Android向けUnityアプリの仮想化

高橋 悠[1,a)]　成見 哲[1]

概要：本研究では，ゲーム開発エンジンである Unity3D を用いて開発された Android 向けアプリを仮想化するシステムである UnityCloud を提案する．UnityCloud は Unity で作成されたゲームをサーバとクライアントに処理を分け，Android クライアントからの入力を受けてサーバ上で演算処理を実行し，その結果を映像として返すことで実現する．その第一段階として，映像出力の方法を 2 通り考案し，どちらがより目的に適しているかの比較検討を行った．また，ネットワーク通信による遅延時間の計測によって，提案した方法がゲームプレイにおいて実用的な応答速度であることが確認できた．今後は操作入力への詳細な対応，より快適なゲームプレイのための動作速度の向上などが必要である．

Virtualization for Android app developed by Unity

Takahashi Yu[1,a)]　Narumi Tetsu[1]

Abstract: In this research, we propose a system named UnityCloud that virtualizes Android apps, which are developed using Unity3D, a game development engine. UnityCloud is composed of a client and a server. The server receives the input data from the user via the client, processes the physical calculation in the game, and sends the video data to the client. As a first step of implementing UnityCloud, we propose two ways of transferring video through a network, and compared them in the point of performance and easy installation. The network latency of the proposed method is in the range of practical tolerance for playing games. In future, we need to implement touch input operation, and to reduce the latency for a more comfortable game play.

1. はじめに

1.1 背景

近年，スマートフォンの普及によって，スマートフォンでゲームを遊ぶ人が増加している．またそれにともない，スマートフォンアプリの開発環境の整備も進み，個人開発者によるゲームの開発も活発になっている．特に，ゲームエンジンおよび統合開発環境である Unity3D[1]（以下 Unity）は，GUI による直感的な編集操作や簡単に物理演算が利用できる点などから多くの開発者に好まれている．しかし，スマートフォン OS として最大のシェアを持つ Android は機種ごとの性能差が大きいため，同じ Android 向けのゲームアプリであったとしても機種によって動作が異なる，または動作しないといった問題が発生する．そのため，対応端末を広げるためにはゲーム内で使える表現方法に制限が出てしまう事が多い．また，複数端末での動作チェックを行う事も個人開発者には負担が大きい．

このような端末性能差による影響を軽減する方法として，クラウドゲーミングという考え方がある．この方法では，ユーザの操作入力がネットワークを通じてサーバに送られ，ゲームの演算処理は全てサーバ上で行われ，その結果は映像としてユーザの端末に配信される．ユーザの端末の演算性能に依存せず，高い処理能力が要求されるゲームであっても，見かけ上は端末で動作させることが出来る．また，ゲームプログラムそのものをインストールする必要が無いので端末のストレージを圧迫しない，複数の端末から同じゲームを同様に楽しむ事ができる，といった利点もある．現在既に発表されているクラウドゲーミングサービスとして NVIDIA GRID[2]，G-cluster[3]，PlayStation Now[4] 等がある．しかし，このようなサービスは企業に

