets」を選択すると、入手したAssetの一覧が出てくるので、NativeToolkitを探します。

Package Manager

+ Packages: My Assets Sort: Name ▾ Filters ▾ Clear Filters

3DThrough	2.0	↓
DOTween (HOTween v2)	1.2.705	+
Final IK	2.2	↓
InGame Code Editor	1.1.3	↓
Japanese Otaku City	1.0	↓
Massive Clouds Atmos - Volumetric Skybox	1.1.0	↓
MoonSharp	2.0.0.1	↓
Native Toolkit	1.2	+
Oculus Integration	54.1	↓
OpenCV for Unity	2.5.4	↓
Paint in 3D	3.0.2	↓
PathMagic	1.2.6	↓
ROS#	1.7	↓
Runtime Inspector & Hierarchy	1.7.0	+
Sakura's Room	1.0	↓
SpaceNavigator Driver	1.5.2	↓
Standard Assets 2018.4 Check out Starter Assets: First Person & Third ...	1.1.6	↓
SteamVR Plugin	2.7.3 (sdk 1.14.15)	↓
Unity Recorder	1.0.2	↓

All 19 packages shown

Last update Jul 6, 01:00

C ▾

Native Toolkit

Second Fury

Version 1.2 - May 22, 2019 [asset store](#)

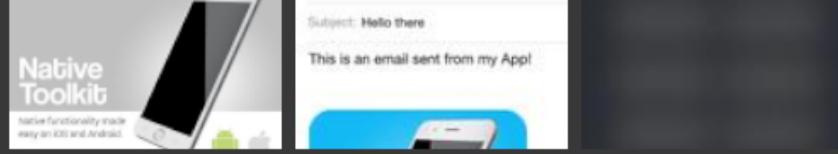
[View in the Asset Store](#) • [Publisher Website](#)

Native Toolkit is a lightweight plugin that makes it easy to integrate native iOS or Android functionality into your Unity project.

>>> GitHub repository here <<< (<https://github.com/ryanw3bb/unity-native-toolkit>)

[More...](#)

Images & Videos



[View images & videos on Asset Store](#)

Package Size

Size: 755.45 KB (Number of files: 37)

Supported Unity Versions

2018.2.20 or higher

Purchased Date

March 14, 2021

Release Details

1.2 (Current) - released on May 22, 2019 [More...](#)

Original - released on November 08, 2018

Assigned Labels

(None)

Import Re-Download

あとは画面の指示に従いプロジェクト内にimportすることで使えるようになります。

実際にボタンを押したときに、その場所の座標にオブジェクトが置かれる用にしてみます。

以下のプログラムを作成します。StartでLocationのサービスを初期化しています。

NativeToolkitは高さを取得するメソッドがないので、そこだけUnity標準のInput.Locationを使うようにします。

placeGameObjectメソッド内で緯度経度を取得しています。取得した緯度経度はそのまま平面直角座標系への変換を行い、オフセットを引いてGameObjectの座標としています。

また、適当なUIボタンを配置して、OnClickのイベントハンドラにplaceGameObjectメソッドを指定します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PhoneGPS : MonoBehaviour
{
    public GameObject go;

    // Start is called before the first frame update
    void Start()
    {
        NativeToolkit.StartLocation();
        Input.location.Start();
    }

    public void placeGameObject()
    {
        double lat = NativeToolkit.GetLatitude();
        double lon = NativeToolkit.GetLongitude();
        double height = Input.location.lastData.altitude;

        (var x, var y) = CoordinateUtil.JGD2011ToPlaneRectCoord(lat, lon, 36d, 139.8333333333d);

        x = x + 29787.4390;
        y = y + 9810.3435;

        Instantiate(go, new Vector3((float)y, (float)height, (float)x), Quaternion.identity);
    }
}
```

適当なGameObjectにアタッチしたら、goに置きたいPrefabやGameObjectを指定して、端末向けにビルドして実行します。

TODO 実際の画面

GPS受信機のデータを重ねる

一般的なGPS受信機ではNMEAという形式で座標や受信している衛星などのさまざまな情報が出力されます。これを保存したファイルを読み込み、座標変換してPLATEAUの3D都市モデル上に重ねて表示します。

現在では、さまざまなGPS受信機が入手できます。たとえば、GARMIN社のGPS受信機（例：[GPSMAP 66i](#)）のように、ディスプレイ付きで単体でナビゲーションできるものから、[GPS-RTK-SMAモジュール](#)のような、モジュールとして組み込むための基板も市販されています。

また、スマートフォンに搭載されているような一般的なGPS受信機は、精度が10m以上の1周波の単体GPSが多いのですが、2周波以上の多波長に対応する受信機や、サーバーと補正情報を通信することで最高精度が数cmにもなるRTKという仕組みを搭載した高精度な受信機もあります。

ここでは、ビズステーション株式会社が販売しているDropperの[RWP](#)パッケージを使用し、ソフトバンク株式会社が提供する[ichimill](#)サービスを補正情報として使用してRTKによる高精度測位でNMEAデータを作成しました。

NMEAでは、行単位のカンマ区切りのテキストデータでGPSに関する情報を表します。

例として、今回使用したDropperのRWPのNMEA出力を示します。

```
$GNGGA,070239.87,3543.9279547,N,13943.5410115,E,5,12,0.64,31.530,M,39.331,M,0.9,0150*50
$GNGGA,070240.00,3543.9278367,N,13943.5410033,E,5,12,0.64,31.578,M,39.331,M,1.0,0150*55
$GNGGA,070240.12,3543.9277397,N,13943.5409928,E,5,12,0.64,31.683,M,39.331,M,1.1,0150*5B
$GNGGA,070240.25,3543.9276340,N,13943.5409879,E,5,12,0.65,31.691,M,39.331,M,1.3,0150*51
$GNGGA,070240.37,3543.9275475,N,13943.5409796,E,5,12,0.64,31.720,M,39.331,M,1.4,0150*53
$GNGGA,070240.50,3543.9274611,N,13943.5409773,E,5,12,0.64,31.792,M,39.331,M,1.5,0150*50
$GNGGA,070240.62,3543.9273552,N,13943.5409905,E,5,12,0.65,31.867,M,39.331,M,0.6,0150*5B
$GNGGA,070240.75,3543.9272637,N,13943.5410161,E,5,12,0.64,31.859,M,39.331,M,0.8,0150*5C
$GNGGA,070240.87,3543.9271658,N,13943.5409850,E,5,12,0.60,31.785,M,39.331,M,0.9,0150*53
$GNGGA,070241.00,3543.9270604,N,13943.5409561,E,5,12,0.60,31.772,M,39.331,M,1.0,0150*5A
```

CSVの時のように、NMEAのファイルをパースして、PLATEAUと合わせて表示するプログラムを書きます。

今回はGPS受信機の軌跡をLineRendererを使って線で表示します。

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ReadNMEA : MonoBehaviour
{
    public TextAsset nmea;

    // Start is called before the first frame update
    void Start()
    {
        LineRenderer lr = GetComponent<LineRenderer>();
        List<Vector3> positions = new List<Vector3>();

        var lines = nmea.text.Split("\n");
        foreach (var line in lines)
        {
            try
            {
                if (line.Contains(","))
                {
                    var tokens = line.Split(",");

                    if (tokens[0] == "$GNGGA")
                    {
                        double lat = double.Parse(tokens[2].Substring(0, 2)) + double.Parse(tokens[2].Substring(2)) / 60d;
                        double lon = double.Parse(tokens[4].Substring(0, 3)) + double.Parse(tokens[4].Substring(3)) / 60d;
                        float height = float.Parse(tokens[9]);

                        (var x, var y) = CoordinateUtil.JGD2011ToPlaneRectCoord(lat, lon, 36d, 139.8333333333d);

                        x = x + 29787.4390;
                        y = y + 9810.3435;

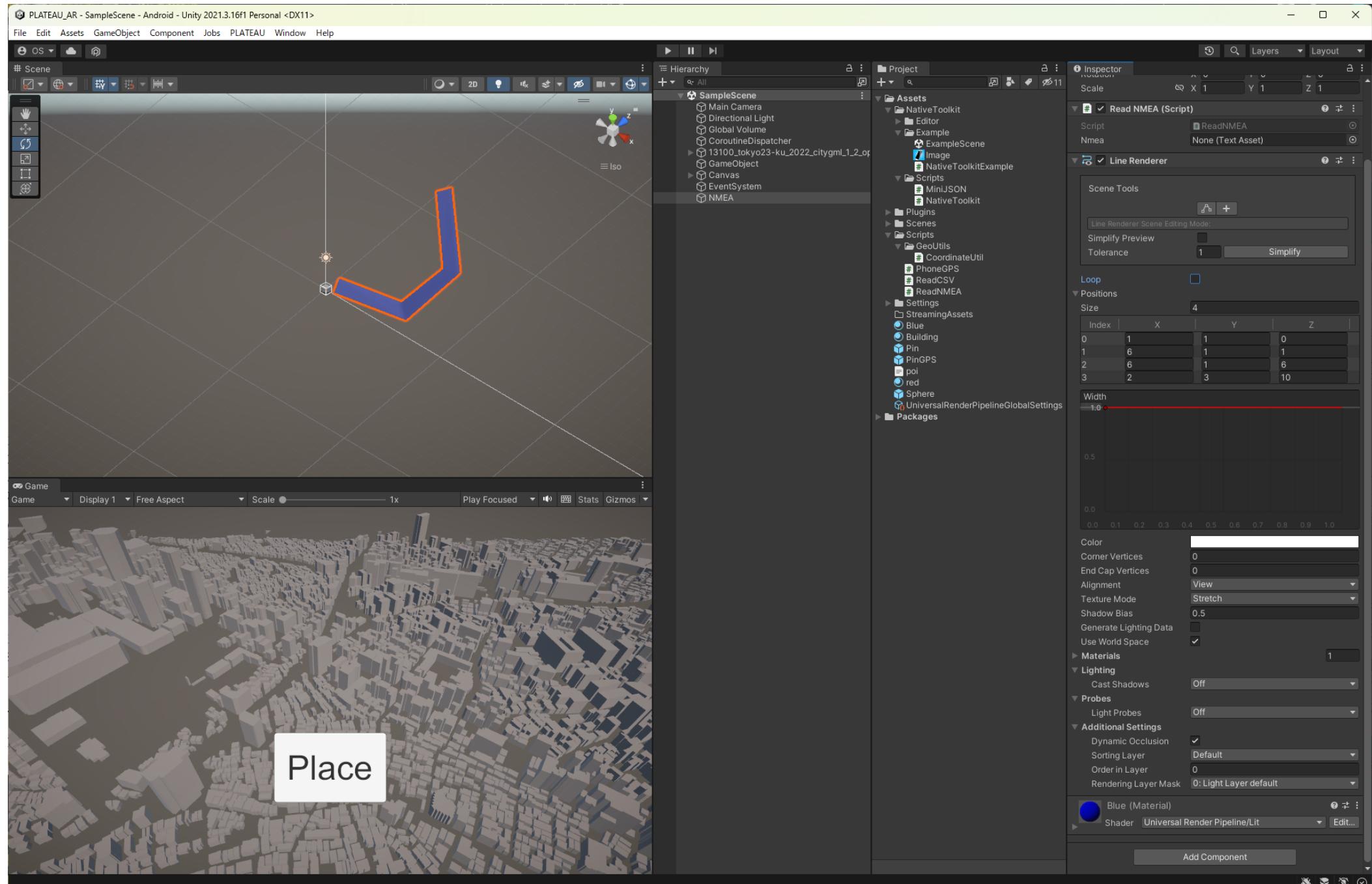
                        positions.Add(new Vector3((float)y, (float)height, (float)x));
                    }
                }
            } catch(Exception e) {
                Debug.LogException(e);
            }
        }
    }
}
```

```
        lr.positionCount = positions.Count;
        lr.SetPositions(positions.ToArray());
    }
}
```

