Java を用いた 3 次元画像コンテンツの開発技術

小林誉　齋藤祐一

Prototyping and Evaluation of Java3D API

Homare Kobayashi　Yuichi Saito

目次

1 序　論

1.1 研究の背景

近年，パーソナルコンピュータが急速に普及し，それに伴ってパーソナルコンピュータは高性能化をとげ，かつては一部の人しか作ることのできなかった 3 次元画像も今や，誰もが作ることのできる環境が整ってきた．さらには，インターネットを中心としたコンピュータ・ネットワークの急速な普及がソフトウェアを大きく変え始めた．今後この 3 次元を表す手法は従来からのエンターテイメント・芸術・科学などの分野に加えて，広報活動や商取引にその強力なイメージ効果を利用しようと考える企業や団体も増えることが予想される．そして，その手法の一つとして Java3D が注目され始めている．

1.2　研究のテーマ

宣伝手段の多様化によってインパクトのある 3 次元画像表現は有効な手段と考えられ，また，商取引なとの場においても 3 次元表示での操作と説明が非常に有効だと考える．そこで，Java 言語というシンプルな言語仕様を使用し，マウスによる回転，移動，拡大縮小，オブジェクトの選択など機能を機能を実装した Java 3D は高い生産性を備えていると考える．しかし，Java3D はまだ出て間もないもののため，まだ広く用いられておらずマニュアル等も不足している．

そこで，Java３D の現状を知り，特徴等を調べ，表現方法の確立と有効な表示・作成の検証を行う．

第 2 章では，Java3D での 3 次元画像表示を実現させるための基礎技術として，Java 言語の説明をする．後に，Java3D の歴史・特徴などの説明を行う．

第 3 章では，Java3D を用いての 3 次元画像の表示方法と，それの Java との連携について実際に作成したプログラムを参照しながら説明する．

第 4 章では，顔表示プログラムの評価を示す．速さ，メモリの使用度，ソースプログラムの量，プログラムの作り易さ等について評価をする．

第 5 章では，本論文の結論を示す．結論として今後の課題，改善個所，改善方法を示す．

第 6 章では，本論文で利用した参考文献を示す．

付録に本プログラム又は例として作成したプログラムのソースプログラムコードを示す．

2 Java3D の基礎技術

Java3D とは Java プログラミング言語を元とした標準的な 3 次元グラフィック API(Application Programming Interface)である．正式名称は Java 3D API という．アプリケーションプログラムから，システム(OS)やライブラリ，言語処理系などに対する，標準的な機能呼び出しの方法を定義したものが API である．

本章では，Java3D API とその元となっている Java について，研究で利用した範囲で説明する．

2.1 Java 言語とは

2.1.1 Java 言語とは

Java 言語は，数多く存在するプログラミング言語の中の一つである．
Java 言語は，図 1.1 に示すように，OAK とよばれる家電製品に組み込むことを前提に開発された言語が前身になっており，そのため OAK は，メモリ消費を少なくする，CPU を選ばない，動作を安定させるなどの要求を満たす必要があり，そのため，C++をベースにしながら次のような言語仕様にした．こうした設計思想は Java にも受け継がれている．

1.ポインタの廃止
2.union の廃止
3.オーバーロードの廃止
4.多重継承の禁止
5.メモリ管理の省略

図 1.1 Java 言語の開発の歴史

1991年頃	プログラミング言語Oakを開発したSun Microsystems社内プロジェクトの開始 その後，OakはJavaとなる
1995年	Javaの正式発表
1995年	NetscapeがJavaのライセンスを受ける
1995年	Java Virtual Machine/Java Language仕様出版
1995年	JDK 1.0リリース
1995年	JDK 1.1リリース
1998年	JDK 1.2（Java2）リリース

2.1.2 Java の特徴

Java 言語は機種依存性が非常に少ない言語である．つまり，あるコンピュータ上の Java 言語で開発したプログラムが他の機種のコンピュータでも動作するというこである．C 言

語や C++言語や他のプログラミング言語では，コンピュータの機種が異なる場合，移植という作業が必要となる．しかし Java 言語は，移植を必要としないように設計されている．

Java 言語は誤りを冒しにくい言語である．Java 言語は型の制約が厳しく，このため，プログラムの記述上の誤りは，コンパイルした時点で可能な限り発見されるようになっている．また，使われなくなったメモリは自動的に回収される仕組みが用意されているで，メモリの開放もれなどが起きないようになっている．

また，Java 言語はオブジェクト指向言語である．ハードウェアが数多くの部品で構成されているように，プログラムも数多くの部品で構成されている．Java 言語では，プログラム全体がオブジェクトという部品によって構成されている．オブジェクト指向とは，オブジェクトを中心にプログラム作る設計方法のことである．オブジェクトをプログラミングの中心に置き，プログラムによってオブジェクトを記述すること，これがオブジェクトと指向である．指向というのは中心に考えるということである．

Java 言語はマルチスレッドを取り扱うとができるプログラミング言語である．すなわち，複数のスレッド（動作している主体）を簡単に扱うことができ，並列プログラミングを行うことができる．

Java 言語は Web ブラウザで動作するプログラム(Applet)を作成できるということで世界中に普及した．そのため Java 言語はアプレットを記述するための専用言語という誤解が一部に生じた．しかし，実際には Java 言語は一般のアプリケーションを作ることができる汎用の言語である．

2.1.3 Java 開発環境（JDK）

JDK（Java Development Kit）は，Java 言語を生んだ Sun Microsystems が提供している開発環境のことである．JDK は，Java 言語で使用ことができる標準的なクラスとその内容を定めているため，実質的に Java 言語の公式仕様になっている．Java 言語の開発環境は各社から出されており，各社が作成したクラスライブラリも製品として出荷されているが，それらはいずれも JDK のどのバージョンに適合したものであるかが明示されている．Java 言語でプログラミングを行う開発者は，自分が書いているプログラムが JDK のどのバージョンなのかを意識する必要がある．ただし，JDK のバージョンで大きく変化するのは，あくまでもクラスライブラリの部分なので，プログラミング言語としての Java は，JDK のバージョンで大きく変化することはない．

2.2 Java3D とは

2.2.1 Java3D の概要について

Java3D とは，Java アプレットや，アプリケーションに対してインタラクティブな 3D コンテンツの作成機能を提供する Java API（Application Programming Interface）である．

Java3D は，高レベルの構造概念を使ってジオメトリを作成し，編集し，構造体を作成した後， ジオメトリをレンダリングする．アプリケーション開発者は，これらの構造概念を使用して非常に大きなバーチャルワールドを記述できる．さらに，Java3D ではそこから必要な情報を収集して， 効率的なレンダリングを行うことができる．

Java3D は，Java の write once run anywhere「一度作成すればどこでも実行できる」という利点を 3D グラフィックスアプリケーションの開発者にもたらす． Java3D API で作成したアプリケーション及びアプレットから，すべての Java クラスにアクセスできるため，今後，インターネットでの利用が広がると考えられる．

Java3D API は，既存のグラフィックス API をベースとしており，これに新しいテクノロジーが織り込まれている． 例えば，Java3D の低レベルグラフィックスの構造概念は，低レベルの API に見られる長所を組み合わせたものである． 同様に，高いレベルの構造概念は, シーングラフをベースとしたシステムに見られる長所を組み合わせたものである．

また，Java3D では，3D 立体サウンドなど，一般的にはグラフィックス環境の要素としては考えれられていない概念も取りこんでいる． Java3D API のサウンド機能を使用すれば臨場感溢れる効果でユーザの心をつかむことができる．

2.2.2 Java3D が登場した背景

Java の開発元であるサン･マイクロシステムズは，これまで（Java2D や Java Media Framework）などのマルチメディア API を提供することにより，Java プラットホームに欠落していたものを埋めてきた．そして, Java3D もそうした目的の下に開発された Java API の１つであり， Java による 3D グラフィックス・アプリケーションの開発を支援する．

2.2.3 Java3D が目指すもの

Java3D の開発にあたっては，いくつかの目標が掲げられている．その中でも重要なのは， 高いパフォーマンスの実現である．設計上のいくつかの決断は， ユーザーに最高のレベルのパフォーマンスを提供することを主眼にしている． 特に，相反する 2 つの用件がある場合には，実行速度の高速化につながる方が選ばれている．

このほかに，Java3D では次の目標が掲げられている．

1. 興味深い 3D ワールドを構築できるよう豊富な機能を提供する．但し，冗長な機能や混乱を招く恐れのある機能は含まれない．

2. 高レベルのオブジェクト指向プログラミング環境を提供する．開発者は洗練されたアプリケーションやアプレットを簡単に作成できる．

3. ランタイムローダをサポートする．これによって Java3D では広範なファイルフォーマット （ベンダー独自の CAD フォーマット，中間フォーマット，VRML 1.0 VRML2.0 など）を扱うことができる．

2.2.4 Java3D の長所と短所について

Java3D のもつ長所，短所として次のようなものが挙げられる．

(1)主な長所

1）Java 言語を使用できる

Java3D は，Java 言語であるため Java を使用して 3D 画像を作成することができかつ，Java アプレットとの連携も可能となる．

2）シーン･グラフの最適化による高速なレンダリング

Java3D ランタイムは，レンダリング機能ビットを使用することにより，可能な限りシーン･グラフを最適化し，レンダリングを高速化できる．

3）シーン･グラフの採用

Java3D は，シーン・グラフをベースにした 3D グラフィックス･モデルを部分的に採用している．シーン･グラフによるアプローチの目的は，グラフィックスや，マルチメディアに関するプログラミングの経験が少ないプログラマーでも，容易に 3D グラフィクスを使いこなせるようにすることである． Java3D は熟練した Java プログラマーであれば，容易に 3D グラフィックスを学習できるようになっている．

4）増えつつある外部ローダ

VRML97 用をはじめ，Java3D ランタイムでの 3D コンテンツのロードを可能にする外部ローダの種類が増えつつある．

5）ベクトル演算機能の提供

Java3D は，ベクトル演算専用のクラス･ライブラリを Javax. vecmath パッケージとし

て提供している．同パッケージは，現時点では Standard Extensions API であるが，将来的にはコア API に移行される予定である．

(2)主な短所

1）現時点ではコアAPI ではない

Java3D は現在，Java のコア API ではなく，Standard Extensions API として提供されている．したがって，Java2D などとは異なり，Java のコア API(Java2）とは別にインストールする必要がある

2）JDK1．1．x には未対応

Java3D は JDK1．1．x や 1．0．x に対応していない．そのため，Java3D に対応するアプリケーションの開発．実行には，Java2 のインストールが必要になる．

3）Win32 と Solaris のみに対応

Java3D に対応する OS は，現時点では　Win32 (Windows95/98，Windows NT 4.0) と Solaris だけである．このことはそもそも，Standard Extensions API としてのステータスから生じる制約である. Java の特徴の１つであるクロスプラットフォーム性を考えると，大きな短所といえる．

4）マニュアルの不足

他のものと比較すると Java3D 関連のマニュアルは不足している．

5）不透明なレタリング･パイプライン構造

高レベルの API であるせいか，レンダリング･パイプラインの詳細が外部からはよくわからない

6）"Heavy-weight"な 3D コンポーネント

Java3D のコンポーネントは，ネイティブ・メソッドによってレンダリングされる．このため，Java3D コンポーネントと Swing などの Light-weight コンポーネントを併用すると GUI 部分の開発が複雑になる恐れがある. ある種の特別な回避策があるわけではないが, 一般には両者を同じコンテナ･オブジェクトやウインドウの中でうまく混在させることは難しい．

2.2.5　プログラミング環境

Java3D はオブジェクト指向の API である．アプリケーションでは，ここのグラフィックス要素を独立したオブジェクトとして作成し，それらをつなぎ合わせてツリー上の構造を形成する． この構造をシーングラフという．

2.2.6 パフォーマンスとは

Java3D のプログラミングモデルは，シーングラフの操作や属性状態の変更管理など，定型的な処理を Java3D API に任せ，アプリケーション側の処理を簡略化できる．これによってパフォーマンスが損なわれる心配はない．Java3D ではより高次の抽象化によって，API が実行すべき処理の総量が押さえられる．しかも， API が実行すべき処理の種類が変わる．抽象化のレベルが低い場合には数々の制約を免れないが，Java3D ではこのようなことはなく， 他の API では不可能だった最適化が実現される．

さらに，レンダリングの細かな部分を Java3D に任せることによって， ハードフェアに合わせたレンダリングの調整が可能になる．例えば， 他の API ではレンダリングの順序が固定されているが， Java3D では順序を変更して操作やレンダリングを並行して実行することができる．シーングラフの中で実行時にに変更されては困る部分を指定しておけば，Java3D はツリーをフラットにし， ネイティブハードウェア形式で表現できるようにジオメトリを実行前に変更することもできる．このとき， 元のデータを保持しておく必要はない．

1.レイヤー構造

シーングラフレベルでの最適化に加え，Java3D のパフォーマンスを左右するさらに重要な要因として， 可視ジオメトリのレンダリングに要する時間が挙げられる． Java3D は，システムに依存しない低レベルのネイティブ API を利用して， その上位に実装されている．特に，Java3D を実装する際には， Direct3D，OpenGL，QuickDraw3D を利用できることが求められる．つまり， これらの低レベル API がサポートされているシステム上であれば， Java3D レンダリングは高速化されるということである．

2.対象プラットフォーム

Java3D は，低価格の PC ゲームカードやソフトウェアレンダリングエンジンをはじめ，ミッドレンジのワークステーション，さらにはハイパフォーマンスの専用 3D イメージジェネレータに至るまで， 広範囲な 3D 対応プラットフォーム（ハードウェア/ソフトウェア）を対象としている．

最近のほとんどの PC, 特に 3D グラフィックスアクセラレータカードを搭載した PC 上であれば， Java3D は十分実用に耐える速度でレンダリングを実行できるものと考えられている． また，ミッドレンジ以上のワークステーションでアプリケーションを実行した場合， ハードウェアパフォーマンスをほぼ完全に引き出すものと考えられている．

Java3D は，今後より高速のハードウェアプラットフォームが登場してもそれに対応できる「スケーラビリティ」を備えている． 次世代 3DPC ゲームアクセラレータは，数年前の高価なワークステーションより，ずっと複雑な仮想空間を表現できるようになっているため，Java3D はこのハードウェアパフォーオマンスの向上を考慮した設計となっている．

2.2.7 アプリケーション及びアプレットの作成のサポート

Java3D は，3D 関連のすべての機能を直接サポートしているわけではない．また， 今後もサポートする予定はない．そのため，必要な機能は Java コードで追加するという方法を取る．

