

マーカレス AR ゲームの開発に向けた ゲームフレームワーク Radish の拡張

久野 剛司^{†1} 新田 直也^{†1}

近年、拡張現実 (AR) のコンピュータゲームへの応用が注目されているが、マーカレス AR ゲームについてはまだ実用レベルに達しているものが見当たらない。そこで本研究では、著者らの研究室で Java を用いて開発した 3D ゲームフレームワーク Radish を拡張し、実用的なマーカレス AR ゲームの実現を目指す。本稿では、マーカレス AR 用に考案した SLAM アルゴリズムの概要を説明し、実験用ゲームを用いて実行速度及びゲームとしての有効性について評価する。

Extending 3D Game Framework Radish for Markerless Augmented Reality Games

TSUYOSHI KUNO ^{†1} and NAOYA NITTA^{†1}

Recently, application of augmented reality technologies to the field of computer games has been attracted attention. To our knowledge, however, there is hardly a markerless augmented reality game at practical level. Therefore, we extend a 3D game framework, named Radish developed in our laboratory, and intend to realize practical games using markerless augmented reality. In this paper, we explain the outline of the algorithm of simultaneous localization and mapping developed in our laboratory, and evaluate the computing performance and the availability for games using a sample game we developed.

1. はじめに

近年、効果的な電子情報提示の枠組みとして拡張現実感 (Augmented Reality, AR)⁴⁾ が

注目されている。拡張現実感とは、現実の環境に仮想の物体を CG 等を用いて付加表示する技術で、コンピュータゲームでの利用の試みも一部では始まっている²⁾。一般に拡張現実感においては、CG をそれがあたかも実在しているかのように現実環境に重ね合わせて表示する必要があるが、CG の重ね合わせ位置を決める手法として、マーカを用いる手法とマーカを用いない手法がある。前者の手法を用いたものをマーカあり AR、後者の手法を用いたものをマーカなし AR またはマーカレス AR とよぶ。マーカあり AR では、現実環境に物理的に配置したマーカをビデオカメラなどの入力機器を用いて識別することによって、重ね合わせ位置を決定する。この手法の問題点としては、実際にマーカを作成し配置することを前提としている点や、マーカが識別範囲にない場合に重ね合わせ位置を決定できないといった点を挙げることができる。一方、マーカレス AR では、ビデオカメラなどで撮影した現実環境の画像などをリアルタイムで解析し、得られた外界やビデオカメラの自己位置に関する情報を基に、マーカを使うことなく重ね合わせ位置を決めることができる。しかし、一般に画像解析には多くの計算処理が必要となるため、リアルタイム性の維持が困難であることや、現実環境の状態によっては重ね合わせ位置の計算で大きな誤差が生じるといった点が問題点として挙げられる。本研究では、拡張現実感を利用したコンピュータゲーム (以下、AR ゲームと略) におけるエンターティメント性の向上を目指して、ゲーム向けマーカレス AR のアルゴリズムを提案する。本アルゴリズムはマーカレスであるにも関わらず、ゲームで利用するために必要なリアルタイム性を有しているという特徴を持つ。また、一般にマーカレス AR では処理速度と位置決め精度の間にトレードオフの関係が見られるが、サンプルゲームを用いた評価実験の結果、一定のゲームジャンルで用いる上では十分な位置決め精度を達成できていることがわかった。

近年、ゲーム開発に要するコストは増大の一途を辿っており、そのため特に海外においては、ゲームエンジンを再利用することによってコストの低減を図る開発手法が一般化しつつある。本研究でも、3D ゲームエンジンとして Java および Java3D を用いたアプリケーションフレームワーク Radish を開発し、3D ゲーム開発のプロジェクト実習において開発コストの低減を図っている。本研究では、マーカレス AR ゲームの開発を容易にするため、本アルゴリズムの Radish への組み込みを行った。これによって、評価実験で用いたサンプルゲームを比較的少ない工数で開発することができた。