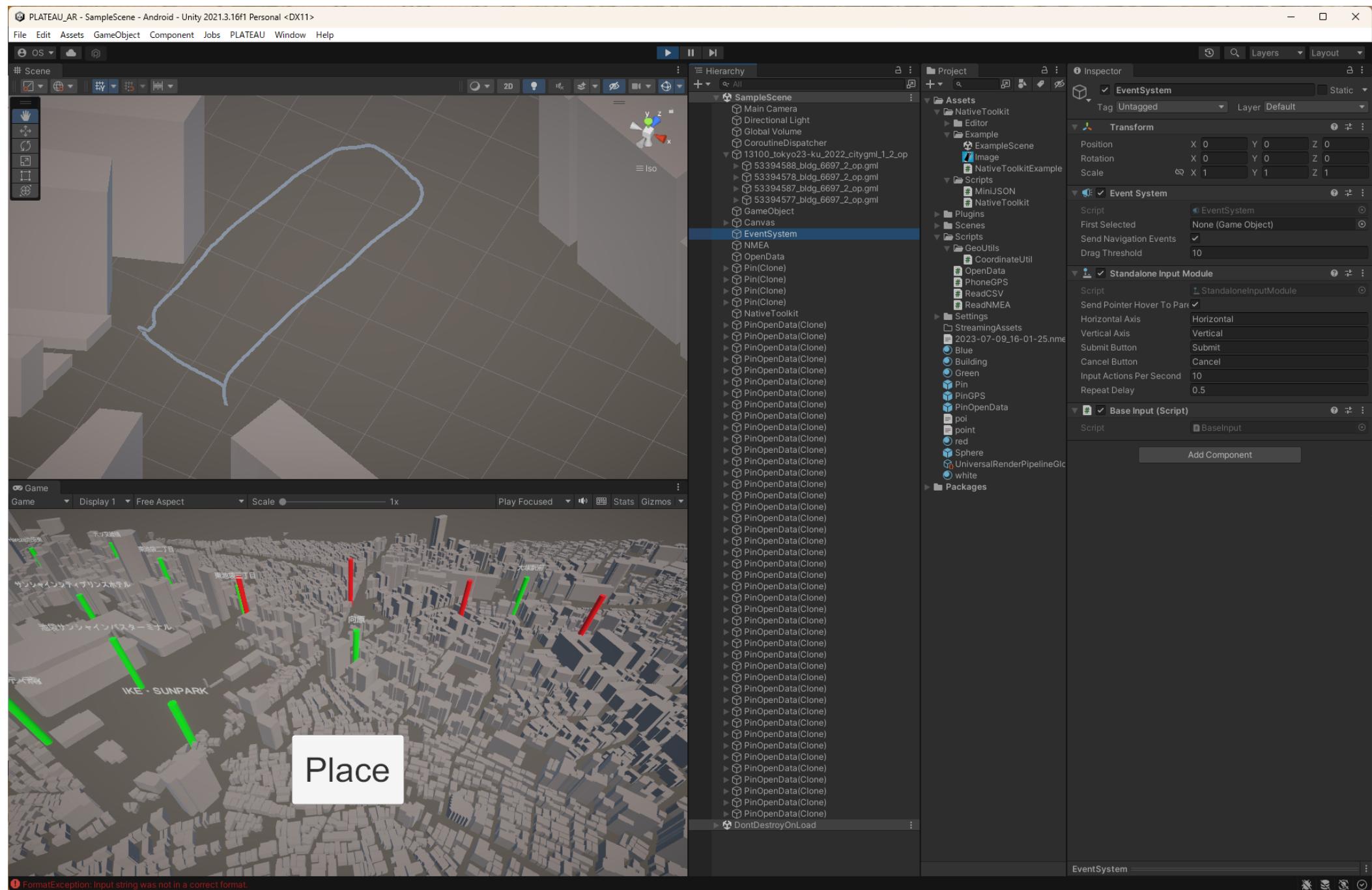
NMEA形式では、緯度、経度の値が、DDMM.MMMMやDDDMM.MMMMのように表されます。これをDD.DDDDDDDの形式に変換するための処理をパース時に実行しています。

また、NMEA形式では、座標以外にも衛星の位置や電波の受信状況などの情報が記載されているため、座標の行だけを判定して処理しています。

UnityのLineRendererは、線を描画するための仕組みで、LineRendererコンポーネントをゲームオブジェクトにアタッチして使用します。



GPSが利用する座標系はWGS84ですが、JGD2011と実用上ほとんど同じため、ここではWGS84からJGD2011の座標系の変換は行いません。GPS受信機からのWGS84の座標をJGD2011とみなして平面直角座標に変換し、表示しています。



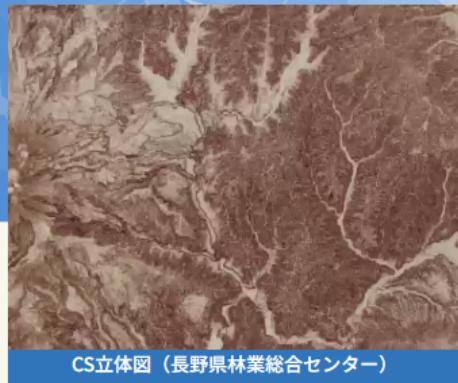
オープンデータを重ねる

近年、自治体やさまざまな会社などが、地理情報データを公開しています。オープンデータと言われるこれらのデータは、個人でもダウンロードして活用することができ、PLATEAUの3D都市モデルを合わせてさまざまなユースケースを作り出すことができるでしょう。

ここでは、オープンデータの紹介と、QGISを使ってデータ変換し、Unityに読み込む方法を解説します。

オープンデータにはさまざまなものがありますが、地理情報のオープンデータの集まっているサイトとして、[G空間情報センター](#)があります。

データの力をまちの力に



CS立体図 (長野県林業総合センター)



断面交通量データ (AIGID)



リアル3D都市モデル (アジア航測)

■法務省 登記所備付地図データに関するお知らせ

1. 初めてご利用の方は、[新規ユーザー登録](#)をお願いします。
※データダウンロードのみの場合は、組織作成申請は不要です。
2. ユーザー登録完了後、G空間情報センターにログインをしてください。
3. 登記所備付地図データのダウンロードは[こちら](#)
4. 当センターでシェープファイル、GeoJSONファイルに変換した登記所備付地図データのダウンロードが可能です。
※ダウンロードにあたりユーザー登録は不要です。
5. XMLデータのダウンロード方法は[こちら](#)にて解説しております。その他、ご不明な点は[FAQ](#)をご確認ください。

データセット数

12,036

ファイル数

70,518

登録組織数

611



また、国土交通省の[国土数値情報](#)にも、多くの地理情報オープンデータが掲載されています。

このサイトでは、地形、土地利用、公共施設などの国土に関する基礎的な情報をGISデータとして整備し、無償で提供しています。

■ 重要なお知らせ

■ 2023/06/02

システムのメンテナンスに伴うホームページの運営について

[一覧を見る](#)

■ データをダウンロードする



国土数値情報

[国土数値情報とは？](#)

位置参照情報

[位置参照情報とは？](#)

国土調査

[国土調査とは？](#)

地図で見る

[地図で見るとは？](#)

今回は、国土交通省の国土数値情報から、バス停留所のデータを変換して表示してみます。

まず、[国土数値情報のページ](#)から、バス停留所のリンクを選択します。

国土数値情報 ▼

位置参照情報 ▼

国土調査 ▼

地図で見る ▼

ジオコーディング (住所⇒緯度経度)

このサイトでは、地形、土地利用、公共施設などの国土に関する基礎的な情報をGISデータとして整備し、無償で提供しています。



新着データ



ランキング



カテゴリー



データ一覧

新着データ (2023年6月時点)

NEW
バスルート

NEW
バス停留所

NEW
国・都道府県の機関

NEW
土地利用3次メッシュ

NEW
土地利用細分メッシュ

NEW
都市地域土地利用細分メッシュ

NEW
土地利用詳細メッシュ

NEW
世界文化遺産

NEW
世界自然遺産

NEW
土砂災害警戒区域

NEW
行政区域

NEW
高速道路時系列

NEW
鉄道

NEW
鉄道時系列

NEW
駅別乗降客数



各データのページ(例としてバス停留所)では、そのデータの諸元が記載されています。座標系やデータ形状、データ構造などが確認できます。また、ライセンスも確認してください。データによっては商用利用が制限されている場合もあります。

国土数値情報 ▼

位置参照情報 ▼

ジオコーディング (住所⇒緯度経度)

国土調査 ▼

地図で見る ▼

千葉 (GML形式)	世界測地系	令和4年	0.34MB	P11-22_12_GML.zip	
千葉 (シェーブ、geojson形式)	世界測地系	令和4年	1.68MB	P11-22_12_SHP.zip	
千葉	世界測地系	平成22年	0.75MB	P11-10_12_GML.zip	
東京 (GML形式)	世界測地系	令和4年	0.44MB	P11-22_13_GML.zip	
東京 (シェーブ、geojson形式)	世界測地系	令和4年	1.83MB	P11-22_13_SHP.zip	
東京	世界測地系	平成22年	0.79MB	P11-10_13_GML.zip	
神奈川 (GML形式)	世界測地系	令和4年	0.38MB	P11-22_14_GML.zip	
神奈川 (シェーブ、geojson形式)	世界測地系	令和4年	1.71MB	P11-22_14_SHP.zip	
神奈川	世界測地系	平成22年	0.69MB	P11-10_14_GML.zip	
甲信越・北陸地方 (GML形式)	世界測地系	令和4年	0.93MB	P11-22_54_GML.zip	
甲信越・北陸地方 (シェーブ、geojson形式)	世界測地系	令和4年	4.12MB	P11-22_54_SHP.zip	
新潟 (GML形式)	世界測地系	令和4年	0.26MB	P11-22_15_GML.zip	
新潟 (シェーブ、geojson形式)	世界測地系	令和4年	1.35MB	P11-22_15_SHP.zip	
新潟	世界測地系	平成22年	0.51MB	P11-10_15_GML.zip	
富山 (GML形式)	世界測地系	令和4年	0.10MB	P11-22_16_GML.zip	



ここでは、「東京（シェープ、geojson形式）」の令和4年版のデータをダウンロードします。Zipファイルなので、適宜展開してください。

ここでダウンロードされるデータは、シェープファイルというGISのファイル形式と、geojson形式のファイルです。

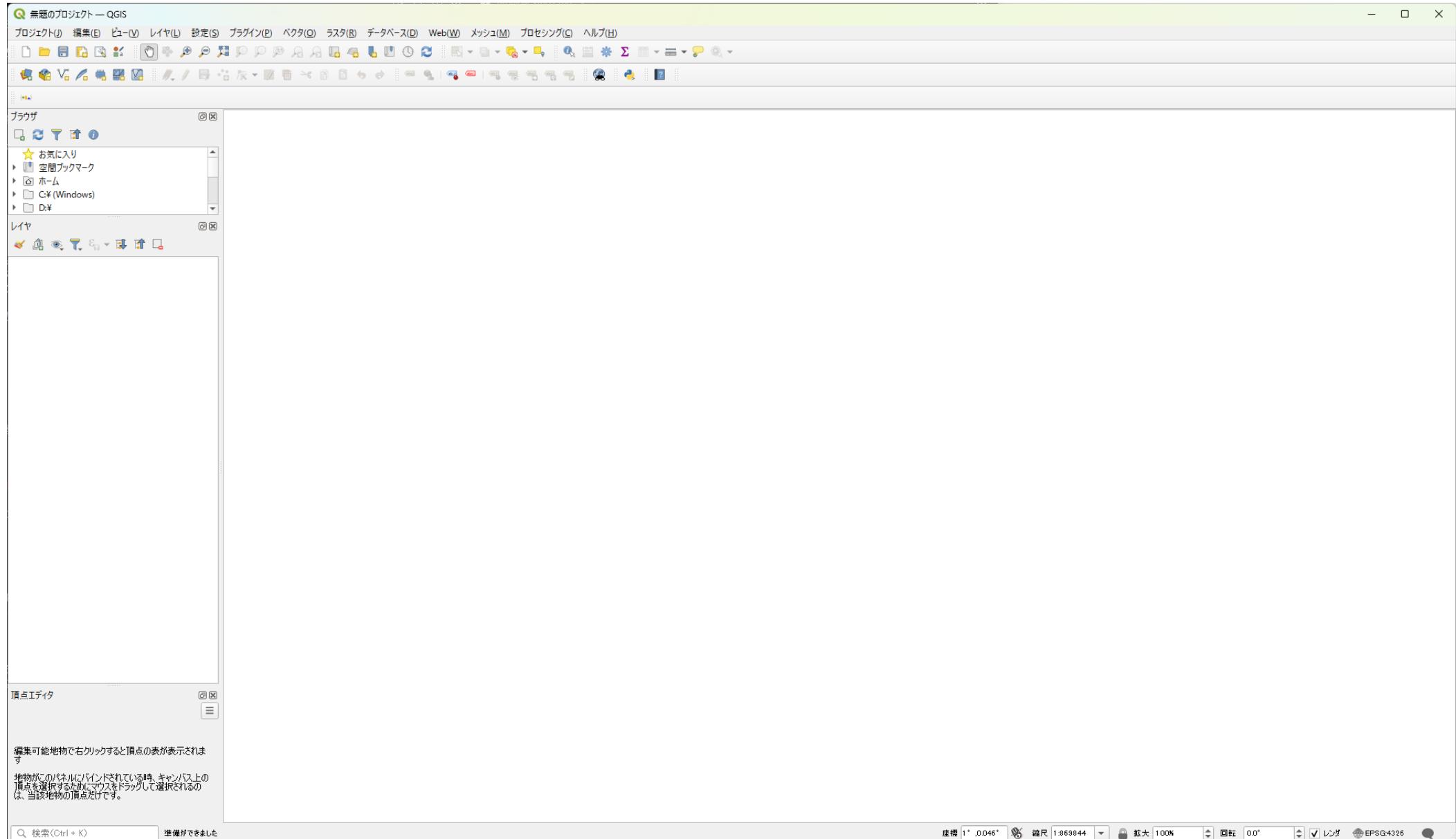
JSONパーサーを使ってgeojsonを読み込むこともできますが、ここではシェープファイルを使うことにします。