Java3D ベースのアプリケーションでは， CAD システム又はアニメーションシステムなどのモデリングソフトで作成したオブジェクトを利用できる．ほとんどのモデリングソフトでは，作成したジオメトリを外部フォーマット（非標準形式の物もある） でファイルにエクスポートすることができる．このファイルフォーマットを読み取り， Java3D でサポートする形式に変換する機能がアプリケーション側に用意されていれば，そのジオメトリ情報を利用できる．

同様に，VRML ローダは VRML ファイルを解析・変換し，そのファイルが表現するコンテンツをサポートするのに必要な， 適切な Java3D オブジェクトと Java コードを生成する．

現在のインターネットブラウザでは，3D データはプラグインや 3D ビューアに渡され，ビューアのウインドウ上でレンダリングされる．現時点でブラウザ上で Java3D を表示させるのは非常に難しいが，今後 3D データをメインウインドウで表示できるブラウザが登場するものと予想される．

3　Java3D を用いての 3 次元画像の表現

Java3D を使って実際に空間を作り，物体を表示させる方法について，順番に説明することにする．
実行環境は下のとおりである

表 3.1 本研究の実行環境

CPU	MMXPentium200MHz
メインメモリ	32MB
OS	Windows98
JavaVM	JDK1.21
Java3D Version	Java3D 1.1 API
表示モード	Retainted Mode を使用

3.1 Java3D の基礎知識

Java3D でグラフィックを表示させるための基礎知識を示す．

3.1.1 モード

Java3D には 3 種類のレンダリングモードが用意されている．それぞれに長所と短所がある．

1）Immediate Mode

1 つの物体の表示など，簡単な機能のみが用意されていて，プログラマはそれを使って複雑な処理を書いていく方法である．描画タイミングの制御などにもっとも融通が利く分，細かいところまで設定する必要があり，プログラマに最も負担のかかる方法である．

2）Retained Mode

シーングラフ(Scene Graph)と呼ばれる木構造を利用する方法である．プログラマは，シーングラフを制御することによって 3 次元処理を行う．Immediate Mode よりも細かい部分の設定も少なく，簡単にきれいなプログラムができる．特別に断わらない限り，本研究開発したプログラムはこのモードを使用している．

3）Compiled-Retained Mode

シーングラフの実行時にコンパイルすることによって高いパフォーマンスを獲得する Retained Mode である．

3.1.2 Java3D の基本構造

java3D の基本的な構造を示す．SimpleUniverse，BranchGroup で構成されるシーング

ラフに物体，背景，視点，などのオブジェクトがぶら下がっている．

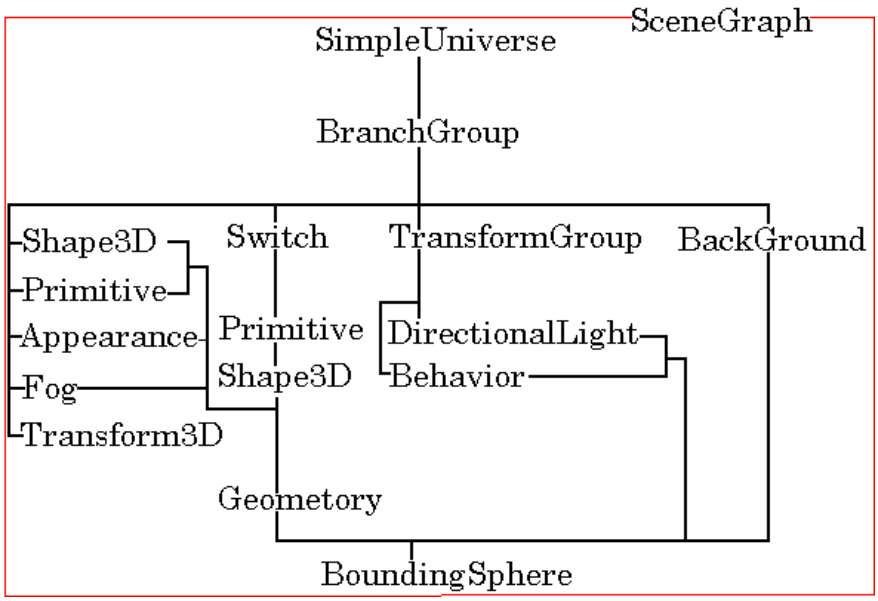
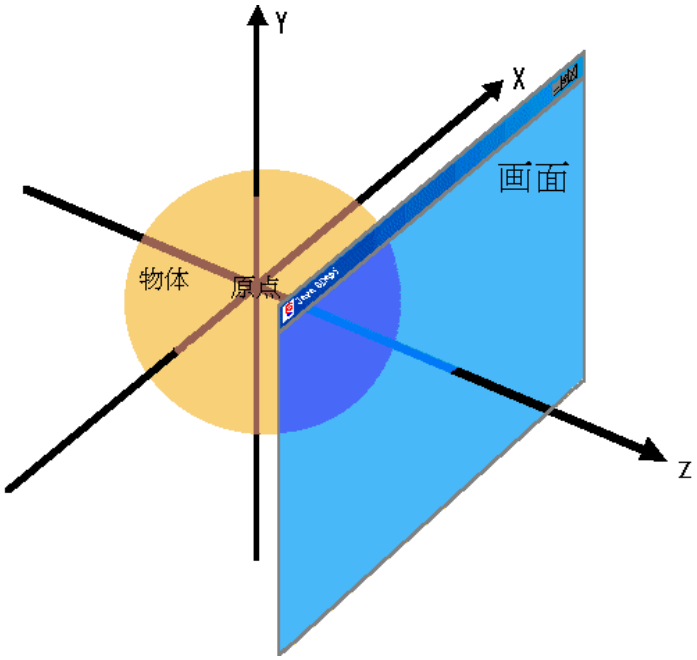


図 3.2Java3D の基本構造

3.1.3 座標系

Java3D で使われる座標系は右手系であり，画面に向かって右方向が x 座標の正，上方向が y 座標の正，画面から手前方向が z 座標の正である.図 3.2

図 3.3Java3D の座標系



3.2 フレームを作る

Java3D をウインドウで表現するためのフレームの設置を以下で示す.

```
public class Window extends Applet {
public Window() {                          //フレームの中身の作成（何もない）
}
public static void main( String[] args ) {
      new MainFrame(new Window(), 600, 480);//メインフレームの作成
}
```

このプログラム Window は”public static void main(String[] args) {“に続く命令ででフレームを定義している.

new MainFrame(new Window(), 600, 480);
MainFrame の引数 new Window(), 600, 480 の部分で実際のフレームの大きさ（ピクセル数）を画面のサイズで表している.
実際にこれを実行すると画面に 600×480 ピクセルのウインドウが表示される. 図 3.3

3.3 何もない空間を作る

フレームを作ったところで Java3D を作ったとはいえない. このフレームの中に, 空間を構築しなくては物体を表示させることはできないのである.

次に空間を構築するためのソースプログラムプログラムを示す.

```
    public Window() {
        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );
        setLayout( new BorderLayout() );
        add(universe.getCanvas(),"Center");
        universe.getViewingPlatform().setNominalViewingTransform();
    }
```

“public Window() 以下の{　　}の中に空間の構築用に必要なソースプログラムプログラムを記述する.
SimpleUniverse は，空間を構築するためのオブジェクトである．ここの情報を示すために，Canvas3D が用意されている．setLayout(new BorderLayout()); add(universe.getCanvas(),"Center");は先ほどのウインドウにこの空間を表示させるための命令となる.
universe.getViewingPlatform().setNominalViewingTransform();は視点を適当な場所に設定するための命令である.
このプログラムを実行すると何もない空間がウインドウ上に現れる，実際には何もないために，黒で表示される.

図 3.3 何もない空間

3.4 空間に物体を表示する

構築した空間に物体を表示するには，BranchGroup オブジェクトを使う.
BranchGroup は，SmpleUniverse オブジェクトの子に当たるオブジェクトであり，その子として TransformGroup オブジェクトや，Shape3D オブジェクトががぶら下がり，TransformGroup オブジェクトの子として，Behavior オブジェクトや，Transform3D オ

ブジェクトがぶら下げられる．(図 3.1 を参照)
これらの木構造がシーングラフと呼ばれ，Java3D の最も基本的な構造となる．
具体的なソースプログラムプログラムを示す．

```
Canvas3D canvas = new Canvas3D(null);
SimpleUniverse universe=new SimpleUniverse(canvas);
add( universe.getCanvas(),"Center");

BranchGroup root=new BranchGroup();
Transform3D translation=new Transform3D();
translation.setTranslation(new Vector3d(1.0,4.0,2.0));
TransformGroup transform=new TransformGroup(translation);
root.addChild(transform);

Shape3D shape = new ColorCube(0.3);
root.addChild(shape);
universe.getViewingPlatform().setNominalViewingTransform();
universe.addBranchGraph(root);
```

まず BranchGroup root=new BranchGroup();で BranceGroup を定義する．
Transform3D translation=new Transform3D();空間における物体の位置を指定するために使われるオブジェクトである．この場合 Transform3D の定義のあとに，3 次元ベクトルを表す Vector3D オブジェクトを介して，原点を中心として（1.0（X 座標）,4.0（Y 座標），2.0（Z 座標））の位置に置く．（translation.setTranslation(new Vector3d(1.0,4.0,2.0));の部分）

TransformGroup transform=new TransformGroup(translation);
Transform3D オブジェクトの内容を実装した，Transform オブジェクトを定義する．
root.addChild(transform);は，BranchGroup オブジェクトに図形の位置を設定するオブジェクトである TransformGroup をぶら下げる．
Shape3D shape = new ColorCube(0.3);は色つきの立方体を定義します
６つの面の色はそれぞれ，赤，緑，青，紫，水色，黄色に振り分けられている．Java3D の中では最も簡単に作ることができる図形である，（0.3）はこの立体の大きさを表している．ここを特に定義しなければ立体の大きさはウィンドウの最大値を示す．
root.addChild(shape);BranchGroup に立体を表すオブジェクト Shape3D オブジェクトをぶら下げる．
universe.addBranchGraph(root);は以上の Brancegroup オブジェクトを SimpleUniverse

に追加することによってウインドウ上に図形が表示される．
このプログラムを実行した場合，真っ黒な空間上に赤い正方形が表示される．位置設定において正面を向くように設定していないため，ただ単に四角形が表示させているかのように，見えるが，これは Java3D によって作られた立方体である．

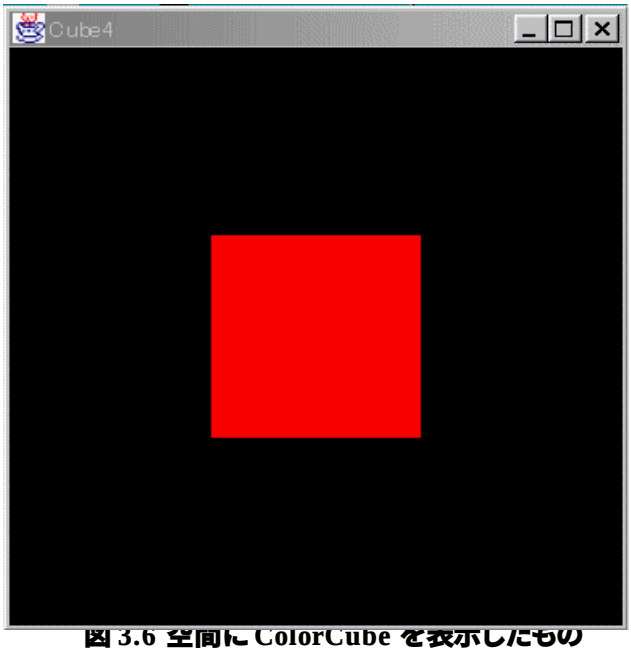


図 3.6 空間に ColorCube を表示したもの

3.5 その他の物体の表示方法

Java3D では，Cube 以外にも様々な立体を作ることができる．

3.5.1 頂点を指定して作られる物体

Shape3D オブジェクトを作るには コンストラクタを使う．その引数に，Geometry オブジェクトを指定するかもしくは引数なしで生成した後に setGeometry(Geometrygeometry)メソッドで, Geometry オブジェクトを指定する必要がある． その指定する Geometry オブジェクトが物体の形状を表わす．Geometry オブジェクトは Geometry クラスのサブクラスでインスタンス化する．3 角形の面で構成される多面体を作りたい時は TriangleArray クラスを，4 角形の面で構成される多面体を作りたい時は QuadArray クラスを使う．
具体的なソースプログラムプログラムは以下のようになる．

```
float[] vertexs={
        0.3f, 0.0f, 0.0f, 0.0f, 0.0f, 0.3f,    0.0f,-0.6f, 0.0f,          // 面 1
        0.0f, 0.0f, 0.3f,-0.3f, 0.0f, 0.0f,    0.0f,-0.6f, 0.0f,          // 面 2
        -0.3f, 0.0f, 0.0f, 0.0f, 0.0f,-0.3f, 0.0f,-0.6f, 0.0f,
```

```
                0.0f, 0.0f,-0.3f, 0.3f, 0.0f, 0.0f, 0.0f,-0.6f, 0.0f,
                0.0f, 0.0f, 0.3f, 0.3f, 0.0f, 0.0f, 0.0f, 0.6f, 0.0f,
                -0.3f, 0.0f, 0.0f, 0.0f, 0.0f, 0.3f, 0.0f, 0.6f, 0.0f,
                0.0f, 0.0f,-0.3f,-0.3f, 0.0f, 0.0f, 0.0f, 0.6f, 0.0f,
                0.3f, 0.0f, 0.0f, 0.0f, 0.0f,-0.3f, 0.0f, 0.6f, 0.0f};// 面 8

        float[] colors={
                0.5f,0.2f,1.0f,0.2f,0.5f,1.0f,0.2f,0.5f,1.0f,
                0.2f,0.5f,1.0f,0.5f,0.2f,1.0f,0.5f,0.2f,1.0f,
                0.5f,0.2f,1.0f,0.2f,0.5f,1.0f,0.2f,0.5f,1.0f,
                0.2f,0.5f,1.0f,0.5f,0.2f,1.0f,0.5f,0.2f,1.0f,
                0.2f,0.5f,1.0f,0.5f,0.2f,1.0f,0.5f,0.2f,1.0f,
                0.5f,0.2f,1.0f,0.2f,0.5f,1.0f,0.2f,0.5f,1.0f
                0.2f,0.5f,1.0f,0.5f,0.2f,1.0f,0.5f,0.2f,1.0f,
                0.5f,0.2f,1.0f,0.2f,0.5f,1.0f,0.2f,0.5f,1.0f};

        TriangleArray geometry=new TriangleArray(
                24,TriangleArray.COORDINATES | TriangleArray.COLOR_3);
                                        //3 (面を構成する頂点の数) ×8 (面の数)
        geometry.setCoordinates(0,vertexs);//頂点座標の設定
        geometry.setColors(0,colors);//頂点の色の設定
        Shape3D shape=new Shape3D(geometry);
        //ダイヤの結晶を 1 方向にのばしたような形をした物体
```

