Vol.2015-HPC-148 No.31
2015/3/3

Unityアプリケーションのクラウド化による高速化

高橋 悠[1]　伊藤 一輝[2]　成見 哲[2,a)]

概要：近年スマートフォンの普及に伴い，モバイル端末でゲームを遊ぶユーザが増えている．また，3Dゲームエンジンおよび統合開発環境である Unity がモバイルゲーム開発に広く使われており，企業に限らず個人開発者も多い．一方で，Unity の特徴である手軽に 3D 処理や物理演算を使えるといった利点は，比較的処理性能の低いモバイル端末では活かすのが難しいという問題がある．そこで，モバイル端末上では重い処理を行わせずネットワーク上に設置する Unity サーバーに処理を肩代わりさせる仕組みを開発した．モバイル端末上で走る専用のクライアントアプリがソケット通信によって操作入力をサーバーに送信し，サーバの実行した処理結果を画像として受信し表示することで，見かけ上は端末上で Unity アプリケーションが動作しているように見える．これにより，通常はモバイル端末で動かせない処理の重いアプリケーションであっても実用的な速度で実行できた．既存のクラウドゲーム環境と違い自分のアプリをクラウド化出来るのも特徴である．

1. はじめに

1.1 背景

近年，多機能型携帯電話スマートフォンやタブレット端末の普及が急速に進んでいる．総務省によると，平成 25 年でのスマートフォン世帯保有率は 62.6 %，タブレット端末世帯保有率は 21.9 %となっている [1]．スマートフォンやタブレットといった多機能型端末では，電話や E メール，ショートメッセージなどの情報通信端末としての用途に限らず，オーディオプレイヤーや web ブラウジング，ゲームアプリケーションといった娯楽目的で利用する場面も増えている．

また，スマートフォンやタブレット端末の普及やゲームユーザの増加にともない，ゲームアプリケーションの開発環境の整備も進んでおり，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[2]（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といった特徴から，企業・個人問わず多くの開発者から好まれている．

一方で，スマートフォンやタブレットといったモバイル端末は，ゲームアプリにおいて重要となるグラフィック性能などは，まだ一般的なコンピュータには及ばない．そのため，低性能端末を持っているユーザは最新のゲームアプリが快適に動かせないという問題が多く見受けられる．逆に，ゲーム開発者からすると，多くの人に遊んでもらおうと思うと低性能な端末でも動くように作る必要が出てくるため，表現の幅に制限がかかってしまう．前述の Unity は 3D グラフィックスや物理演算が手軽に使える事が大きな利点の一つであるが，どのようなモバイル端末でも動くゲームアプリを作ろうと思うと，これらの利点が活かしきれなくなってしまうという問題がある．

1.2 目的

本研究では，Unity で開発されたモバイル端末，特に Android 向けゲームアプリケーションをクラウドゲーム化するシステムを開発することを目的とする．なお，Android に着目する理由としては，モバイル用 OS として最大のシェアがあるためである．このシステムによって，モバイル端末でゲームアプリケーションを見かけ上高速に実行できるようにすることを目指す．

開発するシステムでは，ゲームアプリケーションは高性能なサーバ上で実行する．そのサーバに接続したクライアントとなる端末では，ユーザによる操作入力を検知してサーバに送信，およびサーバ上のゲーム実行の結果となる画面情報を受信し端末画面に描画を行う．これにより，見かけ上はその端末でゲームアプリケーションが動作しているように見えるが，実際は高性能なサーバで実行することによって，端末の性能差による影響を最小限にする事が期待できる．

[1] 電気通信大学　情報理工学研究科
[2] 電気通信大学　情報理工学部
[a)] narumi@cs.uec.ac.jp

2. 既存技術

2.1 画面転送によるモバイルシンクライアントの一検討

水野らは，Android 端末をクライアントとするモバイルシンクライアントシステムの検討と評価を行った [3]．モバイル端末のシンクライアント化により，個人データ保護やコンテンツ保護といったセキュリティ強化，モバイル端末の形式や性能の制限を超えたサービスの提供などが期待されるとしている．このシステムでは，シンクライアントサーバ上で仮想 Android を実行させ，シンクライアント化した Android 端末からネットワークを通じてサーバ上の仮想 Android を操作する．仮想 Android によって生成された画面を画面転送エンコーダによって圧縮しクライアントへ送信，クライアントはこれを受信してデコーダによって復号，画面表示を行う．このモバイルシンクライアントは，テキストメール，web 閲覧，メニュー操作といったアプリケーションにおいて問題のない性能であることを示している．ただし YouTube の再生においてはフレームレートの低下が見られたとしている．

