ARアプリのための現実空間と3D空間の高精度な位置合わせ

OGAWA, Genki / 小川, 元暉

(出版者 / Publisher)
法政大学大学院デザイン工学研究科
(雑誌名 / Journal or Publication Title)
Bulletin of graduate studies. Art and Technology / 法政大学大学院紀要. デザイン工学研究科編
(巻 / Volume)
12
(開始ページ / Start Page)
1
(終了ページ / End Page)
7
(発行年 / Year)
2023-03-24
(URL)
https://doi.org/10.15002/00030236

AR アプリのための現実空間と 3D 空間の高精度な位置合わせ

DEVELOPMENT OF A HIGH-PRECISION POSITIONING SYSTEM BETWEEN REAL SPACE AND 3D SPACE FOR AR APPLICATIONS

小川元暉
Genki OGAWA
主査　岩月正見　　副査　土屋雅人

法政大学大学院デザイン工学研究科システムデザイン専攻修士課程

The development environment for the VPS (Visual Positioning System) method, which estimates the posture of mobile devices and superimposes virtual objects on real scenery such as cityscapes, is recently being prepared. This will enhance the practical value of AR. However, the VPS method currently proposed allows rough alignment, but there is always misalignment. Therefore, while it is possible to place virtual objects in an empty sky, it is difficult to place them along the exterior walls of a building. In addition, it is possible to place a 3D city model and occlude it to make it look as if the virtual object is placed behind the building, but it is necessary to fine-tune the position on the actual device. In this study, we develop a system that uses image recognition to accurately estimate the position and orientation of a device in real space and a virtual camera in 3D virtual space and accurately display virtual objects on real images in a single application.

***Key Words** : Visual Positioning System, Coordinate transformation by homographyMat*

１．はじめに

近年、現実空間に CG やデジタル情報を重ね、現実を拡張する AR（Augmented Reality）技術を用いたコンテンツが増えている. またメタバースという言葉が囁かれるようになり、３次元の仮想空間に VR などで入り込むために現実の風景がテクスチャとして貼られた 3D 都市モデルが作られるようになるなど、XR の分野の開発は進んでいる.

また、最近ではモバイルデバイスの姿勢を推定し、街中などの現実の風景に仮想オブジェクトを重畳させる VPS（Visual Positioning System）手法の開発環境が整いつつある. 例としてソニーセミコンダクタソリューションズが提供する AR MAP（文献 1）は、渋谷の街の上空に大きなスクリーンを設け映画の予告を流す広告に利用された. VPS が確立すれば、広告以外にも道案内情報の 3D 表示や AR サイネージ等に利用でき、AR の実用的な価値を高めることに繋がる.

しかし現状提案されている VPS の手法では、大まかな位置合わせは可能だが必ずズレが生じてしまう. そのため何もない空に仮想オブジェクトを配置することはできるが、建物の外壁に沿って配置することはむずかしい. また 3D 都市モデル配置し、オクルージョンをすることで建物の後ろに仮想オブジェクトを配置しているように見せられるが、実機で位置を微調整する必要がある. モバイルデバイスの精度の甘い GPS 加速度・地磁気センサである程度位置合わせを行った後、微調整を簡単かつ高精度に行うシステムを作ることが急務であると考える. このシステムが確立することで、現実の建物と仮想オブジェクトとの前後関係が分かるので、新しいインタラクションの提案が期待できる.

そこで本研究では、画像認識を用いて現実空間上のデバイスと３D 仮想空間上の仮想カメラの位置姿勢推定を高精度に行い、実画像上に正確に仮想オブジェクトを表示させるためのシステムを一つのアプリケーションで完結するように開発する.