3.4 節で示した ColorCube の代わりに各面の座標を設定した表示の例を上に示す．
float[] colors={ }内で示されている処理は，面を構成する頂点の座標を示している．この場合は X, Y, Z 座標で示される点を 3 つ示している．その 3 点で表示される平面をそれぞれ 8 個定義している．float[] colors={ }以下で示しているのは，その平面の色設定についてである．色は赤，緑，青の三原色で構成されておりその組み合わせで色を決定している．色のパターンについては後述する．
TriangleArray geometry=new TriangleArray
(24,TriangleArray.COORDINATES | TriangleArray.COLOR_3);
三角形の組み合わせなので TriangleArray を定義する，24 は面を構成する頂点の数×面の数を示す．geometry.setCoordinates(0,vertexs); //頂点座標の設定
geometry.setColors(0,colors); //頂点の色の設定をそれぞれ行う．
これを実行すると，ダイヤの結晶を一方向に延ばしたような図形が表示される．

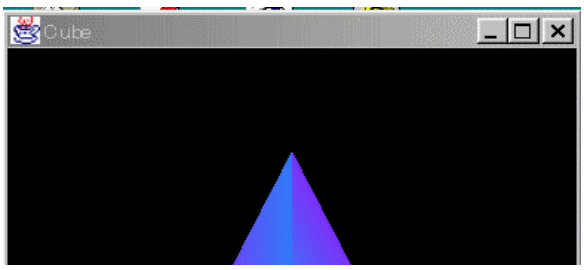
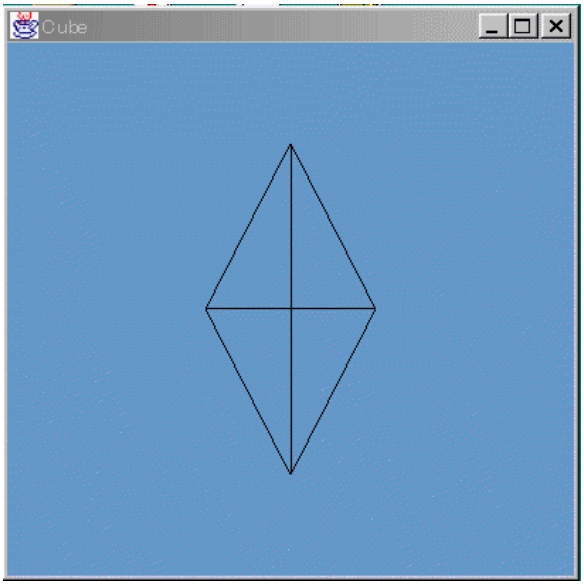


図 3.7 平面を組み合わせた立体



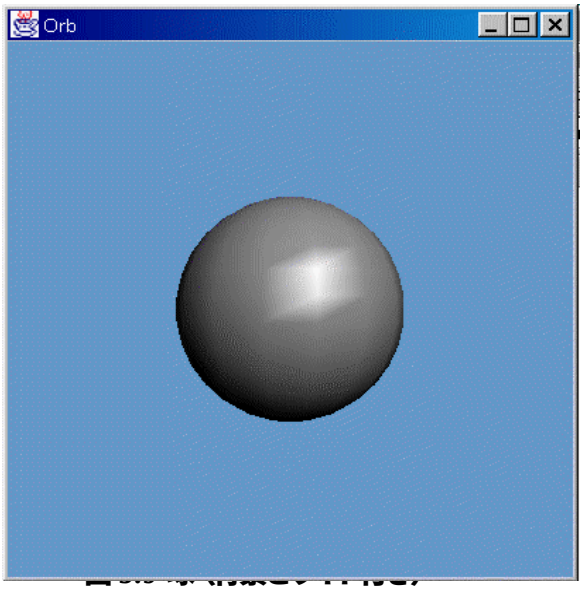
3.5.2 Primitive による球の表示方法

球や円形を持つ立体を表示するときには，3.4 節や 3.5 節で示した shape オブジェクトの代わりに Primitive が使われる. Primitive はユーティリティクラスの一つで基本的な形状の物体を表現する Box (直方体し)，Cone (円錐)，Cylinder (円柱)，Sphere (球)4 種類が用意されている.
実際にソースプログラムの例を示す.

```
Primitive primitive=new Sphere(0.3);//球の
root.addChild(primitive);
```

となる Primitive primitive=new Sphere(0.3);で球を定義し BranchGroup root=new BranchGroup();で BranchGroup オブジェクトにぶら下げている.

物体は複数個設置，（BranchGroup オブジェクトにぶら下げられること）できる．



3.5.3 物体から部品を取り出す

Primitive は実際 Group で，コンストラクタで部品(Shape3D ノード)を作り，その子に追加している．Primitive オブジェクトに対して getShape(int partid)メソッドを使えば，その部品を取り出せる．partid には各 Primitive クラスに定義されている定数を指定する．

```
Primitive primitive=new Cylinder();      //円柱
Shape3D shape=primitive.getShape(Cylinder.BODY);// 側面を取り出す
root.addChild(shape.cloneNode(true));
```

この取り出された部品ノードは Primitive ノードの下にすでにぶら下げられているので，そのままでは SceneGraph に追加できない．cloneNode(boolean forceDuplicate) メソッドで複製して SceneGraph にぶら下げる．図 3.9，図 3.10

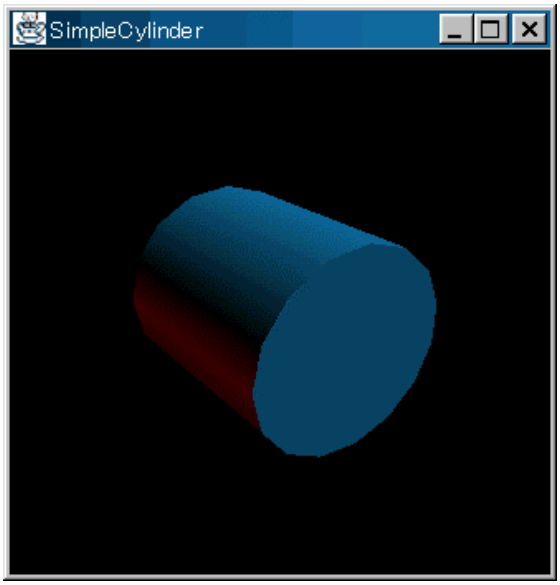
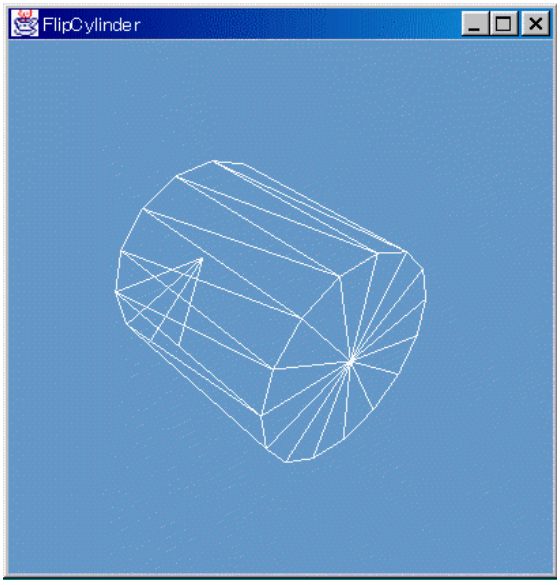


図 3.10 円柱



3.6 物体の外見

Java3D では様々な物体を作ることができる．また，それらの物体に様々な効果を持たせて表現することができる．本節ではそれらの方法について紹介する

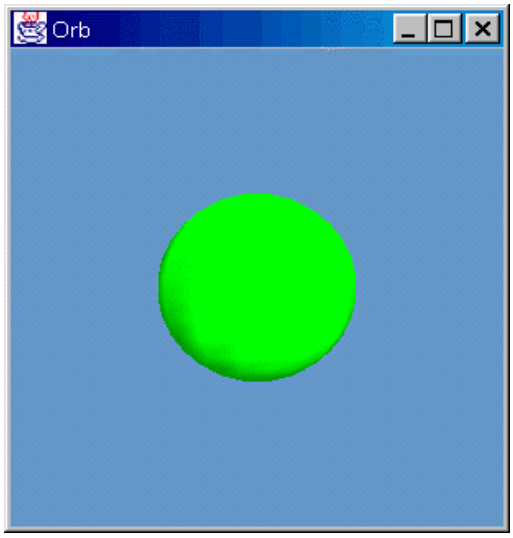
3.6.1 物体に色を付けるには

3.5.2 で説明した方法で，実際に球を作ってプログラムを実行すると問題が発生する．まず，空間そのものが黒いため，物体そのものに色が付けられていないと，黒い画面となって何も表示されていないように見える．（実際は表示されている）．したがって，表示を確認するためにも，物体に色を付けておく必要がある．Appearance オブジェクトが使用される．

具体的な色の設定を以下のソースプログラムで示す．

```
 Appearance ap = new Appearance();
sphere.setAppearance( ap );
Material mt = new Material(
                        new Color3f( 0.0f, 0.4f, 0.0f ),
                        new Color3f( 0.0f, 0.6f, 0.0f ),
                        new Color3f( 0.0f, 0.8f, 0.0f ),
                        new Color3f( 0.0f, 1.0f, 0.0f ),
                        2.0f );
ap.setMaterial( mt );
```

Material mt = new Material()の部分で色を物体の１つのマテリアルとして定義して物体の Appearance オブジェクトにセットしている．それを本体のオブジェクトにセットする．色について説明する．Color3f(0.0f, 0.4f, 0.0f)の部分において()内の数値が色を決定していると説明した．( )内の 3 個の数値はそれぞれ（赤，緑，青）3 原色組み合わせで構成されている．最高値はそれぞれ 1.0f であり．三つの数値が 0.0f であれば，黒，1.0f であれば白になる．全て 0.5f であると灰色になる．その他(1.0f,1.0f,0.0f)で黄色，(1.0f,0.0f,1.0f)で紫（ピンク）(0.0f,0.1f,0.1f)で水色を示す．



3.6.2 半透明な物体の作り方

物体を半透明にするためには Appearance オブジェクトに対して setTransparencyAttributes(TransparencyAttributes transparencyAttributes) メソッドで TransparencyAttributes オブジェクトを指定する．TransparencyAttributes オブジェクトは透明度を指定するオブジェクトである． setTransparency(float transparency) メソッドで透明度を指定する． transparency は 0.0～1.0 の範囲で 1.0 が完全な透明を示す． ソースプログラムで説明する

```
TransparencyAttributes attr=new TransparencyAttributes();
attr.setTransparency(0.5f);          // 半透明

Appearance appearance=new Appearance();
appearance.setTransparencyAttributes(attr);
Primitive primitive=new Sphere(0.5f,appearance);
```

となる．TransparencyAttributes attr=new TransparencyAttributes();

attr.setTransparency(0.5f);で半透明のオブジェクトを定義し，その透明度を次の行で設定している．Primitive primitive=new Sphere(0.5f,appearance);でその情報を球を定義する際に組み込んでいる．

図 3.12 半透明 1　　図 3.13 半透明 2

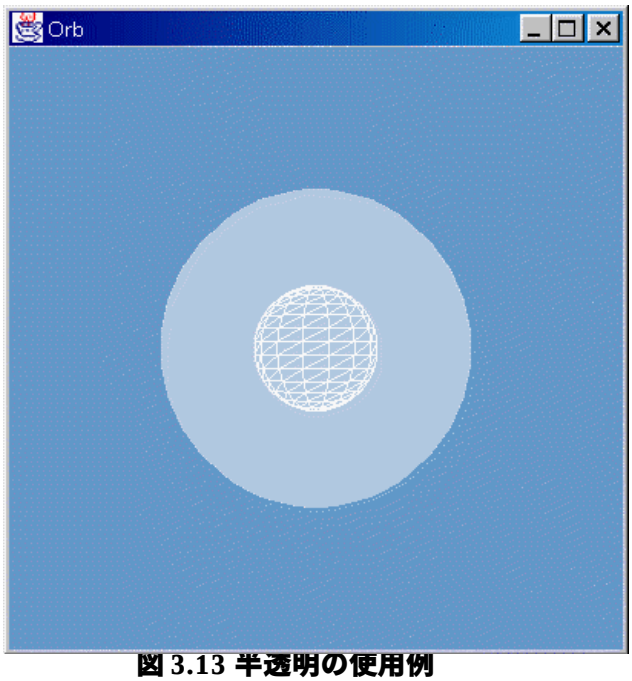
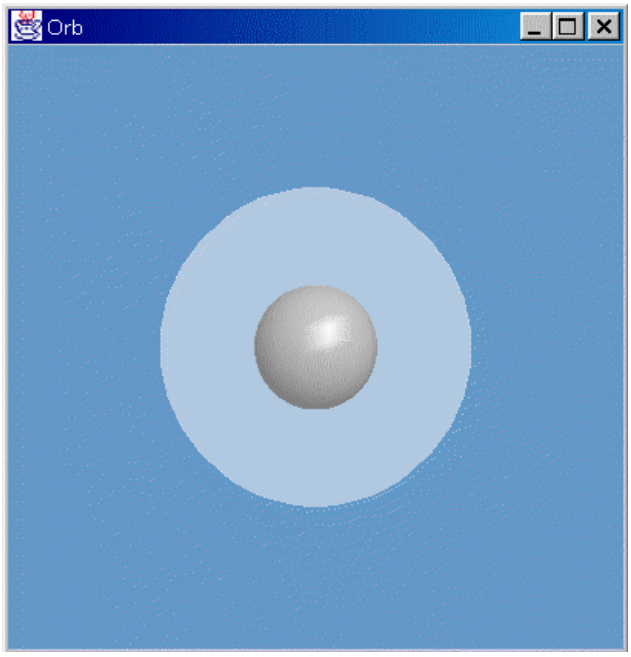


