

Android 向け Unity アプリの仮想化

電気通信大学 情報・通信工学専攻 CS コース 成見研究室

1331060 高橋 悠

1 はじめに

近年、スマートフォンの普及にともない、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が活発になっている。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いため、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。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、個人が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。

そこで、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。

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。

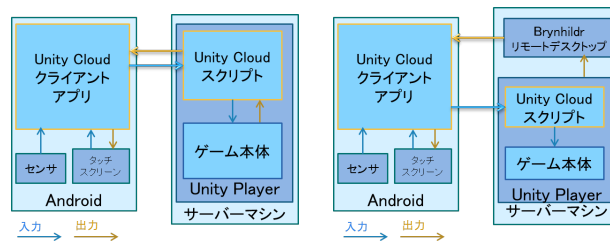


図 1 構成図: Unity スクリプトでの実装

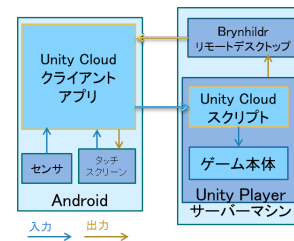


図 2 構成図: リモートデスクトップでの実装

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 Brynhildr の機能で容易に使えるという点である。デメリットは、

表 1 遅延時間

	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

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描画遅延時間と動作速度に関する評価を行った。測定時の実験環境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
- 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

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。

3.1 描画遅延測定

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。画面解像度は 656×449、撮影時間は約 7 秒で、200 フレームを計測した。

3.2 動作速度測定

Unity で負荷がかかった状態のモデルとして、1 フレーム毎に 1 個の球体オブジェクトを生成し、それらが重力下で衝突しあうデモを作成する。そのデモプログラム実行中の動作フレームレートを 0.5 秒毎に計測、記録する。2つの映像出力法に加えて、Android 用にビルドし端末単体でも実行した。

3.3 結果と比較

表 1 に描画遅延の計測結果を、図 3 に動作速度を示す。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。

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。また、動作速度は仮想化した場合は端末単独

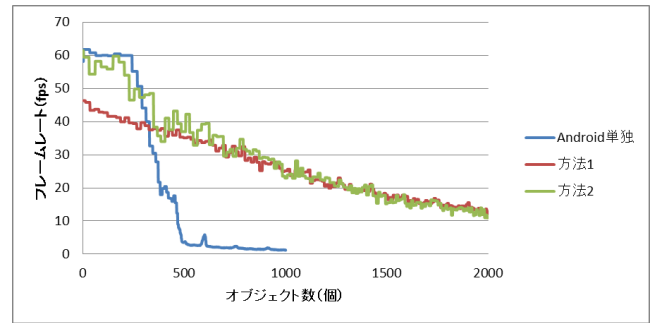


図 3 動作速度

での実行に比べどちらも比較的安定しており、負荷が小さい時は方法 2 が有利であるが、一定以上の負荷がかかった場合スクリーンショット撮影による差はほぼ無視できる事がわかった。よって、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。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られる。

4 まとめと今後の課題

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、動作速度が得られることが確認できた。今後の課題として、詳細な操作入力への対応、スクリプト自動書き換え機能の追加、さらなる遅延時間の削減が挙げられる。

参考文献

- [1] Unity3D
<http://japan.unity3d.com/>
- [2] NVIDIA GRID
<http://www.nvidia.co.jp/object/cloud-gaming-jp.html>
- [3] G-Cluster
<http://gcluster.jp/>
- [4] PlayStation Now
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
<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)