２．概要

（１）提案手法

デバイスに搭載されたカメラ（実カメラ）でチェス盤状のボードを複数枚撮影し、大きさを縦 4096px×横 2048px にリサイズする.それらを元にカメラキャリブレーションを行い、カメラパラメータを算出しておく.ここで得たパラメータは仮想カメラ（Unity カメラ）に反映させることやホモグラフィー行列を分解し、回転行列と並進ベクトルを求める際に使う.つぎに、GPS と加速度・地

磁気センサにより実カメラの現在のおおよその位置姿勢を取得し、そこから仮想３D 都市モデルを捉えた仮想カメラ画像を生成する.GPS、地磁気センサの精度の低さにより、生成された仮想カメラ画像と実カメラ画像とは大きくずれが生じているので、画像処理を行い修正する.はじめに、仮想カメラ画像と実カメラ画像のそれぞれの中に含まれる建物の外壁や看板などのテクスチャからAKAZE（Accelerated KAZE）の手法を使い特徴点を生成する.その後、両画像の特徴点を総当たりで比べ、knn(k-nearest neighbor)マッチを行う.特徴量は距離で表すことができ、小さいほど信頼できるマッチ結果になるため､2 組のマッチング結果を割り算し閾値以下になるものだけを集めソートする.信頼性の高い特徴点からホモグラフィー行列を求め、実カメラと仮想カメラ間の回転・並進行列に分解する.得られた回転と並進行列により仮想カメラを移動させ、そこから捉えた仮想３D 都市モデルで先に配置しておいた仮想オブジェクトをオクルージョンしたものを実画像に重ね合わせて表示させる.これらによって人の手で実機の姿勢推定を微調整することなく、実画像と仮想画像のずれをなくして表示させることができる.

（２）アプリケーションのシステム

システムの構成図を、図１に示す.

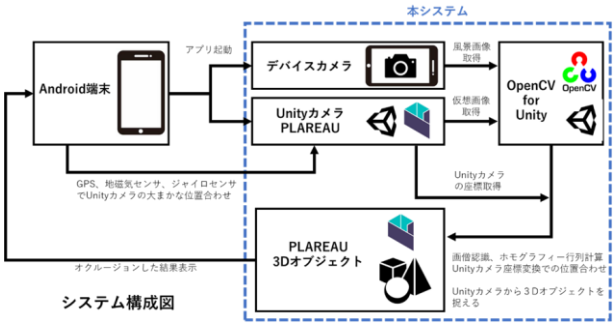

図１　本研究 VPS のシステムのシステム構成図

アプリケーションの使用方法は以下の通りである.

① アプリケーション起動

② GPS での仮位置合わせ(図 2)

図２　起動から GPS 位置合わせをする手順

③ マッチング用の写真を取得(図 3)

図３　GPS 位置合わせ後、マッチング用写真の取得手順

④ マッチングによる位置合わせ(図 4)

図４　マッチング用写真から位置合わせを行う手順

⑤ オクルージョン(図 5)

図５　位置合わせ後オクルージョンする手順

３．理論解説

（１）カメラキャリブレーション

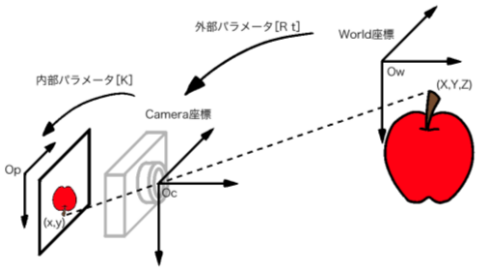

図６　ピンホールカメラモデル

図 6 のようにカメラで撮影する場合、スケールを W、3 次元の点を 2 次元に変換するカメラマトリックスを P、レンズ焦点距離などの内部パラメータを K、カメラの姿勢を表す外部パラメータを R(回転行列)、t (並進行列)としてレンズのないピンホールカメラモデルは以下のような式で表すことができる.

$$W\begin{bmatrix}x\\y\\1\end{bmatrix}=P\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} \tag{1}$$

$$P=K\cdot[R\quad t] \tag{2}$$

ピンホールカメラとは違い通常のカメラは、レンズの湾曲により画像が歪む現象が起こるが、近年のスマホカメラには歪みを補正して表示する機能があるので本研究では考慮しない.

3 次元の位置情報と 2 次元の画像情報を対応させるためには内部パラメータ、外部パラメータを計測する必要がある.それらの導出をカメラキャリブレーションと呼び、OpenCV では形状が既知のチェス盤状のマーカーを複数枚別角度から観測することによって、マーカー上の各点と画像上の各点の対応を知る Zhang の手法が採用されている.この内部パラメータは個々のカメラによって変わるので、使用する機材によって求め直す必要がある.

