グラフィックスライブラリへの３次元音場生成ライブラリの融合

永山　悟　小池英樹

電気通信大学大学院 情報システム学研究科

Email：satoru,koike@vogue.is.uec.ac.jp

３次元グラフィックスによる仮想空間において，より現実感を表現するためには３次元音場の利用が必要である．しかし実際に３次元音場を利用する場合，別々に用意されているグラフィックス作成のライブラリと３次元音場生成のライブラリを使わなければならず，簡単には作成することができない．近年VRMLにおいてこれらを同時に利用できるようになり，今まで３次元音を利用した仮想空間をなかなか作成できなかったものが用意に作成できるようになった．しかしこの３次元音は簡易的なものであり，より現実的な音とはいえない．そこで本研究では，３次元グラフィックスライブラリと３次元音場生成のライブラリを融合した．作成したライブラリを用いることでより現実感の得られる仮想空間の構築が可能になる．

Integrating 3-D Sound Library into Graphics Library

Satoru Nagayama and Hideki Koike

Graduate School of Information Systems
University of Electro-Communications

To make a realistic virtual world, 3-D sound capability is getting more important. However, programs with 3-D sound tend to be complex since graphics library and 3-D sound library exist independently.

This paper proposed a basic framework for integrating 3-D sound library and graphics library in scene graph API. Programmers can add realistic 3-D sound just by adding a 3D-sound node to the scene graph which is used to render the scene.

1. はじめに

今日，仮想現実感は多くの分野に利用されている．宇宙開発や軍用機開発，遠隔ロボットの制御のような高い技術を必要とする分野の補助システムとしての高度な仮想現実感から，アミューズメントやアートのような身近にあるようなものまで様々である．またグラフィックスライブラリの普及に伴い，いままで専門の技術者によって作成されていた仮想空間を，３次元グラフィックスを用いて誰にでも作成することができるようになった．

これらの仮想空間をより現実的にするためには，３次元音場を利用することが重要である．３次元音場生成のための研究も多く行われている．原音場の物理特性を比較的多数のスピーカを用いて模倣する考え方に基づく音場再生技術[1)2)3)]や，受聴者の耳入力信号を原音場での収録時に得られた耳入力信号に一致させる考えに基づく音場再生技術[4)5)]など様々な手法による３次元音場生成の試みがなされている．

このように仮想現実感を表現するために多くの研究がなされているが，この２つを同時利用して仮想空間を作成するようなものはそれほど多くはない．この理由の１つに，３次元グラフィックスと３次元音場生成のライブラリが別々に用意されているためと考える．例えば，仮想空間にオブジェクトを描き，そのオブジェクトの位置から音を出力したい場合，オブジェクトの描画の部分と３次元音場生成のために必要な位置情報などのパラメータを計算する部分が別々に行われるなどして面倒である．

これに対し近年VRML[9)]において，３次元グラフィックスと３次元音を同時に利用できるようにサポートしている．VRMLでは，後述するシーングラフにSoundノードとAudioClipノードの２つのノードを，オブジェクトを描画するノードの子供にすることで，VRMLで作成した仮想空間に３次元音を埋め込むことができる．