本研究では，アプリケーション単位でクラウド化を行うことで，モバイル端末のみでも簡単に処理できるアプリケーションは通常通り実行できるようにする．

2.2 NVIDIA GRID

NVIDIA GRID[4] は NVIDIA が提供するゲーム，仮想デスクトップ，クラウドアプリケーション向けの GPU アクセラレーションを可能とする技術である．NVIDIA GRID GPU を搭載したサーバをクラウドとして稼働することで，高品質なクラウドゲーミングサービスを提供出来るとしている．実際に NVIDIA GRID を導入しクラウドゲーミングを提供しているサービスとしては，G-cluster[5] などがある．

既存のクラウドゲーミングサービスは個人で開発したゲームなどを手軽に配信出来ない．そのため，本研究では個人開発者が自作のゲームアプリケーションをクラウド化し配信出来るようにするシステムを開発する．

2.3 Amazon AppStream

Amazon AppStream[6] は Amazon Web Services が提供する，アプリケーションのストリーミングサービスである．Amazon AppStream を利用したい開発者は，専用の開発キット（Amazon AppStream SDK）を用いてアプリケーションを開発する．また，アプリケーションはサーバ側とクライアント側の両サイドを開発する必要がある．開発したアプリケーションは，Amazon AppStream に登録することによって，Amazon Web Services のクラウド上で実行されるようになる．アプリケーションを利用したいユーザは，そのアプリケーションの対応する専用クライアントアプリケーションを端末に導入することによって，ストリーミング配信を利用できる．

本研究では，スマートフォン向けゲーム開発に広く活用されている Unity で開発されたアプリケーションをクラウド化出来るようにする．

3. Unity の概要

Unity とは Unity Technologies 社が開発・提供するゲームエンジンおよび統合開発環境である．Unity はマルチプラットフォームのゲーム開発が可能であり，開発したゲームは Windows，MacOS X，Linux，iOS，Android，WindowsPhone，Windows ストアアプリ，ウェブブラウザプラグイン，Flash やコンシューマゲーム機など非常に多くのプラットフォーム向けにビルド可能である．また，物理演算エンジンの PhysX[7] を内蔵しており，物理演算が簡単に利用可能であるという特徴もある．Unity 内のスクリプトは C ♯，JavaScript，Boo のいずれかの言語を用いて記述出来，.NET フレームワークに基づくプラットフォームである Mono[8] の上で実行される．

4. システム概要

4.1 Unity Mobile Streaming

本研究で開発したシステムを「Unity Mobile Streaming」と呼称する．Unity Mobile Streaming は Unity を用いて開発された Android 向けゲームアプリケーションをクラウド化するためのシステムである．通常は Android 端末内にて実行されるアプリケーションを，アプリケーションの処理を行うサーバサイド，端末の入出力を行うクライアントサイドに分けることで，アプリケーションの処理自体は端末側のスペックによる影響を受けなくする．よって，クライアント側となる Android 端末よりも十分に高性能なコンピュータをサーバとして利用することによって，ゲームアプリケーションがスクリプト処理や物理演算，レンダリングといった点で比較的高い処理性能を求める内容であったとしても，Android 端末はクライアントアプリケーションの動作要件を満たすスペックさえあれば，ユーザがゲームアプリケーションを快適に遊べるように出来る．

4.2 システム全体の構成

Unity Mobile Streaming のシステム構成図を**図 1** に示す．Unity Mobile Streaming はサーバ・クライアント型のシステムである．サーバとクライアント間の通信は IP によるソケット通信を用いて行う．クライアントとなる Android 端末，すなわちユーザの利用する端末には，Unity Mobile Streaming の専用クライアントアプリを導入する．クライアントアプリはタッチ情報やセンサ情報といったゲームの操作に必要となる入力情報を端末から取得し，接

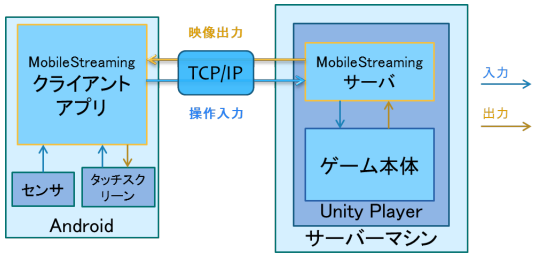

図 1 Unity Mobile Streaming のシステム構成図

・スクリプト
Scripts
├ MobileStreamingManager.cs
├ Input.cs
├ ScreenCapture.cs
└ Structs.cs

