マルチメディア通信と分散処理　85－12
（１９９７．　１１．　６）

分散サーバによるVR空間の実現

小野 仁† 西村 浩二‡ 相原 玲二‡
†広島大学大学院工学研究科 ‡広島大学総合情報処理センター

現在、エンターテイメントや軍事シミュレーション等、さまざまな目的でマルチユーザのVR（Virtual Reality：仮想現実）環境を構築する研究が行なわれている。そのための方法として、単一サーバによるクライアント・サーバ型が多く用いられているが、この方法ではユーザ数の増加に従ってサーバがボトルネックになり、接続可能なユーザ数が制限される。本研究では、サーバを複数設け、一つのサーバが管理する領域を分割し、多数のユーザをサポート可能な分散型サーバを提案する。また、この方式に基づくサーバおよびクライアントのプロトタイプを試作したので報告する。

キーワード：仮想現実、VRML、マルチユーザ、分散型サーバ

VR Space on Distributed Servers

Hitoshi Ono† , Kouji Nishimura‡ and Reiji Aibara‡

† Graduate School of Engineering, Hiroshima University

‡ Information Processing Center, Hiroshima University

Much research effort has been devoted to the issue of constructing multi-user Virtual Reality environments, for example, entertainment and military simulation. A centralized server model which manages the whole world is often employed to construct such environments. However, in this approach, the number of users connecting to the server is restricted because server's performance may become bottleneck as the number of users increases. In this paper, we propose a method using several servers instead of using the centralized server, and dividing the whole world into several regions, each of which is assigned to one of the distributed servers. This method can support large number of users by distributing the load to several servers. We demonstrate a prototype system based on the proposed method.

keywords:Virtul Reality, VRML, Multi-user, Distributed Server

1 はじめに

さまざまな目的でマルチユーザのVR環境を構築する研究が行なわれているが、現在はそのための方法として、単一サーバによる集中型のクライアント・サーバ方式（以下、「集中型サーバ」）が多く用いられている。この方法では、一貫性の制御が比較的容易になるという利点があるが、ユーザ数の増加に従ってサーバがボトルネックになり、接続可能なユーザ数が制限される。一方、サーバが存在しない完全分散型のものとして、SICS(Swedish Institute of Computer Science)のDIVE[1]、米国海軍大学院のNPSNET[2]等が提案されている。DIVEでは、一貫性保証が可能であるが、スケーラビリティに問題がありユーザ数が厳しく制限されている。また、NPSNETは非常に多くのユーザがシステムに参加できるが、一貫性制御は行なっておらず、使用しているDIS[3]プロトコルは軍事シミュレーションに大きく依存したものとなっている。

本研究では、サーバを複数設けて一つのサーバが管理する領域を空間により分割し、サーバに対する負荷を減らすことで全体として多数のクライアントをサポート可能な方式（以下、「分散型サーバ」）を提案する。また、ユーザすなわちクライアントプロセス（以下、「クライアント」）が処理を続行するために必要とする空間（以下、「ターゲット空間」）の情報をサーバプロセス（以下、「サー

バ」）から取得することにより情報を選択し、通信量を減らすことも考える。

2 分散型サーバ

2.1 分散型サーバの概要

通常、仮想空間におけるマルチユーザシステムではシーンにクライアントの存在や動作を反映することにより、複数のクライアントが同じシーンを見た時、他のクライアントの動きを知ることができる。そのシーン内でクライアントの存在を表示するには通常アバタと呼ばれる人の形をした3次元オブジェクトを用いる。

集中型サーバではクライアントが何らかのナビゲーションを行ない、シーン内のアバタの状態（位置や方向等）が変化した時にサーバに対して更新情報を送出し、サーバがその情報を他のクライアント全てに送る（図1(a)）。この場合、サーバが全体を管理するので、一貫性の保証は比較的容易であるという利点があるが、クライアントは一つのサーバに接続することを強いられるのでクライアント数が多くなるとサーバに負荷が集中し、接続可能なクライアント数が制限される。