現在でも多くの地理情報がシェープファイルで提供されているので、その扱い方をここで解説します。

読み込んだシェープファイルはQGISで読み込み、CSVに変換してUnityに表示しましょう。

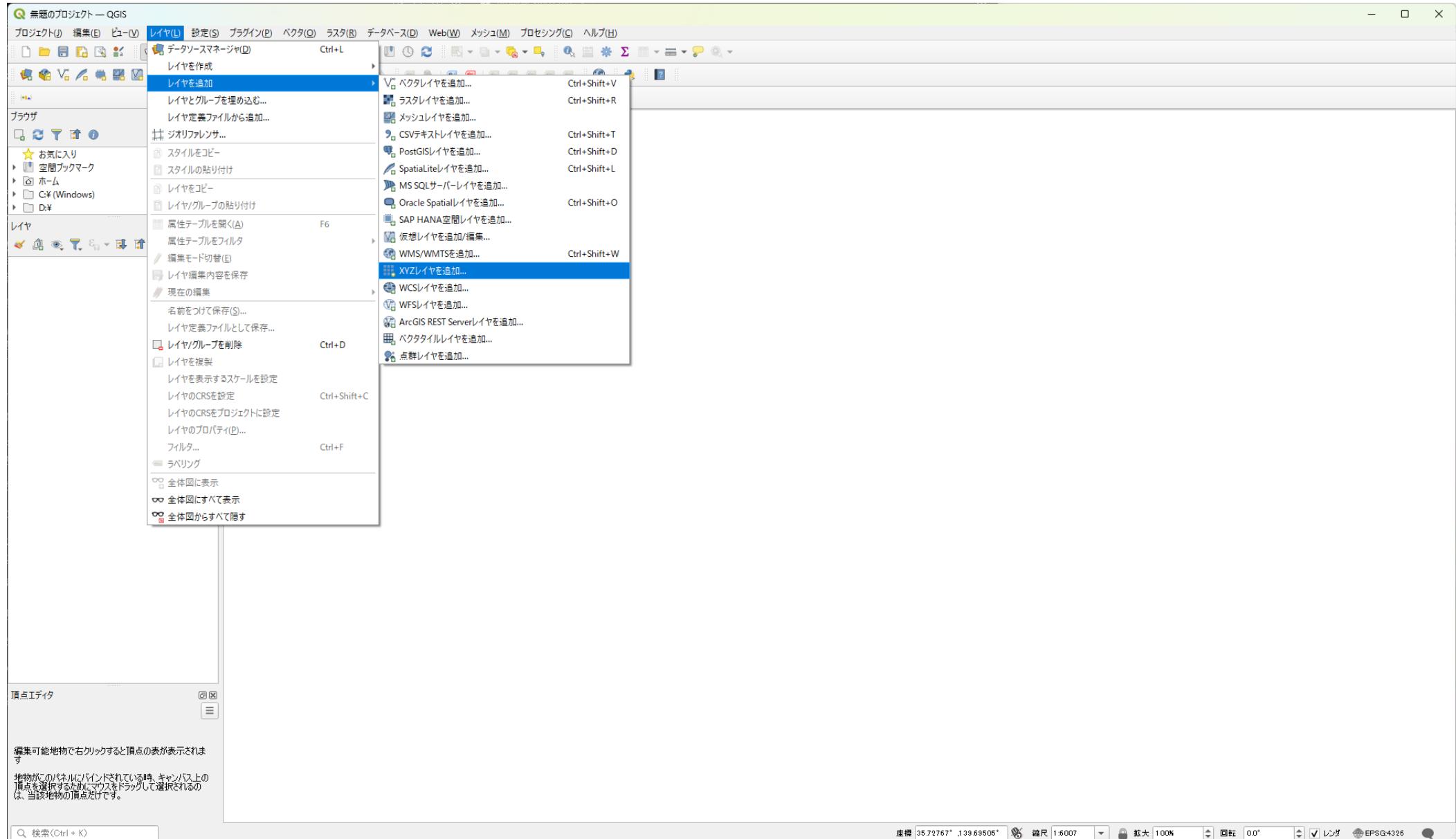
QGISを開き、新規プロジェクトを作成します。

QGISの基本的な操作は、22年度PLATEAUチュートリアルコンテンツの[TOPIC 5 GISで活用する \[1/3\]](#) も参考にしてください。

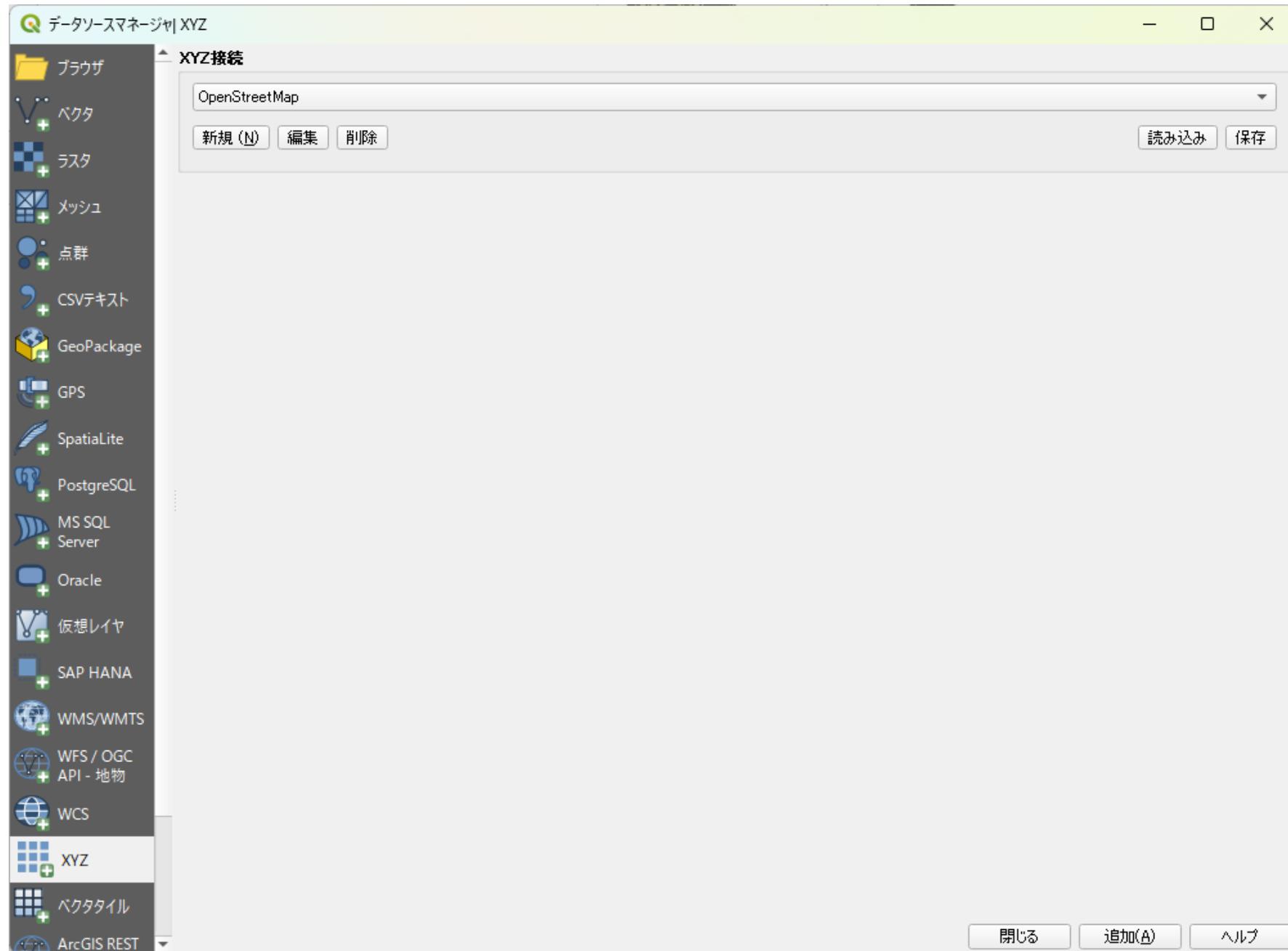


最初にベースマップとなる地図を読み込みます。[オープンストリートマップ](#)というオープンな地図データを使います。オープンストリートマップは、マッパーという有志のコミュニティによって構築されているオープンデータです。

「レイヤ」→「レイヤを追加」→「XYZレイヤを追加」を選択します。



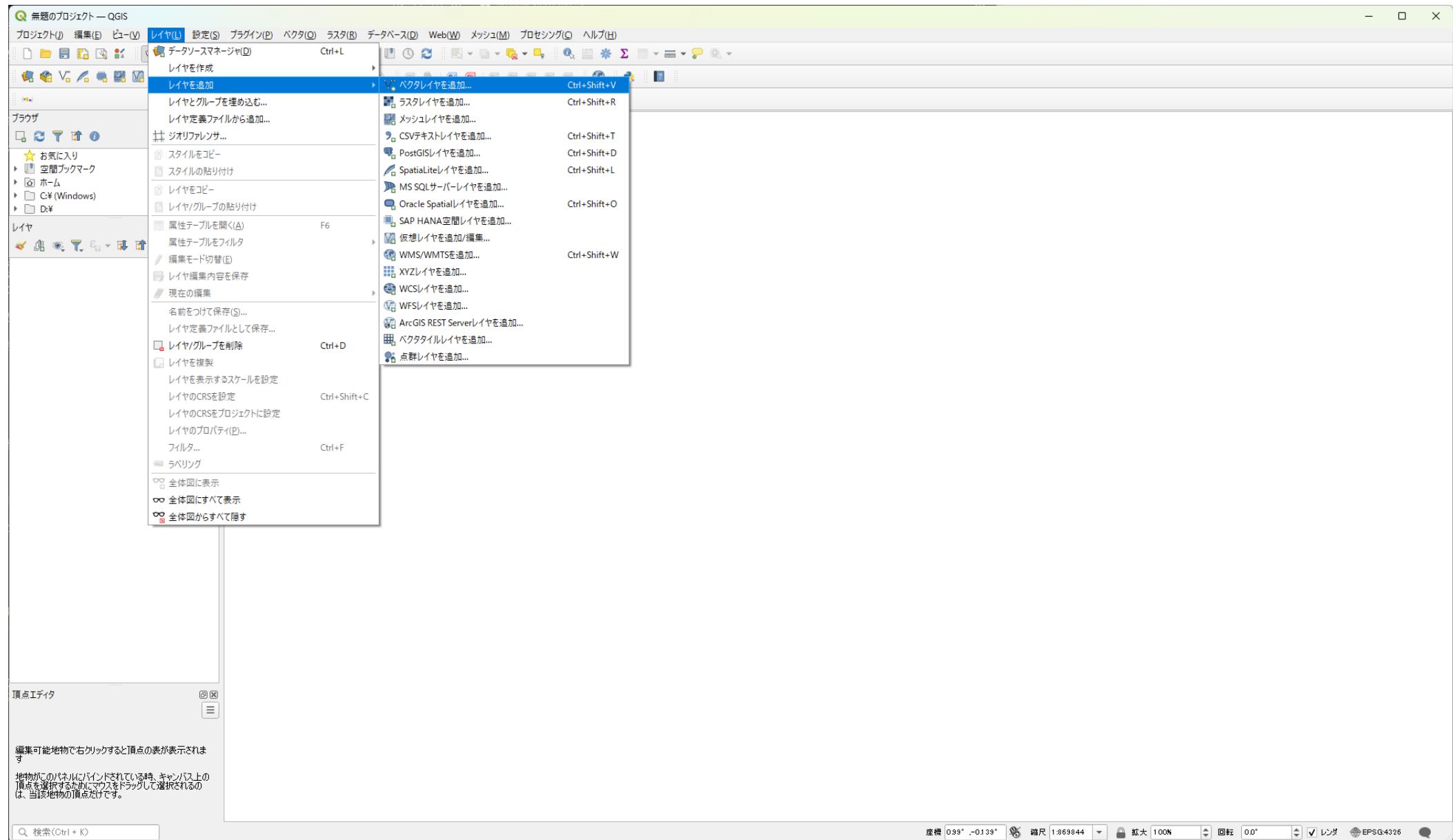
プルダウンメニューでプリセットの中にあるOpenStreetMapを選択して、「追加」ボタンを押してください。