・プレハブ
UnityMobileStreaming

図 2 Unity Mobile Streaming プラグインの構成

続しているサーバへ送信する．サーバ側は送信された入力情報を元にゲームアプリケーションを動作させる．また，実行しているゲームアプリケーションの画面を定期的に画像としてキャプチャしクライアントに送信する．クライアントアプリは送信されてきた画像を端末のディスプレイに出力することで，ユーザは自分の操作しているゲームの画面を確認できるようになっている．

4.3 サーバ側アプリケーション

ゲーム開発者は開発したゲームの Unity プロジェクトに Unity Mobile Streaming プラグインを導入することによって，そのアプリケーションをサーバ化する事ができる．Unity Mobile Streaming 用サーバアプリケーションとなったプロジェクトはサーバの OS に合わせて Unity のスタンドアローンプログラムとしてビルドする．Unity Mobile Streaming プラグインの構成は**図 2** のようになる．C ＃言語によるスクリプト群と一つのプレハブから構成されている．

ゲーム開発者は以下の手順によって Unity Mobile Streaming プラグインの導入，およびサーバの稼働を行える．ただし，プロジェクトのスクリプトは C ＃にて書かれており，Android 向けを想定して開発されているものとする．

- 手順 1　Unity Mobile Streaming パッケージのインポート
- 手順 2　MobileStreaming プレハブのシーンへの追加
- 手順 3　ポート番号の設定
- 手順 4　スクリプト内の Input クラスに関わる部分を書き換える
- 手順 5　サーバの OS に合わせたスタンドアローンプロジェクトとしてビルド
- 手順 6　ビルドした実行ファイルを実行

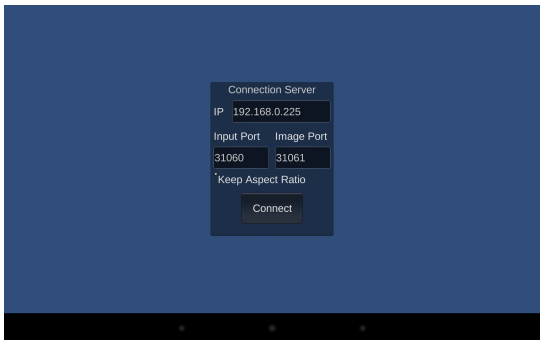

図 3 Unity Mobile Streaming クライアントアプリの初期画面

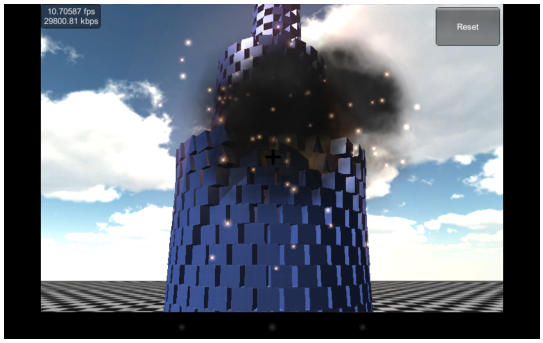

図 4 Unity Mobile Streaming クライアントアプリのゲーム画面

4.4 クライアント側アプリケーション

Unity Mobile Streaming サーバ化されたアプリケーションを遊ぶためには専用のクライアントアプリを用いる．このアプリはサーバの IP アドレスとポートを指定し接続する（**図 3**）ことで，そのサーバで稼働しているアプリケーションを遊ぶ（**図 4**）ことが出来る．クライアントアプリ自体は操作入力情報のサーバへの送信と，サーバから送られてきた映像の描画しか行わないため，1 つのクライアントアプリで Unity Mobile Streaming サーバ化されたアプリケーションは全て遊ぶことが出来る．

Unity Mobile Streaming の動作を**図 5** に示す．クライアントアプリは初期画面で接続先サーバの IP アドレスとポート番号を指定する．接続ボタンが押されるとゲーム画面へ遷移し，サーバとの接続をリクエストする．この時，接続に失敗した場合は再び初期画面へ戻る．ゲーム画面では，メインスレッドで画面の描画とセンサ値やタッチ情報の取得を行い，通信はそれぞれ専用のスレッドで行う．操作入力情報を送信するスレッドでは，UDP によるソケット通信を用いて定期的にサーバに各種センサ値とタッチ情報をバイナリ列に変換し送信する．画面情報の通信を行うスレッドでは，まず TCP による接続をサーバと確立する．

