NICOGRAPH 2012, pp. 1234-1237

NUI デバイスを利用した 3D 仮想空間上での リアルタイムアニメーションの研究

井関 文一

東京情報大学総合情報学部

Research of the Real Time Animation in 3D Virtual Space using NUI Device

Fumikazu Iseki

Faculty of Informatics, Tokyo University of Information Sciences

iseki＠solar-system.tuis.ac.jp

アブストラクト

本研究では，Second Life や OpenSim などの 3D 仮想空間内で，リアルタイムにユーザの動きをアニメーションとして実行するシステムについて論じる．アニメーションデータの入力では，Kinect や Xtion などの Natural User Interface（NUI）と呼ばれるデバイスを利用している．

1. はじめに

現在，メタバースと呼ばれる 3D 仮想空間サービスの内，最も代表的なものは 米 Linden Lab 社の提供する Second Life[1] と，その Second Life と互換性のある OpenSimulator[2]（以後 OpenSim）である．

Second Life/OpenSim 内ではアバターに任意のアニメーションを実行させることが可能であるが，それらのアニメーションデータは予めサーバにアップロードして置かなければならない．そのため，アニメーションを利用する場合の自由度はそれ程高くない．またアニメーションデータはアップロードの際，Second Life/OpenSim の内部形式に変換されるため，元のアニメーションを忠実に再現できない場合がある．

もし，アニメーションデータを外部から直接入力し，それをリアルタイムに実行することが可能であれば，アニメーションを利用する際の自由度が増し，またその場での試行錯誤によるアニメーション映像（マシニマ等）の作成も可能となる．さらに，仮想空間内でのアバター間のコミュニケーション手段に幅を持たせることも可能となる．

本研究では，Second Life/OpenSim の Viewer に対して，外部から直接アニメーションデータを取り込んでリアルタイムにアニメーションを実行できるように若干の修正を行い，さらにネットワーク上でアニメーションの同期を取るために，アニメーションデータの中継を行うアニメーション中継サーバの開発を行った．

アニメーションデータを入力するためのデバイスとしては，NUI(Natural User Interface) 若しくは NI(Natural Interaction)と呼ばれる安価な MicroSoft社の Kinectや ASUS 社の Xtionを利用しているが，外部のアニメーションデータを取り込む部分のインターフェイスに従えば，他のデバイスやアプリケーションを利用することも可能となっている．

本研究と類似した研究としては，NUIデバイスを使用してユーザのジェスチャーを検出し，それを Second Life/OpenSim のキー入力の代替およびアニメーションのトリガーとして使用するシステムの構築[3][4]が挙げられるが，本研究ではユーザの動きそのものを3D仮想空間内で再現することを目的としている．

2. アニメーションデータ

2.1 アニメーションデータの流れ

Second Life や OpenSim ではアニメーションデータは BVH ファイルとして，Viewer を経由して SIM(Region)サーバにアップロードされる（図 1）．Viewer に読み込まれた BVH ファイルは，Viewer 内で独自の内部形式[5]（Second Life の通信プロトコルをエミュレーションする OpenMetaverse[6] や OpenSim では，これをbinBVHと呼ぶ）に変換されてからSIMサーバに転送され，

保存される（図1 ①, ②）.

Viewer でアニメーションを再生する場合,Viewer はアニメーションが選択された時点でキャッシュファイルを調べ，指定されたアニメーションデータがキャッシュされていなければ，サーバにデータ転送のリクエストを送信し（図 1 ③），受信した binBVH アニメーションデータをキャッシュファイルに保存する（図1 ④）.

その後，アニメーションの再生ボタンがクリックされると，キャッシュファイルからデータを読み込んで，それを各ジョイントの相対位置座標，回転のクォータニオンに変換してリストデータ（JointMotionList）に記憶する（図1⑤）. アニメーションの実行時には，タイムフレームに従って，各ジョイントのデータが随時 JointMotionList から読み出され，実行される（図1 ⑥）.