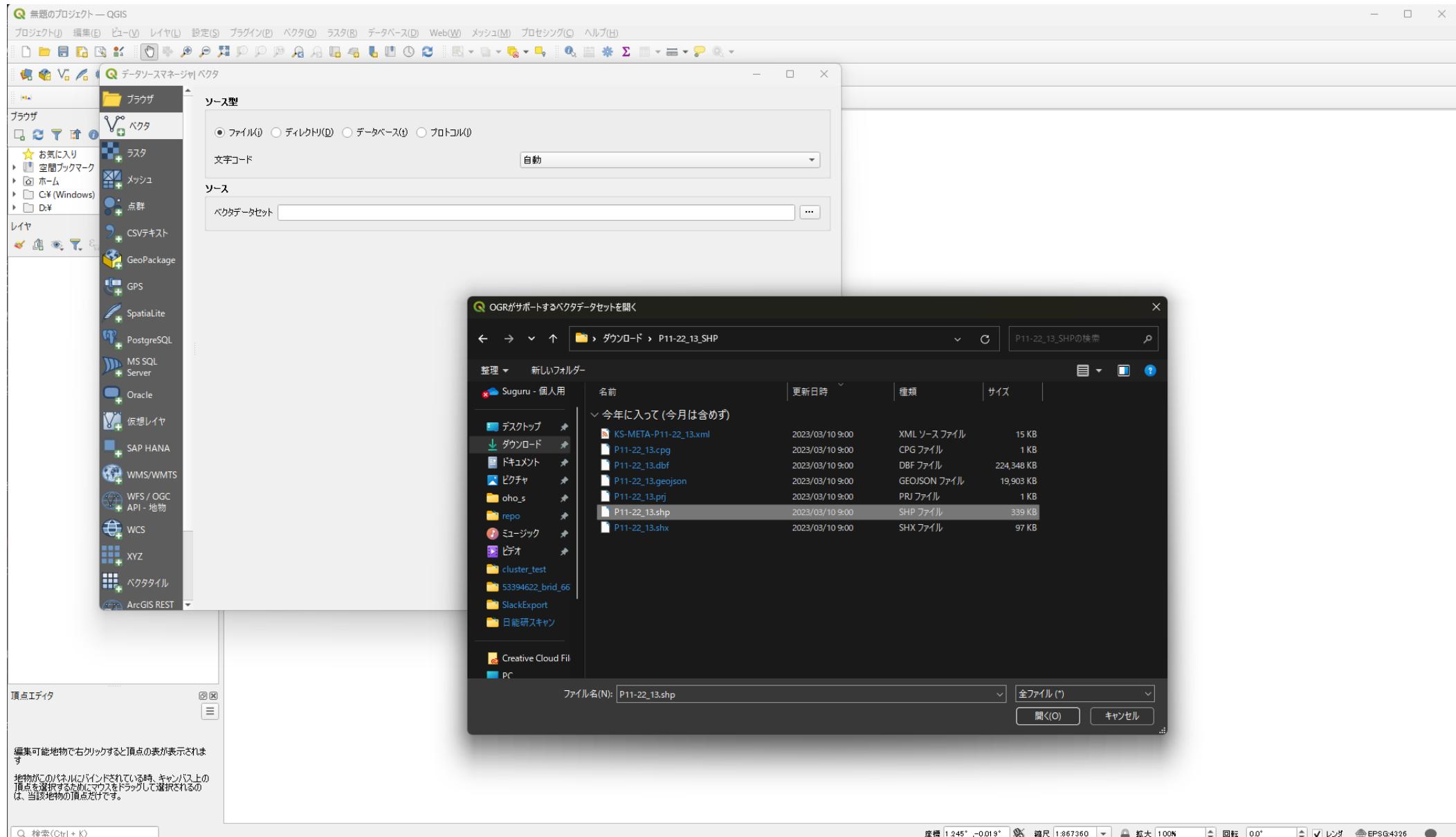
ダイアログは、「閉じる」を押して閉じます。

次に、ダウンロードしたバス停留所のデータを読み込みます。

「レイヤ」→「レイヤを追加」→「ベクタレイヤを追加」を選択します。

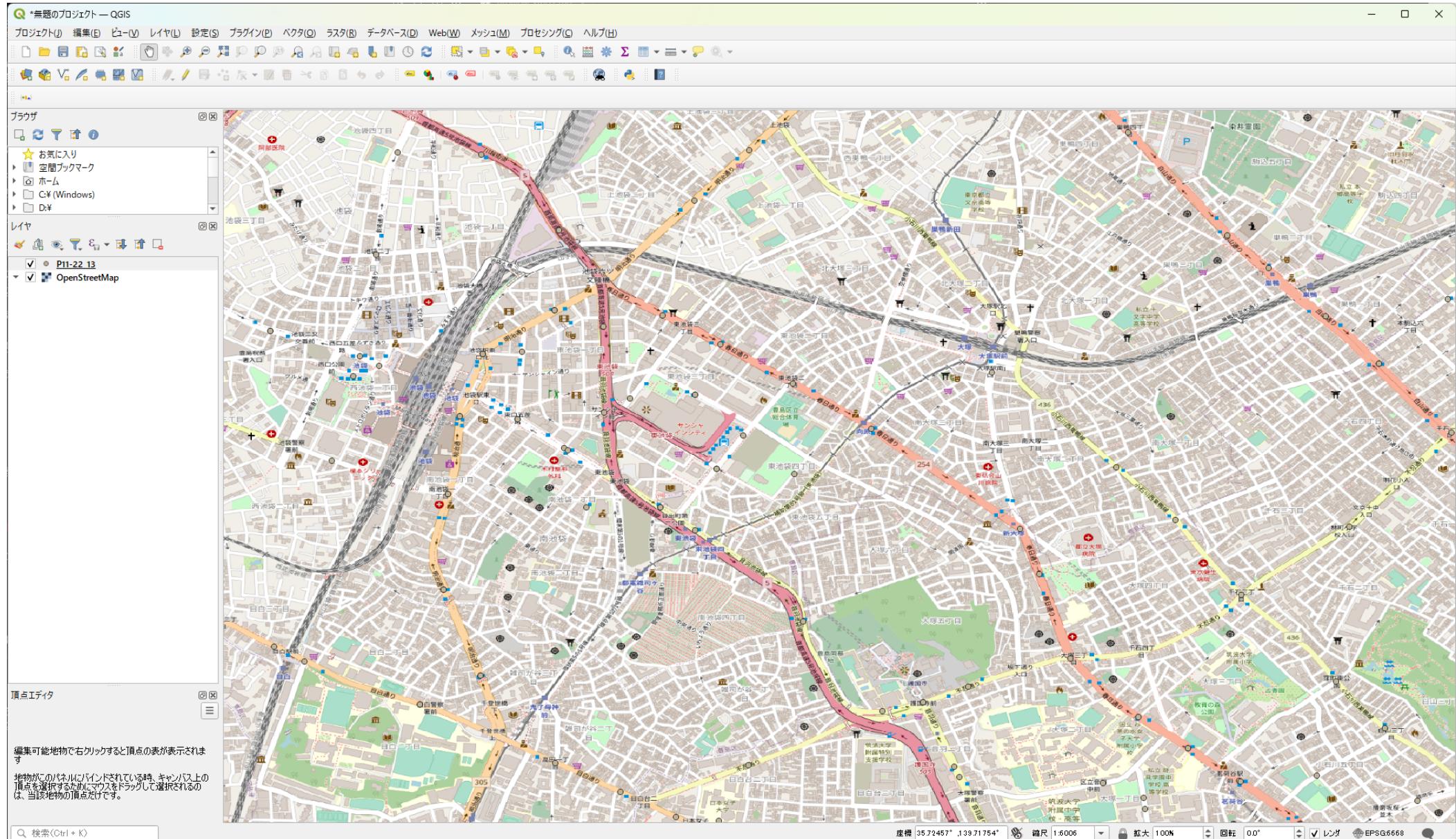


「ソース」のファイル選択ダイアログで、ダウンロードして展開したフォルダの中の拡張子がshpのファイルを選んでください。



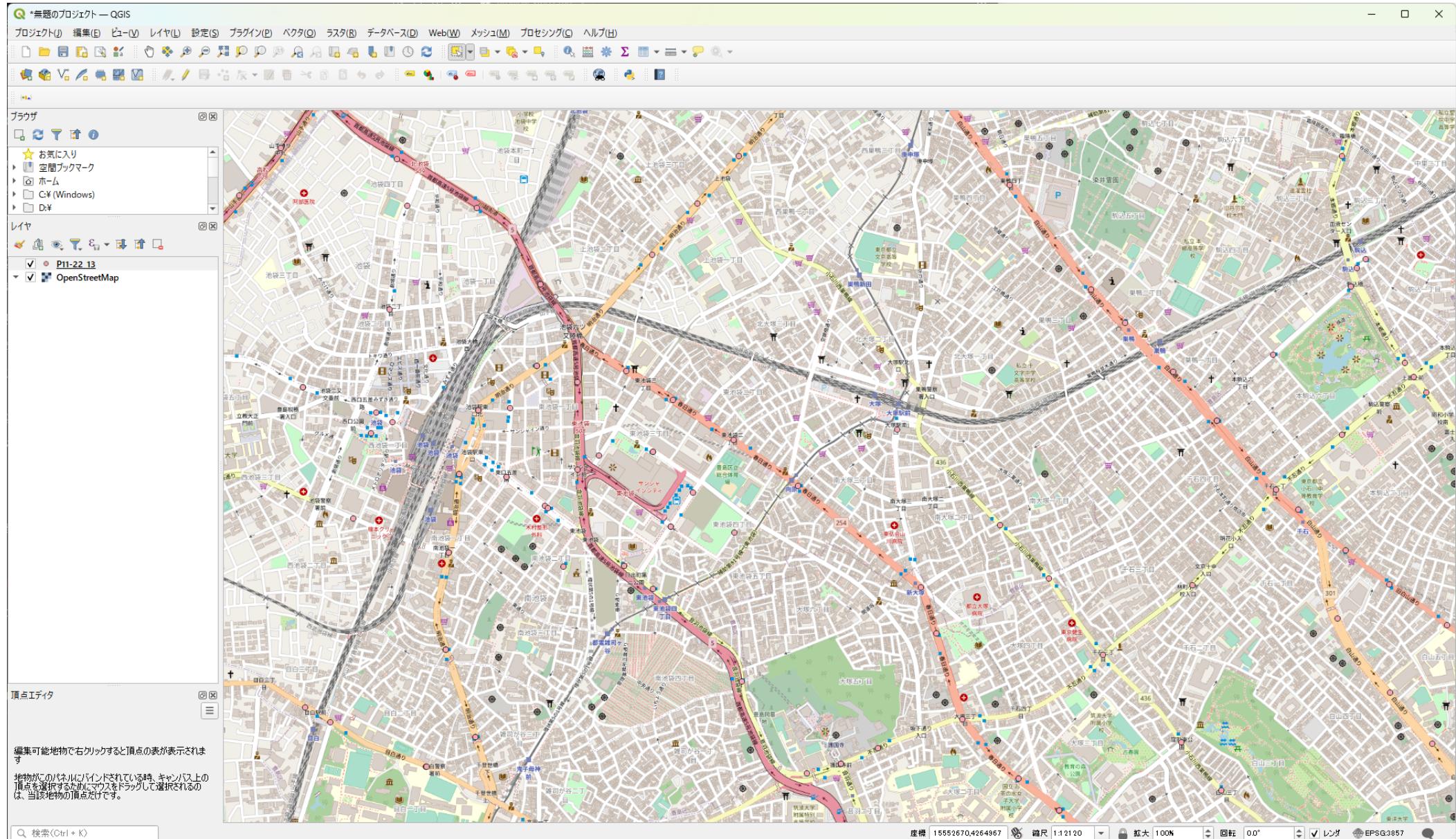
最後にダイアログの「追加」ボタンを押すと、データが読み込まれます。

東京都のあたりにズームインしていくと、図のように、オープンストリートマップの上にバス停の点が表示されています。

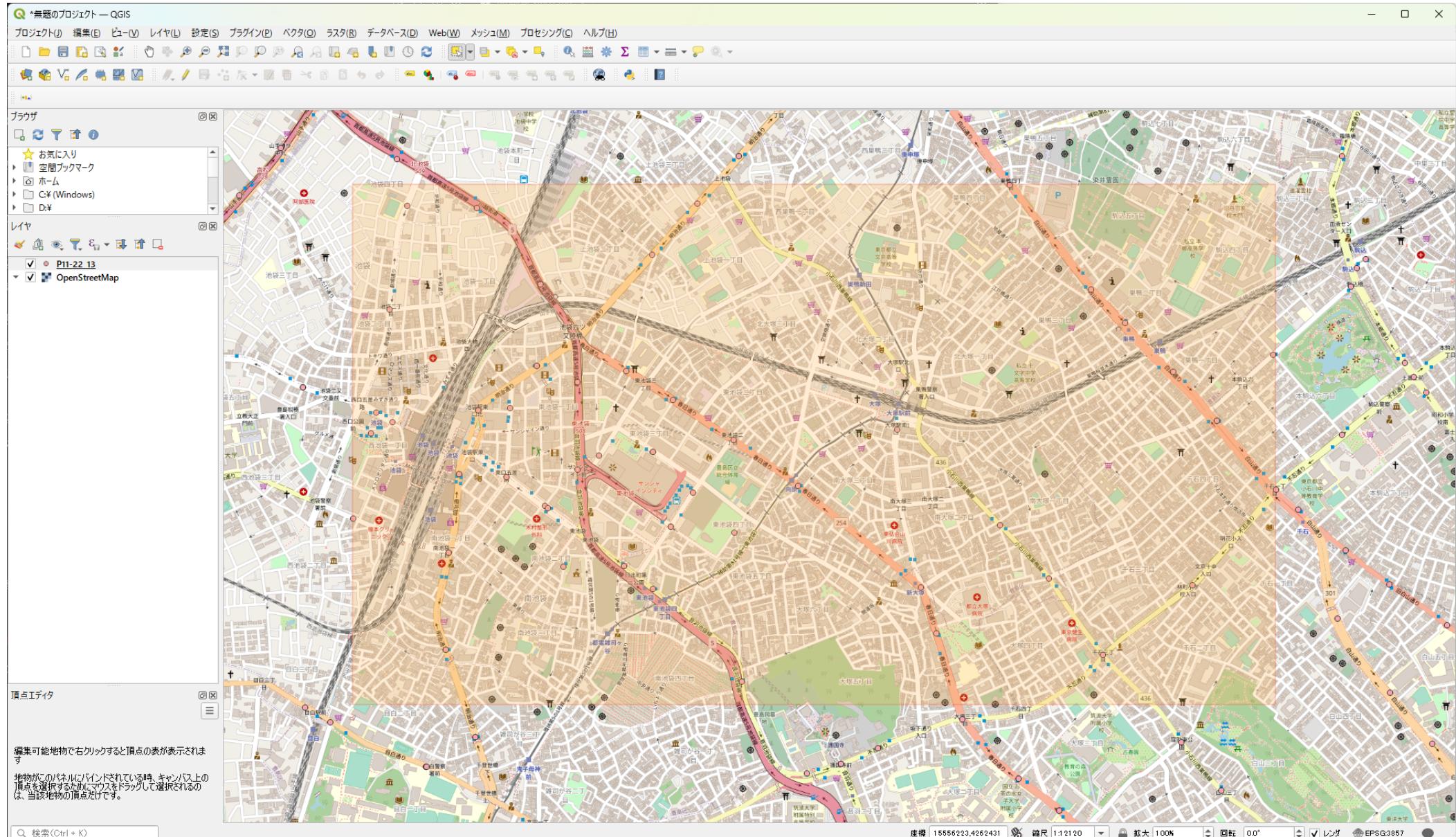


次に、重ね合わせたいデータを切り取り、CSVに保存します。

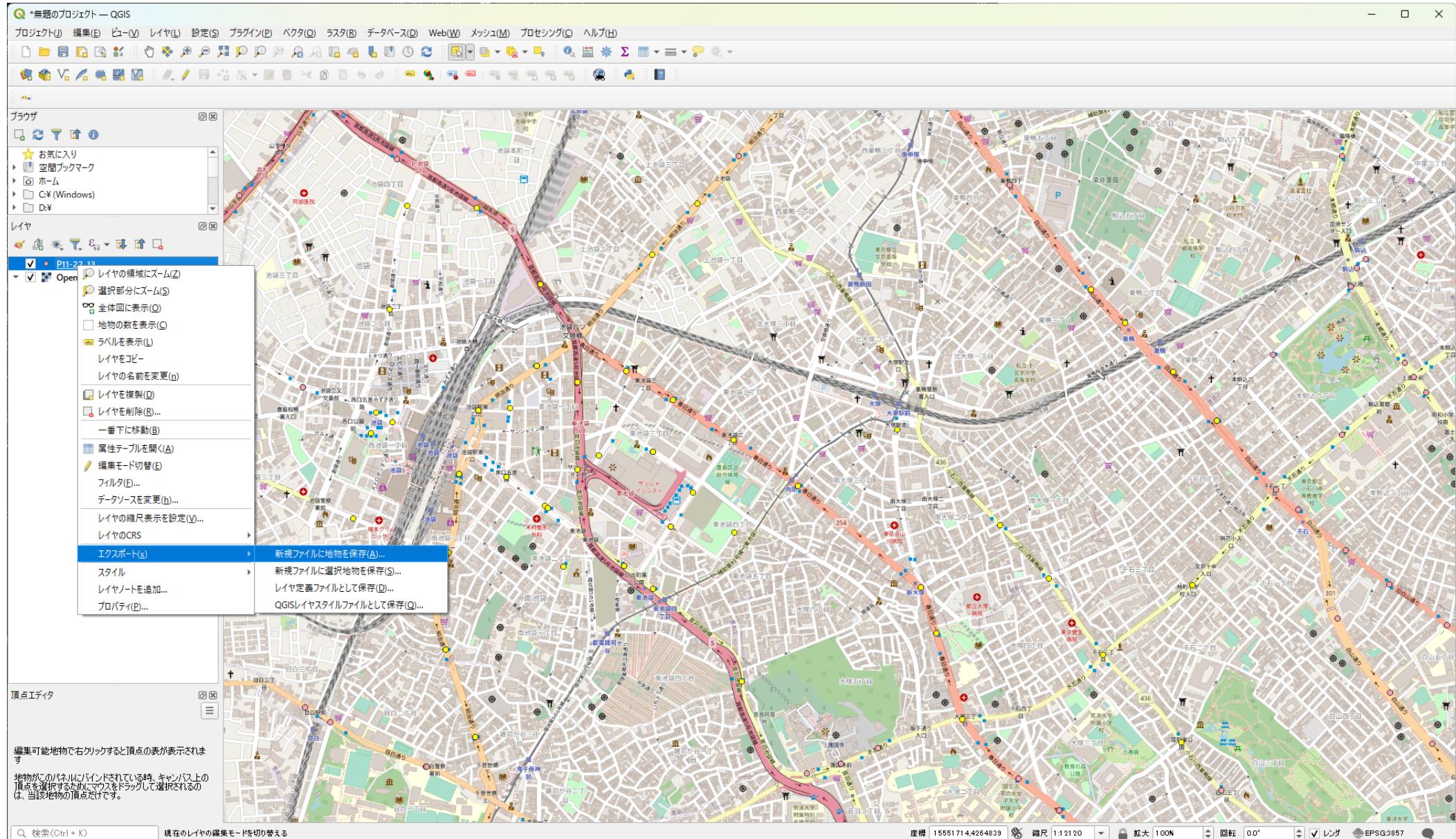
地物の選択ツールを選択します。



マウスでドラッグ＆ドロップすることで範囲内の点を選択できます。



点が選択できたら、読み込んだレイヤの上で右クリックして、「エクスポート」→「新規ファイルに地物を保存」を選択します。



エクスポートのダイアログで、ファイル名の指定と「選択地物のみ保存」のチェックを入れ、そのほかの設定も図のよう



これで、つぎのようなCSVファイルが作成されます。