図 3.13 半透明の使用例



3.5.3 ワイヤーフレームで表示する

ワイヤーフレームとは，（様々な平面ポリゴンでできている）Java3D の立体を構成する物体を平面の継ぎ目のみで立体を構成するものである．

物体をワイヤーフレームで表示するためには，Appearance オブジェクトに対して setPolygonAttributes(PolygonAttributes polygonAttributes)メソッドで PolygonAttributes オブジェクトを指定する．
PolygonAttributes オブジェクトは レンダリング方式などを指定するオブジェクトである． setPolygonMode(int polygonMode) メソッドの引数に POLYGON_POINT, POLYGON_LINE,POLYGON_FILL のいずれかの定数を指定して物体の描き方を指定する．実際に使用する．

```
PolygonAttributes attr=new PolygonAttributes();
attr.setPolygonMode(PolygonAttributes.POLYGON_LINE);
Appearance appearance=new Appearance();
appearance.setPolygonAttributes(attr);
Primitive primitive=new Sphere(0.5f,appearance);
```

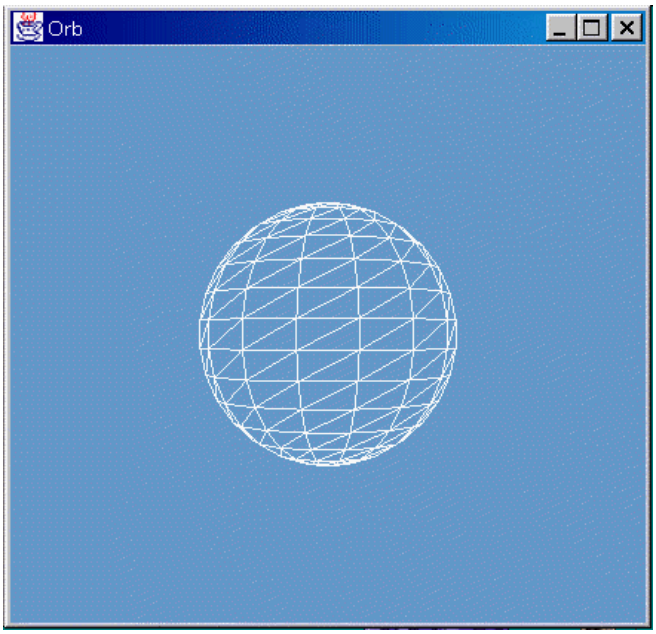
考え方は基本的に半透明の時と同じである．

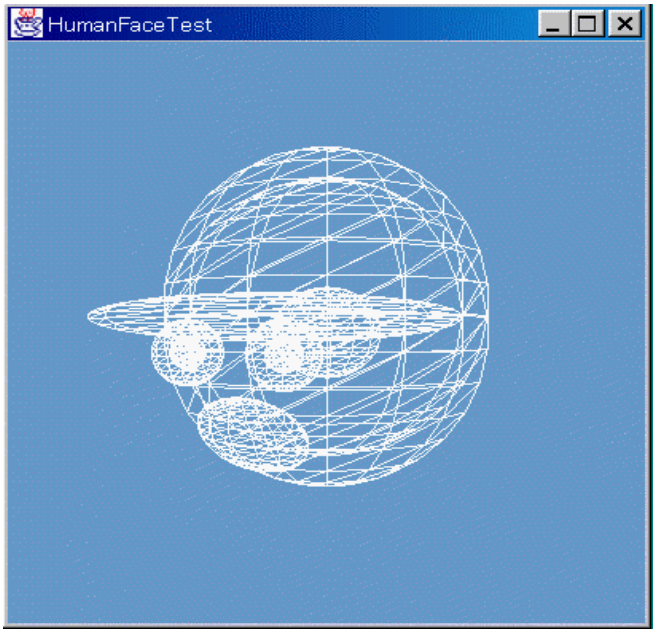
```
PolygonAttributes attr=new PolygonAttributes();
    attr.setPolygonMode(PolygonAttributes.POLYGON_LINE);
```

でワイヤーフレームの定義と組み込みを行い，

```
Appearance appearance=new Appearance();
    appearance.setPolygonAttributes(attr);
    Primitive primitive=new Sphere(0.5f,appearance);
```

の部分でワイヤーフレームの情報 Appearance オブジェクトを介して球に組み込む．





3.6.4 物体にテクスチャを貼る

物体にテクスチャを貼るには Texture オブジェクトを使用する．Appearance オブジェクトに対して setTexture(Texture texture)メソッドで Texture オブジェクトを指定する．ファイル名で指定した画像ファイルを元に Texture オブジェクトを作る．
また，そのテクスチャーを貼り付ける Shape3D オブジェクトにはテクスチャ座標 を設定する必要がある． Primitive の場合はコンストラクタの primflags に GENERATE_TEXTURE_COORDS を指定する事で自動的に設定する．

```
Texture   texture=new TextureLoader
          ("image/texture.gif",component).getTexture();
Appearance appearance=new Appearance();
          appearance.setTexture(texture);
Primitive primitive=new Sphere1.0f,
           Sphere.GENERATE_TEXTURE_COORDS,appearance);
```

Texture texture=new TextureLoader
("image/texture.gif",component).getTexture();の部分でテクスチャオブジェクトを定義し，同時に表示させる画像ファイルを呼び出す．Appearance オブジェクトを介して Primitive オブジェクトにその情報を組み込んでいる．テクスチャを使うことによって物体に，自由な画像を張り付けることができる．

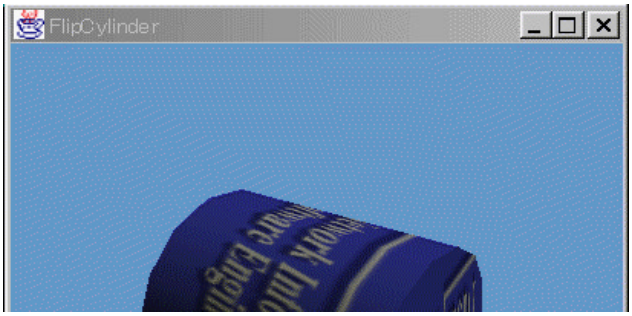
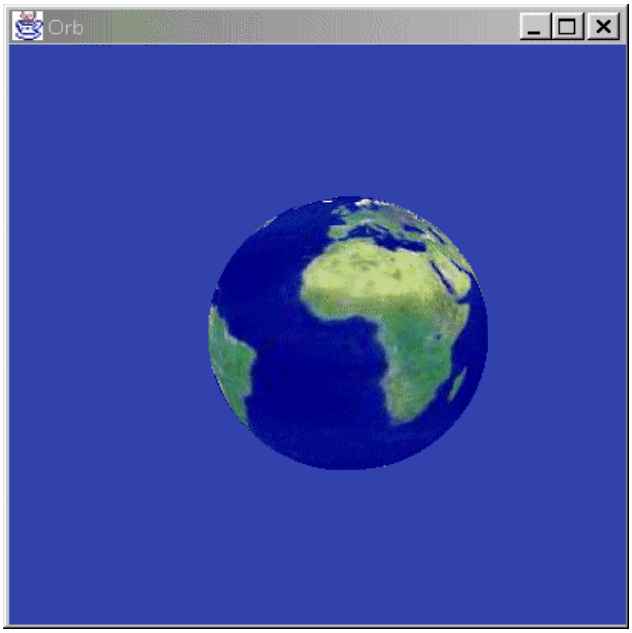


図 3.17 テクスチャを貼った円柱



3.7 物体以外のグラフィックの表現

物体の作り方について紹介してきたが, 次にその物体を彩る設定方法について紹介する.

3.7.1 背景の設定

Java3D における背景とは, 無限方向にあるものを示すものである. つまり物体のない空間の設定のことである. 背景を設定するには Background オブジェクトを設定する. SceneGraph に Background オブジェクトをぶら下げる.

Background オブジェクトに色を設定した場合, 背景はその色で埋め尽くされる. コンストラクタ Background(Color3f color) かメソッド setColor(Color3f color) または

setColor(float 赤, float 緑, float 青)で色を設定する．
背景の設定を含むソースプログラムを以下に示す．

```
BranchGroup root=new BranchGroup();
Background background = new Background
            (new Color3f(0.4f,0.6f,0.8f));

BoundingSphere bounds=new BoundingSphere(
             new Point3d(0.0,0.0,0.0),100.0);
background.setApplicationBounds(bounds);
root.addChild(background);

Primitive primitive=new Sphere(0.4f);
root.addChild(primitive);
```

まず，Background background = new Background
(new Color3f(0.4f,0.6f,0.8f));の部分で背景の定義と色の設定を行う．
BoundingSphere bounds=new BoundingSphere
(new Point3d(0.0,0.0,0.0),100.0);
background.setApplicationBounds(bounds);この部分は背景の適用範囲を設定し（この場合は(0.0,0.0,0.0)なので原点を中心として半径 100 の範囲），それを，背景の定義に組み込む．
root.addChild(background);において，背景の設定を BranchGroup オブジェクトにぶら下げる．
その空間内に，物体を，定義する．このプログラムの場合，空色の背景に色の定義されていない黒い球が表示される．

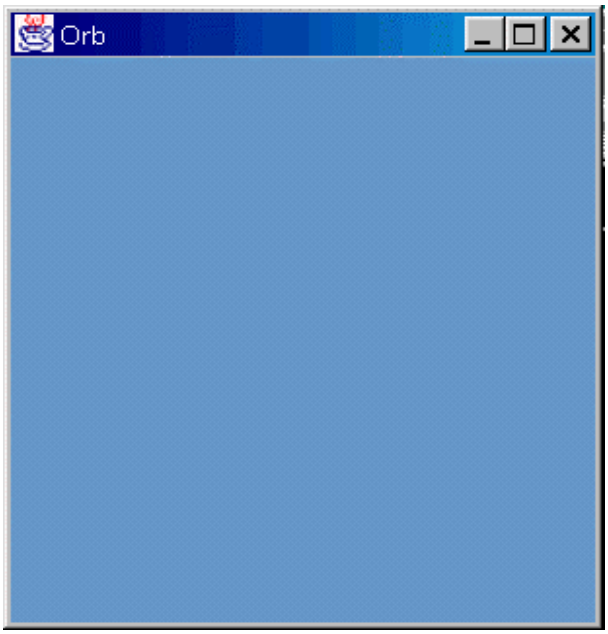
オブジェクトに ImageComponent2D オブジェクトを設定した場合，背景に画像が表示される． コンストラクタ Background(ImageComponent2D image) か メソッド setImage(ImageComponent2D image) を使う．ImageComponent2D オブジェクトはこのような場合に使う画像を参照するオブジェクトである．

```
ImageComponent2D image=new ImageComponent2D(
                    ImageComponent2D.FORMAT_RGB,someImage);
Background background=new Background(image);
```

また背景にテクスチャを張り付けることも可能である．方法としては「5.4 物体にテクス

チャを貼る」で説明したやり方を応用してテクスチャーを定義し，背景に組み込んでいる

```
TextureLoader bgTexture = new TextureLoader(
        "../images/sora.gif", this);
Background background = new Background(
        backgroundTexture.getImage());
background.setApplicationBounds(bounds);
```



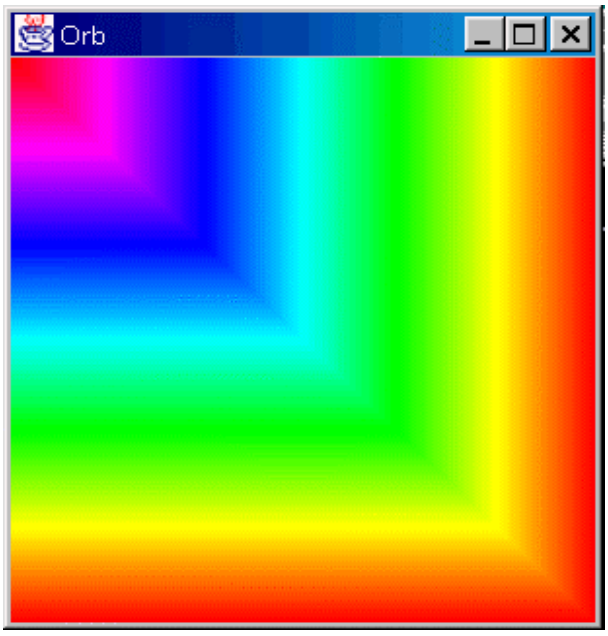


図 3.20 テクスチャを貼った背景

3.7.2 霧の効果

Java3D では，背景や物体をぼやかす霧の効果が用意されている．
霧，効果を設定するには Fog オブジェクトを設定する．SceneGraph に Fog オブジェクトをぶら下げる．
Fog オブジェクトは 実際には Fog クラスのサブクラスをインスタンス化する． Fog クラスのサブクラスには霧のかかり方の違う ExponentialFog, LinearFog の 2 種類が用意されている．背景同様に Fog オブジェクトは適用範囲を設定する必要がある．範囲は Bounds オブジェクトが表現するので, それを setInfluencingBounds(Bounds region)メソッドで設定する．

```
BoundingSphere bounds=new BoundingSphere(
        new Point3d(0.0,0.0,0.0),100.0);
Fog fog=new LinearFog(new Color3f(1.0f,1.0f,1.0f),1.7f,3.0f);
fog.setInfluencingBounds(bounds);
root.addChild(fog);
```

解説するとこれは,
BoundingSphere bounds=new BoundingSphere(
new Point3d(0.0,0.0,0.0),100.0);で適応範囲を指定する．
Fog fog=new LinearFog(new Color3f(1.0f,1.0f,1.0f),1.7f,3.0f);において霧の定義を行う．
この場合，白い霧をに出現させる．後方の数値 1.7f,3.0f は霧の薄さを表している．
それを root.addChild(fog);で BranchGroup オブジェクトにぶら下げる．