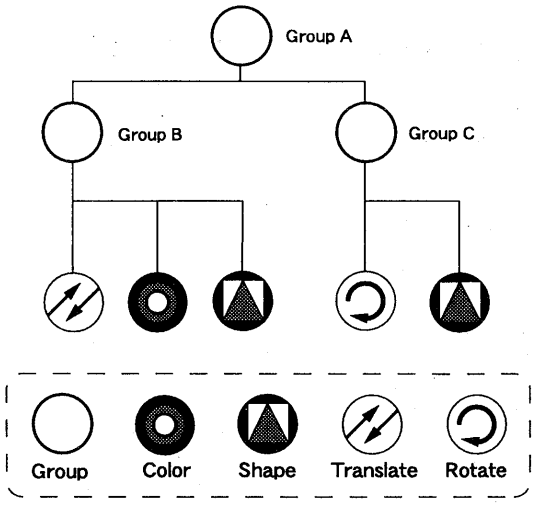

図1　シーングラフ

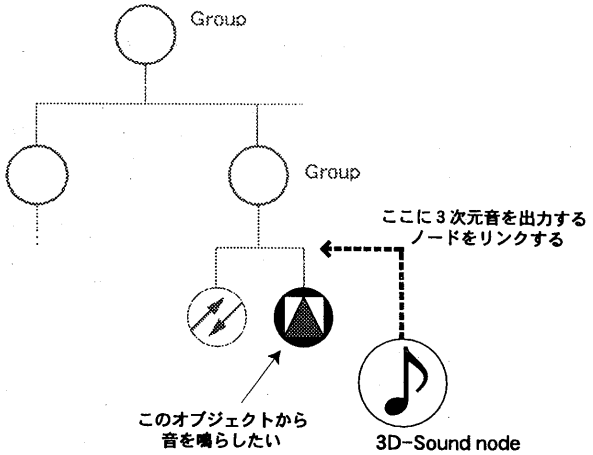

図2　音ノードの概念

しかしVRMLで利用できる3次元音は，強弱をつけることで音の移動感を持たせる疑似的なものであり，現実的な3次元音を生成することはできない．

そこで本研究では，3次元グラフィックスライブラリと真の3次元音場生成ライブラリを融合することで，仮想空間内に3次元音場を生成できるライブラリの作成を行った．3次元音場生成装置は後述するRoland RSS-10[11]を用いる．そのとき，できるだけ容易に仮想空間に3次元音場を生成するために，VRMLのようにシーングラフの概念を用いて構成し，Soundノードを作成することで実現した．

本論文では，2章ではグラフィックスライブラリと3次元音場生成ライブラリを融合するための基本方針を述べ，3章では両ライブラリの融合の説明をし，簡単なプログラム例を示す．4章ではVRMLと今回作成したライブラリとの簡単な比較を行った結果について述べ，5章では本ライブラリで作成したアプリケーションを簡単に説明する．最後に6章で課題を述べる．

2. 基本方針

2.1 シーングラフ

前述したVRMLやOpen Inventor[10]において用いられている手法で，Shape, Material, Translate, Scale, Rotate, Groupのようなグラフィックスオブジェクトを各々ノードとして表現し，このノードを用いて階層的なグラフ(ツリー)構造を構成する．この構成された階層構造をシーングラフと呼ぶ．そしてシーンの描画時にこのシーングラフを先行走査(深さ優先順走査)し，その順に描画作業が行われる．図1のシーングラフでは，Group AからGroup Bに移動し，座標移動した位置にcolorノードで指定された色でshapeノードの図形が描画される．次にGroup Cに移り，回転移動したあとにshapeノードの図形が描画され終了する．

シーングラフは，現在3次元グラフィックスを作成するための手法として一般化しつつある．そこでこのシーングラフを拡張して3次元音を利用できるように設計を行った．

2.2 3D-Soundノードの概念

前節で述べたシーングラフに対して音が利用できるように設計する場合，独自のSoundノードを作成することが考えられる．そのSoundノードを作成するときに，次のことを考慮に入れることでより使いやすいノードになると考える．

- Soundノードを音を鳴らしたいオブジェクトの存在するGroupノードにつなぐだけで音が鳴るようにする．図2にその概念を示す．Soundノードを作成するときに位置やオブジェクトの情報をユーザが指定するのではなく，作成されるシーングラフからその音源に対する位置情報や対象オブジェクトを認識し，3次元音場が生成できるようにする．
- 簡便さのため，できるだけ少ないノード数で3次元音場生成のプログラミングができるようにする．
- RSS-10での3次元音場生成にはさまざまな値を設定し，デバイスに送信しなければならない．この設定をプログラム中に行うのは非常に面倒である．そこで，できるだけ簡単にデバイスの設定が行えるようにする．
- ユーザが構築したシーングラフで用いているSoundノード以外からも3次元音が出力できるようにする．