X,Y,P11_001,P11_002,P11_003_01,P11_003_02,P11_003_03,P11_003_04,P11_003_05,P11_003_06,P11_003_07,P11_003_08,P11_003_09,P11_003_10,P11_003_11,P11_003_12,P11_003_13,P11_003_14,P11_003_15,
139.72143740043,35.7377671299951,上池袋三丁目,東京都,"深夜02,王40甲,王40出入,王55,草63,草64",,,,,,,,,,,,"2,2,2,2,2",,,,,,,,,,,
139.710143356298,35.7377110240603,池袋小学校,国際興業(株),池07,,,,,,,,,,,,"1",,,,,,,,,,,
139.712844798114,35.7376363769469,豊島清掃事務所,国際興業(株),"池07,池55,赤51,赤97,光02",,,,,,,,,,,,"1,1,1,1,1",,,,,,,,,,,
139.71967678249,35.7361567137618,上池袋一丁目,東京都,"深夜02,王40甲,王40出入,王55,草63,草64",,,,,,,,,,,,"2,2,2,2,2",,,,,,,,,,,
139.715334410769,35.7352150452483,健康プラザとしま,国際興業(株),池07,,,,,,,,,,,,"1",,,,,,,,,,,
139.736766495305,35.7350547517637,とげぬき地蔵前,東京都,"草63,草64",,,,,,,,,,,,"2,2",,,,,,,,,,,