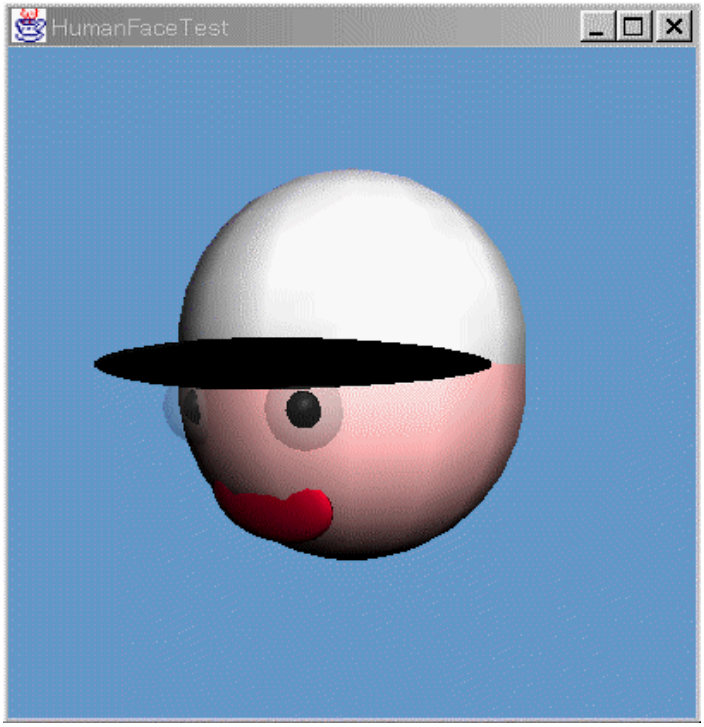
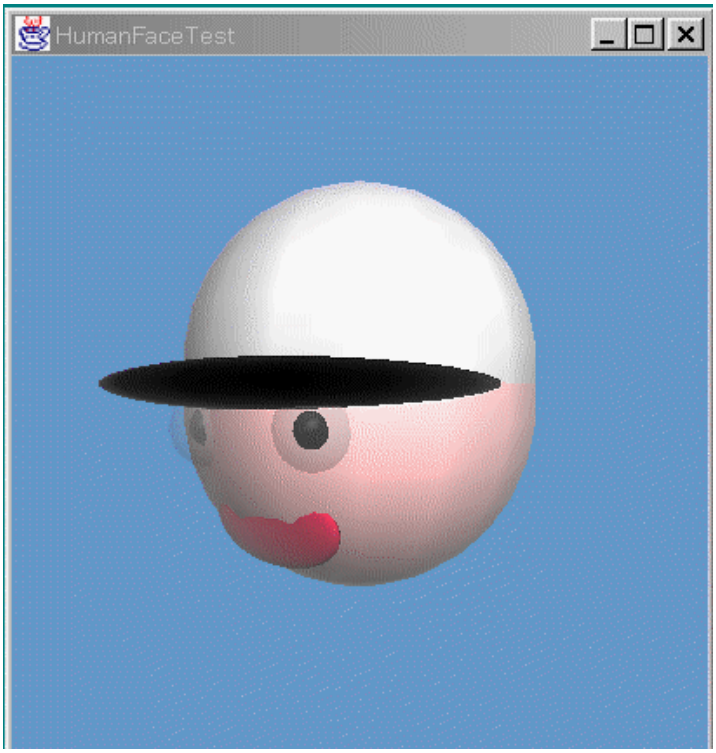


図 3.21 霧がかかっていない物体



3.7.3 物体に光を当てる

これまで説明した方法で物体を定義した場合，色の設定をしなければ物体には初期設定されたときの色となる．そのため色が平坦で立体を出現させているという実感に乏しいものである．物体が立体であることを実感できる表現にするには，物体に光を当て，その質感を表現させたいものである．

このため Java3d には光源機能が用意されている．

光源は Light クラスのサブクラスによって表現されている．SceneGraph (BranchGroup オブジェクトまたは TransformGroup オブジェクトの下) にぶら下げる

```
BranchGroup root=new BranchGroup();
BoundingSphere bounds=new BoundingSphere(
                    new Point3d(0.0,0.0,0.0),100.0);
```

```
Light light=new PointLight(
        new Color3f(1.0f,0.0f,0.0f),
        new Point3f(2.0f,1.0f,1.0f),
        new Point3f(1.0f,0.0f,0.0f));      // 点光源
light.setInfluencingBounds(bounds);
root.addChild(light);
```

この場合は赤い光を座標軸（1.0f,0.0f,0.0f）と(1.0f,0.0f,0.0f)の二点からから当てるというものである．
このように，光自体に色を設定することが可能である．特に黒い光は影となる．

light.setInfluencingBounds(bounds);で光源の範囲を指定し，
root.addChild(light);で BranchGroup オブジェクトにぶら下げている．
物体の表面に光源から光を当て質感を表現するには，Material オブジェクトを作り，setLightingEnable(boolean state) メソッドで光があたるようにし，物体の Appearence オブジェクトにその Material オブジェクトを組み込む．

```
Material material=new Material();
material.setLightingEnable(true);

Appearance appearance=new Appearance();
appearance.setMaterial(material);
Primitive primitive=new Sphere(1.0f,appearance); // 球
root.addChild(primitive);
```

3.8 視点の設定について

視点を移動させるには ViewPlatform オブジェクトを作り SceneGraph に追加する．その ViewPlatform オブジェクトはコンストラクタで直接作るのではなく View オブジェクトを作り getViewPlatform()メソッドで取得する．
ViewPlatform オブジェクトは TransformGroup オブジェクトを間に挟んで SceneGraph にぶら下げる．TransformGroup オブジェクトに組み込まれている，Transform3D オブジェクトで視点の位置を決定する．
SceneGraph のルートオブジェクトとして SimpleUniverse を使った場合の視点移動を説明する．
SimpleUniverse オブジェクトが作られると自動的に ViewingPlatform オブジェクトが作られて SceneGraph にぶら下げられる．ViewingPlatform オブジェクトは

TransformGroup,ViewPlatform オブジェクトをまとめたものである．
SimpleUniverse オブジェクトの getViewingPlatform()メソッドで ViewingPlatform オブジェクトを取得し，ViewingPlatform オブジェクトの getViewPlatformTransform() メソッドで TransformGroup を取得する．

```
Canvas3D canvas=new Canvas3D(null);
SimpleUniverse universe=new SimpleUniverse(canvas);

TransformGroup transform;
Transform3D translation=new Transform3D();

translation.setTranslation(newVector3d(0.0,1.0,0.0));
transform=universe.getViewingPlatform(.getViewPlatformTransform();
transform.setTransform(translation);
```

translation.setTranslation(newVector3d(0.0,1.0,0.0));の部分で視点を(0.0,1.0,0.0)に指定し，transform=universe.getViewingPlatform( ).getViewPlatformTransform();で SimpleUniverse でその内容を ViewPlatform で取得し，その情報を transform.setTransform(translation);で Transform に組み込んでいる．

3.9 アニメーションについて

3.8 節までは空間と物体を表示させるものであり．実際にウィンドウの中の世界は見るものにとって平面的な 1 枚絵であった．本節ではこの 1 枚絵が立体であることを実感するために，簡単な動きをつけることにする．
Java3D でアニメーションの方法として Interpolator クラスが用意されている．Interpolator クラスは必ず Alpha オブジェクトを指定する．Alpha オブジェクトは時間情報の保持を行うものであり，0.0 から 1.0 の範囲のαの値を持ち時間とともにそれを変化させる．
回転運動をする Interpolator のサブクラスである，RotationInterpolator についてソースプログラムの例で説明する．RotationInterpolator オブジェクトはα値に応じて TransformGroup の持つ Transform3D オブジェクトに Y 軸を中心とした回転を設定するものである．
αが 0.0 時の回転角は 0，1.0 の時は回転角 2πである．

```
TransformGroup rotation = new TransformGroup();
Root.addChild(rotation);
```

```
        rotation.addChild(new ColorCube(0.3));

        Alpha alpha = new Alpha();
        Interpolator interpolator =
                new RotationInterpolator(alpha,rotation);
        rotation.addChild( interpolator );
        BoundingSphere bounds = new BoundingSphere(
                New Point3d(0.0,0.0,0.0),100.0);

        interpolator.setSchedulingBounds( bounds );
```

このプログラムを実行すると，ColorCube が Y 軸を中心に反時計回りで回転する．

```
TransformGroup rotation = new TransformGroup();
Root.addChild(rotation);
rotation.addChild(new ColorCube(0.3));
```

において TransformGroup に図形 ColorCube をぶら下げる．

```
Interpolator interpolator =
                new RotationInterpolator(alpha,rotation);
rotation.addChild( interpolator );
```

で Alpha の情報を取得した RotationInterpolator を TransformGroup にぶら下げている．

```
BoundingSphere bounds = new BoundingSphere(
        New Point3d(0.0,0.0,0.0),100.0);
interpolator.setSchedulingBounds( bounds );
```

RotationInterpolator の適用範囲を指定している．

3.10 マウスによる物体の操作

これまでは空間を作り，その中に物体を作って物体や空間に対して行うことのできる様々な処理を紹介してきた，本節ではマウスの操作による物体の位置変更について紹介する．

マウスでの操作に応じて物体などの位置を変更するために，MouseBehavior クラスが用意されている．

MouseBehavior クラスのサブクラスのオブジェクトを生成し，SceneGraph にぶら下げて使用する． どの MouseBehavior もコンストラクタまたはメソッド setTransformGroup(TransformGroup transformGroup)で TransformGroup オブジェクトを指定する．マウスの操作に応じてこの TransformGroup が書きかえられる．

```
        BoundingSphere bounds = new BoundingSphere();
```

```
bounds.setRadius( 10.0 );

TransformGroup objtrans = new TransformGroup();
objTrans.setCapability( TransformGroup.
                        ALLOW_TRANSFORM_WRITE );
objTrans.setCapability( TransformGroup.
                        ALLOW_TRANSFORM_READ );
root.addChild( objTrans );
BoundingSphere bounds = new BoundingSphere();
Transform.addChild(new ColorCube(0.3));//色つき立方体
```

まず操作したい図形の設定や範囲を指定しておく．
範囲は Bounds オブジェクトが表現する．それを setSchedulingBounds(Bounds region) メソッドで設定する． この範囲と視点 (ViewPlatform) が重なった時のみ操作が可能となる．
さらに TransformGroup の設定を書き換えるため TransformGroup には Capability bit（後述する）として，ALLOW_TRANSFORM_READ, ALLOW_TRANSFORM_WRITE を設定する必要がある．これがないと実行時に禁止されたオブジェクトのぶら下げによる例外が発生する．

3.10.1 回転移動

回転運動を行うには，MouseBehavior クラスのサブクラスの MouseRotate オブジェクトを使用する．

```
MouseRotate behavior1 = new MouseRotate( objTrans );
objTrans.addChild( behavior1 );
behavior1.setSchedulingBounds( bounds );
```

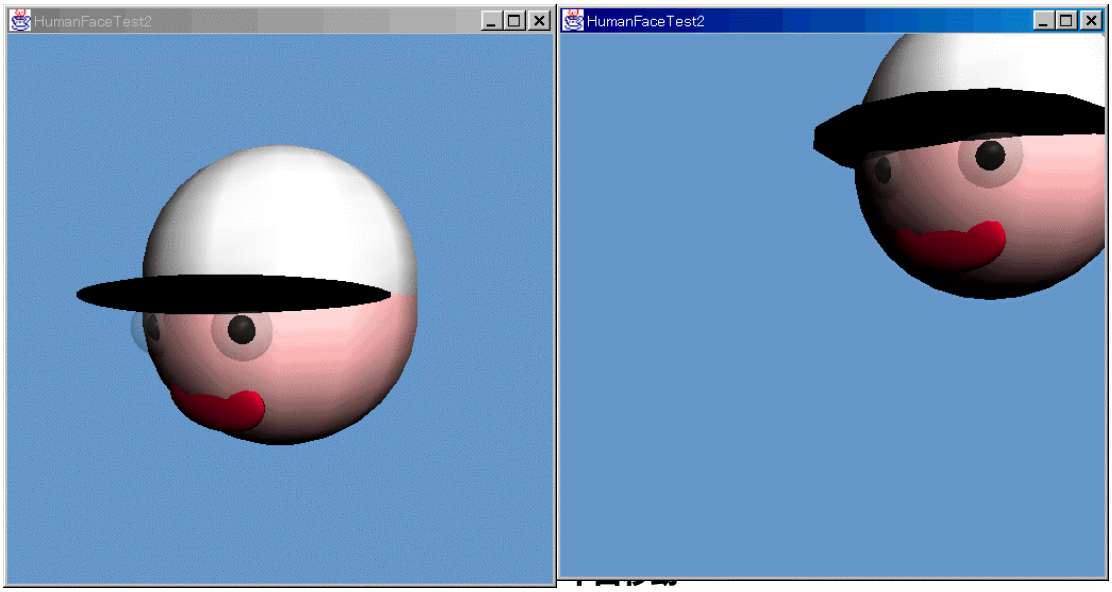
objTrans.addChild(behavior1);で TransformGroup に MouseRotate をぶら下げ，behavior1.setSchedulingBounds(bounds);でその範囲を設定する
このソースプログラムを実行すると BoundingSphere オブジェクトで指定された範囲で物体が自由に回転することができる．

3.10.2 平面移動

平面移動を行うには，MouseBehavior クラスのサブクラスの MouseTranslate オブジェクトを使用する．

```
MouseTranslate behavior2 = new MouseTranslate( objTrans );
objTrans.addChild( behavior2 );
behavior2.setSchedulingBounds( bounds );
```

objTrans.addChild(behavior2);で TransformGroup に MouseTransrate をぶら下げ，
behavior2.setSchedulingBounds(bounds);でその範囲を設定している
このソースプログラムを実行すると BoundingSphere オブジェクトで指定された範囲で物体が X,Y 平面上を自由に移動することができる．



3.10.3 拡大縮小

拡大縮小を行うには，MouseBehavior クラスのサブクラスの MouseZoom オブジェクトを使用する．

```
MouseZoom behavior3 = new MouseZoom( objTrans );
objTrans.addChild( behavior3 );
behavior3.setSchedulingBounds( bounds );
```

objTrans.addChild(behavior3);で TransformGroup に MouseZoom をぶら下げ，
behavior3.setSchedulingBounds(bounds);でその範囲を設定する
このソースプログラムを実行すると BoundingSphere オブジェクトで指定された範囲で物体が．
Z 軸方向すなわち，画面の手前から奥への移動を行い拡大縮小する効果が得られる．

拡大すると物体は視点であるところを突き抜けその断面を表示することになる．物体の中は空であり，背景と同じ状態になっている，つまり物体の中身から図形をはかり知ることはできない．
なお，実際には 3.10.1 から 3.10.3 の方法を同時に定義することができる．しかし 2 個しかボタンのないマウスでは 1 度に 2 つの機能までしか実行できない．マウスボタンの実行の割り当ての順番は定義順になっている．