[1] 電気通信大学
The University of Electro-Communications
[a)] yuu.takahashi@uec.ac.jp

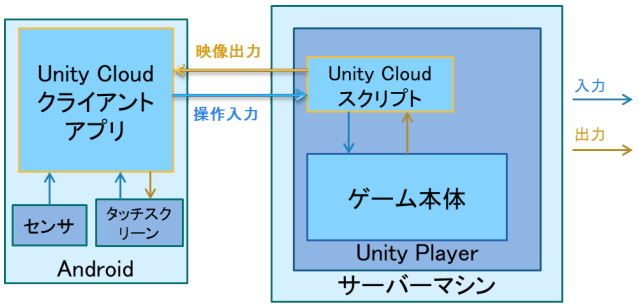

図 1 構成図：Unity スクリプトでの実装
Fig. 1 System structure:Implementation by Unity scripts

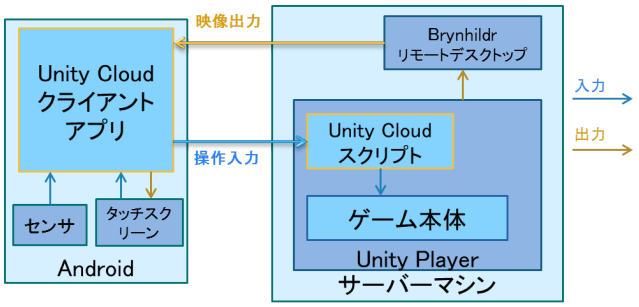

図 2 構成図：リモートデスクトップでの実装
Fig. 2 System structure:Implementation by Remote desktop

よって開発されたゲームをユーザに提供することが目的であり，個人の開発者が気軽にクラウドゲームを提供する事は出来ない．

1.2 目的

本研究では，個人開発者が手軽に自作のゲームをクラウド化出来るシステムを開発することを目的とする．システムの対象は，利用者が多いという点および拡張性が高いという点から，Unity を用いて Android 向けに開発したゲームアプリとする．本研究ではこのシステムを UnityCloud と呼称する．

クラウドゲーミングシステムにおいて映像出力は欠かすことの出来ない要素である．そこで，UnityCloud 開発の第一段階として，2 通りの映像出力の方法について実装，比較を行った．

2. システム構成

UnityCloud はサーバ・クライアント式の構成となっている（図 1，2）．サーバとクライアントはソケット通信によって 2 つのポートを使って情報のやりとりを行う．片方は操作入力を扱うポート，もう片方は映像出力を扱うポートとなる．それぞれの通信はマルチスレッドによって非同期的に行う．

2.1 クライアント

クライアントは Android アプリとして作成した．サーバの IP アドレスとポート番号を指定して接続する．サーバから受信した映像を画面に表示するとともに，センサ情報を取得し操作入力としてサーバへ送信する．2 種類の映像出力方法に対応して 2 種類のクライアントを作成した．

2.2 サーバ

サーバは主に Unity のスクリプトで作成した．このスクリプトを Unity プロジェクト内で動作させることで，その Unity プロジェクトをサーバ化する事ができる．

Unity スクリプトで実装された UnityCloud クラスは，クライアントから受け取った操作入力情報をプロパティとして保持する．UnityCloud クラスは static クラスとして定義されているため，これらのプロパティは外部のスクリプトから直接参照することが出来る．

映像出力は 2 通りの方法で実装した．

2.2.1 Unity スクリプトでの実装

Unity スクリプトによって Unity 実行画面のスクリーンショットを連続で取得し，クライアントへ送信する（**図 1**）．この方法のメリットは，サーバ側が Unity 内部で完結できる点，サーバの OS が Unity が動作する環境であれば Windows に限らず MacOS，Linux でも可能な点があげられる．デメリットとしては，スクリーンショットを撮影する処理によって Unity の動作に負荷がかかる点，マウス入力・音声出力が Unity の機能だけでは実装が難しい点がある．

2.2.2 リモートデスクトップソフトを利用した実装

IchiGeki 氏の開発したリモートデスクトップソフトウェアである Brynhildr[5] を利用して映像出力を行う（**図 2**）．Brynhildr は通信プロトコルを無償公開しているため，自由にクライアントを開発する事ができる．この方法のメリットは，マウス入力，音声出力が Brynhilfr の機能で容易に使えるという点である．デメリットは，Brynhildr が Windows 専用ソフトであるためサーバの OS が Windows に限定されるという点，Unity 以外のソフトを利用することからサーバ環境の構築に手間が増える点があげられる．

3. 性能評価

2 つの映像出力の方法に対して遅延時間に関する評価を行った．

3.1 測定法

Unity 内にてタイマーを表示し，サーバ画面とクライアント画面が共に写るように動画を撮影する．その動画をフレーム単位で切り出し，各フレームにてクライアントの表示がサーバの表示から何秒遅れているかを計測する．測定時の実験条件は以下のとおりである．