このように，Second Life/OpenSimでは，実行するアニメーションデータは予めサーバにアップロードして置かなければならず，使い勝手が良いとは言えない. また，BVHファイルから内部形式であるbinBVHに変換する際に変換が完全に行われず，アップロード前のBVHのアニメーションを元通りに再生することができない場合もある.

2.2 Viewer の修正

Second Life/OpenSim 上においてリアルタイムにアニメーションを実行するためには,Viewer の修正が必要である. つまり，Viewer の修正により，通常とは別の手順でアニメーションデータを実行する必要がある.

具体的な手法としては図1の JointMotionList の内容を直接書き換える方法と，JointMotionList からデータを読み出す部分で，データを外部のアニメーションデータに置き換える方法（図1 ⑦）が考えられる.

JointMotionList の内容を書き換える方法では外部データを一旦 binBVHに変換しJointMotionListを書き換えた後,Viewer がそれを再度座標ベクトルとクォータニオンに変換するため，計算の処理に無駄が生じる. そこで本研究では，後者の方法を採用した.

採用した手法の簡易的なアルゴリズムを図2に示す. 図2の Viewer のフローチャートにおいて，灰色の部分が今回の修正部分である. なお，外部プログラムから Viewer にアニメーションデータを渡す手段としては共有メモリを用いており，外部プログラム(Rinions)は逐次必要なデータを共有メモリに書き込む.

この手法では，予め取り替えの対象となる（ターゲットの）アニメーションデータを選定し，そのデータをサーバ上に無限ループのアニメーションデータとしてアップロードしておく必要がある. なお，このデータには，リアルタイムのアニメーションを実行する際に動かす各ジョイントの（ダミー）データが必ず含まれていなければならない. Viewer のアルゴリズムとして，アップロードされた BVH に含まれていないジョイントについては，アニメーション実行中にそのジョイントに関するデータの読み取りは一切行われないからである.

Viewer 側ではアニメーションの実行ループ中に，実行中のアニメーションの UUID が共有メモリのインデックス領域に存在するかどうかを確認し，もし同じ UUID が存在すれば Viewer は JointMotionList ではなく，共有メモリからアニメーションデータを読み込み，それを実行する. これにより，リアルタイムアニメーションが可能となる.

2.3 共有メモリ

ジョイント毎の共有メモリの構造を 図3に示す. 共有メモリは,チャンネルとアニメーション UUID の対応を取るインデックス領域と，ジョイントのデータそのものを格納するデータ領域に分かれる.

チャンネルはアニメーション毎のデータのメモリ上の位置を表す指標で，今回作成したシステムでは，デフォルトでチャンネル 0～21 が存在する. チャンネル 0 はローカルなデータ用であり，1～21 がネットワーク経由で受信するデータ用である. 21 個のネットワーク用チャンネルの内，1 つには自分自身のデータを格納するので,実質的にはネットワーク上で 20 個の外部ノードをサポートする.

データ領域にはジョイント毎にサイズが8の double の配列が用意してある. 各チャンネルの最初の 8byte は作業用で，続く 24byte がジョイントの相対位置座標の XYZ 成分，その後の 32byte がジョイントの回転を表すクォータニオンの各成分である.