2. 拡張現実感ゲーム

コンピュータゲームで AR を用いる試みはすでに始まっており、商品化されているもの

^{†1} 甲南大学 自然科学研究科 情報システム工学専攻
Graduate School of Natural Science, Konan University

もいくつかある。たとえば SONY が開発したブレイステーション 3 用の THE EYE OF JUDGMENT⁵⁾ では、カード上に印刷されたマーカをカメラが読み取りその上に CG のキャラクターが重ね合わされるといった表現手法が用いられている。他にも、マーカあり AR を用いたゲームはいくつか商品化されており^{6),7)}、今後も増えていくものと予想される。一方マーカなし AR を用いたゲームは、今のところ少ない。たとえば、Enjoy Gaming 社が開発した任天堂 DSi 用の System Flaw⁸⁾ は、プレイヤーの周囲に居る“見えない敵”をカメラで探して撃ち落とすゲームであるが、特にマーカ等は使用しない。数は少ないが、他にもマーカレス AR を用いたゲームは存在するが⁹⁾、いずれも

- カメラの自己姿勢を推定しているのみで、自己位置までは推定していない、
- CG で表現しているものが空中に漂っているものばかりなので、カメラの自己姿勢の推定誤差が目立ちにくい

といった特徴がある。詳細は後述するが、カメラの自己位置と自己姿勢の両方を精度よく推定するためには、ロボット工学分野で広く研究されている SLAM(Simultaneous Localization And Mapping) 問題を解く必要がある。すでにロボット工学分野では、SLAM 問題を解くアルゴリズムがいくつか考案されているが、ゲームとロボットでは要求される実行速度や推定精度が異なると考えられるため、既存のアルゴリズムの利用が必ずしも適切であるとは限らない。そこで、本研究では SLAM 問題を、一定のジャンルのゲームで必要とされる精度を満たしつつ効率良く解くアルゴリズムを考案し、それを用いてサンプルゲームの実装を行う。SLAM 問題を効率良く解くことで、カメラの自己位置をリアルタイムで推定することができ、それによって敵の攻撃をかかわず、敵の背後に回り込むといったより多彩な動作を実現することが可能になる。

3. Radish フレームワーク及びその拡張

本研究室では、3D ゲームの開発効率の向上および 3D ゲームエンジンのアーキテクチャの研究を目的として、Java および Java3D を用いた 3D ゲーム用アプリケーションフレームワーク Radish を開発している³⁾。Radish は、アーキテクチャが公開されている、アプリケーションフレームワークの形態を採用しているため適合するゲームジャンルにおいてはアプリケーションの生産性が高い、Java および Java3D を用いているため OpenGL 環境でも DirectX 環境でも動作し、OS に依存しない等の特徴を持つ。ただし、現在のところ教育・研究用に開発されたものであるため、市販の 3D ゲームエンジンと比較して機能の数では劣る。本研究では、エンターテインメント性の高いマーカレス AR ゲームの開発を目指している

ため、Radish にマーカレス AR 機能を付加することを考える。マーカレス AR を実現するシステムとしては PTAM(Parallel Tracking and Mapping for Small AR Workspaces)¹⁾ が良く知られているが、PTAM が C++ で記述されており、Java で記述されている Radish との統合が困難であることが予想されること、PTAM はゲームでの使用を前提としていないため、実行速度および推定精度において今回の目的に適合しない可能性があること、などの理由により、独自にマーカレス AR のアルゴリズムを考案し Radish に組み込む形で実装を行った。ただし、今回考案したアルゴリズムは、Radish での使用を前提とするものでも、Java での記述を前提とするものでもない。またマーカレス AR のアルゴリズムの実装にあたっては、開発効率を考慮して、現在もっとも広く用いられているオープンソースの画像ライブラリである OpenCV 2.0 を使用するようになった。OpenCV 2.0 は、C++ で記述されているが、Radish からの呼び出しは JNA(Java Native Access) を用いることで対処した。