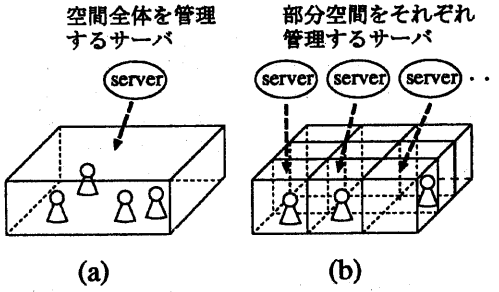

図 1: 集中型サーバと分散型サーバ

一方、分散型サーバでは空間を分割し、各部分空間に対してサーバを割り当てる（図1(b)）。サーバは自身が受け持つ空間は上で述べた集中型のクライアント・サーバ型と同様に管理する。これにより、クライアントが各空間に均等に分散するような状況においてはサーバの負荷が分散される。また、あるサーバの管理するクライアントが他の空間からの情報に対する要求を行なった場合、該当する空間を管理するサーバから情報を送信してもらい、その情報をクライアントに転送する。クライアントの要求する情報の空間的広がりは、メディアにより異なると考えられるが（図2）、本稿ではその一例として視覚をとり挙げている。

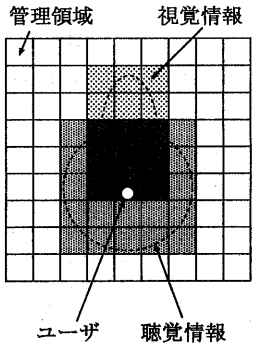

図 2: クライアントの要求する情報

2.2 集中型サーバとの比較

コネクション数と通信量の概算を行ない、分散型サーバと集中型サーバを比較する。ここでは、クライアントは各管理領域に対し均等に分散されているとし、クライアントは隣接する8近傍の領域から情報を要求すると仮定する。クライアント数を c、サーバ数を s とする。サーバに接続しているクライアント数 n は $n \approx c/s$ となる。またクライアントは常に i(Mbps) で情報を送信しているとする。

まずサーバのコネクション数について評価する。クライアントは8近傍の情報を要求するが、これはサーバ間通信により実現するため、1つのサーバにおけるコネクション数は $n+8$ となる。$c = 1000$、$s = 50$ とすると、分散型サーバでは一つのサーバのコネクション数は28となる。一方、集中型サーバでは全クライアントが一つのサーバに接続するため1000のコネクションが必要となる。

次に通信量について評価する。一つのサーバは、接続されているクライアントから $n \cdot i$ (Mbps)、隣接する8つのサーバから $8n \cdot i$ (Mbps) の情報を受信する。すなわち、$9n \cdot i$ (Mbps) の情報を受信する。$c = 1000$、$s = 50$、$i = 0.02$ とすると 3.6(Mbps) となる。一方、集中型では全クライアントからの情報を受信する必要があるため $c \cdot i = 20$(Mbps) の情報を受信する。

3 システム構成

3.1 システム構成概要

このシステムでは、複数のサーバを用いるが、それぞれのサーバは以下の情報を管理する。

1. 管理空間に存在するアバタの位置座標、向き、操作しているユーザの名前、色
2. 空間の形状データ（地形データ）
3. オブジェクトのデータ（形状、位置座標）
4. 隣接空間を管理するサーバのアドレスとポート番号

1～3はクライアントがローカルな空間を構築、更新するために必要な情報である。4はクライアントが隣接する他の空間に移動するために必要となる。クライアントが隣接空間に移動するとき、現在接続しているサーバとのコネクションを切断し、サーバから得た4の情報（隣接サーバ情報）により、次のサーバに接続する。

サーバは、このような情報を管理しておくことで、新しいクライアントが接続してきた際に現在のサーバでのオブジェクトやアバタの位置などをクライアントに知らせ、一方、新規クライアント情報を既接続クライアントへ通知する。またサーバは一貫性のためサーバ間で管理情報の交換を行なう必要がある。