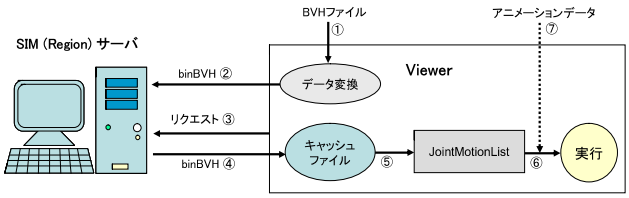

図1. アニメーションデータの流れ

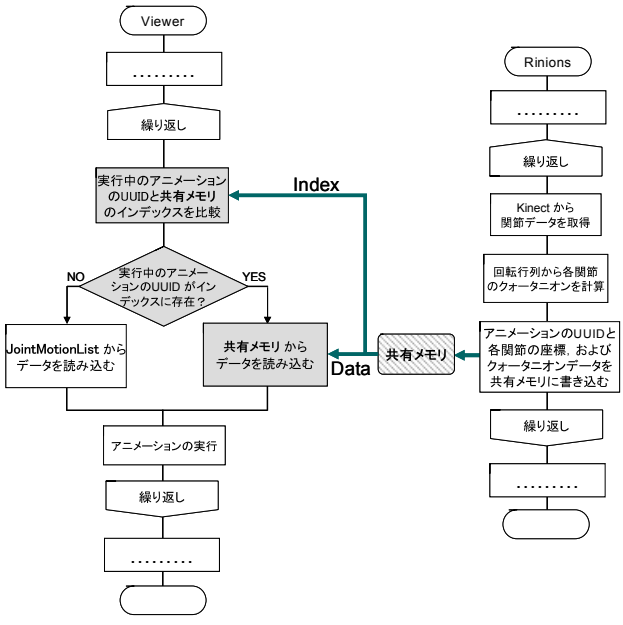

図2. システムのフローチャート
Viewer の灰色の箇所が修正部分

なお，それぞれの共有メモリの名前は，対応するSecond Life /OpenSimのジョイント名となっており，もし，Viewerがジョイントのデータの読み取りを行ったにも関わらず，そのジョイント名の共有メモリが存在しない場合は，そのジョイントの固有の動きは無効となる（親ジョイントの動きには従う）．

3. Rinions

本研究では，外部データの入力用機器としてMicrosoft 社のKinect または ASUS 社の Xtion を使用している． Kinect と Xtion はNUI(またはNI)用のデバイスで，PC でも使用可能な低価格の 3D モーションキャプチャ用機器である．

SDK としては Kinect ではMicrosoft 社の Kinect SDK[7]を，Xtion ではオープンソースの OpenNI と NITE [8]を使用することができる．Kinect SDK と OpenNI ではいくつかの違いは存在するが，主たる機能はほぼ同じである．

今回作製したシステム Rinions (Real Time Input from NI/NUI and Output to the Network and Shared Memory System) [9] では，Kinect SDK と OpenNI の両方をサポートし，インストールされているライブラリを自動識別して動作するようになっている．

Rinions はそれぞれの SDK から得られた各ジョイントの座標データを元に，図 4 に示すような各ジョイントの対応を考慮しつつジョイントの回転のクォータニオンを計算し，それを前述の共有メモリに書き込む．

クォータニオンの計算では，3 点のジョイントの座標から中央のジョイントのクォータニオンを求める．例えば，図 5 のジョイント J1, J2, J3 において，J2 のクォータニオン Q2 は，ベクトル a とベクトル b の外積ベクトル c と， a と b の内積角 θ から Q2 = (c*sin(θ/2), cos(θ/2)) と計算できる (a, b, c は単位ベクトルとする)．ただし，これは世界座標系でのクォータニオンであるので，ルートジョイントのクォータニオン Qr を用いて，~Qr*Q2*Qr としてルートジョイントに対する相対的なクォータニオンに変換しなければならない．

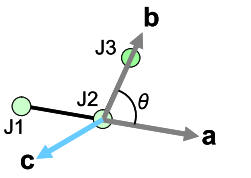

図 5. クォータニオンの計算

座標データからクォータニオンを計算する場合，例えば図 5 において，ベクトル b を回転軸とする回転を知ることはできないので，手足のねじれに関しては曖昧さが残る．

現時点では Rinions は，Kinect/Xtion から得られた座標データに対して，移動平均による平滑化以外には特に補正を行わずに，そのままクォータニオンを計算している．そのため，アバターのジョイントの動きが不自然，またはあり得ない動きになる場合があり，これらは今後の課題である．

4. アニメーションデータの中継

4.1 データの中継

Kinect/Xtion と Rinions のみを用いたシステムでは，アニメーションの再生はローカルな Viewer のみに留まる．そこで，ローカルで取得したデータをネットワーク上で同期させるために，図 6 に示すようなアニメーションデータの中継サーバの作成を行った．

アニメーション中継サーバはグループキーによるグループ化機能を有し，各クライアントソフト (Rinions) のローカルなデータを UDP データとして受信し，そのグループ内の全てのクライアントに転送する．これにより，グループ内でのアニメーションの同期を可能にする．