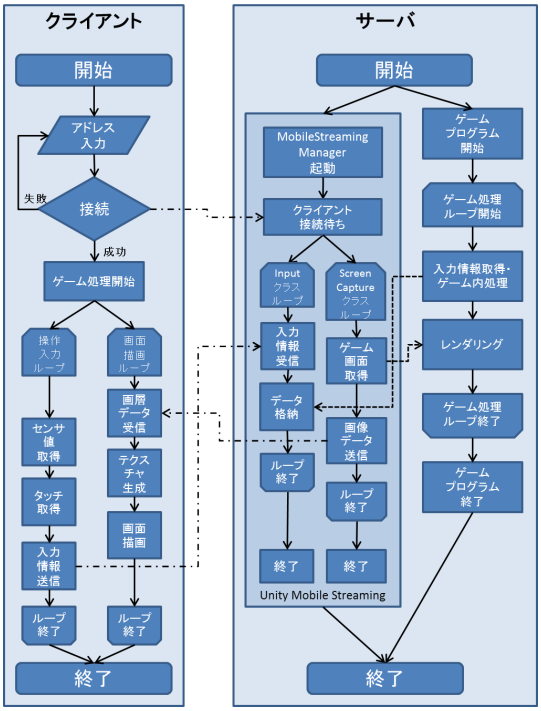

図 5 Unity Mobile Streaming の動作フローチャート

接続を確立した後は，クライアントとサーバでヘッダ情報をやりとりしたのちサーバから画像を受信を繰り返し行う．ヘッダ情報は受信する画像の解像度，およびファイルサイズが含まれた 16 バイトのバイナリ列である．画像は png 形式にエンコードされたデータで送られてくるため，受信したバッファをそのまま描画する事で画面の表示ができる．ゲーム画面は元のアスペクト比を維持して表示するか端末側に合わせてフルスクリーンで表示するかが選べる．

4.4.1 Android 向けクライアントアプリ

クライアントアプリ自体も Unity によって実装しているため，センサ値やタッチ情報の変換は必要なく，そのまま値を送信すれば良いという利点がある．Java による Android アプリとして実装するのに比べて，画像を表示する際に，一度バッファに保持してからテクスチャを生成・描画するためメモリ効率は落ちてしまうが，実行速度に大きな差は見られなかった．ここで，Unity スクリプトでソケット通信を行う場合.NET フレームワークのソケット通信を使うことになるが，この機能を Android 向けにビルドして使う場合，UnityPro ライセンスおよび Unity Android Pro ライセンスが必要になる．そのため，ソケット通信部分だけ Java コードの Android プラグインとして実装し，Unity から外部 Java コード呼び出し機能を使って通信を行わせた．このため，外部呼び出しにはその度に 1 フレームの遅延が発生するため，通信の遅延が Java での実装よりも大きくなってしまう．この問題は UnityPro 環境での開発なら容易に解決できると考えられる．

4.4.2 Windows タブレット向けクライアントアプリ

Unity で実装したクライアントアプリは Android プラグイン部分を除き，他はそのままでマルチプラットフォームに対応できる．そこで，OS に Windows8.1 を搭載したタブレット端末向けのクライアントアプリの開発も行った．Windows のスタンドアローンアプリケーションであれば.NET フレームワークのソケット通信が利用できるため，通信部分を Android プラグインから Unity スクリプトでの実装に書き換えた．しかし，現時点の Unity では Windows8.1 タブレットのタッチ入力やセンサ情報を取得することが出来なかったため，操作入力が出来ずクライアントアプリとしては動作しなかった．この問題は，将来的に Unity のアップデートによって解決する可能性がある．

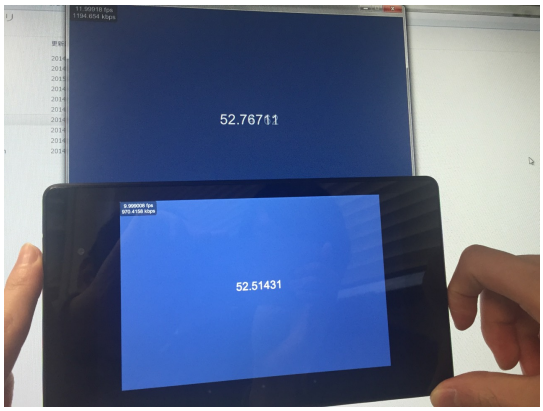

図 6 画面出力遅延の測定方法

5. システム評価

Unity Mobile Streaming の性能について各項目について実験評価を行った．

5.1 動作確認環境

動作の確認や性能評価は**表 1** の環境で行った．サーバとクライアントは同一のローカルネットワークによって接続されている．

5.2 画面出力の際に発生する遅延

画面を出力する際にネットワークの通信によって発生する遅延を計測した．ここでは，ゲーム画面のレンダリングが終わった時点から，実際にクライアント端末の画面に描画されるまでの時間を計測する．すなわち，サーバ側でゲーム画面のレンダリングが終了してから，画面のキャプチャ，画像の送信，クライアント側の画面描画にかかる時間が含まれている．