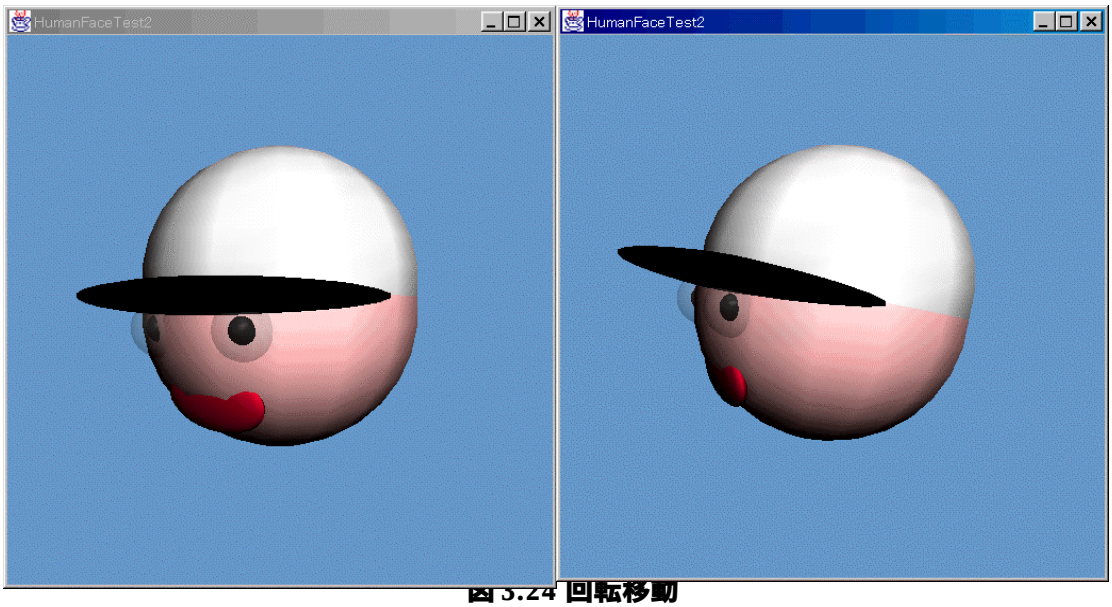


図 3.24 回転移動

3.11 Capabiility bit について

SceneGraph にオブジェクトをぶら下げた後に，そのオブジェクトの属性を変更しようとすると CapabilityNotSetException が発生する．これは Capability bit を設定していないために起こるものである．
例えば 3.10 節で紹介した，マウス操作によって物体を動かすことがそれに当たる．この場合，視点が変更されるため実行時に例外が発生する．
このような場合のために Capability bit と呼ばれる SceneGraph の Node 属性変更を許可するためのフラグを用いる．属性変更は，通常どの属性の変更も許されていないので，Capability bit をセットする必要がある．
SceneGraph を構成するクラス(SceneGraphObject クラスのサブクラス) に Capability bit をセットするメソッド setCapability(int bit)がある．この引数の bit には各クラスに用意されている ALLOW_で始まる定数を必要な数だけ OR 演算して指定する．
また Java3D では高速化などの効率を上げるための内部処理のために，一度 SceneGraph

にぶら下げられたノードやコンパイルされたノードに対して別のノードをぶら下げたり取り除いたりできるのは BranchGroup ノードのみに制限している.
動的に他のオブジェクトをぶら下げたい場合 BranchGroup オブジェクトの下にそのオブジェクトをぶら下げた後に SceneGraph に BranchGroup オブジェクトをぶら下げる.
親には Capability bit の ALLOW_CHILDREN_EXTEND をセットする.

```
root.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
// オブジェクトの追加を許可
root.compile();
BranchGroup newGroup=new BranchGroup();
newGroup.addChild(new ColorCube());// 色付き立方体
root.addChild(newGroup);// 追加できる
```

動的にオブジェクトを取り除く方法は，前もってそのオブジェクトを Capability bit の ALLOW_DETACH をセットした BranchGroup オブジェクトの下にぶら下げておいて，BranchGroup オブジェクトごと detach() メソッドで取り除く.

```
root.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
root.compile();
Shape3D shape = new ColorCube( 0.5 ); // newGroup2 を root にぶら下げる
newGroup2.addChild( shape );
root.addChild( newGroup2 ); // newGroup2 を root から取り除く

newGroup2.setCapability( BranchGroup.ALLOW_DETACH );
newGroup2.detach();
```

このソースプログラムを実行すると BranchGroup (root)にぶら下がった BranchGroup (newGroup)に ColorCube がぶら下がるのだが，その後，BranchGroup (newGroup)が BranchGroup (root)より，取り除かれるために空間に物体は表示されなくなる.

3.12 効率化するためには

ここでは，Java3D の処理を切り替えるなどして効率よく処理できるようにするオブジェクトを紹介する.

3.12.1 SharedGroup クラス

同じ物体(中身が同じ Shape3D オブジェクト)を複数箇所に表示させたい時， Shape3D

オブジェクトを複製する必要がある．しかし同じオブジェクトを作ることはメモリーを余分に消費したり処理速度が落ちたりして非効率的である．例えば色を変更したい時に全てのオブジェクトの色を変更しなければならない．
SharedGroup クラスと Link クラスを使いこれらの無駄をなくすことができる．シーングラフでリンクを実現するものである．
Shape3D オブジェクトなど SceneGraph に複数ぶら下げたいオブジェクトを SharedGroup オブジェクトにぶら下げる．

```
SharedGroup shared=new SharedGroup();
shared.addChild(new ColorCube());
```

そして，Link オブジェクトを SceneGraph にぶら下げる．

```
root.addChild(new Link(shared));
Transform3D transform=new Transform3D();
transform.setTranslation(new Vector3d(1.0,0.0,0.0));
TransformGroup translation=new TransformGroup(transform);
translation.addChild(new Link(shared));
root.addChild(translation);
```

なお，SharedGroup オブジェクトは SceneGraph にはぶら下げない．

3.12.2 Switch クラス

図形を切り替えたい，背景を切り替えたい，もしくは高速化のためにといった理由で，特定のオブジェクトを SceneGraph から一時的にはずしたりしたい時がある．
しかし，SceneGraph へのオブジェクトの追加や削除自体に時間がかかったり， Java3D の最適化も適応できなくなるなどの問題点がでてくる．
Java3D には Switch クラスが用意されている．このクラスは Group の 1 つでその下にぶら下げられているオブジェクトの有効/無効を切り替えるものである．
有効/無効にするオブジェクトの指定方法には 2 種類の方法が用意されている．
1 の方法は有効とするオブジェクトをインデックスで指定する方法である． Switch オブジェクトは通常の Group と同じように扱えるが，setWhichChild(int child)メソッドまたは，コンストラクタ Switch(int whichChild)で有効にするオブジェクトのインデックスを指定する．インデックスは Switch オブジェクトに追加した順に 0,1,2・・・・・と付けられる．
具体的に，次のソースプログラムのようになる．

```
BranchGroup root = new BranchGroup();

Switch selecter=new Switch();//スイッチの定義
selecter.addChild(new Sphere(0.3));          // 球 (0 に指定)
selecter.addChild(new ColorCube(0.3));// 色付き立方体 (1 に設定)
selecter.setWhichChild(0);          // 球を表示させる

root.addChild(selector);
```

この場合，インデックスで指定されないオブジェクトは無効となる．インデックスには全てのオブジェクトを有効にする定数 CHILD_ALL や全てのオブジェクトを無効にする定数 CHILD_NONE を指定する事もできる．

もう 1 つの方法としてオブジェクトごとに有効/無効を設定する方法がある．上の方法でインデックスに定数 CHILD_MASK を指定し，setChildMask(BitSet childMask) メソッドで 有効/無効 を一括設定する．

具体的に，次のソースプログラムのようになる．

```
BranchGroup root = new BranchGroup();

Switch selecter=new Switch(Switch.CHILD_MASK);
selecter.addChild(new Sphere(0.3));          // 球 (0 に指定)
selecter.addChild(new ColorCube(0.3));//色付き立方体(1 に指定)

BitSet flag=new BitSet(2);
flag.set(0);        //0 を有効に
flag.clear(1);      //1 を無効に
selecter.setChildMask(flag);

root.addChild(selector);//BranchGroup に Switch をぶら下げる．
```

このソースプログラムを実行すると空間には球が表示されることとなる．Switch は図形そのものを入れ替えたりする場合に威力を発揮する．

3.12.3 OrderedGroup クラス

SceneGraph にぶら下がっている全てのオブジェクトは，自動で最適な順に処理される．

しかし，指定した順番に処理させることも可能である．
そのためには OrderedGroup クラスを利用する．このクラスは Group の 1 つで，その下にぶら下げられているオブジェクトは必ずぶら下げられた順に処理される．
ソースプログラムの例を示す．

```
BranchGroup root = new BranchGroup();

OrderedGroup orderedGroup=new OrderedGroup();
orderedGroup.addChild(new Sphere());//(1 番目に処理される)
orderedGroup.addChild(new ColorCube());// (2 番目に処理される)

root.addchild(orderedGroup);
```

このソースプログラムの場合，球が表示されてからその後に色つき立方体が表示される．

3.13 Java と Java Applet の組み合わせによるJava3D

今までは Java3D に用意されているオブジェクトやクラスを使った表現方法を紹介してきたが，基本的に Java3D は Java 言語の Java Applet を使用して表現されていることが多い．今回の研究においても Java Applet を利用することによって JavaD を表現している．

3.13.1 Java 言語と組み合わせる

Java3D も Java 言語の 1 つであることに変わりはない．従って通常の Java 言語と組み合わせて，プログラムを作ることができる．やり方は通常の Java 言語の命令を，ソースプログラムの中に組み込んでいく．プログラム中に数値の入力を指定したり，変数の数値の表示をさせることができる．例えば System.out.println();がある．ただし，グラフィックが表示される GUI ウィンドウではなく，Dos ウィンドウに表示される．

3.13.2 Java Applet との組み合わせによる，各種ボタンの指定方法

前述したとおり Java3D は Java Applet を使用して表現されていることが多い．そのため Java Applet に用意されている機能やクラスを Java3D でも利用できる．ボタンもその一つであり，ボタンをマウスで押したり，選択することによって，Java3D の物体や，背景等の表現変化させることが可能である．
ボタンの表現方法を，幾つかに紹介する．

3.12.3 通常のボタンの作り方

まず，ボタンの使用を許可させるために extends Applet 以下に

implements ActionListener {

を加える．これはボタンごとに違うので” , ”によって複数指定が可能である．
次にボタンの定義を行う．

```
Button rotateA = new Button("左");
Button rotateB = new Button("右");
```

ボタン 1 つごとに 1 つずつボタンの定義を行い，文字を入力したいときは（“ ”）に文字を入力する．この場合は「左」「右」という 2 個のボタンになる．

```
Panel p = new Panel();
p.add( rotateA );
p.add( rotateB );
add("South",p);
```

パネルを定義し,そこに先ほど定義したボタンを指定する.パネルの表示位置は North（上），South（下），West（左），East（右），Center（中央）となる．
ボタンは表示の順に並ぶことになる．なお，ここで行を変えたり文字やパネルの色を指定することもできる．

```
rotateA.addActionListener( this );
rotateB.addActionListener( this );
```

ここで implements ActionListener にボタンを加える．

```
public void actionPerformed( ActionEvent e ) {
    if ( e.getSource() == rotateA ){
        angle +=  Math.toRadians( 10 );
        trans.rotX( angle );
        objTrans.setTransform( trans );
    }

    if ( e.getSource() == rotateB ){
        angle -=  Math.toRadians( 10 );
        trans.rotX( angle );
        objTrans.setTransform( trans );
```

```
            }
        }
```

このようにしてボタンを押したときの処理を定義する，この場合「左」のボタンを押すと物体が左に 10°，「右」のボタンを押すと右に 10°回転するように設定してある．if 文や if～else 文を使って表現するとが多い．

3.13.4 ラジオボタンの作り方

ラジオボタン（チェックボタン）を設置するときも基本的に通常のボタンの設置と同じような処理を行う．ただし多少異なる部分がある．

まず通常ボタン用に用意した implements ActionListener { を implements ItemListener{（ラジオボタン用）に変更する．

```
        CheckboxGroup Box = new CheckboxGroup();
```

ラジオボタンは複数の選択肢から 1 つのみ選択するので，まず，ラジオボタンの所属するグループを定義する．

```
        Checkbox box1 = new Checkbox( "通常色 1", Box,true );
        Checkbox box2 = new Checkbox( "通常色 2", Box,false );
```

または

Checkbox box1, box2;

（）内は(“ボタンに表示させたい文章や，所属する CheckboxGroup の名前,ボタンの初期状態（true 最初にチェックが存在する，false がチェックされていない）となっている．

```
        p.setLayout(gl);
        p.add( box );
        p.add( box );
        add("South",p);
```

または

```
        p.setLayout(gl);
        p.add(new Checkbox( "通常色 1",Box,true ))
        p.add(new Checkbox( "通常色 2",Box,false ));
        add("South",p);
```

の形式でパネルに設置する．

```
box1.addItemListener(this);
box2.addItemListener(this);
```

通常ボタンの ActionListener がラジオボタン用の ItemListener に変更になる.

```
public void itemStateChanged( ItemEvent e ) {
                root2.detach();
            universe.addBranchGraph( root1 );
        }

        if( box2.getState() ) {
                root1.detach();
                universe.addBranchGraph( root2 );
        }
}
```

通常ボタン用の public void actionPerformed(ActionEvent e) {　を
ラジオボタン用の public void itemStateChanged(ItemEvent e) {　に変更する.
if (e.getSource() == rotateA){も if(box1.getState()) {という形式に変更する.
if 内の処理を説明すると BranchGroup を Capability bit に detach()を使って切り替えている.