4. マーカレス AR ゲーム向け SLAM アルゴリズム

4.1 アルゴリズムの概要と SLAM

今回、著者らが考案したアルゴリズムは、ビデオカメラより取得した動画像から、三次元復元とカメラの自己位置及び姿勢推定を同時に行う SLAM 問題を解くアルゴリズムである。このアルゴリズムの処理の流れは以下の通りである。

- (1) カメラより取得した現在のフレームの画像から特徴点を取得する。ここで、特徴点の数を N 個とし、 i 番目 ($1 \leq i \leq N$) の特徴点の画像平面上の座標を (X_i, Y_i) とする。
- (2) 前回のフレームの画像と現在のフレームの画像から特徴点の移動量 (オプティカルフロー) を求める。ここで i 番目の特徴点のオプティカルフローを $(X_i, Y_i, \Delta X_i, \Delta Y_i)$ とする。
- (3) (2) で求めたオプティカルフローと現在までのフレームで求めた特徴点の奥行き情報及びその分散からカメラの運動パラメータを推定し、カメラの自己位置及び姿勢推定を行う。(詳細は 4.2 節を参照。)
- (4) (3) で求めたカメラ運動パラメータとオプティカルフローの情報から、各特徴点の奥行き情報を再計算し、奥行き情報の精度を高める。(詳細は 4.3 節を参照。)

上記処理をフレーム毎に繰り返し行う。上記処理を繰り返すことで各特徴点の奥行き情報及びカメラの運動パラメータの精度は反復的に向上していきと考えられる。各特徴点の奥行きは、初めて出現した特徴点は距離に対して等確率分布に従うと仮定して、次のフレーム以降は、ガウス分布に従うと仮定して算出している。

4.2 奥行きが既知の場合のカメラ運動推定

奥行きが既知であると仮定して、各特徴点の動きから、カメラの並進運動量及び回転運動量を推定し、カメラの自己位置及び姿勢を求める。以下では3次元座標系として右手座標系 (x, y, z) を用い、簡単のため現在のフレームにおける画像平面上の X 軸と x 軸、 Y 軸と y 軸がそれぞれ平行かつ、カメラの位置を原点と仮定する。また特徴点の奥行きを表すために d 軸を導入し、 $d = -z$ とする。図1と図2はそれぞれ、カメラが並進運動、回転運動した時の図である。

• 並進運動の場合:

カメラが $x-d$ 平面上で $(\Delta Cx, \Delta Cd)$ だけ並進運動して、図1のように画像平面上の i 番目特徴点が X_i から ΔX_i^T 移動したことが観測されたとすると、以下の式が成り立つ。ただし、ここで i 番目の特徴点の奥行き d_i は既知であるとする。