以上の概念をもとに作成したノードについて次章で説明する．

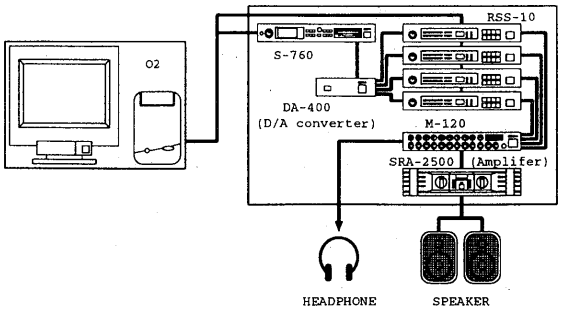

図3　システム構成図

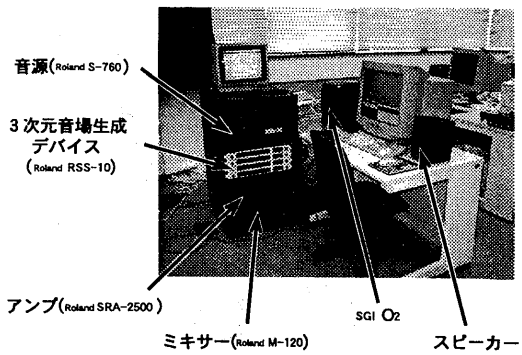

図4　システムの概観

3. 実　　装

3.1 システムの構成

本システムのハードウェア構成を図3に示す．また，基本的な装置の説明を以下に示す．

- Silicon Graphics O2 (Graphics Workstation)
  仮想空間を作成するために用いる．またその作成された仮想空間で利用する音の出力や3次元音場生成のための各パラメータを送信している．
  - CPU : MIPS R5000(180MHz)
  - Memory : 64Mbytes
  - OS : IRIS Version 6.3
- Roland S-760 (Sound module)
  音を出力する装置である．音の出力の命令はO2より行われ，送信されてきたMIDIメッセージをもとに制御される．S-760はディジタルサンプラーであり，実世界の音をサンプリングして音源として利用することが可能である．
- Roland RSS-10 (Sound space processor)
  3次元音場を生成するものである．RSS-10は距離感，方向感による3次元音場生成のほかに，直接音と壁や床などで反射した反射音もあわせて聞くことで，さらなる距離感を得ることができる．また，空間内の残響感を出すこともできる．これらにより，フランジング効果やドップラー効果を実現でき，より現実的な音場の生成を行うことができる．O2より送信されたMIDIメッセージからシステムの設定をし，S-760より送信された音を加工し，出力する．
- Roland M-120 (Mixer)
  各RSS-10で生成された3次元音を合成するミキサーである．

3.2 3D-Sound ノードの作成

本研究では，3D-Soundノードを2つのノードにより実現する．以下で説明する**Sound**ノードと**Device**ノードがそれである．Soundノードは，主に音の出力に関する設定を行うノードである．一方Deviceノードの方は，先に説明したRoland RSS-10を操作するために必要な各種設定を行うためのノードである．

3.2.1 Sound ノード

本システムで用いているS-760は，MIDIメッセージにより音の出力・停止を制御できる．また出力に使うチャンネル，そしてプログラム番号もMIDIメッセージで命令することで指定可能である．チャンネルとは，使用できるMIDIキーボード数と考えると解りやすい．また，プログラム番号というのは出力する音の種類番号である．MIDIを使うときに利用できるチャンネル数は16である．この16個のチャンネルに使いたい音(プログラム)を割り当てることで利用できる．また，S-760では1つのチャンネルに複数の種類の音を割り当てることができる．以上のことから，音を出力するときに必要となる最低限の情報は『このチャンネル(channel)をこの楽器(program)に設定し，その楽器のこの音程(note)をこれくらいの音量(velocity)で出力したときの音』というものである．そこでchannel，program，note，velocityの指定が必要となる．

ここまでが音の出力に関する説明だが，その音を出力するタイミングは，仮想空間内では以下の2つが考えられる．

(1) 時間により音の出力タイミングが決定
　時計の時報や遮断機の音，鳥の鳴き声や飛行機の音のような環境音はこの時間によるタイミングで音の出力を開始する．

(2) ユーザのマウスイベントに対して音の出力タイミングが決定
　ドアをノックする音やチャイムを押したときの音のような人間が操作することで音が出力されるようなものはイベントで音の出力を開始する．