内部パラメータは以下の式で表し、焦点距離 (fx,fy)、光学的中心 (cx,cy) の情報を含んでいる.このパラメータを元に外部パラメータを算出できるが、本システムでは使用しない.

$$\begin{bmatrix}fx & 0 & cx\\0 & fy & cy\\0 & 0 & 1\end{bmatrix} \tag{3}$$

本研究では図 7 の交点が 7×10 存在する正方形の一辺が 20.5 ㎜のキャリブレーションボードを印刷し、13 枚の写真を撮影した.

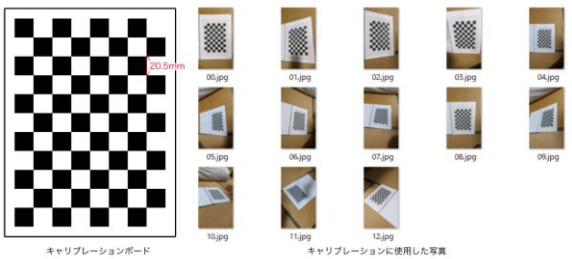

図 7　キャリブレーション素材

OpenCV for Unity では 2 の累乗サイズの画像しか扱うことができないため、それらを縦 4096px×横 2048px にリサイズしてからキャリブレーションを行い、以下の内部パラメータを得た.(図 8)

$$\begin{bmatrix}2900.4438596553432 & 0 & 988.84434666077698\\0 & 3249.1257111724326 & 2057.7716315116736\\0 & 0 & 1\end{bmatrix}$$

図 8　Pixel 5a のカメラ内部パラメータ

OpenCV で取得したカメラパラメータを Unity で使うためのスクリプト (文献 2) を使用し、モバイルデバイスカメラと Unity 内のカメラを一致させた.

(2) AKAZE 法

KAZE 法を高速化させた特徴点検出の手法の一つであり、SIFT や SURF と比べ特許問題がなく OpenCV にオープンソースとして実装されている.他の特徴点検出の手法に比べ、スケール変化、回転、照明変化に対して影響されにくく、信頼性が高い.AKAZE 法で特徴点検出した様子が図 9 である.本研究では実カメラと仮想カメラの画像をマッチングさせるための特徴点検出に使用している.

図 9　AKAZE 特徴点

(3) knn マッチング

総当たりで特徴点を比較し、訓練特徴量をどれだけ保持するかを設定できるマッチング方法である.マッチングした特徴点同士には近ければ近いほど信頼できる距離が存在する.それを使いマッチング結果をソートする手法としては閾値を設け、1 つ目を 2 つ目で割った値が閾値以下になる前後の差が大きい特徴点(1 つ目の距離が小さい特徴点)を集めるものある.本研究では上位 2 個を保持するよう設定していて、割った値が閾値の 0.5 以下になるような点のみを集めている.特徴点マッチングのみしたものとソートを行ったものの様子を図 10 に示す.

図 10　マッチング結果のソート

（４）ホモグラフィー行列

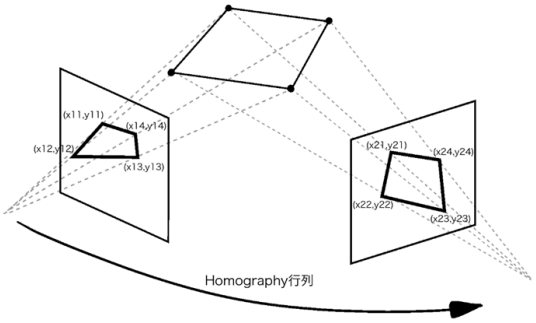

図 11　ホモグラフィー行列による座標変換

図 11 のように同じものを映した 2 枚の画像がある場合、対応点が各画像 4 点ずつ分かれば、撮影したカメラ座標間の座標変換行列を求めることができ、これをホモグラフィー行列(3×3 行列)と呼ぶ.ホモグラフィー行列を H として座標変換は以下の式で表す事ができる.