4.2 UDP データ

通信に UDP を使用するのは，データの転送速度を優先させるためである．UDP パケットデータはヘッダ (64byte) と複数のジョイントデータからなる可変長のデータである (図 7)．

ジョイントのデータの各要素は float(4byte)で転送され，ジ

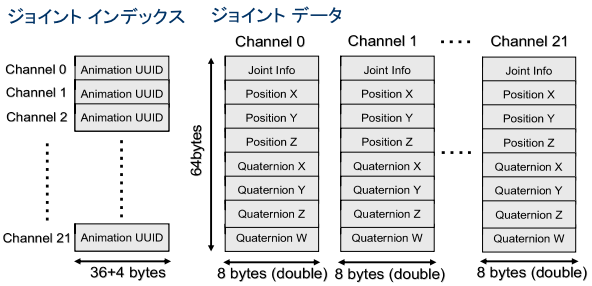

図 3. ジョイント毎の共有メモリの構造

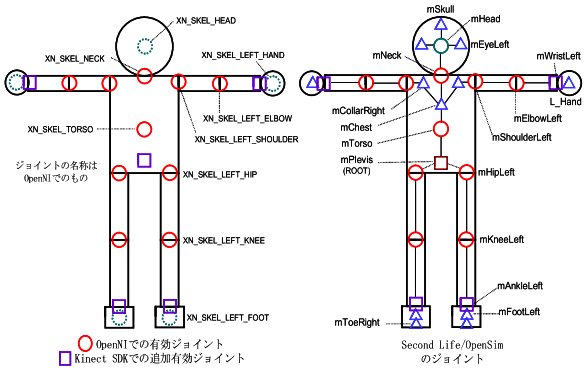

図 4. ジョイントの対応

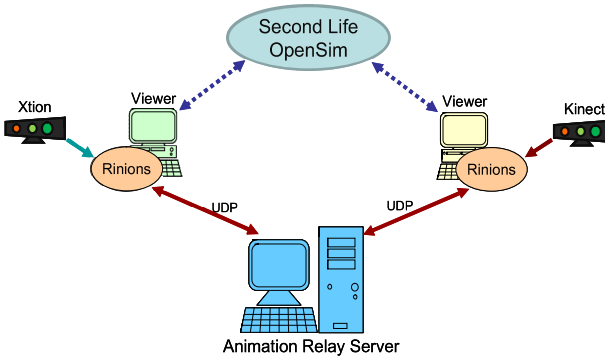

図 6. アニメーション中継サーバ

ョイントの座標データまたはクォータニオンデータはどちらか一方は省略可能である．即ち，ジョイントデータのサイズは通常は 32byte であるが，座標データを省略した場合は 20byte，クォータニオンを省略した場合は 16byte となる．また将来の拡張用に特殊ジョイントとして，指（左右），表情およびその他の情報を格納する 32byte×4 個のデータを付加している．

例えばジョイント数が 15（Kinect SDK での有意なジョイント数）で座標データはルートジョイントのもののみを転送するとすれば，全体のデータ長は 64 + 1×32 + 14×20 + 4×32 で 504byte となる．このデータサイズは IPv4 ではフラグメンテーションを起こさないことが保障されており（512byte 以下），高速な通信が期待できる．

また Rinions では，データ受信用ポートから一定間隔でサーバに対して Keep Alive パケットによる UDP ホールパンチングを行うので，一般家庭の BB ルータによる NAPT 内であっても何の設定も無しに使用することが可能である．

4.3 フロー制御

UDP には TCP のスライディングウィンドウ方式によるフロー制御のような機能は存在しないので，UDP データが大量にネットワーク上を流れると輻輳を起こしやすい．

本システムでは，サーバにおいてはグループ内のユーザ数に対して O(n^2)のオーダーで通信量が増加し，クライアント(Rinions)においては O(n)のオーダーで通信量が増加する．そのため何らかのフロー制御が必要となる．