よって，Soundノードを作成するときに，この2つの区別がはっきりするように2つの関数を用意して設定を行うことにした．(1)の時間による出力のタイミング

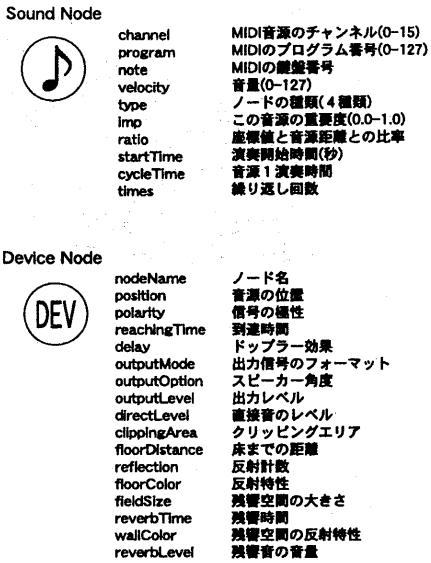

図5 Sound ノードと Device ノード

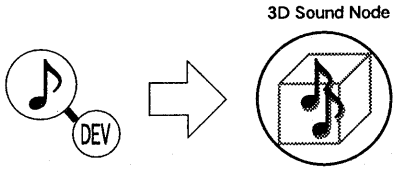

図6 3D-Sound ノードのシンボル

の取り方は，プログラム起動時を0としたときのスタートタイム (startTime) を設定することで音の出力が開始される．設定値単位は秒であるが，小数の値で設定することでより細かな設定もできる．(2) のマウスによる出力のタイミングの取り方は，クリックに対して次のような音の出力の仕方が考えられる．

- マウスをクリックしたら音が出力する
- マウスがクリックされるごとに音が **ON/OFF** する
- マウスを押している間だけ音を出力する

この3種類の出力法 (type) を実現するために，音の出力法を変化させるためのフラグを用意する．

また音には一定の長さがあり，その音が1度鳴り終ったら終了するような状況ばかりではない．例えば，スーパーでは同じコマーシャルが繰り返されているのをよく聞く．そこで，このような繰り返しをサポートするために，繰り返しの回数を設定 (times) することにする．また，この繰り返しがどのくらいの間隔 (cycleTime) で行われるのかも設定することになる．この2つのパラメータの設定で，この音を何秒ごとに何回鳴らすという設定ができる．以上より図5のようなノードを作成した．

3.2.2 Device ノード

音の出力・停止，その他の各種設定は前節に設計した Sound ノードを用いる．しかし，それはあくまでもサウンド・モジューラである S-760 に必要なパラメータであり，3次元音場を生成する RSS-10 の情報ではない．そこで RSS-10 で3次元音場を生成するために必要なパラメータを保持するために作成したのが Device ノードである．

Device ノードは図5に示すようなパラメータを持つ．それらは，ドップラー効果，音の出力レベル，床での反射，残響感に関するものである．このノードを設定し，パラメータを送信することで様々な状況を想定した3次元音場を生成できる．

この Device ノードは RSS-10 のデバイス数だけ用意するのではなく，各 Sound ノードに1つ用意しなければならない．これは，作成したい音源ごとに，表現したい音が微妙に異なるためである．また，自分が生成したい音場を指定のデバイスで変換するのではなく，任意のデバイスにより行われるので，どうしても Sound ノードに1つの Device ノードは必要となる．そこで Device ノードは図6のように，各 Sound ノードにリンクされて利用する．

この Device ノードのパラメータを，プログラム作成中に Sound ノードを作成した個数分設定を行うのは面倒であり，決して容易な作業ではない．そのため，Device ノードは Sound ノードを作成したときに，デフォルト値に設定されたものが自動的に作成される．

Device ノードは，ユーザによって自由に設定が変えられるように種々の関数を用意してある．しかし，関数を使って設定するには非常に手間がかかる．また音の微妙な設定は，経験による感が必要であり，理想に近い3次元音を得るまでに各パラメータの設定を繰り返さなければならず，決して簡単に利用できるとはいえない．そこで Device ノードが作成される際，自動的にデバイス値設定メニューを作成するようにした．このメニューを利用することで，プログラム実行時に音を聞きながら好みの音に設定にできる．さらに，その設定した各 Device ノードのパラメータは，プログラム終了時にデータファイルに保存されるので，次からは Device ノードの設定が不要となり，直ちにユーザの理想の音を提供できる．

このように，デバイスに関する設定の大部分はメニューにより行われる．そのため，実際にはこの Device ノードを意識しなくてもプログラミングを行うことができる．そこで，Sound ノードと Device ノードを組み合わせられたものを1つの **3D-Sound** ノードと定義する．この 3D-Sound ノードのシンボルを図6に示す．