ここまでたら、CSVファイルの読み込みのプログラムを多少変更してこのデータをUnityに読み込みます。

高さが含まれないので、高さは0としています。また、最初の行がヘッダーになっているので、数値としてパースできなかった時の例外処理をするようにして対応しました。

```

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OpenData : MonoBehaviour
{
    public TextAsset csv;
    public GameObject go;

    // Start is called before the first frame update
    void Start()
    {
        var lines = csv.text.Split("\n");
        foreach (var line in lines)
        {
            var tokens = line.Split(",");
            try
            {
                double lon = double.Parse(tokens[0]);
                double lat = double.Parse(tokens[1]);
                double height = 0d;
                string name = tokens[2];

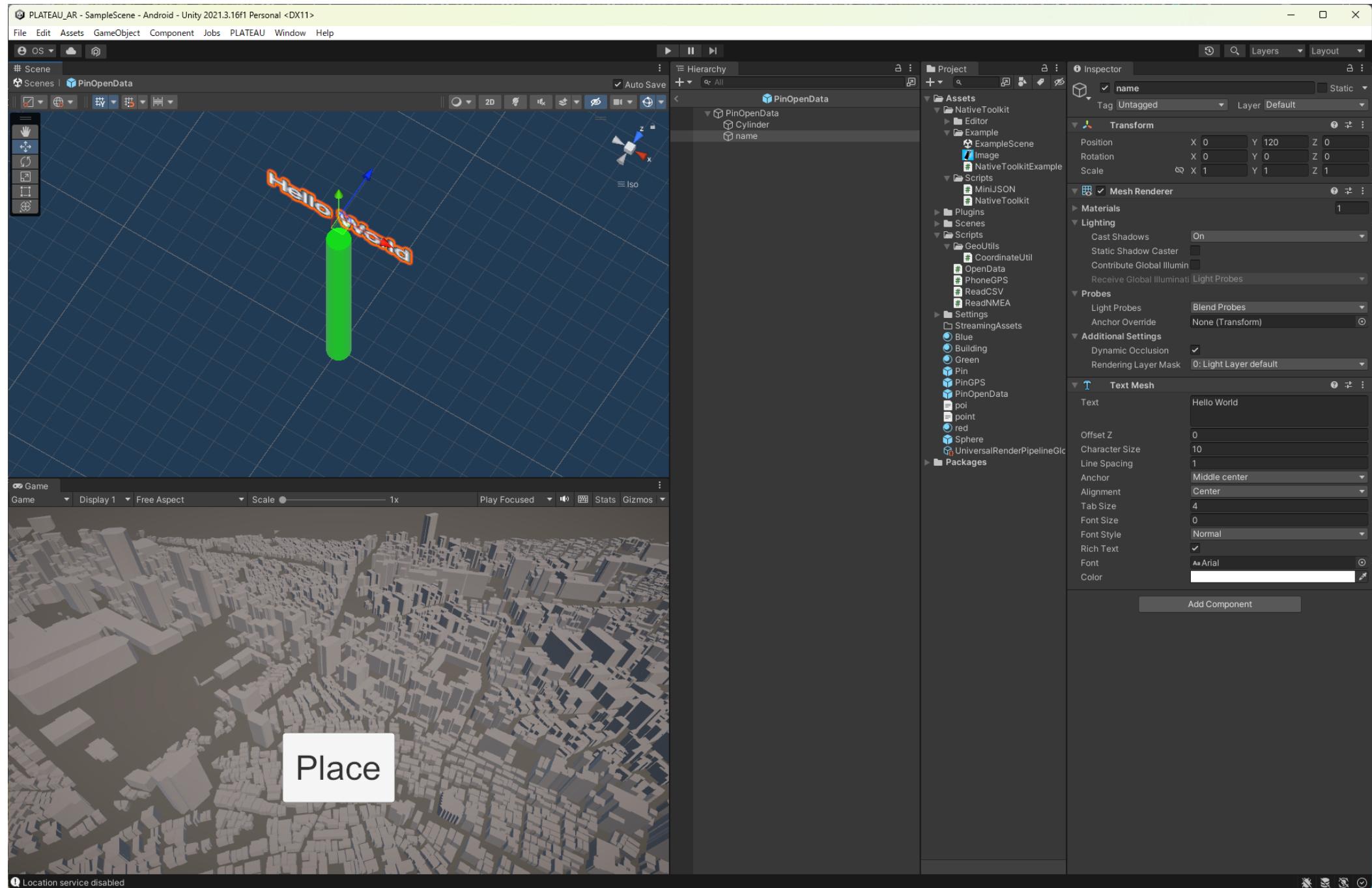
                (var x, var y) = CoordinateUtil.JGD2011ToPlaneRectCoord(lat, lon, 36d, 139.8333333333d);

                x = x + 29787.4390;
                y = y + 9810.3435;

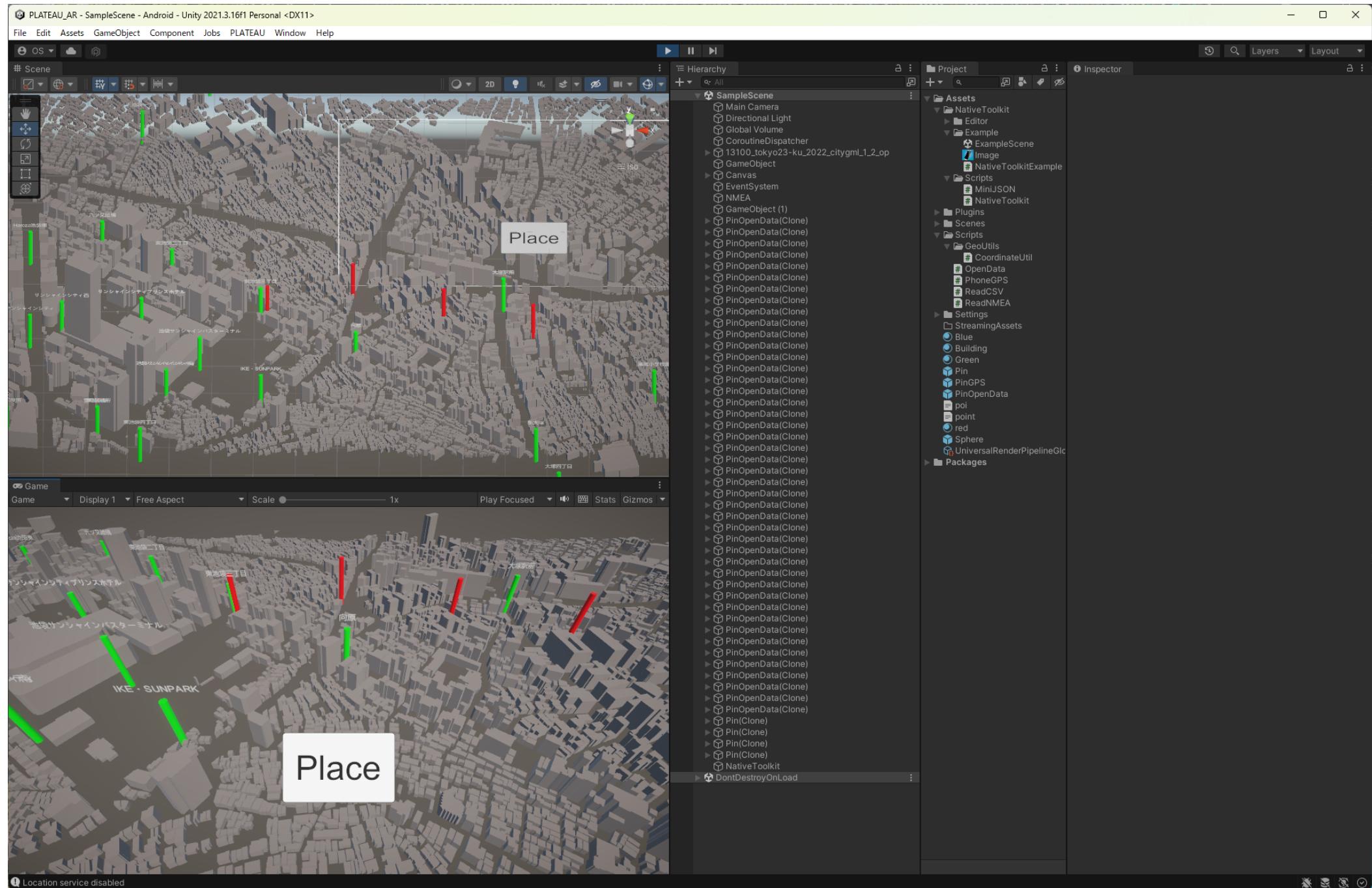
                var obj = Instantiate(go, new Vector3((float)y, (float)height, (float)x), Quaternion.identity);
                obj.transform.GetComponentInChildren<TextMesh>().text = name;
            } catch (Exception e)
            {
                Debug.LogException(e);
            }
        }
    }
}

```

また配置するPrefabのゲームオブジェクトにテキストを追加して、バス停の名前を表示するようにしています。



ここまでで実行すると図のようになります。



コラム 点以外のジオメトリの読み込みについて

シェープファイルには、点以外にも、線、面などのジオメトリが格納されていることがあります。また、シェープファイル以外でも、さまざまな形式のデータで、これらの線や面のジオメトリを読み込みたいときがあります。線に関しては、NMEAの項で説明したLineRendererで描画する方法もありますが、UnityのMeshでは、各頂点をスクリプトから設定する仕組みがあり、それを利用して線や面を描画することもできます。

面の場合は、Unityでは三角形以外の多角形を直接描画できないので、多角形を三角形に分割するアルゴリズムを利用します。Earcut(参考：[MapboxのEarcutアルゴリズムのJavaScript実装](#))などが有名なので、参考にしてください。

また、[WKT\(Well Known Text\)](#)というジオメトリ情報をテキスト化してアプリ間をとりまわすためのフォーマットが定義されています。QGISでもエクスポート時にこれを選択でき、Unity側でパーサーを作成し読み込むことができるでしょう。

ARアプリの開発

最後に、ここまでさまざまな位置情報を表示してきたものを実際のARアプリにまとめます。ここでは、あらかじめアプリに内蔵した位置情報をAR表示する機能、新しい位置情報を端末のGPSから記録してAR表示する機能、3D都市モデルを表示する機能を持ったARアプリを作成します。

PLATEAUモデルの調整

ARで見やすいように3D都市モデルの表示を工夫します。

今回は、完全に透明なオクルージョンではなく、3D都市モデルが半透明で見えるようなマテリアルを作成します。ただ単に半透明で表示すると、建物同士の重なりが多い部分が白飛びして見えにくくなるので、深度を考慮した半透明のシェーダーを書きます。

仕組みは、まず深度バッファのみ更新し、その上に深度バッファを検証しながら半透明で描画する2パスのシェーダーとなります。

今回のプロジェクトがURPなので、URPに対応したシェーダーを書きます。また、このシェーダーを使うマテリアルを作成します。

```
Shader "Custom/Building"
{
    Properties
    {
        _Color("Color", Color) = (1,1,1,1)
    }
    SubShader
    {
        Tags {"Queue" = "Transparent" "IgnoreProjector" = "True" "RenderType" = "Transparent"}
        LOD 200

        Pass {
            Tags {"LightMode" = "SRPDefaultUnlit"}
            ZWrite On
            ColorMask 0
        }

        Pass
        {
            Tags {"LightMode" = "UniversalForward"}
            Tags{ "QUEUE" = "Transparent" "IGNOREPROJECTOR" = "true" "RenderType" = "Transparent" }
            Blend SrcAlpha OneMinusSrcAlpha
            ColorMask RGBA
            ZWrite Off
            ZTest LEqual

            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            fixed4 _Color;

            struct appdata
            {
                float4 vertex : POSITION;
                float3 normal : NORMAL;
            };

            struct v2f
            {
```

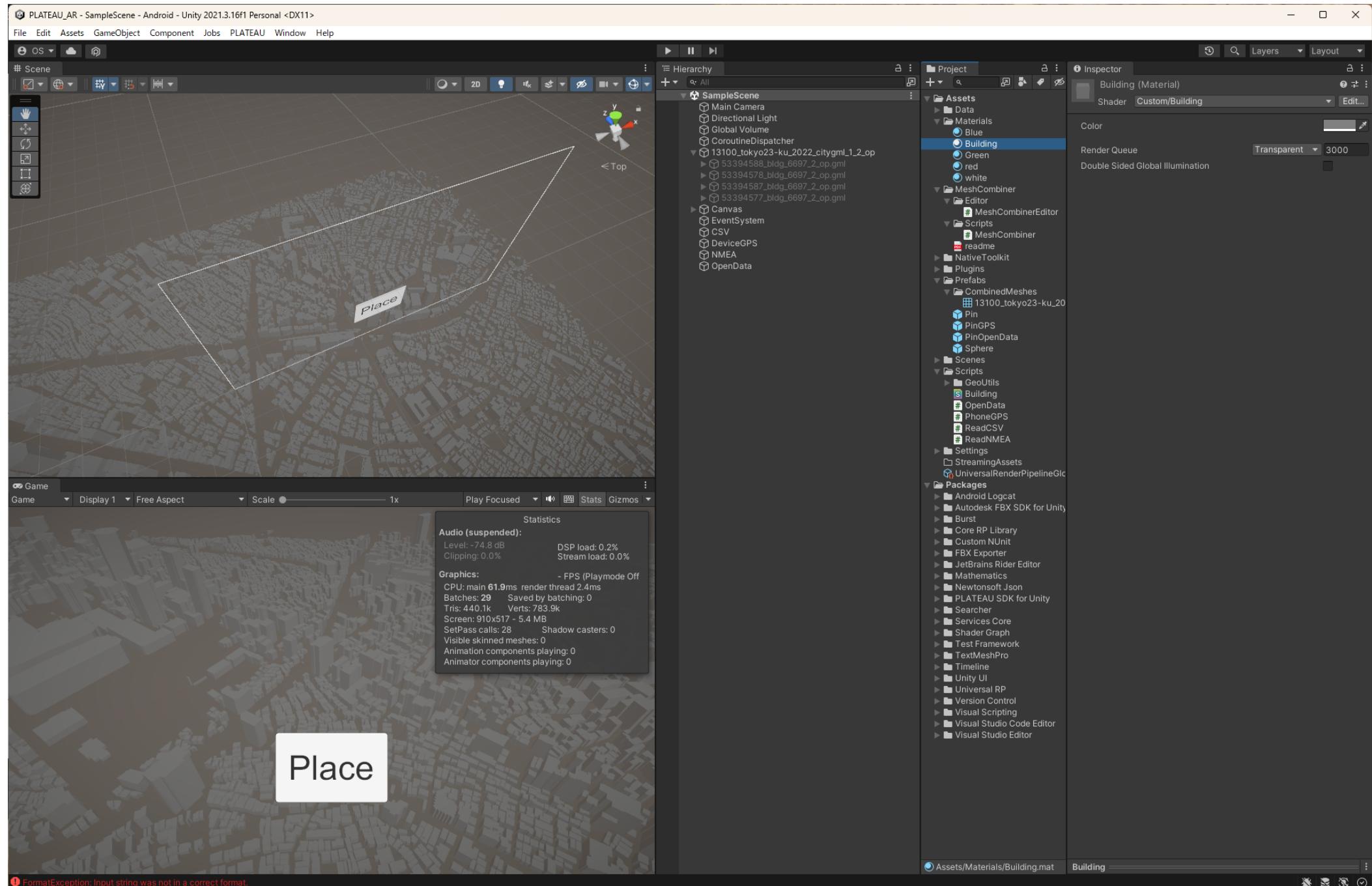
```
float4 vertex : SV_POSITION;
float3 worldNormal : TEXCOORD0;
};

v2f vert(appdata v)
{
    v2f o;
    o.vertex = UnityObjectToClipPos(v.vertex);
    o.worldNormal = UnityObjectToWorldNormal(v.normal);
    return o;
}

fixed4 frag(v2f i) : SV_Target
{
    float3 L = normalize(_WorldSpaceLightPos0.xyz);
    float3 N = normalize(i.worldNormal);
    fixed4 col = _Color * max(0, dot(N, L) * 0.5f + (1 - 0.5f));
    return col;
}

ENDCG
}

}
```