$$\begin{pmatrix} x21 \\ y21 \\ 1 \end{pmatrix} = H \begin{pmatrix} x11 \\ y11 \\ 1 \end{pmatrix} \tag{4}$$

（５）座標変換行列

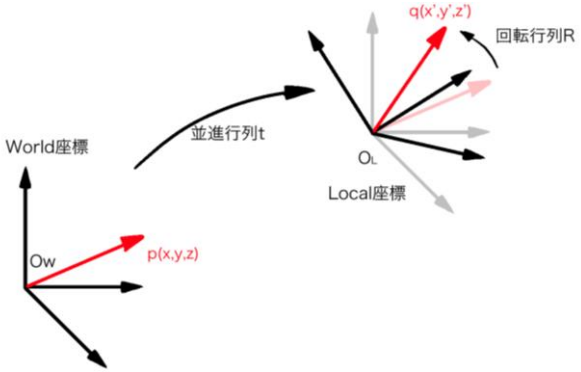

図 12　座標変換行列による座標移動の様子

図 12 のようにベクトルをカメラの直交座標系(ワールド座標)から任意の座標系(ローカル座標)に移動させる場合、基底ベクトルから別ベクトル座標に変換することになる.この時、基底ベクトルに行列をかけて変換を行うことができ、この行列を座標変換行列と呼ぶ.座標変換行列は回転行列成分 R(3×3 行列)と並進行列成分 t(3×１行列)に分けられ、変換前の座標を表す位置ベクトル p=(x,y,z)を変換後の座標を表す位置ベクトル:q=(x`, y`, z`)に移動したい場合、以下の式で表すことができる.

$$q = Rp + t \tag{5}$$

前述したホモグラフィー行列は座標変換行列にあたり、本研究では OpenCVforUnity でホモグラフィー行列を計算した後、decomposeHomographyMat 関数を使い並進ベクトルと回転ベクトルに分解している.また図 13 のように Unity は左手座標系であり、OpenCV は右手座標であるため、画像処理で得られたベクトルは全て Y 軸を反転させる.

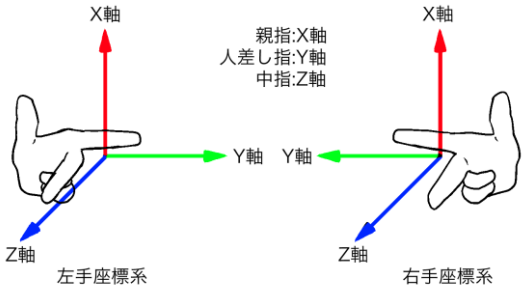

図 13　座標系

Unity の回転はクォータニオンで定義されるため、ホモグラフィー行列の分解により求めた回転ベクトルを Quaternion.AngleAxis 関数で変換してカメラオブジェクトにアタッチすることでカメラの回転を実装している.また、OpenCV で得られる並進ベクトルは回転後の座標系から変換後の座標系原点から返還前の座標系原点までの距離を示しているため、ワールド座標の x 軸 y 軸 z 軸の単位ベクトルに並進ベクトルを掛け合わせてカメラの移動を実装している.

４． 開発環境

（１）Unity

Unity とは Unity Technologies 社が開発、提供している統合開発環境を内蔵したプラットフォームであり、IOS、Android など様々な形式に対応している.本研究では Unity2019.2.13f1 を使用している（文献 3）.

（２）OpenCV for Unity

Open CV for Unity とは Enox Software 社が提供する画像認識プラグイン（文献 4）であり、オープンソースの OpenCV を Unity に移植し、C＃での開発を可能にしている.本研究では、Version 2.4.7 をインポートしていて、OpenCV4.5.5 の機能を使うことができる.

（３）AR Core

AR Core は、Google 社が提供する AR(Augmented Reality 拡張現実）機能を構築するためのプラットフォームであり、Unity にプラグインファイルとして読み込ませることで開発を行うことができる.プラグインファイル「AR Core SDK for Unity」は公式サイト（文献 5）で取得でき、バージョン 1.26.0 を使用している.