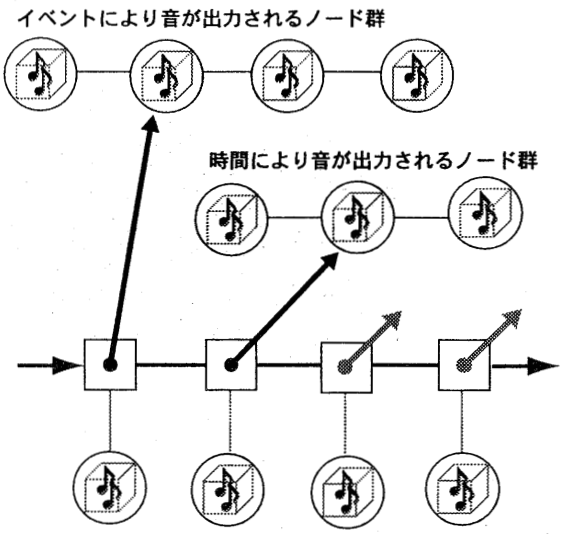

図7　3D-Sound ノードの出力構造

3.3　3 次元音場の生成

3D-Sound ノードはシーングラフ作成後，図 7のようにノード同士がリンクされる．このとき，ユーザの操作 (イベント) により出力される 3D-Sound ノードと時間により出力のタイミングをとる 3D-Sound ノードに分類され，2 つのリストを作成する．

ユーザのマウス操作で選択されたオブジェクトに対応した 3D-Sound ノードをシーングラフ内から探す際，作成したシーングラフを Root から走査するのは時間がかかる．最悪の場合，グラフを最後まで辿らなければならない．まだ小さなシーングラフであれば，マウスで選択されてから，目的の 3D-Sound ノードを捜しあて，S-760 と RSS-10 にパラメータを送信し，音が実際に出力されるまでの時間差はさほど気にならない．しかしシーングラフが大きなものになると，シーングラフの走査に時間がかかってしまい，出力のタイミングが気になってしまう．そこで，3D-Sound ノードをリストにすることで，このシーングラフの走査を行わないようにしている．さらに 2 つにリストを分割することで，イベント用の 3D-Sound ノードの探索が行いやすくしている．

その 3D-Sound ノード群から実際に音を出力する作業に移行するために，スイッチを用意した．このスイッチが指している 3D-Sound ノードがデバイスを占有することになる．このスイッチが切り替わることで出力音を切り替えている．このスイッチは利用する RSS-10 の数だけ用意する必要がある．また，このデバイス数の設定はユーザにより設定することもできるが，基本値は 4 台を利用するように設定してある．

利用できるデバイス数より出力される 3D-Sound ノードが多い場合，3D-Sound ノードの割り当てられているオブジェクトの位置とリスナーとの距離，また各 3D-Sound ノード持つ優先度 (imp) を用いて計算し，出力する音源を選出する．優先度の計算は以下の式で行われる．

$$order = \frac{imp}{(distance)^2}$$

出力を待たされているノードは待機状態となり，デバイスを割り当てられている 3D-Sound ノードが出力を終わりデバイスが空き状態になるか，優先度が現在利用中の 3D-Sound ノードを上回るかしたときに出力される．

3.4　環境音の作成

以上までは，音をオブジェクトに対応させて出力することを考えて設計してきたが，オブジェクトとは関係なく音を操作したいときも考えられる．例えば，仮想空間に流れる BGM のようなものである．また実世界においても，どこから音がしているのか分からないような音が存在する．強風や雨の音などである．これらの音は実体を持たないものであったり，複数の音が混ざりあってできる音である．そして，ユーザが移動をしても音の位置が変わるようなものでもない．

そこで，シーングラフに依存しない音情報の出力が行えるように，前節で説明した出力ノードを選別するスイッチにあらかじめ 3D-Sound ノードを持たせた．このノードに自由に書き込みを行い，自分の意図したノードを作成できる．例に挙げた雨の音のようなものは，仮想空間をみるためのカメラの位置とシンクロさせることでどこでも同じ音を実現できる．また，このノードの重要度を高めに設定しておけば，ユーザがどこに移動しても音を鳴らしておくことができる．