図 3.2 ボタン(左), ラジオボタン(右)

3.14 背景を切り替える方法

3.14.1 detach()をボタン処理に据えた場合

MouseBehavior やボタンを設定することにより, 外的制御によって物体の性質を複数変

更することが可能となった．ここでそれらの機能を利用して物体の背景をボタンを使って変更する方法を紹介する．
まず，Capability bit の変更を行えるようにすることである．
背景 BackGround は，Scene Graph にぶら下がっている．そのため，背景がぶら下がっているオブジェクトから背景のオブジェクトを切り離し，別の背景のオブジェクトをぶら下げる必要がある．
例えば

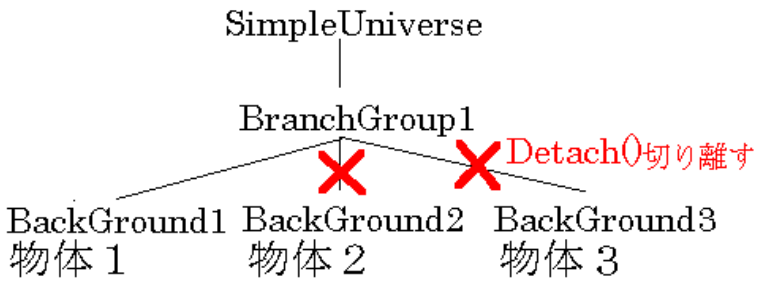
```
BackGround bg1 = new BackGround( new color3f( 1.0f, 1.0f, 1.0f ) );
bg1.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
bg1.setCapability(BranchGroup.ALLOW_DETACH);
bg1.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

BackGround bg2 = new BackGround( new color3f( 0.0f, 0.0f, 0.0f ) );
Bg2.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
Bg2.setCapability(BranchGroup.ALLOW_DETACH);
Bg2.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
                    •
                    •
                    •
                    •
BackGround bgN = new BackGround( new color3f( R, G, B ) );
BgN.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
BgN.setCapability(BranchGroup.ALLOW_DETACH);
BgN.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
```

ソースプログラムのように，Capability bit を利用してオブジェクトの追加，削除，書き込みを，切り替えたい背景の数だけ設定する．
その上で，複数の背景をぶら下げていく．
以下ボタン処理設定の部分で次のような処理を行う．

```
if(box1.getState()){
    bg2.detach();
      •
      •
    bgN.detach();
    root.addChild( bg1 );
```

```
        }
        else if(box2.getState()){
            bg1.detach();
                •
                •
                •
            bgN.detach();
            root.addChild( bg2 );
        }
                •
                •
                •
        else if(box2.getState()){
            bg1.detach();
            bgN.detach();
                •
                •
                •
            root.addChild( bgN );
        }
```

と入力し，ボタンを切りかえるたびに，ぶら下げられているオブジェクトを切り離し，選択したオブジェクトをぶら下げることができる．



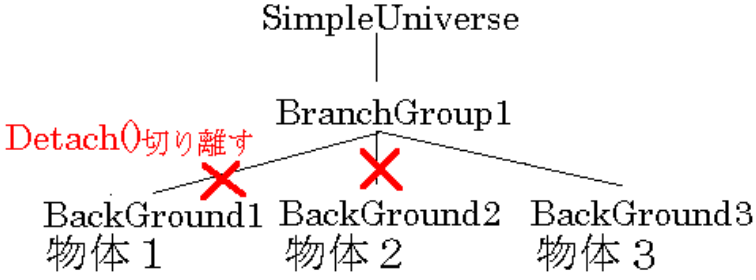


図 3.26Capability bit と Detach()による切り替えの仕組み

3.14.2　Switch をボタン処理に置換する方法

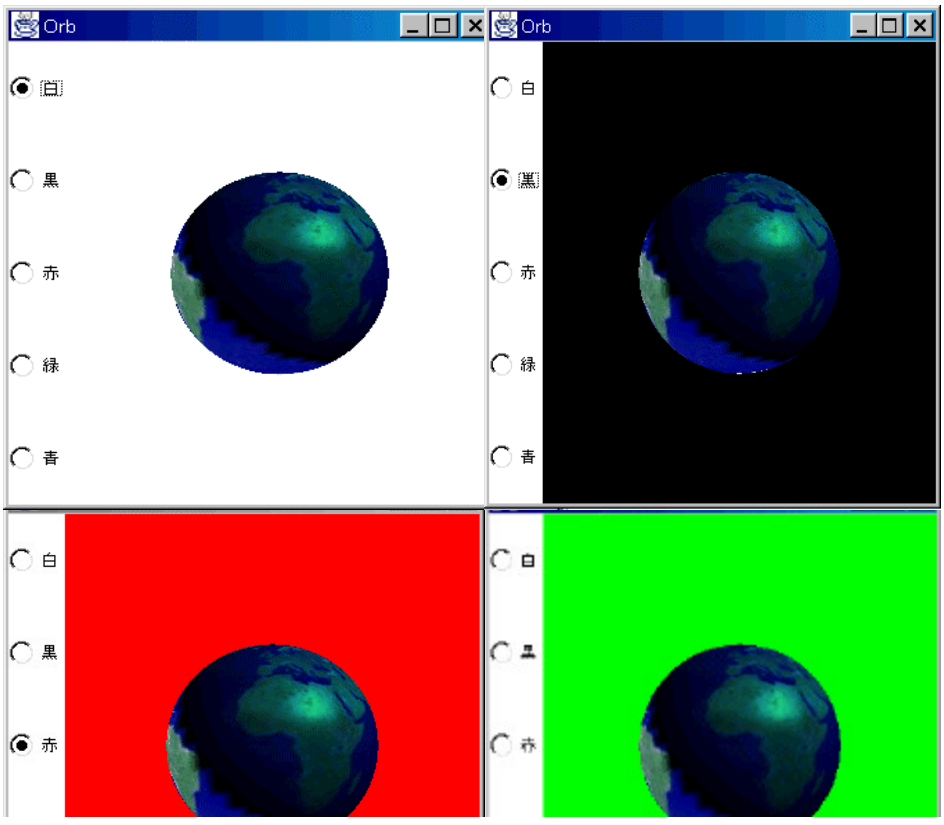
detach の変わりにボタン部分に Switch オブジェクトを使って背景を切り替える方法がある．3.11 節で説明した Capability bit の設定に Switch の追加，切り離しに関する設定を追加する．

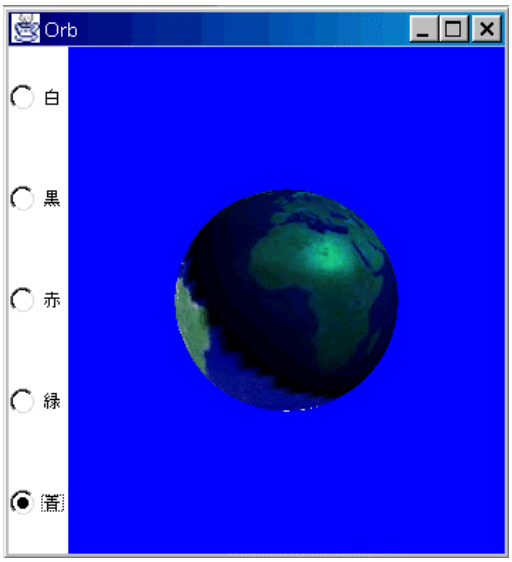
```
selecter.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
/
selecter.setCapability(BranchGroup.ALLOW_DETACH);
selecter.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
```

次にボタン設定部分で次のように処理する．

```
if(boxB.getState()){
    root.detach();
    flag.clear(0); // インデックス 0 を有効に
    flag.clear(1); // インデックス 1 を無効に
    flag.set(2);   // インデックス 1 を無効に
        selecter.setChildMask(flag);
   universe.addBranchGraph( bg1 );
}
```

とする．いずれの場合にしても，選択肢の数だけ背景を用意しなければならない．





3.15 物体を切り替える

3.14.で説明した方法は物体にも適応することができる．物体のみならず BranchGroup オブジェクトにぶら下がっているものは，すべて変更がが可能である．ただし注しなければならないのは，切り離したオブジェクトにほかのオブジェクトがぶら下がっていたときは，それ以下のオブジェクトに関しても選択肢分用意しなくてはならない．

```
Shape3D shape1 = new ColorCube( 0.1 );
Shape3D shape2 = new ColorCube( 1.0 );

shape1.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
shape1.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
shape1.setCapability( BranchGroup.ALLOW_DETACH );

shape2.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
shape2.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
```

```
shape2.setCapability( BranchGroup.ALLOW_DETACH );

root.addChild( shape1 );
root.addChild( shape2 );
```

としたする．次にボタン設定部分で，以下のようにする．

```
if( boxA.getState() ) {
    shape2.detach();
    universe.addBranchGraph( root );
}

if( boxB.getState() ) {
    root1.detach();
    universe.addBranchGraph( root2 );
}
```

背景同様に Switch を使用して，図形を切り替えることもできる．背景と物体を別々に切り替えることも可能である．これらの機能を用いて Java3D を表現することができる．

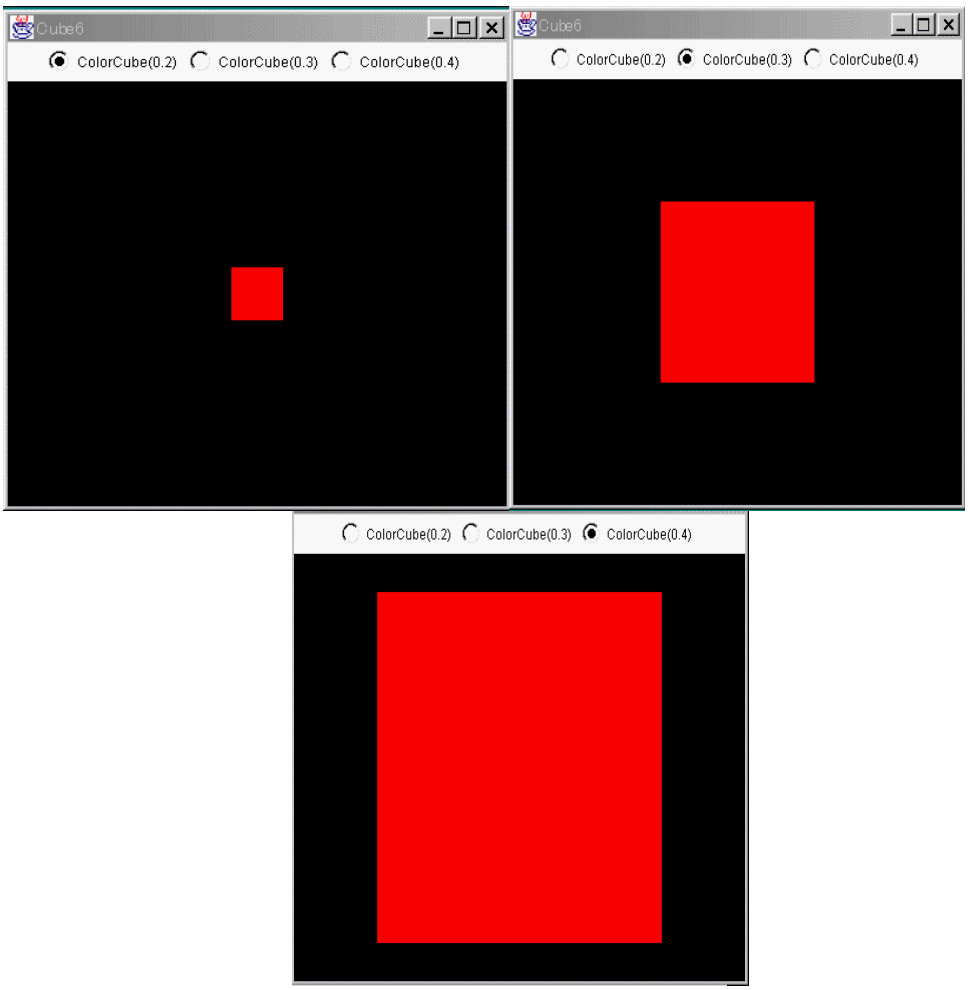


図 3.28 物体の変更例

3.16 Java3D のクラス

Java3D の各機能を使うためにさまざまなクラスが用意されている．
それらをまとめるものとする．

javax.media.j3d.* クラス

javax.media.j3d	Java3D の主要なクラスはここに含まれている．
Canvas3D	3 次元グラフィックスが描画される AWT である．
VirtualUniverse	シーングラフの根となるクラスである． 簡単な 3 次元処理を行う時は，そのサブクラスの com.sun.j3d.utils.universe. SimpleUniverse クラスが使われる．
BranchGroup	この下にさまざまな Node をぶらさげる．
TransformGroup	Transform3D オブジェクトを持つ Group である．
Transform3D	位置情報を所有する 4×4 の行列である．回転や拡大縮小，平行移動を表わすものである．物体の位置とは原点からの平行移動で表現するものである．
Shape3D	物体を表わすクラスである．Geometry オブジェクトと Appearance オブジェクトの情報を持っている．
Geometry	物体の形を表わす．使う時はそのサブクラスを使う．
Appearance	物体の見かけ(テキスチャー・色など)を表現するクラス．実際には数種類のオブジェクトへの参照で，情報を保持している．
GraphicsContext3D	Immediate Mode を使う場合に必要なクラスである．AWT の Graphics クラスに対応するクラスである．
SharedGroup	同じ Node をシーングラフの複数箇所にぶら下げたい時に使う Group である．Link クラスと一緒に使われる．
Switch	ぶら下げた Node を個別に有効/無効にできる Group である．
Background	背景を表わすクラスである．
Fog	霧(フォグ)効果を表わすクラスです．
Light	光源を表わすクラスである． 使う時は そのサブクラス を使う．
Link	シーングラフオブジェクトへの参照を表わすクラスである．
GeometryArray	配列を使って定義できる Geometry のスーパークラスである．
TriangleArray	3 角形の面で構成される多面体を表示する．
QuadArray	4 角形の面で構成される多面体を表示する．

LineArray	線分の集まりを示す.
PointArray	点の集まりを示す.
IndexedQuadArray	インデックスを使った 4 角形の面で構成される多面体である.
IndexedPointArray	インデックスを使った点の集まりである.
IndexedGeometryArray	インデックスと配列を使って定義できる
IndexedTriangleArray	インデックスを使った 3 角形の面で構成される多面体である.
IndexedLineArray	インデックスを使った線分の集まりである.
Light	光源クラスのベースとなる抽象クラスである.
AmbientLight	環境光である．周囲の環境による散乱光，周囲の物体による相互反射によって生じる間接光を示している.
DirectionalLight	平行光線である．太陽光のように無限遠点に光源がある場合に使われる.
SpotLight	スポットライトである．点光源の特殊形として定義されている.
PointLight	点光源である．電球のような比較的小さな光源がある程度離れた物体を照射する時に光源の大きさを無視しても問題ないような場合に使われる.
SpotLight	スポットライトである．点光源の特殊形として定義されている.
Material	光源下での物体の見え方を指定するクラスである.
Texture	テキスチャーを示すクラスである.
TransparencyAttributes	透明度を示すクラスである.
PolygonAttributes	レンダリング方式などを指定するオブジェクト.
Interpolator	Interpolator クラスのベースとなる抽象クラスである.
RotationInterpolator	Y 軸を中心とした回転運動をする Interpolator である．回転角=α値×2π
PositionInterpolator	X 軸に沿った移動を行う Interpolator である．X 軸=α値
ScaleInterpolato	スケール変更を行う Interpolator である．スケール=α値
TransparencyInterpolator	透明度変更を行う Interpolator である．透明度=α値
PositionPathInterpolator	指定した折れ線状の経路をたどる Interpolator である.
RotationPathInterpolator	指定した姿勢の線形補間をする Interpolator である.
Bounds	Bounds クラスのベースとなる抽象クラス.
BoundingBox	直方体によって表現される Bounds である.
BoundingSphere	球によって表現される Bounds である.
BoundingPolytope	凸多面体によって表現される Bounds である.