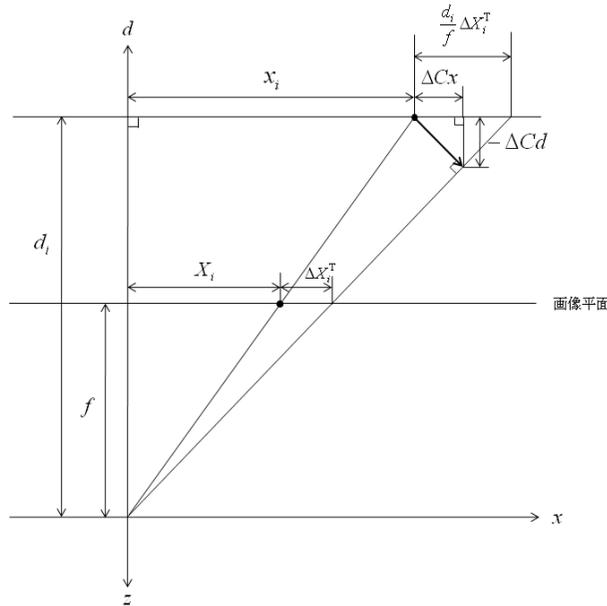


図1 並進運動の場合のオプティカルフロー

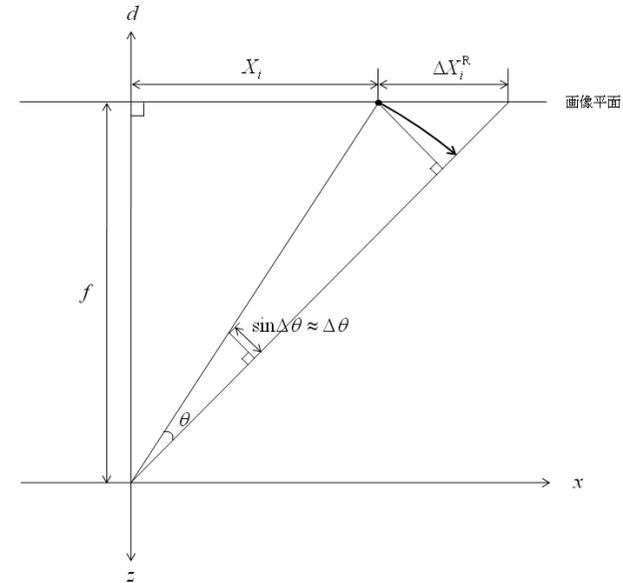


図2 回転運動の場合のオプティカルフロー

$$\frac{d_i}{f} \cdot \Delta X_i^T = \Delta Cx - \Delta Cd \cdot \frac{X_i + \Delta X_i^T}{f}$$

$\frac{\Delta X_i^T}{f}$ はフレームレートが高いと仮定すると0とみなせるので、並進運動は

$$d_i \cdot \Delta X_i^T \approx \Delta Cx \cdot f - \Delta Cd \cdot X_i \quad (1)$$

を満たす。

• 回転運動の場合:

カメラが $\Delta\theta$ だけ回転運動して、図2のように、画像平面上的特徴点が X_i から ΔX_i^R 移動したことが観測されたとすると、

$$\Delta X_i^R = \sin \Delta\theta \cdot \sqrt{f^2 + X_i^2} \cdot \frac{\sqrt{f^2 + (X_i + \Delta X_i^R)^2}}{f}$$

フレームレートが高いと仮定すると $\Delta\theta$ が0に近い値となり $\sin \Delta\theta \approx \Delta\theta$ とみなせる。

同様に、 $(X_i + \Delta X_i^R) \approx X_i$ とみなせることから、

$$\Delta X_i^R \approx \Delta\theta \cdot \frac{f^2 + X_i^2}{f} \approx \Delta\theta \cdot f + \frac{\Delta\theta}{f} \cdot X_i^2 \quad (2)$$

● 一般の場合:

(1)(2) よりそれぞれの式を加えたものが一般の運動で成り立つ式となる.

$$d_i \cdot \Delta X_i \approx \Delta Cx \cdot f - \Delta Cd \cdot X_i + \Delta \theta \cdot f \cdot d_i + \frac{\Delta \theta}{f} \cdot d_i \cdot X_i^2 \quad (3)$$

ここで, (3) 式の右辺の第 4 項は他項と比較して係数が小さいため無視し, 得られた $d_i \cdot \Delta X_i \approx \Delta Cx \cdot f - \Delta Cd \cdot X_i + \Delta \theta \cdot f \cdot d_i$ を $(X_i, d_i, d_i \cdot \Delta X_i)$ の 3 変数による平面の方程式と考える. 実際に計測されたオプティカルフローは一般にこの理論式から外れるため, 3次元の重み付き最小二乗法を用いて, 理論式を推定しカメラの運動パラメータ $(\Delta Cx, \Delta Cd, \Delta \theta)$ を求める.

4.3 奥行きとカメラ運動推定の反復的高精度化

4.2 節で求めたカメラの運動パラメータと X_i と ΔX_i から, 奥行き情報 d_i を再計算し, 奥行き情報として更新する. このとき, 更新の度に d_i を保存しておき, 上の 3次元の重み付き最小二乗法の計算で過去の奥行きの平均値を d_i , 標準偏差の逆数を重みとして用いる. これは, これらのパラメータのうち d_i が最も不確かであるためである.