（４）PLATEAU

PLATEAU は国土地理院が無償で提供している実寸大の 3D 都市モデルのオープンソースである.東京 23 区やその他の都市の 3D データを公式サイト（文献 6）で取得できる.本研究では､Unity 内に仮想の都市を構築するために新宿駅周辺の Lod2 データを obj 形式で使用した.

（５）Android

Android は、Google 社が開発するモバイル端末向けの OS（Operating System）（文献 7）であり、オープンソースで提供されている.Unity で開発したアプリケーションを実装させることができるプラットフォームの 1 つである.本アプリケーションの実機テストでは「Pixel 5a」を使用する.

（６）Blender

Blender とはオープンソースの 3 次元コンピュータグラフィックスソフトウェア（文献 8）の一つであり、3D モデルの作成、レンダリングのほかにアニメーション、コンポジット機能も備えている.本研究では、PLAREAU で取得した 3D 都市モデルの切り取り､テクスチャの再設定のために Version2.8.3 を使用している.

５．プログラム

本システムを構成するアプリケーションの各機能は、Unity で動作する C#言語のスクリプト（文献 9）をプログラムとして実装し、アプリケーションの動作全般を管理する.画面を構成する要素である「Game Object」に対し、それぞれの機能に応じたスクリプトを適用させ、画面を構築する.ダイアログ表示を含めたアプリケーションの終了処理、文字や図形で構成されるインタフェースの調整、変数の宣言と演算など、Unity や C#言語の基本的な要素を用いた機能はもちろんのこと､AR や画像処理など外部のプラグインファイルによって実装する機能も Unity の C#スクリプトを用いる.

（１）スマホセンサによる大まかな位置合わせ

PLAREAU 内に Unity アプリ内の仮想カメラを移動させるため、仮想カメラに親オブジェクトを作る.親オブジェクトが移動すると子オブジェクトも同じだけ移動する.大まかな位置合わせを実装するために、デバイスの GPS を起動し現在の緯度経度を得た後、平面直角座標と緯度経度の相互変換をするライブラリ（文献 10）を使用して xyz 座標を取得する.PLAREAU は緯度経度を元に建物の輪郭が構成されていて、その座標変換も上記のものと同じであるため、取得した xvz 座標に親オブジェクトを移動することで並進移動が実現できる.また、回転には地磁気センサによってアプリ起動直後の磁北極に基づく度単位の行方向を取得し、カメラの親オブジェクトの y 軸回転から引くことで必ず北を向いた状態で起動するようにしている.起動後の回転はジャイロセンサの値を親オブジェクトに反映させて実装している.

（２）画像の取得

OpenCVforUnity で扱う画像は全て 2 の累乗のサイズであり､Texture2D というクラスでなければならない.本研究では Unity 内の AR カメラ(デバイスカメラと同じ画像を映す)と UnityCamera(仮想カメラ)両方とも同じ内部パラメータを持つように設定しており、それぞれのカメラに写るもののスクリーンショットを Texture2D に変換することで実装している.

（３）ホモグラフィー行列による位置合わせ

UnityAsset 内に OpenCVforUnity.unitypackage を導入し、using ライブラリで指定することで OpenCV の機能を C#言語で扱えるようになる.本研究では、OpenCV の機能である CoreModule(OpenCV で使用するクラスが定義された物)、UnityUtilsusing(Unity の画像を OpenCV に読み込ませるための物) 、Features2dModule(特徴点生成やマッチングに関する関数が定義された物)、Calib3dModule(座標変換行列を計算する変数が定義された物)を参照している.

AKAZE.create 関数に先に取得した実画像と仮想画像を入力し、それぞれの特徴点が格納された KeyPoint を計算する.つぎに取得した KeyPoint を BFMatcher.knnMatch 関数に入力することで、マッチングを行い 1 つの特徴点に対し 2 つのマッチング候補を持つ特徴点群を計算する.特徴点には小さいほど信頼性が高い距離が存在し、第 1 候補を第 2 候補で割った値が小さければ小さいほど信頼性の高いものと推定できるので、閾値(本研究では 0.5)以下のマッチング結果だけを集める.信頼性の高いマッチング結果を findHomograpy 関数に入力することでホモグラフィー行列を推定し、これを decomposeHomographyMat 関数で併進行列と回転行列に分解する.それぞれの行列を Unity で使用できる形に整えた後、UnityCamera に反映させ、親オブジェクトを基準に現実のカメラがある位置に座標変換できる.