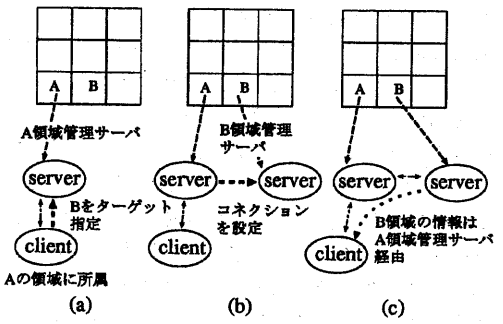

図 3: サーバ・サーバ間通信が起こる場合の例

クライアントは、分割された空間の一つに所属し、一つのサーバと接続している。ターゲット空間を指定するなどの依頼は接続しているサーバとの通信により行なう。管理空間をまたがるオブジェクトの移動などがあった場合や、クライアントがターゲット空間を指定した場合、サーバ間で通信が発生する (図 3)。

3.2 ターゲット空間変更による情報選択

クライアント側では、サーバから送信されるデータに基づいてローカルな 3 次元空間を更新する。通常、空間が広くなってくると、遠い所での変化はクライアントにとって必要で無いことが多く、近い所の変化のみを知りたいことが多い。そこで、アバタの位置や向きにより自動的にターゲット空間を指定し、必要な情報のみを受信することで全体の通信量を減らす。本システムでは、ターゲットは、当分割された空間ブロック単位で指定し、複数指定する場合このブロック単位で複数指定する。

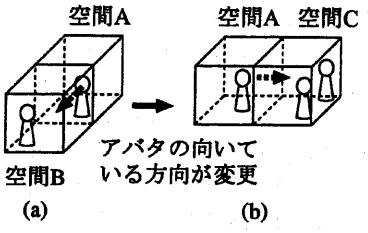

図 4: ターゲット空間の指定

図 4の例は、現在向いている方向の空間一つをターゲット空間として自動的に指定する場合を示している。最初、空間 A に存在するクライアント（アバタ）は空間 B の方向を向いている。このとき、空間 A にいるクライアントは空間 B をターゲット空間としている状態で、空間 B で起こった変化（空間 B 内にいるクライアントが動いた時に発生する位置情報等）は、A のクライアントに対して送信される。この例では、その後、A のクライアントは C の方に向きを変え、B を管理するサーバとのコネクションを切断し C をターゲット空間として指定している。この例では、前方の一つの空間しか指定していないが、複数の空間を指定することも可能である。

3.3 サーバの構成

サーバではまず最初にメインスレッドである Server スレッドが、クライアントからの接続要求

を待つスレッド (ClientListener)、サーバからの接続要求を待つスレッド (ServerListener)、全空間の位置を知るサーバ (WorldObserver) からの接続要求を待つスレッド (WOListener) を生成する。これらのスレッドは、異なったポート番号でコネクション要求を待っている。

ClientListener スレッドは、クライアントからの要求があると、クライアントとの間の通信を行なうためのスレッド (ClientManager) を生成する。以後、クライアント・サーバ間の通信はこのスレッドとクライアントとの間で行なわれる。クライアントが複数接続してくると、その数だけこのスレッドが生成される。図 5ではサーバ B において 2 つのクライアントから接続要求があり 2 つの ClientManager スレッドが生成されている。クライアントがコネクションを切断すると対応するスレッドは消滅する。

ServerListener スレッドは、他のサーバからの要求があると、そのサーバとの通信を行なうためのスレッド (ServerManager) を生成する。このスレッドが、サーバで起こった変化 (ユーザの位置座標の更新、オブジェクトの移動など) を、接続要求を出してきたサーバに対して送信する。図 5の例ではサーバ B がサーバ A に対して接続し、対応する ServerManager スレッドが生成されている。

WOListener スレッドは、空間どうしの接続の関係を知っている WorldObserver から、サーバ間の接続状態を示した情報を受けとるためのスレッドである。WorldObserver とサーバ間は、常にコネクションを確立している必要はないため、このスレッドから新たなスレッドは生成されず、情報を受けとってはそのまま処理する。この情報により、前の節で述べた、サーバでの管理情報の一つである他の空間を管理するサーバの情報を得る。