4.4 カメラ位置と姿勢の周期的補正

上記アルゴリズムのみでは, カメラが移動していくにつれて, カメラの自己位置及び姿勢, 追跡している特徴点の奥行き情報のすべてに誤差が蓄積してしまう. そこでその蓄積誤差を軽減するため, 周期的にカメラから特徴点までの距離の再計算を行う補正をかける. 具体的には M フレーム毎に補正処理を行うとした場合, 補正を行うフレームの M フレーム前の画像平面上の特徴点座標から算出した特徴点の方向ベクトル \vec{v}_i , 現フレームにおける特徴点の方向ベクトル \vec{v}_i' , M フレームの間のカメラの推定位置の変位ベクトルを \vec{t} とし図 3 のように特徴点の奥行き d_i を求め, この値で奥行き情報を更新する.

5. 評価実験

5.1 本アルゴリズムの精度評価実験

本アルゴリズムの精度評価実験を行う. 実験はカメラの向きを固定したままカメラの位置が円軌道を 10 周するように手で動かし, カメラの推定運動とのズレを確認する. カメラの物理的な運動はカメラにペンタブレット専用のペンを取り付け, ペンタブレット上でカメラを運動させて測定する. 特徴点の最大抽出数と, 4.4 節の周期的補正の有無を変えて実験を行った. カメラの物理的軌跡と運動推定による軌跡を比較した結果を図 4~ 図 6 に示す. これらの図からわかるように, いずれの条件においても推定精度が高いとはいえないが, こ

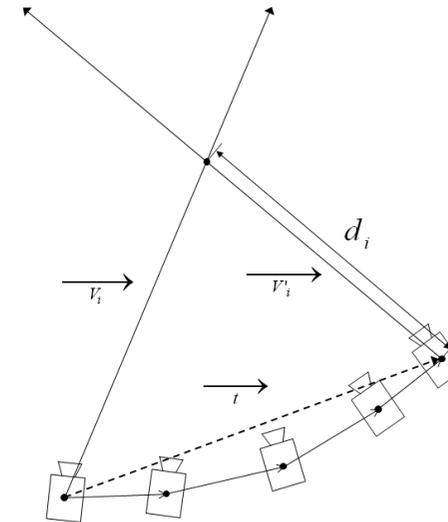


図 3 カメラから i 番目の特徴点までの距離の再計算

れらの中では最大特徴点数 100, 補正有の場合に比較的推定精度が高いことがわかる. このことから最大特徴点数を増やすことと補正を有効にすることの両方が推定精度の向上に貢献していることが分かる.

5.2 サンプルゲームによるユーザビリティ評価実験

5.2.1 サンプルゲームの仕様

本アルゴリズムを実装し, マーカレス AR 機能として Radish に組み込んだ. Radish のマーカレス AR システムとしてのユーザビリティを評価するためサンプルゲームを実装した (図 7 参照). 仕様の概要を以下に示す. (タブレット PC 上での操作を想定している.)

- プレイヤーは web カメラを動かして自分の位置及び視点を操作する.
- ディスプレイ上には web カメラで撮影された現実環境の上に CG で表現された敵の戦闘機やミサイルが重ね合わせて表示される.
- プレイヤーはタッチパネルをタッチし, 視線方向にミサイルを発射する.
- プレイヤーから発射されたミサイルが敵の戦闘機に当たれば敵の戦闘機は撃墜されスコアが加算される.
- 敵が発射したミサイルがプレイヤーに当たればゲームオーバーになる.