（４）オクルージョン

仮想カメラから捉えた仮想 3D 都市モデルで仮想オブジェクトをオクルージョンしたものを実画像に重ね合わせるためマテリアルを変更するスクリプトを制作した.オクルージョンの際は一時的に 3D 都市モデルを自オブジェクトと自オブジェクトが重なっている部分を透明化させるカットアウトマテリアルに変更し、街中のオブジェクト表示を実装している.また、本システムでは最初に PLATEAU 内に実画像上に表示させたいオブジェクトを配置する必要があるが、マッチングの際にノイズにならないようオクルージョンの際は表示し、マッチングの際は非表示にさせている.

６．考察

本研究にて、風景画像と仮想画像を画像認識によるマッチングを行うことでスマホの位置を推定する VPS を開発し、Android 端末向けに実装した.

従来の VPS では処理が軽いものの、ずれて表示されても違和感がないように建物に干渉しない上空や広い空間にオブジェクトを配置しなければならない課題があったが、本研究の現実の建物と 3D 都市オブジェクトを重ね合わせる手法により、オクルージョンすることで、建物の壁面や背後など自由にオブジェクトを配置できるようになったため改善できたと考える.

実際の風景画像と本アプリケーションで位置合わせ後に 3D 都市オブジェクトでオクルージョンした画像を図 14 に示す.

図 14　実際の風景に仮想画像を重ねて表示する様子

本アプリケーションはマッチング後のスマホの移動はジャイロセンサにて回転は取得できるが並進移動を取得できない.加速度センサの値を 2 回積分することで位置を求める算段を立てていたが、基本的にスマホに搭載されるものでは計測誤差成分が生じているため、現実世界のスマホの移動とアプリ内のカメラの移動が一致させることは困難だった.そのため本アプリケーションは現実世界の位置を更新するごとに GPS による大まかな位置合わせ、画像認識による位置合わせと手順を踏まなくてはいけないので、リアルタイムの移動は実装していない.テンプレートマッチングを使い、マッチング範囲を限定することで射影変換行列をリアルタイムで求められるので、それを元に移動を実装すれば解決する可能性がある.また、ホモグラフィー行列の計算はスマホでは数十秒、PC でのテストでは数秒かかってしまう.この待ち時間は総当たりマッチングの影響が大きく、検出する特徴点の個数を制限できる ORB 法を使ってパターン数を減らすことで精度と引き換えに短縮できる.その他にも本研究のシステムを作る際に OpenCV C++で試作したものがあり、これを RestAPI として呼び出すことで通信が必要になるが数秒の短縮が期待できる.

PLAREAU の 3D 都市モデルは現状建物にテクスチャが張られている Lod2 データが新宿駅周辺のみの配布のため、本システムを使用できる場所が限られている.そして建物の輪郭はある程度実物と近いものであるが、テクスチャは低解像度の物が大まかに貼られているため、そのままのデータではマッチングに使用できない.また、昼夜によっても日光と街灯の明かりに違いが出るため、日中と夕暮れ時両方のテクスチャを用意する必要があると考えた.PLAREAU でダウンロードしたままの 3D データの画像と本研究でマッチングさせるため日中のテクスチャを貼った 3D データの画像、夕暮れのテクスチャを貼った 3D データの画像を図 15 に示す.

図 15　各テクスチャの比較

それぞれのテクスチャの場合で日中・夕暮れの風景画像とマッチングさせたときの様子を図 16 に示す.

図 16　日中と夕暮れ時でのマッチング結果

日中の写真でマッチングを行う場合、昼夜のテクスチャに関係なくマッチングを行うことができる.しかし夕暮れ時の写真の場合、日中のテクスチャでマッチングを行うことはできるが精度が低く、ホモグラフィー行列を求めるための必要数を検出できなかった.夕暮れ時の写真×夕暮れ時のテクスチャでのマッチングでは日中×日中ほどではないが、ホモグラフィー行列の計算は可能で

ある.暗い写真は境界があいまいになり、特徴点量も減るためなるべくテクスチャを合わせる必要があることを確認できる.そのため時間帯により、テクスチャを変える必要があり、本システムの使用のためには、新宿以外のLod2データ作成とテクスチャの設定の改善が望まれる.