Receiver スレッドは、クライアントがターゲット空間を指定してきた際、その空間を管理するサーバに対して接続し、そのサーバからの更新情報を受けとる。Receiver スレッドは ClientManager スレッドが生成し、Receiver スレッドは、ターゲット空間として指定された空間を管理するサーバに対して 1 対 1 に生成される。一つでも、その空間をターゲット空間と指定しているクライアントがいれば、Receiver スレッドはそのサーバから更新情報を受信し、その空間をターゲット空間に指定してるクライアントがいなくなれば、対応する Receiver スレッドは消滅する。図 5ではサーバ B においてサーバ A からの更新情報を受信するために Receiver スレッドが生成されている。

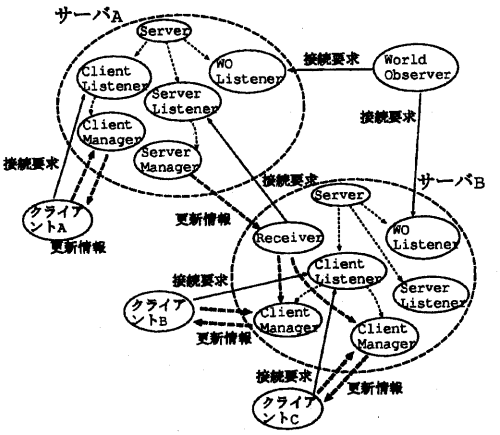

図 5: サーバの構成

クライアントがターゲット空間を指定してきた場合、まず、サーバはクライアントが新しく指定したターゲット空間は、すでに他のクライアントが指定しているかを調べる。他のクライアントが指定していなければ、その空間を管理するサーバに対してコネクションを設定し、更新情報を送信してもらうように要求を出すコマンド (YOU_ARE_INTERESTED_IN) を送り、Receiver スレッドを新たに生成する。他のクライアントがすでに指定していれば、ターゲットとして指定した空間を管理するサーバとの間には、すでにコネクションが設定されており、そのサーバから更新情報は送られてきているはずであるので、それを新たにターゲットを指定してきたクライアントに対して送信するように登録する。図 5ではクライアント A、B がサーバ A の管理する空間をターゲット空間として指定しておりサーバ A から送信された更新情報をそれぞれの ClientManager からクライアント A、B に送信している。クライアントがターゲット空間を指定してきた際、空間形状ファイルが必要となるので、サーバ間で空間形状ファイルのやりとりも行なわれる。ターゲット空間として指定されたサーバに対して、空間の形状情報を送信するよう要求してサーバから形状情報を受信し、クライアントに転送する。

3.4 クライアントの構成

クライアントは、サーバから送られてくる情報により、世界を更新する必要がある。さらに情報が、どこのサーバから送出されたかを知る必要がある。サーバから送られてくる情報は、サーバの管理する空間に対する相対位置座標、ID、コマンド、更新情報である。この空間の相対位置座標と、IDにより、どのアバタやオブジェクトに対応するクライアントからの情報かが判るため、アバタやオブジェクトを更新することが可能となる。

クライアント・サーバ間における通信は、TCP (Transmission Control Protocol) により行なう。クライアントはある空間を管理するサーバに接続した時、その空間内でクライアントを識別するためのIDをサーバから割り当ててもらい、空間内にいる間はその空間を管理するサーバとコネクションを確立しておく。以後、クライアントからの情報には、このIDをつけてサーバに送出する。また、サーバからクライアントへの情報にもこのIDが負荷される。これはマルチキャストやブロードキャスト通信に対応するためである。その後、その空間の形状ファイル、オブジェクトのファイル、隣接空間を管理するサーバのアドレスを受信する。ターゲット空間の指定のように、クライアントはサーバに対してコマンドを送出することでサーバに処理を依頼する。クライアント・サーバ間において、やりとりされるコマンドには現在以下のものがある。

CHANGE_SERVER クライアントが空間を移動し、隣接サーバに接続を切替えるとき、サーバに対して送出する。

ADD_TARGET クライアントがターゲットを指定する時に送出する。このコマンドの後に、3つの整数値 (x,y,z 座標) によりターゲット空間を相対的に指定する。例えば、クライアントが前方（北、z 軸の負の方向）を指定する時、コマンドの後に 3 つの整数値 (0,0,-1) を送出する。また、このあとに、空間の形状ファイルを要求するかしないかを示す論理値 (1bit) を送信する。