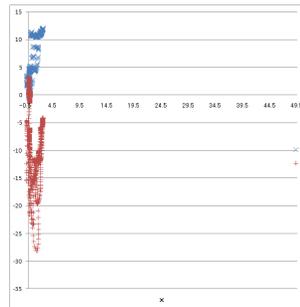


図 4 最大特徴点数 20 補正有における物理的軌跡とカメラの運動推定による軌跡

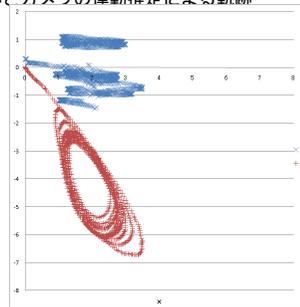
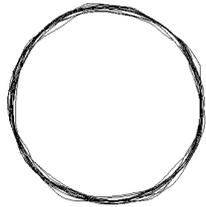


図 5 最大特徴点数 100 補正有における物理的軌跡とカメラの運動推定による軌跡

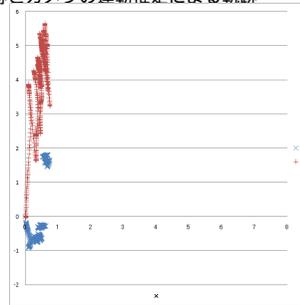


図 6 最大特徴点数 100 における物理的軌跡とカメラの運動推定による軌跡

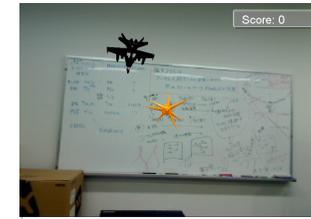


図 7 サンプルゲームのプレイ画面

5.2.2 ユーザビリティ評価

サンプルゲームを用いて、ダブルブラインドテストを行う。ダブルブラインドテストとは被験者が評価する対象が 2 種類存在し、そのいずれを評価しているのかが、被験者にも試験者にも分からないようにした評価テストのことをいう。サンプルゲームがゲームとしてプレイができ、CG で表示された可動物が不自然な挙動をしないか否かを同一被験者に OK と NG で判定させる。各テストでは被験者にサンプルゲームを約 1 分間プレイさせる試行を 15 セットずつ行う。試行毎にあらかじめ決められた 2 つのパラメータのいずれかを乱数で選択しゲームを開始する。最初から 5 セット分は被験者のゲームの挙動に対する慣れによる影響を避けるため除外することにする。残りの 10 セットで t -検定を行い、 t 値を算出して、被験者がパラメータの差異を区別出来ているか判断する。結果を表 1~表 3 に示す。

表 1 最大特徴点数 100 と 500 による比較テスト (補正有)

| 回数 | 特徴点数 | 評価 |
|-------|-------------|----|
| 1 | 100 | OK |
| 2 | 500 | OK |
| 3 | 500 | OK |
| 4 | 100 | OK |
| 5 | 100 | OK |
| 6 | 500 | NG |
| 7 | 100 | OK |
| 8 | 100 | NG |
| 9 | 500 | OK |
| 10 | 100 | OK |
| t 値 | 0.395734833 | |

表 2 最大特徴点数 20 と 100 による比較テスト (補正有)

| 回数 | 特徴点数 | 評価 |
|-------|------|----|
| 1 | 100 | OK |
| 2 | 20 | NG |
| 3 | 20 | NG |
| 4 | 20 | NG |
| 5 | 20 | NG |
| 6 | 20 | NG |
| 7 | 100 | OK |
| 8 | 20 | NG |
| 9 | 100 | OK |
| 10 | 100 | OK |
| t 値 | 0.0 | |

表 3 最大特徴点数 100 で補正の有無による比較テスト

| 回数 | 補正 | 評価 |
|-------|----------|----|
| 1 | 無 | NG |
| 2 | 無 | OK |
| 3 | 有 | OK |
| 4 | 無 | NG |
| 5 | 無 | NG |
| 6 | 有 | NG |
| 7 | 無 | NG |
| 8 | 有 | OK |
| 9 | 有 | OK |
| 10 | 有 | OK |
| t 値 | 0.033344 | |