この実験は操作入力について問題がある Windows 版クライアントアプリでも同様に行うことが出来るため，Android クライアント端末と Windows8.1 クライアント端末の両方について行った．

5.2.1 実験方法

アプリケーションが実行されてからの時間をミリ秒単位

表 1 動作確認環境

サーバマシン	
OS	Windows7 64bit
CPU	Intel Core i3 530 2.94GHz
メモリ (RAM)	4GB
GPU	NVIDIA GeForce 8800 GT
クライアント端末 (Android)	
機種	Nexus7（2013 年版）
OS	Android4.3
CPU	Qualcomm Snapdragon S4 Pro 1.5GHz
メモリ (RAM)	2GB
GPU	Adreno 320 400MHz
ネットワーク	Wi-Fi IEEE 802.11 a/b/g/n
リンク速度	65Mbps
搭載センサ	加速度，GPS，周囲光 コンパス，ジャイロスコープ
バッテリー容量	3950mAh, 3.8V, 15.5Wh
クライアント端末 (Windows8.1)	
機種	geanee EDP-71
OS	Windows8.1 with Bing
CPU	Intel Atom Z3735G 1.33GHz
メモリ (RAM)	1GB
GPU	Intel HD Graphics
ネットワーク	Wi-Fi IEEE 802.11 b/g/n
リンク速度	65Mbps
搭載センサ	加速度
ネットワーク	
無線 LAN ルータ	BUFFALO Airstation WZR-HP-AG300H
準拠規格	IEEE 802.11 a/b/g/n
データ転送速度	最大 300Mbps
グラフィッククオリティ	
レンダリング	
Pixel Light Count	2
Texture Quality	Full Res
Anisotropic Texture	Per Texture
Anti Aliasing	Disabled
Soft Particles	false
シャドウ	
Shadows	Hard and Soft Shadows
Shadow Resolution	Medium Resolution
Shadow Projection	Stable Fit
Shadow Cascades	Two Cascades
Shadow Distance	40
その他	
Blend Weights	2 Bones
VSync Count	Every VBlank
Lod Bias	1
Maximum LODLevel	0
Particle Raycast Bud	256

で表示するゲームアプリケーションを実装する．このアプリケーションを Unity Mobile Streaming を使用して実行し，サーバ側のゲーム画面とクライアント側のゲーム画面が同時に写るように外部からビデオ撮影を行う（**図 6**）．この時，サーバ側ゲーム画面に表示されている時間とクライアント側ゲーム画面に表示されている時間の差が画面出力の際に発生する遅延である．約 20 秒間 30fps のカメラで撮影した全 600 フレームのうち，任意の 120 フレームについて遅延を計測した．また，サーバ側のゲーム画面解像度は 720 × 480 で実験を行った．

表 2 画面出力の遅延

	平均遅延時間（秒）
Android 版クライアント	0.24
Windows8.1 版クライアント	0.11

表 3 操作入力実験の平均計測結果

	計測時間（秒）
Unity Mobile Streaming 利用時	0.65
Android 端末単独	0.38

5.2.2 結果

画面出力遅延時間について，結果を**表 2** に示す．

Android 版クライアントと Windows8.1 版クライアントの遅延時間の差は，主に.NET フレームワークのソケット通信を利用しているか否かによる影響が大きいと考えられる．そのため，Unity Pro ライセンスのある環境下で.NET フレームワークのソケット通信が利用できれば，Android 版クライアントも Windows8.1 版と同等の性能になることが期待できる．

5.3 操作入力の際に発生する遅延

ユーザがクライアント端末で操作入力を行ってからサーバ側でゲーム内に反映されるまでのネットワークの通信によって発生する遅延時間を計測した．ここでは，クライアント端末がタッチ入力を検知してからサーバ側のゲーム内スクリプトがその入力を取得するまでの時間を計測する．

5.3.1 実験方法

ランダムなタイミングで画面に「Touch」の文字が表示され，文字が表示されてからタッチされるまでの時間を計測するゲームアプリケーションを実装する．非常にシンプルなアプリケーションのため，ゲーム内でのスクリプト処理やレンダリングにかかる時間は十分小さく無視できるものとする．このアプリケーションを，Unity Mobile Streaming を利用して実行した場合の結果と，Android のスタンドアローンアプリケーションとしてビルドして実行した場合の結果から，入力遅延時間を求める．Unity Mobile Streaming を利用した場合と，Android 端末単独で実行した場合で，それぞれ 30 回づつ試行した．また，サーバ側のゲーム画面解像度は 720 × 480 で実験を行った．