- サーバ CPU：Intel Core i3 530
- サーバ GPU：NVIDIA GeForce 8800GT

表 1 遅延時間
Table 1 Latency

	方法 1	方法 2
平均（ms）	101	91
最大（ms）	207	232
最小（ms）	69	47
標準偏差	27	32

- サーバ OS：Windows7
- クライアント端末：Nexus7（2013 年版）
- クライアント OS：Android4.3
- クライアントの Wi-Fi リンク速度：65Mbps
- 画面解像度：656×449
- 撮影時間：約 7 秒
- 計測フレーム数：200 フレーム

なお，サーバとクライアントは同一ローカルネットワーク内に存在している．

3.2 結果と比較

表 1 に 2 通りの映像出力方法の計測結果を示す．なお，方法 1 は Unity スクリプト内で実装した場合（2.2.1），方法 2 はリモートデスクトップソフトを利用した実装の場合（2.2.2）をそれぞれ表す．

平均して，方法 1 では 101ms，方法 2 では 91ms の遅延が発生していた．リモートデスクトップを利用した方法 2 が遅延時間は少ないが，その差は 10ms であり，標準偏差の範囲に収まっていることから大きな差であるとは言えない．その上で，個人開発者が手軽に導入できるようにするという本研究の目的から，Unity のみで実装できる方法 1 が UnityCloud に適切であると考えられる．

Yeng-Ting Lee らの研究 [6] では，クラウドゲームにおける遅延時間の増加は体感品質の低下を招くとしている．いくつかのゲームジャンルについて平均した場合，遅延時間の増加と体感品質の低下は比例関係にある事がわかる．しかし，アクションゲームなど厳密なリアルタイム性が低いジャンルについては，遅延なしの場合と遅延 100ms の場合で体感品質の低下がほぼ見られないものもあった．そのため，遅延時間 100ms は必ずしも無視できるとは言えないが，実用的な応答速度であると考えることができる．

4. まとめと今後の課題

本研究は，Unity で作成された Android 向けアプリをクラウド化するシステム UnityCloud を提案し，映像出力について 2 通りの方法を比較した．Unity 内部で実装できる方法でもクラウドゲームとして実用的な応答速度が得られることが確認できた．今後は以下の課題の解決を目指す．

(1) 操作入力への対応
現在，操作入力は一部センサ値の送信のみに限られている．ゲームプレイのためには各種操作入力への対応は必須である．タッチ入力への対応，Unity 側でセンサ値を保持するデータ型の整理などを行う．

(2) UnityCloud 導入時のスクリプト自動書き換え
現在の設計では，UnityCloud からの操作入力を Unity スクリプトで取得する場合，UnityCloud の専用クラスを参照するため，通常の Android アプリ向けの記述からスクリプトを書き換える必要がある．これはシステム導入の手間を増やしてしまうため，自動化することが望ましい．Unity エディタの機能拡張によって実現できると思われる．

(3) 遅延時間の削減
厳密なリアルタイム性が求められるようなゲームでは 50ms の遅延ですら致命的となりうる．幅広いゲームへの対応を目指す場合，遅延時間の削減が必要になる．

(4) マルチプレイヤーへの対応
現在は，一つのサーバに対して一つのクライアントしか接続することが出来ない．サービスとして提供することを考えると，この制約は実用的ではない．一つのサーバで複数のユーザを処理できる仕組みを考案する必要がある．

参考文献

[1] Unity3D
入手先 ⟨http://japan.unity3d.com/⟩

[2] NVIDIA GRID
入手先 ⟨http://www.nvidia.co.jp/object/cloud-gaming-jp.html⟩

[3] G-Cluster
入手先 ⟨http://gcluster.jp/⟩

[4] PlayStation Now
入手先 ⟨http://www.jp.playstation.com/info/release/nr_20140108_playstation_now.html⟩

[5] Brynhildr - 極高速リモートデスクトップ
入手先 ⟨http://blog.x-row.net/?p=2455⟩

[6] Yeng-Ting Lee, Kuan-Ta Chen, Han-I Su, and Chin-Laung Lei: *Are All Games Equally Cloud-Gaming-Friendly? An Electromyographic Approach* IEEE, Venice, Annual Workshop on Network and Systems Support for Games (2012)