Rinions ではサーバに対して最大転送速度のリクエストを送ることが可能である．また自らの出力フレームレートを調整することも可能であり，これらの機能を必要に応じて使用することによりフロー制御を実現する．

4.4 通信遅延

本システムではネットワークを通してアニメーションデータを転送するため，アニメーションの遅延が発生する．Kinect/Xtion ではデータの取得スピードが 約 30FPS であるので，遅延が発生しても 33.3m 秒未満であれば，遅延は 1 フレーム内に収まるので特に問題は発生しない．

この遅延許容時間は現代のネットワーク環境では，サーバと同じ AS 内やネットワーク的に近距離であればほとんど問題にはならない．しかしながら，遠地点同士（海外など）の通信においては検証が必要であり，これも今後の課題である．また，現時点ではセキュリティに関しても十分に高いとは言えない．

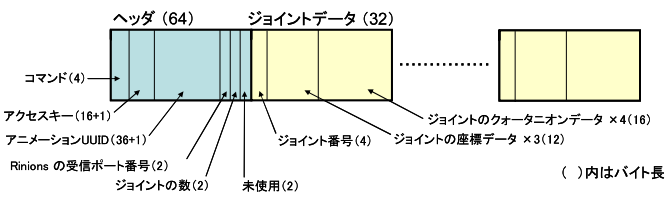

図 7. UDP パケットの構造

5. まとめ

本研究では，Second Life/OpenSim の Viewer に若干の修正を加え，さらにアニメーション中継サーバを構築することにより，NUI/NI デバイスから入力されたアニメーションデータをネットワーク上でリアルタイムに再生することが可能な事を示した．

図 8 に，実際にシステムを利用している画面の例を示す．画面では，手前の男性のアバターをローカルなユーザが操作し，奥の女性のアバターをネットワーク越しにリモートユーザが操作している．

本研究では，リアルタイムアニメーションの実行に際して，システムの通信遅延の評価などがまだ行われていない等の問題点が存在するが，Second Life/OpenSim などの 3D 仮想空間内でリアルタイムアニメーションが容易に可能になれば，マシニマの作成やアバター間のコミュニケーション手段の広がりに有益であると期待される．また，3D 仮想空間を使用したゲームなどにも応用が可能であると考えられる．

今後の課題としては以下の項目が挙げられる．

- 通信遅延の測定と評価
- セキュリティの考慮
- 骨格構造を考慮したジョイントデータの補正

なお，本研究で開発したRinionsはオープンソースとして公開している[9]．

参考文献

[1] http://secondlife.com/
[2] http://opensimulator.org/
[3] Evan A. Suma, Belinda Lange, et al. FAAST: The Flexible Action and Articulated Skeleton Toolkit, IEEE Virtual Reality 2011, pp.245-246, March 2012, Singapore, ISBN: 978-1-4577-0037-8
[4] John McCaffery, Alan Miller and Colin Allison, Gonna Build Me a TARDIS: Virtual Worlds for Immersive Interactive Experience, PGNET2012, June 2012, Liverpool, ISBN: 978-1-902560-26-7
[5] http://wiki.secondlife.com/wiki/Internal_Animation_Format
[6] http://openmetaverse.org/
[7] http://www.microsoft.com/en-us/kinectforwindows/
[8] http://openni.org/
[9] http://www.nsl.tuis.ac.jp/xoops/modules/xpwiki/?Rinions

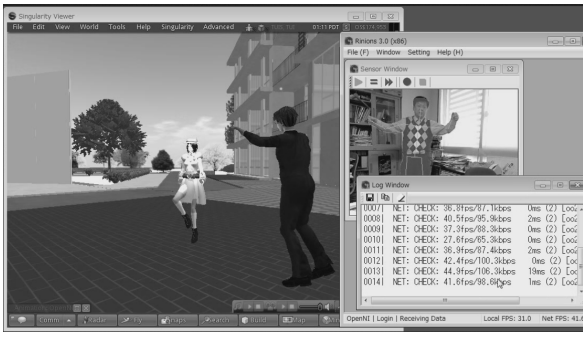

図 8. ネットワーク上での Rinions の動作例