5.3.2 結果

それぞれ 30 回の試行の平均を**表 3** に示す．

この時，Unity Mobile Streaming を利用した場合の結果には，「タッチと表示されてから実際にタッチする，被験者の認知にかかる時間」と「入出力通信の遅延時間」が含まれている．また，Android 単独実行した場合には「被験者の認知にかかる時間」のみが含まれている．よって，Unity Mobile Streaming を利用した場合の結果から，単独実行した場合の結果と節 5.2 で求めた出力遅延を引いた値が，Unity Mobile Streaming の入力遅延の値となる．

よって，$0.65 - 0.38 - 0.24 = 0.03$ 秒が Unity Mobile Streaming の操作入力通信にかかる遅延時間である．

5.4 アプリケーション実行速度の比較

ゲームアプリケーションを，クライアント端末単独で実行した場合と Unity Mobile Streaming を通じてサーバマシンで実行した場合の実際の動作速度を比較した．ゲーム内のスクリプトや物理演算処理，レンダリングなどによってある程度の負荷がかかっている状況を想定する．

5.4.1 実験方法

ある程度の演算処理の負荷がかかる Unity アプリケーションとして，瀬戸口らの研究 [9] 内で開発された，「Unity による剛体の衝突シミュレーション」を利用した．この剛体シミュレーションは，一辺の大きさが 128 である立方体の空間内において，大きさ 1 質量 1 を持つ球体によって行われる．球体および空間の端となる壁の跳ね返り定数は 1 であり，静止摩擦および動摩擦ともに存在しない．実験では，まず初めに n 個の粒子に初速度を x, y, z の各軸方向に-50 から 50 の間でランダムに与え生成する．それ以降，各粒子が等速直線運動をする中で，粒子同士または壁との衝突を繰り返す．このプログラムを 1000 フレーム実行するのにかかった時間 t を計測し，$1000/t$ によってフレームレート（fps）を求める．

粒子数 n は 128，256，512，1024，2048，4096 の 6 通りを，各 5 回づつ試行する．この実験を Android 端末単独での実行，サーバマシン単独での実行，Unity Mobile Streaming を利用しての実行それぞれで行う．また，サーバ側のゲーム画面解像度は 720×480 で実験を行った．

5.4.2 結果

Android 端末単独で実行した場合，サーバマシン単独で実行した場合，Unity Mobile Streaming を利用して実行した場合のそれぞれの結果を，各粒子数毎に平均を求めグラフに示す（**図 7**）．グラフより，Android 端末で実行する場合，実行速度が大きく落ち込んでいる事がわかる．また，Unity Mobile Streaming を利用した場合の実行速度はサーバマシン単独で実行した場合と殆ど同じであるが，画面キャプチャ処理が加わることによって若干実行速度が落ちている．

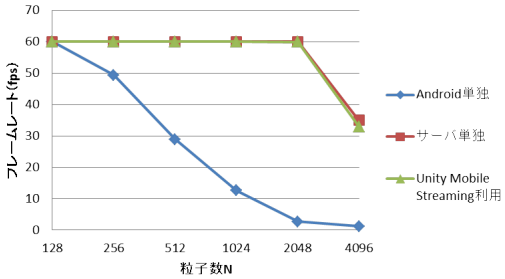

図 7 動作速度の比較

表 4 Unity Mobile Streaming の描画フレームレートと転送速度

粒子数	フレームレート (fps)	転送速度 (kbps)
128	11.0	1310.2
256	11.4	1513.6
512	10.9	1723.6
1024	11.0	2201.5
2048	10.6	2708.6
4096	10.5	3712.0

5.5 描画フレームレートと転送速度

Unity Mobile Streaming を利用する場合，ゲームの実行速度とは別に描画フレームレートがユーザの体感に影響をおよぼす．Unity Mobile Streaming は毎フレーム画面をキャプチャするが，通信はゲーム動作とは非同期で行われるため，実際にクライアントに送信している画像は通信時の最新の画面のみである．そのため，クライアントで描画されるフレームレートは実行フレームレートより低くなる．Unity Mobile Streaming では約 0.5 秒毎にその間実際にクライアントに送信した画像の数からフレームレートを，送信した画像の合計バイト数から転送速度を求めている．節 5.4 の実験の際に，実行中の描画フレームレートと転送速度を記録し，各粒子数の時の実行時平均を求め**表 4** にまとめた．このアプリケーションでは，描画フレームレートはおよそ 11fps で安定している．転送速度は，粒子数の多いほどより複雑な画像となりファイルサイズが大きくなるため，より高い転送速度となっている．転送速度によって描画フレームレートは変化していないため，転送速度がネットワークの通信速度を超えない限り，描画のボトルネックにはならないと考えられる．