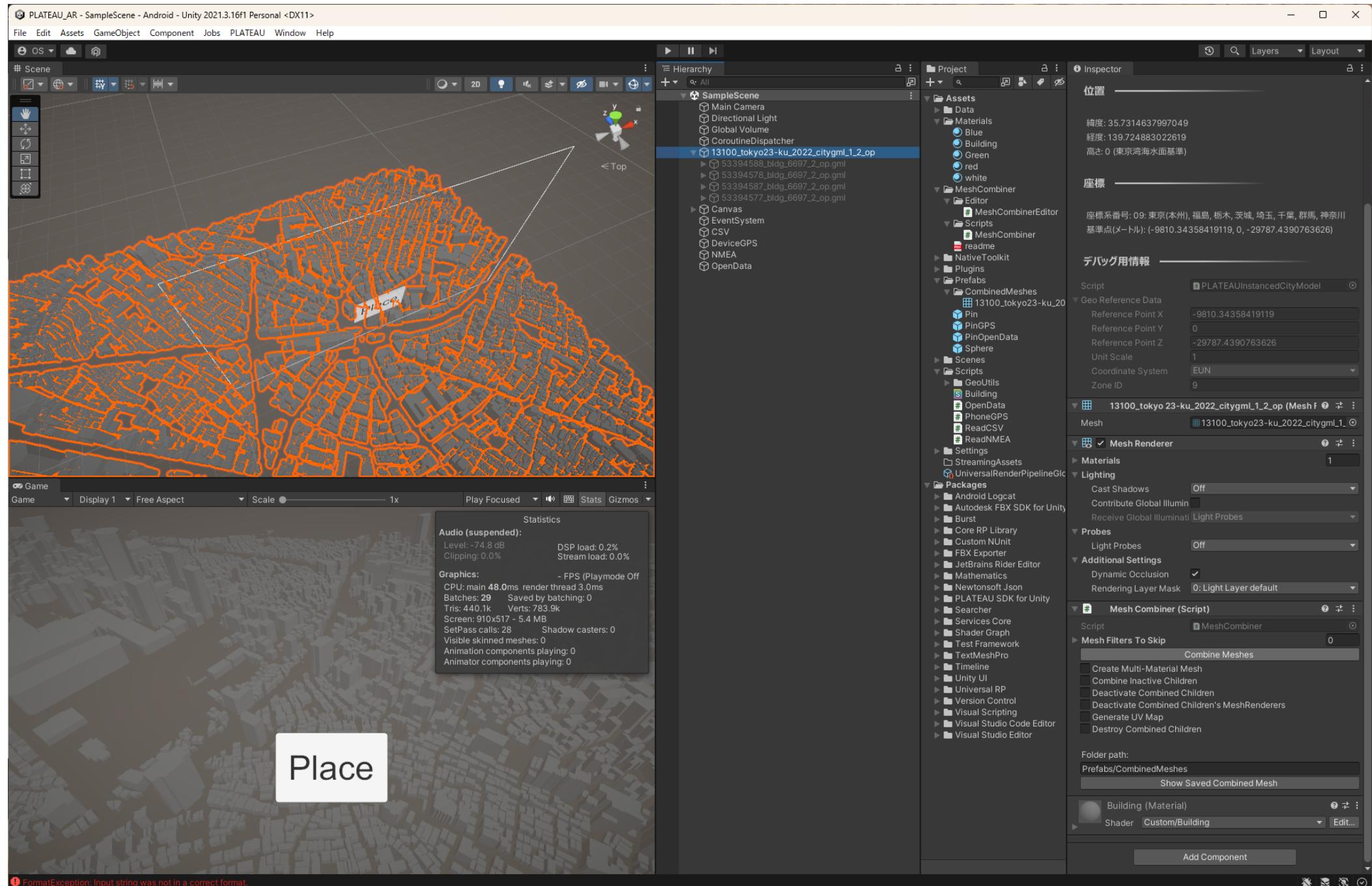


Colorには、任意の色を設定してください。画面では灰色にしています。

さらに、このシェーダーは1つのメッシュでしか有効ではありません。2パスでの描画の場合、メッシュ一つごとに2つのパスが交互に実行されるため、そのメッシュ内では背後が隠れますが、他のメッシュで描画したものは透過してしまうためです。もともとの3D都市モデルは、区画ごとにメッシュが分割されているため、全体を1つのメッシュにまとめてしまいます。

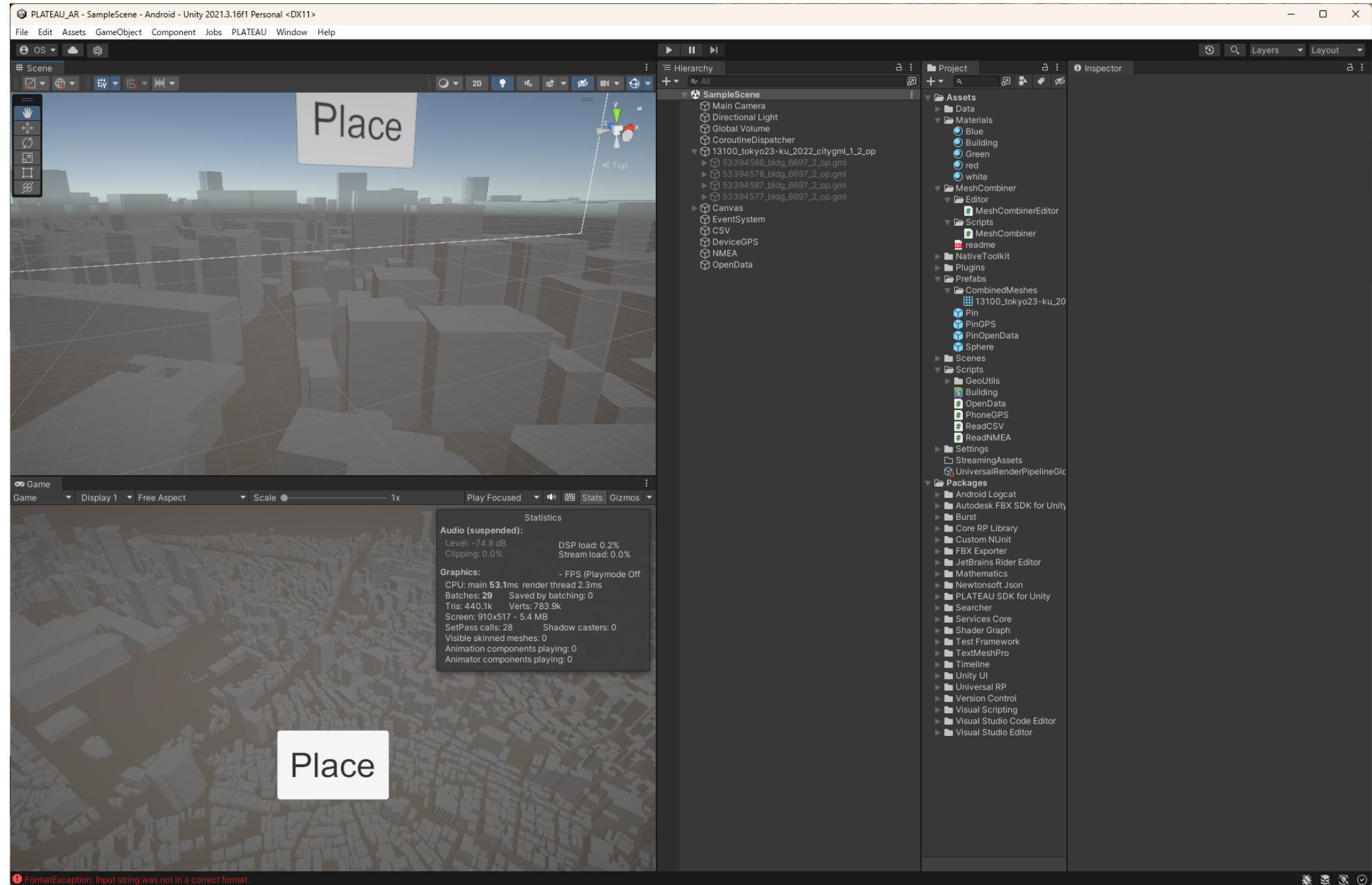
メッシュをまとめるやり方はいくつかありますが、ここでは、Unity Asset Storeから[MeshCombiner](#)をインストールして使います。AssetStoreで入手したらPackageManagerからインストールしてください。

読み込んだ3D都市モデルのゲームオブジェクトにMesh Combinerスクリプトをアタッチし、図のような設定にして「Combine Meshes」ボタンを押すと、すべてのメッシュが1つにまとめられます。



子についている、「53394588_bldg_6697_2_op.gml」などのオブジェクトはDisableにするか、消すなどしておきます。

これでメッシュが統合され、先ほどのマテリアルを設定することで半透明だけど背後は描画されないという意図した見え方になります。



コラム メッシュの統合について

メッシュをまとめるとGame Objectの階層構造の削減やDraw Callの削減により描画が効率化することで、だいたいパフォーマンスが向上します。しかし、広大な範囲のメッシュをまとめてしまうと、まとめる前は表示されないと判断されて描画していなかつた部分の計算がまとめてしまったために判定しなければならなくなり、場合によってはパフォーマンスが悪化します。

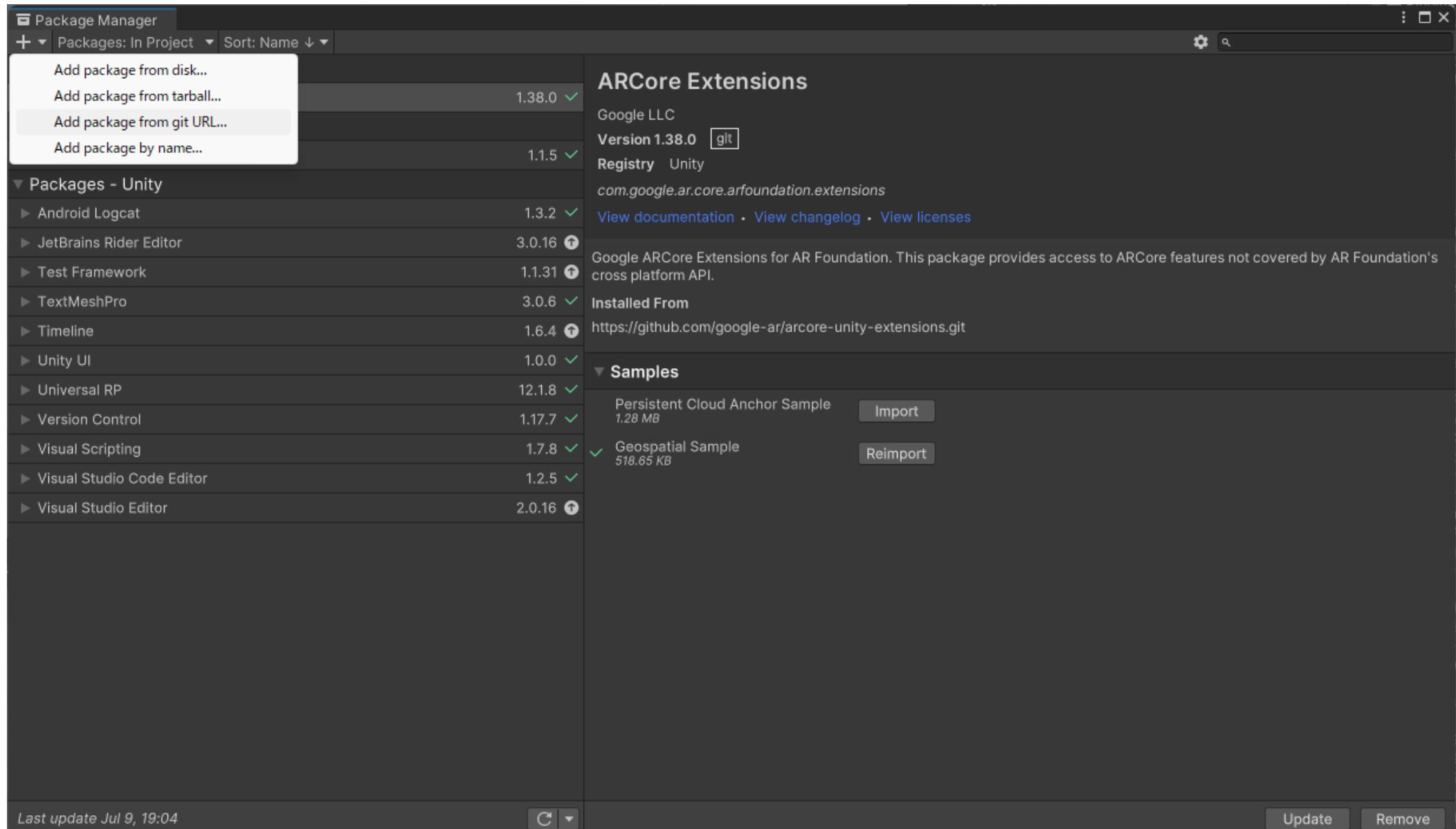
あまりにも広大な範囲のメッシュをまとめるとときは気を付けてください。

GeospatialAPI・ARFoundationの設定

次に、ARFoundationとGeospatialAPIを導入します。詳細は、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。