3.5　簡単なプログラム例

では，実際に作成したライブラリで簡単なプログラム例を示す (図 9)．このプログラムは，図 10(右) のように，視点の上空をテクスチャーのヘリコプターが旋回するものである．このプログラムで作成されるシーングラフは図 10(左) である．

このシーングラフを作成するとき，接続させたいノード同士を addChild() 関数によって接続している．図 9の 26 行目の **makeSound()** 関数で 3D-Sound ノードを作成している．この関数の引数は，『ノード名』，『チャンネル番号』，『プログラム番号』，『ノート番号』，『音量』となっている．ここで作成したノードは『チャンネル 1 をプログラム・ナンバー 126(MIDI 音源でヘリコプター音に定義されている) 音程 60 を音量 100 で出力する』ように設定をした．このノードを，音を出力したいオブジェクト (テクスチャーのヘリコプター) と

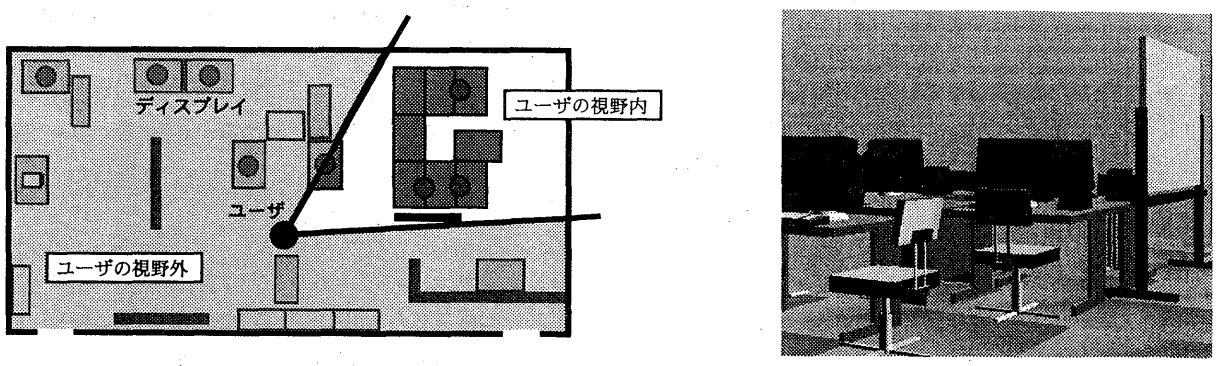

図8　アプリケーションで用いた仮想空間

同じGroupノード(Group2)と接続することで(図9の27行目)，3D-Soundノードを利用できる．

また図9の34~38行目はアニメーションを行っている箇所であり，revolutionノードとrotateノードの回転角度を変化させることで行っている．

4. VRMLのSoundノードと3D-Soundノードの比較

この章では，本論文にて説明してきた3次元音場生成ライブラリと，シーングラフAPIで疑似的に3次元音を利用可能なVRMLとの簡単な比較実験を行った．比較は目の前を横断する列車音，正面から向かってきて上昇していくジェット機，そして右上方から左下方へと進むヘリコプターの3つをヘッドホンで聞いてもらい，どのように移動しているか・臨場感はどうかなどのコメントをもらった．以下にまとめる．

(1) VRMLで作成したモデル
- 横への移動はわかる．しかし上下方向，前後方向の移動はわかりにくい．
- 元々のwavファイルの音とさほど変わらない．
- オブジェクトの移動動作を見ながら聞くと，ある程度そのように聞こえてくる．

(2) 本ライブラリで作成したモデル
- 横の移動だけでなく，上下・前後方向の移動もある程度わかる．
- 実世界で聞くような音になっている．
- 前方の感覚があまりしない．
- オブジェクトの移動動作と合わせると，より臨場感がでた．

グラフィックスとサウンドを融合して実験を行った結果，人間の視覚からの情報により，音の移動感が補間されどちらを用いても移動しているようには聞こえる．しかし音質は，本ライブラリを用いて作成した音の方が臨場感があり，仮想空間をよりリアルに演出することができるという結果を得た．以上のことからVRMLで利用できる3次元音よりも，リアルな3次元音を提供できることがわかった．