5.6 消費電力の比較

Android 端末単独で実行する場合と比較し，Unity Mobile Streaming を利用する場合にどの程度消費電力が変化するかを調べた．Android 端末のバッテリー満充電時から，節 5.4 で利用した剛体の衝突シミュレーションを連続稼働した時，バッテリーが空になるまでの時間を計測した．粒子数 n は 2048 で実験を行い，Android 端末のディスプレイ輝度は常に最高にした．それぞれ，実験に使用した Android

表 5　消費電力の比較

	連続稼働時間	消費電力 (W)	消費電力量 (mWh/kf)
Unity Mobile Streaming 利用時	3 時間 52 分	4.00	19
Android 端末単独	4 時間 9 分	3.73	380
サーバマシン（1 台ホスト時）	-	140	650
サーバマシン（2 台ホスト時）	-	155	719

端末のバッテリー容量から消費電力を，さらに消費電力と節 5.4 の 2048 粒子の結果より 1000 フレーム処理するのにかかる電力量を求め，結果を**表 5** にまとめた．また，Unity Mobile Streaming を利用する場合新たにサーバマシンの電力消費が加わるため，ゲーム稼働中のサーバの消費電力も計測した．クライアント単体で見ると，Unity Mobile Streaming を利用することによって消費電力はおよそ 1.07 倍になっているが，処理能力あたりの消費電力量はおよそ 5 %と大きく削減出来ていることがわかった．サーバマシンの消費電力量を合わせると，$650 + 19 = 669$（mWh/kf）と Android 端末単独で実行するよりも不利となる．また，1 つのサーバで 2 つのゲームを稼働し 2 台のクライアントをホストした場合の消費電力とフレームレートを調べたところ，フレームレートは同等であった．しかし消費電力は，$719 + 19 \times 2 = 757$(mWh/kf) となり，Android 端末単独で 2 台動かす場合の $380 \times 2 = 760$（mWh/kf）よりも有利な結果となった．

5.7　実用性の考察

5.7.1　実行速度

Kuan-Ta Chen ら [10] によると，ユーザがゲームを遊ぶ際に感じるアプリケーションの実行速度は，ユーザの操作入力からそのレスポンスの出力までにかかる時間によって表せる．すなわち，Unity Mobile Streaming によって，サーバ上で高速に演算処理やレンダリングを行うことによって減った処理時間が通信の遅延よりも大きい時，Unity Mobile Streaming は有効なシステムであると言える．

ここで，節 5.4 の結果より，Android 端末単独で実行した場合と Unity Mobile Streaming を利用して実行した場合のフレームレートから，1 フレーム処理するのにかかる時間を計算する．1000/フレームレートによって 1 フレーム処理するのにかかった時間（ミリ秒）を求めると**表 6** のようになる．また，節 5.2 より画面出力にかかる時間は 240 ミリ秒，節 5.3 より操作入力にかかる時間は 30 ミリ秒とすると，Unity Mobile Streaming の通信によって増える遅延時間は 270 ミリ秒と言える．そのため，表 6 の Unity Mobile Streaming の値に 270 を加えた値が，Unity Mobile Streaming を利用している時の操作入力からレスポンスの出力までにかかる時間であると言える（**表 7**）．なお，クライアント端末そのものが入力を検知するのにかかる時間と画面を描画するのにかかる時間は十分小さく，また Android 端末単独で実行する場合も Unity Mobile Streaming を利用する場合も同等の時間がかかるため，無視する．表 7 より，2048 粒子のときおよそ 1.28 倍，4096 粒子のとき 2.8 倍高速である．よって，2048 粒子以上相当の負荷がかかる場合において Unity Mobile Streaming が有効である事が示せた．

表 6　1 フレーム処理するのにかかる時間（ミリ秒）

	粒子数					
	128	256	512	1024	2048	4096
Android 端末	16	20	34	78	367	840
Unity mobile Streaming	16	16	16	16	16	30

表 7　操作入力からレスポンスの出力までにかかる時間（ミリ秒）

	粒子数					
	128	256	512	1024	2048	4096
Android 端末	16	20	34	78	367	840
Unity Mobile Streaming	286	286	286	286	286	300