[Window] メニューから [Package Manager] を開きます。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の追加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ください。

ARCore Extensionsをインストールすると、設定したビルドプラットフォームに応じて、必要な依存関係の拡張も合わせてインストールされます。

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

Project Settings

Adaptive Performance

▼ Analysis

- Android Logcat Settings
- Audio
- Burst AOT Settings
- Editor
- Fbx Export

▼ Graphics

- URP Global Settings
- Input Manager
- Memory Settings
- Package Manager
- Physics
- Physics 2D
- Player
- Preset Manager
- Quality
- Scene Template
- Script Execution Order

▼ Services

- Ads
- Cloud Build
- Cloud Diagnostics
- Collaborate
- In-App Purchasing
- ShaderGraph
- Tags and Layers
- TextMesh Pro
- Time
- Timeline
- UI Builder
- Version Control
- Visual Scripting

▼ XR Plug-in Management

- ARCore
- ARCore Extensions
- ARKit

XR Plug-in Management

Initialize XR on Startup

Plug-in Providers [?](#)

- ARCore
- Oculus
- Unity Mock HMD

Information about configuration, tracking and migration can be found below.

[View Documentation](#)



Project Settings

Adaptive Performance
Analysis
Android Logcat Settings
Audio
Burst AOT Settings
Editor
Fbx Export
Graphics
URP Global Settings
Input Manager
Memory Settings
Package Manager
Physics
Physics 2D
Player
Preset Manager
Quality
Scene Template
Script Execution Order
Services
Ads
Cloud Build
Cloud Diagnostics
Collaborate
In-App Purchasing
ShaderGraph
Tags and Layers
TextMesh Pro
Time
Timeline
UI Builder
Version Control
Visual Scripting
XR Plug-in Management

XR Plug-in Management

Initialize XR on Startup

Plug-in Providers [?](#)

ARKit

Information about configuration, tracking and migration can be found below.
[View Documentation](#)



APIキーは、Google Cloudのコンソールで作成し、設定してください。

Project Settings

Adaptive Performance

▼ Analysis

- Android Logcat Settings
- Audio
- Burst AOT Settings
- Editor
- Fbx Export

▼ Graphics

- URP Global Settings
- Input Manager
- Memory Settings
- Package Manager
- Physics
- Physics 2D
- Player
- Preset Manager
- Quality
- Scene Template
- Script Execution Order

▼ Services

- Ads
- Cloud Build
- Cloud Diagnostics
- Collaborate
- In-App Purchasing
- ShaderGraph
- Tags and Layers
- TextMesh Pro
- Time
- Timeline
- UI Builder
- Version Control
- Visual Scripting

▼ XR Plug-in Management

- ARCore
- ARCore Extensions**
- ARKit

ARCore Extensions

iOS Support Enabled

Android Authentication Strategy

Android API Key

iOS Authentication Strategy

iOS API Key

Optional Features

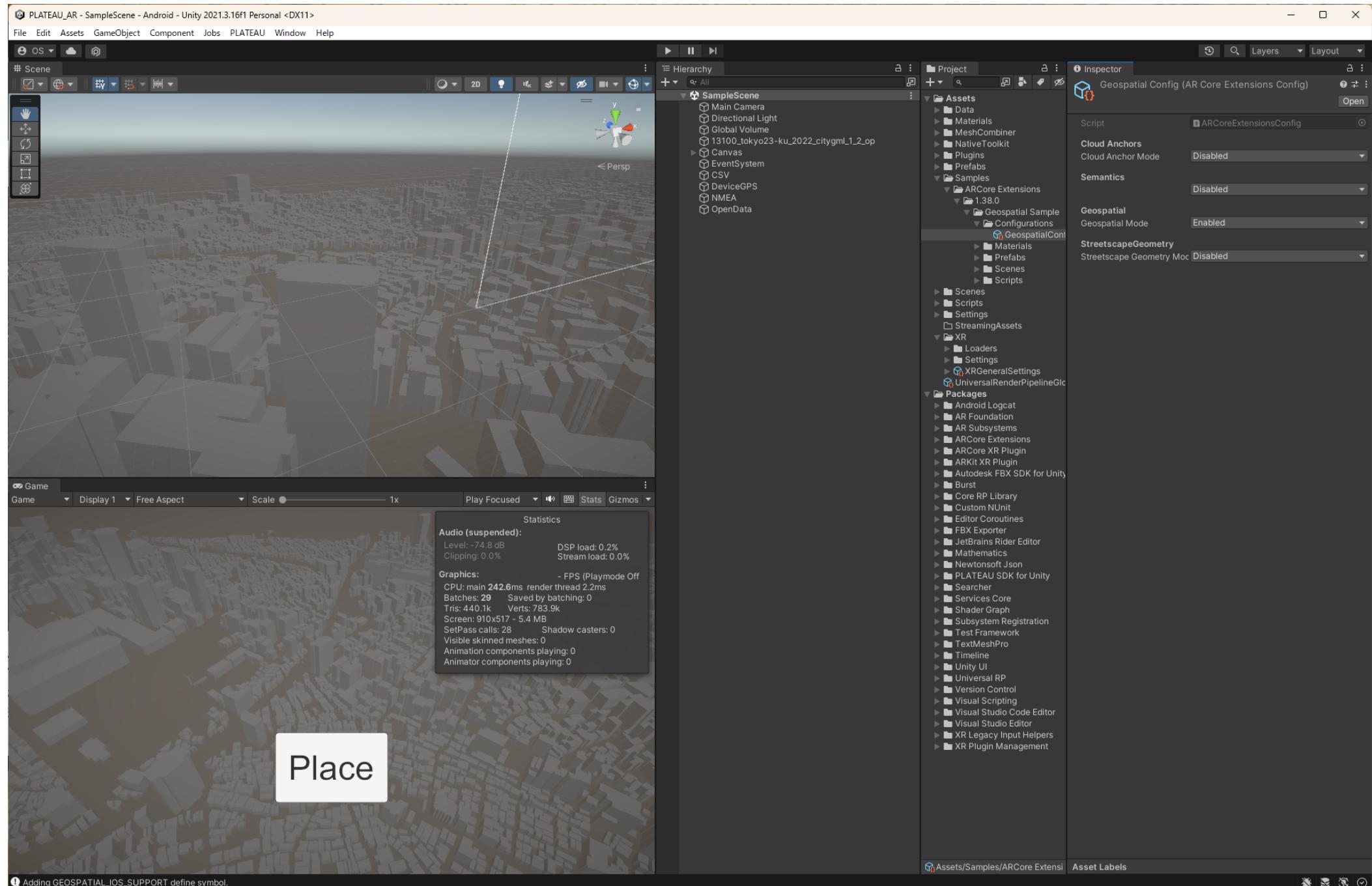
Cloud Anchors

Geospatial

When using the Geospatial API, add location dependencies on Android, and import Geospatial CocoaPod on iOS. Note: precise location permission is required at runtime, otherwise, enabling the Geospatial API may fail with a permission not granted error.

Geospatial Creator





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。

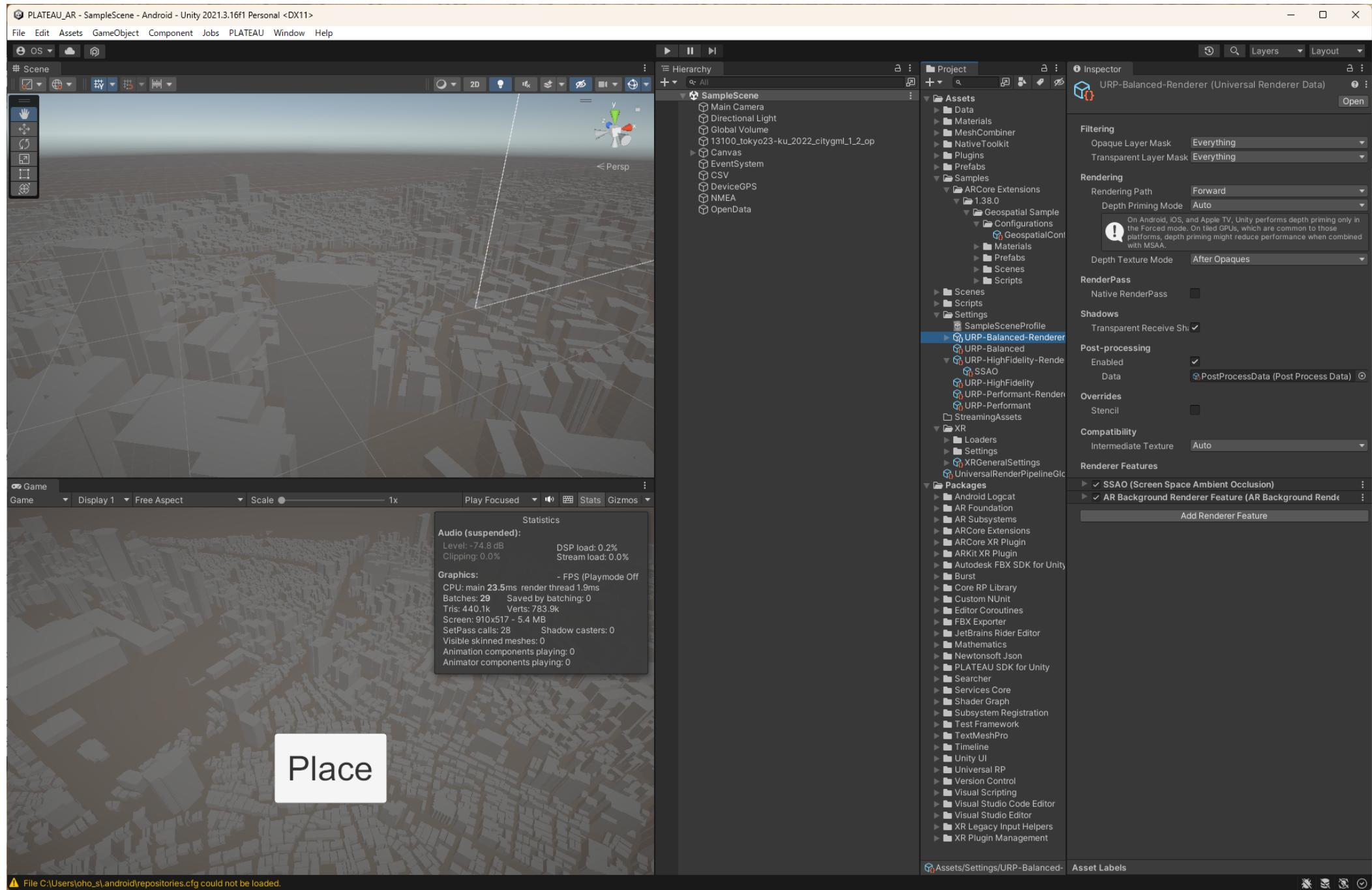
【Androidの場合】

- Minimum API Levelは、28程度が妥当です。
- IL2CPPでビルドします。
- Graphics APIは、OpenGL ES3以上が必要です。

【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。

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。



ヒエラルキーの[Assets]-[Samples]以下のGeospatialAPIのサンプルの中から、Geospatialシーンを開き、これまで作ってきたシーン内の、3D都市モデルとデータの読み込みのオブジェクトなどをコピーします。

アプリ完成