5. アプリケーション例

仮想空間内で，視界に入らないオブジェクトの位置を判断することは難しく，あちこちさまよってしまうことがある．たとえば，図8(左)のような仮想空間で，ユーザの視野外にあるオブジェクト(ディスプレイ)を探す場合，ユーザは仮想空間のどのあたりにオブジェクトが存在するのかすぐにはわからない．そこで仮想空間のオブジェクトの位置座標から3次元音を出力し，オブジェクトの位置をユーザに知らせる簡単なアプリケーションを構築した．

実際に作成したモデルの一部を図8(右)に示す．これは研究室の簡単なモデル(図8(左))であり，机上の各計算機に3D-Soundノードを割り当てている．ユーザはこの仮想空間内で計算機を探すとき，メニューに登録してあるホスト名を選択することで，その計算機の位置から音を出力することができる．そしてその出力された音を指標とし，その計算機を視野にいれることができる．

今回は簡単なプロトタイプを作成したが，実際に仮想展示室のようなものに適用可能であり，自分のみたい作品の方から音がしてくるようなものが考えられる．また，ただ音を出すだけでもつまらないので，その作品に対する説明などを流しても面白い作品になると考える．

6. 課　　題

6.1 音源のデバイス数への依存

本研究で用いたRoland RSS-10はデバイスの性能上，移動音源の生成に1台必要である．実際に実現したシステムでは，4台のRSS-10を利用しているので，同時に生成される3次元音は4ヶ所と制限されてしまう．

しかし，同時に多数の3次元音を定位させても，同時に聞きとることは非常に困難であり，逆に聞きづらい仮想空間となってしまう恐れがある．

6.2 描画時間による遅延の考慮

人が画像の描画と音の聞き取りで起こる遅延に対して違和感を持たずにいられる時間は約0.2秒である．しかし，より現実的な仮想空間を作成すると，描画するオブジェクト数が増加し，1フレームに対する描画時間が増加してしまう．それに伴い音データの送信も送れてしまい，移動感がうまく表現できなくなってしまう．そこで描画時間を短縮するために，LOD(Level Of Detail)や井上[8]の手法を用いて描画数を削減することで遅延時間を小さくさせる必要がある．

6.3 アプリケーションの実世界への拡張

今回のアプリケーションは，仮想空間内での位置認識の補助を行った．これは実世界にも利用できると考える．そこで今後3次元音を実世界のオブジェクト情報の認識に利用するためのアプリケーションの作成を行いたい．

7. おわりに

本研究では，3次元グラフィックスライブラリと3次元音場生成のためのライブラリの融合を目指し，シーングラフの拡張を行った．3D-Soundノードを用いることで容易に仮想空間に3次元音を生成できる．これによって，より現実的な仮想空間を手軽に構築することが可能になる．