5.2.3 実行速度の測定

サンプルゲームを最大特徴点数を変えて実行し、実行速度を測定した。結果を表 4 に示す。

表 4 最大特徴点数と実行速度の関係

| 最大特徴点数 | 平均フレーム実行時間 | 平均フレームレート (FPS) |
|--------|------------|-----------------|
| 2000 | 154.2 | 6.484 |
| 1000 | 79.02 | 12.66 |
| 500 | 67.59 | 14.79 |

5.3 考 察

表 1 及び表 2 の結果からわかるように、被験者がゲームのプレイにあたって支障を感じないためには最大特徴点数が 100 は少なくとも必要で逆に 100 以上の数にしても、体感にあまり差が出ないことが分かった。また表 3 による結果から、補正がなければゲームとしてのプレイに支障が出ることも分かった。このユーザビリティ評価の結果は 5.1 節の精度評価の結果とよく一致するため、今回のサンプルゲームのような CG 上の可動物と自分との相対的な位置関係によって進行するようなゲームにおいては SLAM 問題を厳密に解かなくても、最大特徴点数が 100 以上でかつ補正をかけた場合の精度でゲームとしてプレイする上では必要十分であると考えてよい。また、表 4 で示した結果からわかるように、最大特徴点数を減らしていけばゲーム中の実行速度が上がっていくが、最大特徴点数が 500 以下ではフレームレートが 15FPS から変化しなかった。この原因は OpenCV2.0 の 15FPS 以上の頻度で画

像をキャプチャできないという制限によるものと考えられる。仮にこの制限が解除されたとすれば、最大特徴点数 100 の場合で、ゲームをプレイする上で十分な実行速度を達成できると推測される。

6. おわりに

カメラの自己位置および姿勢推定を行うマーカレス AR ゲームの開発に向けて、本研究室で開発された 3D ゲームフレームワーク Radish の拡張を行った。拡張に際しては独自の SLAM アルゴリズムを考案し、実行速度および推定精度の評価と、サンプルゲームを利用したユーザビリティ評価を行った。その結果、CG 上の可動物と自分との相対位置だけで勝敗が決まるゲームにおいては、十分な実行速度と推定精度を達成できていることがわかった。今後、本フレームワークを再利用して、よりエンターテインメント性の高いマーカレス AR ゲームが開発されることが期待される。今後の課題としては、PTAM との間で定量的な比較を行うこと、より広いゲームのジャンルで利用できるように SLAM アルゴリズムにおける推定精度の向上を図ること、三次元復元した情報を利用して、衝突判定、物理演算、オクルージョン処理等ができるようにすること、C++ に移植し更なる高速化を図ること、OpenCV を使用しないようにしマルチプラットフォーム性を高めることなどが挙げられる。

参 考 文 献

- 1) Georg Klein and David Murray: Parallel Tracking and Mapping for Small AR Workspaces, In Proc. ISMAR'07, 2007.
- 2) Pietro Azzari and Luigi Di Stefano: Vision-Based Markerless Gaming Interface, In Proc. ICIAP'09, 2009.
- 3) 新田 直也, 久野 剛司, 久米 出, 武村 泰宏: 3D ゲームエンジン Radish の開発とそのアーキテクチャ比較への応用, 日本デジタルゲーム学会, 採録決定.
- 4) Stephen Cawood and Mark Fiala: Augmented Reality: A Practical Guide, 2008.
- 5) Sony Computer Entertainment Inc.: THE EYE OF JUDGMENT, <http://www.jp.playstation.com/scej/title/eoj-ppsp/>, 2010.
- 6) Sony Computer Entertainment Europe Inc.: EyePet, <http://www.eyepet.com/>, 2009.
- 7) Sony Computer Entertainment Inc.: InVizimals, <http://www.invizimals.com/>, 2009.
- 8) Enjoy Gaming ltd.: System Flaw, <http://www.systemflaw.com/>, 2009.
- 9) A Different Game: Ghostwire, <http://www.ghostwiregame.com/>, 2009.