REMOVE_TARGET ターゲット空間として指定していたが、もうそこからの情報を必要としない場合に送出する。

TELL_ME_NEAR_AREAS 隣接するサーバのアドレスとポート番号を教えてもらうためのコマンドで、通常最初にサーバと接続した時に送出する。

GET_WORLD_SHAPE サーバに保存されている空間ファイルを要求するためのコマンドである。これも、通常最初にサーバと接続した時に送出する。

DISCONNECT クライアントが終了するときに送出する。

3.5 管理領域の移動によるサーバ切替

現在いる空間を出て、隣接する次の空間に移るために、クライアントはサーバを切替える必要がある。クライアントは、今いる空間を出る時、サーバに対して、サーバを切替える旨を示すコマンド (CHANGE_SERVER コマンド) を送る。サーバは、要求を受けとったクライアントとの間のコネクションを切断し、クライアントに対応する Client-Manager スレッドを終了させ、以後、サーバにおける更新情報はクライアントに対して送られなくなる。サーバでは、現在設定されている他のサーバに対するコネクションを調べ、そのコネクションが設定されている先のサーバに対して、どのクライアントもターゲットとして選んでいない場合、メッセージを送信し、コネクションを切断する。

4 プロトタイプシステムの試作

本稿で提案した分散型サーバのプロトタイプシステムを試作した。クライアント用のブラウザとして Sony Community Place (Version1.1 Beta6)[5] を用い、3次元オブジェクトの記述には VRML2.0 [4] を用いた。VRML2.0 は以下のような特徴を持つ。

- 3次元オブジェクトの動的な追加・削除が可能
- Java 言語など、他の言語を用いて通信等様々な機能を実現できる

試作システムでは、クライアントの通信機能を Java で記述し、VRML より利用している。また

サーバもJava言語を用いて記述している。図6に試作システムのクライアント画面の例を示す。地面の色が分散型サーバの管理領域に対応している。

図 6: 試作システムのクライアント画面

5 おわりに

本研究では、サーバを複数設け、サーバの管理領域を分割することによりサーバに対する負荷を減らし、全体として多数のユーザをサポートするシステムを提案した。現在プロトタイプシステムを実装中である。試作システムでは空間を等分割しサーバを割り当てたが、空間の分割方式については検討が必要である。空間を分割し、サーバを割り当てる場合の問題としては以下の2つが考えられる。一つは負荷バランスの問題である。すなわち、分割領域に所属するクライアント数が均等になるとは限らず、最悪一領域に全クライアントが集中することもあり得る。もう一つは、仮想空間上の配置と実際のクライアントやサーバのネットワーク的な配置の関係である。これらは必ずしも一致しないため、場合によっては通信路を大きく圧迫することもあり得る。

その他、サーバを分割するために生ずる一貫性の保証とサーバ間通信方式に関するトレードオフについて考察する必要がある。

謝辞 本研究の遂行にあたり、貴重な御意見を頂いた広島市立大学有川正俊助教授、天野晃助教授、前田香織講師、開和生助手、大阪大学下條真司助教授に深く感謝致します。

参考文献

[1] O.Hagsand: Interactive MultiUser VEs in the DIVE System, IEEE Multimedia Magazine, Vol.3, No.1, 1996, http://sics.se/dce/dive/dive.html

[2] M.R.Macedonia, M.J.Zyda, D.R.Pratt, P.T.Barham, S.Zeswitz: NPSNET: A Network Software Architecture for Lage Scale Virtual Environments, Presence, Vol3, No.4, http://www-npsnet.cs.nps.navy.mil/npsnet

[3] Distributed Interactive Simulation (DIS), http://dis.pica.army.mil/index.html

[4] The Virtual Reality Modeling Language Specification Version2.0, ISO/IEC CD 14772 August 4, 1996, http://vag.vrml.org/Specifications/VRML2.0/

[5] Virtual Society on the Web Community Place, http://vs.sony.co.jp