参考文献

1) E.Mayer, W.Burgtorf and P.Damaske : An apparatus for the electroacoustical simulation of sound fields. Subjective auditory effects at the transition between coherence and incoherence, *Acustica*,15,pp.339-344 (1965).
2) 横山 健司：モノラル，ステレオに次ぐ第3の音〈オムニサウンド〉実測音場データを格納した音場創生機 DSP-1, *JAS*,J.26(8),pp29-36 (1986).
3) 八木 哲：新しい音場再生 DSP, *JAS*,J.29(4),pp10-15 (1989).
4) D.H.Cooper and J.L.Bauk : Prospects for transaural recording, *J.Acoust.Soc.Am*,41,pp263-264 (1989).
5) 浜田 晴夫：バイノーラル音場再生系について，日本音響学会誌,48,pp250-257 (1992).
6) 大木 直人, 亀倉 龍, 阿部 圭一, 岡田 謙一, 松下 温：人工現実感を用いた音情報検索, 情報処理学会研究報告,Vol.93,No37 IM-11-1,pp1-8 (1993).
7) 大木 直人, 阿部 圭一, 寺本 邦男, 岡田 謙一, 松下 温：VCP: 音による仮想空間を用いた情報環境の提案, 情報処理学会論文誌,Vol.36,No6,pp.1342-1349 (1995).
8) 井上正行, 小池英樹：フラクタルを用いたシーンの自動簡略化手法, グラフィックスと *CAD* シンポジウム予稿集,pp67-72 (1997).
9) Jed Hartman, Josie Wernecke : The VRML 2.0 Handbook, Addison-Wesley Publishing Company.
10) Josie Wernecke : The Inventor Mentor, Addison-Wesley Publishing Company.
11) Sound Space Processor RSS-10 取扱説明書, Roland.

```
1:  Dlist *revolution, *rotate;

2:  void main (int argv, char** argc) {
3:    Dlist *Root, *Group1, *Group2, *translate;          /* 変数宣言 */
4:    Dlist *floor, *helicopter, *Sound;
5:    VGcamera *cam;
6:    VGlight  *light;

7:    fvInit(argc, argv, 500, 500);                       /* 3次元グラフィックスの初期化 */
8:    a3dInit(1);                                         /* 3次元空場の初期化    *
                                                           * 引数は利用デバイス数 */
9:    Root = makeGroup();
10:     cam = makePerspective(0.0, 0.0, 1.0,              /* カメラの作成／登録 */
                              0.0, 0.0, 0.0,
                              30.0, 1.0, 1.0, 50.0);
11:   addCamera(Root, cam);
12:   light = makeLight(20.0, 20.0, 20.0, 0, GL_LIGHT0);  /* ライトの作成／登録 */
13:   addLight(Root, light);

14:   floor = makeShape(makefloor(10.0, 10.0));          /* 床面の作成／登録 */
15:   addChild(Root,floor);
16:   Group1 = makeGroup();
17:     revolution = makeRotation(0.0,0.0,1.0,0.0);
18:     addChild(Group1, revolution);

19:     Group2 = makeGroup();
20:       translate = makeTranslate(0.0,4.0,20.0);
21:       addChild(Group2, translate);
22:       rotate = makeRotation(0.0,0.0,1.0,0.0);
23:       addChild(Group2, rotate);
24:       helicopter = makeTexture("helicopter.rgb");     /* ヘリコプターの作成／登録 */
25:       addChild(Group2,helicopter);
26:       Sound = makeSound("helicopter",0,125,60,100);   /* ヘリコプター音作成／登録 */
27:       addChild(Group2, Sound);
28:     addChild(Group1, Group2);
29:   addChild(Root, Group1);
30:  addWorld(Root);

31:  fvMainLoop();
32: }

33: GLfloat revolutionValue, rotateValue;

34: void Idle (void) {                                    /* アニメーションを行う関数 */
35:   revolutionValue += 2.0; rotateValue += 10.0;
36:   setRotation(revolution, revolutionValue, 0.0, 1.0, 0.0);
37:   setRotation(rotate, rotateValue, 0.0, 1.0, 0.0);
38: }
```

図9　プログラミング例

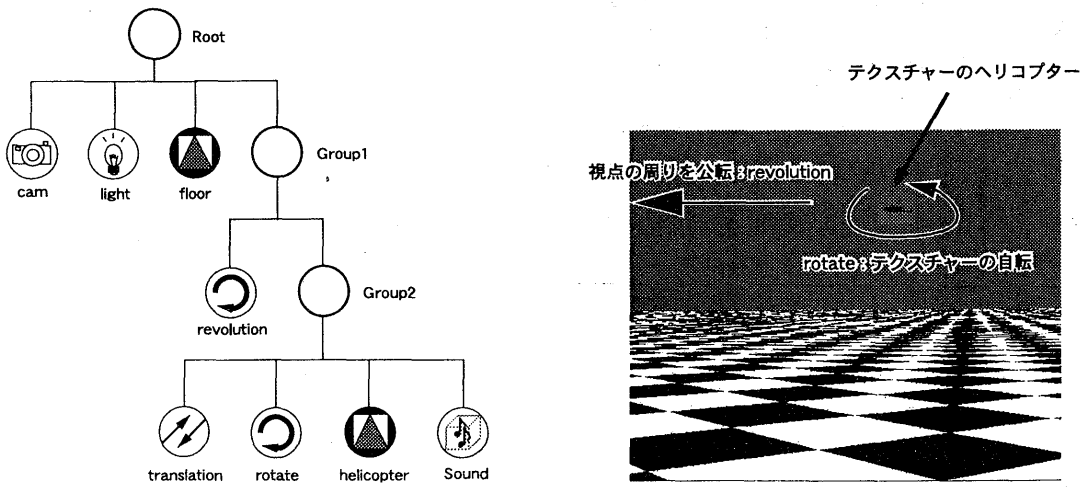

図10　プログラミング例のシーングラフと実行画面