７．結論

本研究では、3D都市モデルを使って画像認識によるVPSのシステムを提案した.通常バーチャル秋葉原などのメタバースに没入するためにはVRのヘッドセットが必要になり、機材の為に場所が限られてしまうという欠点がある.あまり VR やメタバースが万人に普及していない理由はこの大がかりさが原因だと考える.しかし、本アプリケーションでは気軽に街中を拡張できるため、メタバースの世界をスマホから覗き込むような体験ができる.

また、本研究では Unity 内にインポートした PLAREAU のオブジェクトの中にテキストや３D モデルを配置し、オクルージョンすることで街中を拡張した写真を撮影した.そのため位置合わせのみを実装していてオブジェクトを配置する機能はアプリケーション内に実装していない.レイキャストを使って建物の壁面にスプレーを塗布するような物や、マテリアルの変更によって壁のテクスチャを全て広告に変える演出などの実装により街中の風景が一変する新たな体験ができると考える.特に前者のようなレイキャストを使った演出は近距離の場合は Depth カメラで実装できるが、街中などの広範囲で実装するためには本システムが最適であると考える.また、オブジェクトにメッシュコライダーを設定することで壁に跳ね返るボールを射出するアプリケーションを制作できる等今までの VPS では実装困難な演出が可能になる.

広告やエンタテインメント以外にも活用の機会が存在する.道案内 AR として道に矢印を表示させるものがあるが、GPS の精度が低い場合、ずれて表示されてしまう.しかし本システムでは現実の建物と、3Dの建物が一致している為ずれることなく表示できる.また、店舗レビューやキャンペーンの情報を AR で壁面に出すことや、目的地の３D モデルのテクスチャを変えることによって強調表示させるなど今までにない街中の AR 表現が期待できる.そして城などの観光地では建設当時の建造物の3Dモデルを作っておくことにより、現在と過去の建造物を比較できるなど活用の余地が多い.

このように本アプリケーションは、建物のLod2データさえあればARによって街中を拡張し、いつもの通り道を特別なものに変えられる力を秘められていて、AR がより実用的になることで豊かな社会が実現できると考える.

謝辞

本研究の遂行にあたり、指導教員である岩月正見教授には、最先端な事例の紹介、研究の着想、実装までの手法、論文執筆まで終始適切なご指導を賜りました.深く感謝申し上げます.そして、副査として土屋雅人教授には適切なご助言をいただきました.最後に、スマートマシン研究室の皆様のご協力により、修士論文の完成まで至ることができました.ここに感謝いたします.

参考文献

1)「AR MAP」の VPS を利用した「スパイダーマン：ノー・ウェイ・ホーム」AR アプリの技術解説が公開。他のサービスとの比較や開発における Tips なども解説，ゲームメーカーズ，https://gamemakers.jp/article/2022_06_16_8113/,（参照 2023-2-8）.
2) OpenCV で取得したカメラパラメータを Unity で使う，かみのメモ，https://kamino.hatenablog.com/entry/unity-import-opencv-camera-params,（参照 2021-1-6）.
3) Unity，https://unity.com/,（参照 2023-2-8）.
4) OpenCV for Unity，https://enoxsoftware.com/opencvforunity/,（参照 2023-2-9）.
5) AR Core，https://developers.google.com/ar?hl=ja,（参照 2023-2-9）.
6) PLATEAU，https://www.mlit.go.jp/plateau/,（参照 2023-2-9）.
7) Android，https://www.android.com/intl/ja_jp/,（参照 2023-2-9）.
8) Blender, https://www.blender.org/,（参照 2023-2-9）.
9) Unity C#，https://unity.com/ja/how-to/programming-unity,（参照 2023-2-8）.
10) 平面直角座標と緯度経度の相互変換をするライブラリ，yubeneko，https://www.blender.org/,（参照 2023-2-8）.