5.8　遅延時間

一方で，最も理想的な実行速度，すなわち 60fps でゲームが動作している場合，レスポンスの出力までにかかる時間は 16 ミリ秒であるため，通信によって増える遅延時間の 270 ミリ秒は小さいとは言えない．Yeng-Ting Lee らの研究 [11] 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．この研究では，どのようなジャンルのゲームがクラウドゲーミングとの相性が良いかを，筋電図によるアプローチで評価している．彼らの，遅延時間と顔面筋電図（fEMG）ポテンシャルの関係図によると，遅延時間 270 ミリ秒の時は，殆どのゲームタイトルにおいて，fEMG ポテンシャルの値が大きく（体感品質が低く）なっている．よって，現状の Unity Mobile Streaming では快適に遊べるとは言いがたい可能性がある．しかし，節 5.2 で Windows タブレット端末を用いた実験での結果を考えると，Android 版クライアントアプリでも.NET フレームワークのソケット通信を利用できれば出力遅延は 110 ミリ秒まで削減出来ると思われる．そうすれば，入力遅延と合わせて通信にかかる遅延時間は 140 ミリ秒となる．遅延時間が 140 ミリ秒の場合，ファーストパーソンシューティングのような厳密なリアルタイム性の求められるゲームジャンルでは，やはり快適に遊ぶのは難しいようであるが，アクションゲームやロールプレイングゲームでは fEMG ポテンシャルの値に大きな変化の見られないタイトルもあるため，ある程度快適に遊べるのではないかと思われる．

6. まとめと今後の課題

本研究では，モバイル端末で Unity アプリケーションを実行する際，処理性能によってゲーム開発に制限がかかるという問題の解決を目指し，Android 向けに開発された Unity アプリケーションをクラウドゲーム化するシステム Unity Mobile Streaming を開発した．Unity Mobile Streaming は Unity アプリケーションをサーバ化するプラグインと，サーバに接続するためのクライアントアプリから構成され，サーバ・クライアント間はソケット通信によって接続される．

Unity Mobile Streaming の性能について，通信による実行速度の面と遅延時間の面から評価実験を行った．実行速度については，アプリケーションを Android 端末単独で実行する場合と比較し，高い負荷のかかる状況において十分処理の高速化が出来たため，有効性が有ることを示せた．遅延時間については，現状ではまだ快適なプレイが出来るとは言いがたいが，Windows タブレット端末を用いた実験により，アクションゲームやロールプレイングゲームといった厳密なリアルタイム性が必要ないゲームジャンルにおいては快適にプレイ出来る可能性が示せた．

今後の課題として，遅延時間の削減，音声出力への対応，GUI 入力への対応，Android 以外のモバイル OS 版クライアントの開発などが挙げられる．

また，Unity Software License Agreement 4.x では，「ストリーミングおよびクラウドゲーミングに関する制限」として，Unity から別途許諾されない限り，直接的にも間接的にも，Unity によって開発したインタラクティブなコンテンツをストリーミング形式で頒布することを禁じている．そのため，本研究で開発した Unity Mobile Streaming を利用してクラウドゲームを頒布することは出来ない．よって，現状のライセンスでは，インターネット上に公開せず家庭内 LAN 限定で個人的に利用する，といった用途に限られる．

参考文献

[1] 総務省：平成 26 年版 情報通信白書，入手先 ⟨http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h26/index.html⟩
[2] Unity3D, Unity Technologies, 入手先 ⟨http://japan.unity3d.com/⟩
[3] 水野大輔，小澤一範，高田巡，仙田裕三，大塚壮一，佐保達也，木村司：”画面転送によるモバイルシンクライアントの一検討”，電子情報通信学会総合大会，通信講演論文集 2，p.166 (2011)
[4] NVIDIA GRID, NVIDIA. 入手先 ⟨http://www.nvidia.co.jp/object/nvidia-grid-jp.html⟩
[5] G-cluster, Broadmedia, 入手先 ⟨http://gcluster.jp/⟩
[6] Amazon AppStream, Amazon Web Services, 入手先 ⟨http://aws.amazon.com/jp/appstream/⟩
[7] PhysX, NVIDIA, 入手先 ⟨http://www.nvidia.co.jp/object/physx_new_jp.html⟩
[8] Mono, Mono Project, 入手先 ⟨http://www.mono-project.com/⟩
[9] 瀬戸口幸寿，成見哲：”仮想物理世界上で動く論理回路の実装と高速化の検証”，電気通信大学，情報・通信工学科卒業研究 (2014)
[10] Kuan-Ta Chen, Yu-Chun Chang, Hwai-Jung Hsu, De-YuChen, Chun-Ying Huang, Cheng-Hsin Hsu: ”On the Quality of Service of Cloud Gaming Systems”, IEEE, IEEE Transactions on Multimedia, Vol.16, No.2, pp.480-495 (2013)
[11] Yeng-Ting Lee, Kuan-Ta Chen, Han-I Su, Chin-Laung Lei: ”Are All Games Equally Cloud-Gaming-Friendly? An Electromyographic Approach”, 11th Annual Workshop on Network and Systems Support for Games, pp.1-6, Venice (2012)