パッケージの概要を示すクラス

・javax.vecmath	3 次元計算に役立つ幾何学クラス(行列やベクトル)がある. javax.media.j3d パッケージクラスのメソッド引数にこのパッケージのクラスを使う事がある.
・com.sun.j3d.*	javax.media.j3d と javax.vecmath パッケージだけで 3 次元処理のプログラムは書けるが，非常に長いコードになったりすることがある．そのため良く使われるであろうクラスがこ

	のパッケージにまとめられている.

com.sun.j3d.utils.* クラス

universe.SimpleUniverse	VirtualUniverse クラスと Locale クラスを 1 つにまとめた働きをするクラスである. SceneGraph の根になる.
geometry.ColorCube	色付き立方体である.
applet.MainFrame	アプレットとして設計したクラスをアプリケーションとしても実行できるようにするためのクラスである. main メソッドでこのコンストラクタを呼ぶようにする.
.image.TextureLoader	BufferedImage オブジェクトやファイル名で指定した画像ファイルを元に Texture オブジェクトを作る.
geometry.Primitive	Primitive のベースとなる抽象クラスである.
geometry.Box	直方体を表示する.
geometry.Cone	円錐を表示する.
geometry.Cylinder	円柱を表示する.
Geometry.Sphere	球を表示する.

4 評価

この 4 章では，顔を表示させ，ボタン操作で顔の色と背景を変えることができるプログラムを異なる手法のものを 2 種類作成し，2 つのプログラムをいろいろな観点から評価を行うことにする．

4.1 顔表示プログラムの作成

顔表示プログラムは顔の持つそれぞれの部分のクラス，つまり目・口・鼻そして，輪郭を構成するクラスと，帽子とそのつばを構成するクラスを FaceSetting というクラスで組み立て，表示用のクラスで表示するものになっている．下図 4.1.1 に大まかな構成の図を示す．

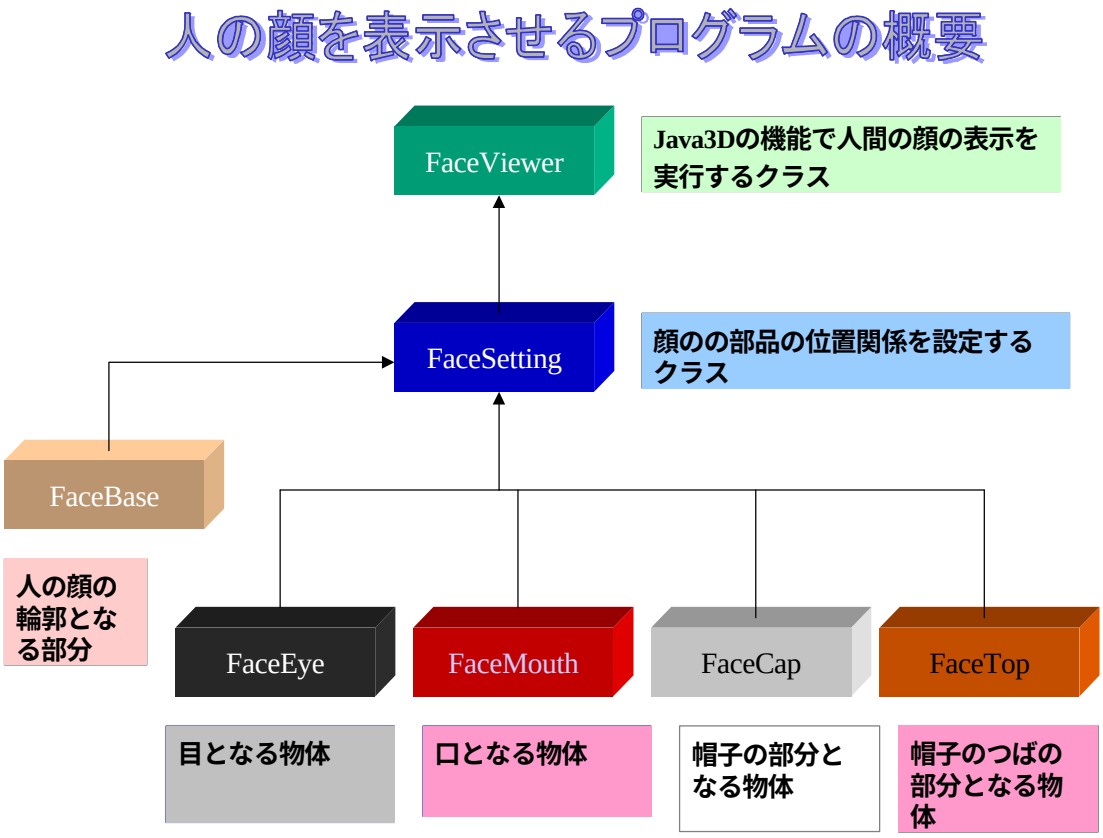


図- 4.1.1 顔表示プログラムの構成図

各顔の部品のクラスではその物体の形状・色を決定し，それを組み立てる FaceSetting で顔の部品の位置決めを行っている．また，表示させるクラス FaceViewer から変数を送り，その数値を元にして顔の大きさと顔の色を決定するようになっている．

4.2　2 つの手法による顔の色の変更

Java3D では，シーングラフに存在する物体を表示する．それより，物体の内容の変更を行う方法として次の 2 つを考えた．

1.あらかじめ変更する物体のオブジェクトを生成しておき，ボタンを押したときにそのシーングラフにぶら下げられているオブジェクトを変更したい物体のオブジェクトにつなぎなおす方法
　（以後，この方式をセレクト型と書く）

2.ボタンを押すまでオブジェクトの生成は行わず，変更のボタンを押された時点で新たにその物体のオブジェクトを作り直して表示する方法
　（以後，この方式をクリエイト型と書く）

1 のセレクト型の場合，表示したい物体をあらかじめオブジェクト生成して置きつなげ直すだけという方式であるため，ボタンを押して物体の表示を変える際の処理は速いが，メモリを多く使用することが予想される．

一方 2 のクリエイト型の場合，表示するたびにオブジェクトのの生成を行うために，メモリの使用率を低く押さえられるが，ボタンを押した際の処理に時間がかかると考えられる．

4.3　プログラムの起動時間とボタンを押した時の処理時間の計測

4.3.1　Java３D 1.1 ,JavaVM ,メインメモリ 64M バイトでの測定

4.2 で説明したプログラムに起動時間とボタンを押したときの時間を測るために，時間を測るコマンドを入れて，起動時間と処理の時間を測定した．

下に，起動時間を含めた処理時間のグラフと，処理時間のみのグラフを示す．

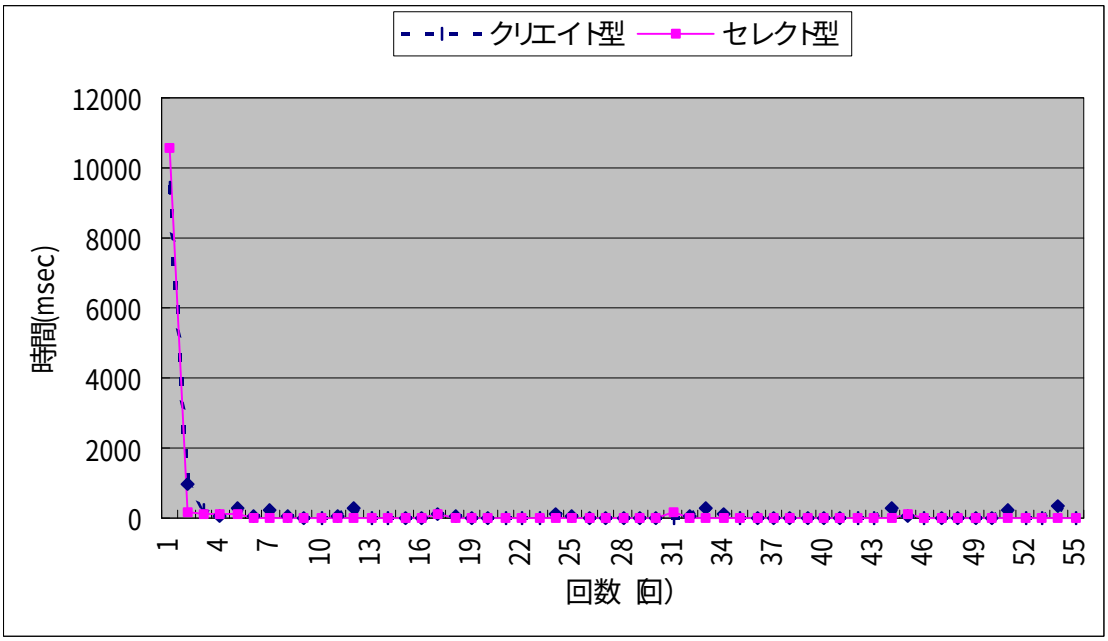


図-4.3.1　起動時間と処理時間の比較

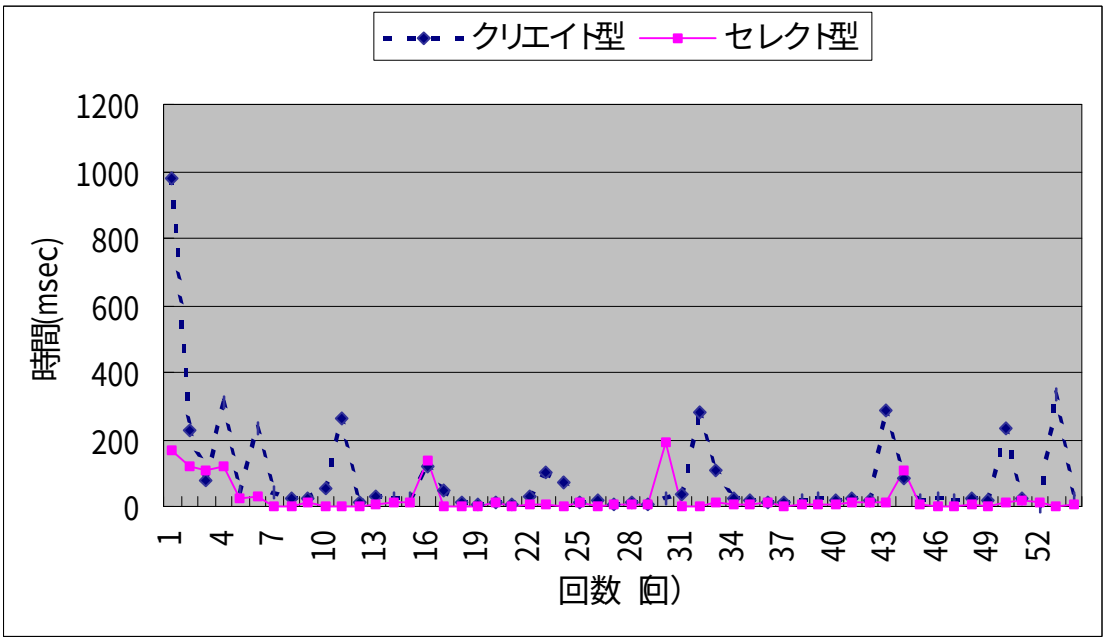


図-4.3.2　(**図** 4.2.1)の処理時間のみを表したグラフ

図 4.3.1 より，クリエイト型の方が起動時間は平均で約 1100msec 速くなっている．これはクリエイト型ではボタンを押すまで顔のオブジェクトを作成しないため，起動時に顔の数だけオブジェクトを生成するセレクト型より速くなったと思われる．

また，クリエイト型は 1 回目の実行の際に初めて顔のオブジェクトを生成するため，平均で 979msec となり，セレクト型の 165msec と比較して約 1/5 になった．

それ以降の回の測定でも，ボタンを押すたびにオブジェクト生成を行うクリエイト方はセレクト型に比べて全体的に処理に時間がかかっている．

また，両方の型で，特定の回数で処理時間が多くかかるところが見られた．これは１つの予想として，この回のタイミングでガーベッジコレクションが働いたと考えられる．

4.3.2　Java３D 1.1 ,JavaVM ,でメモリを増やした場合の測定

現在のまで行った環境である 64Mバイトのメインメモリを増やして計測を行うとどう変わるかをメモリ以外同じ環境で測定を行った．

結果をメモリが 96M バイトの場合、160M バイトの場合それぞれ下に示す。

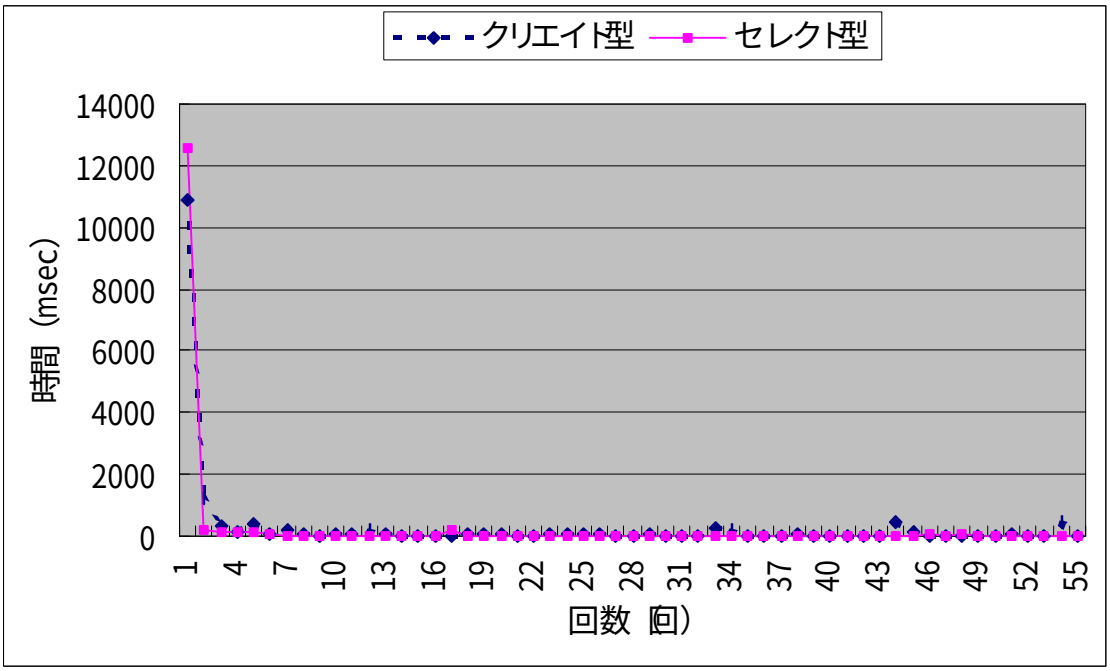


図-4.3.3　メモリ 96M バイト時の起動時間と処理時間

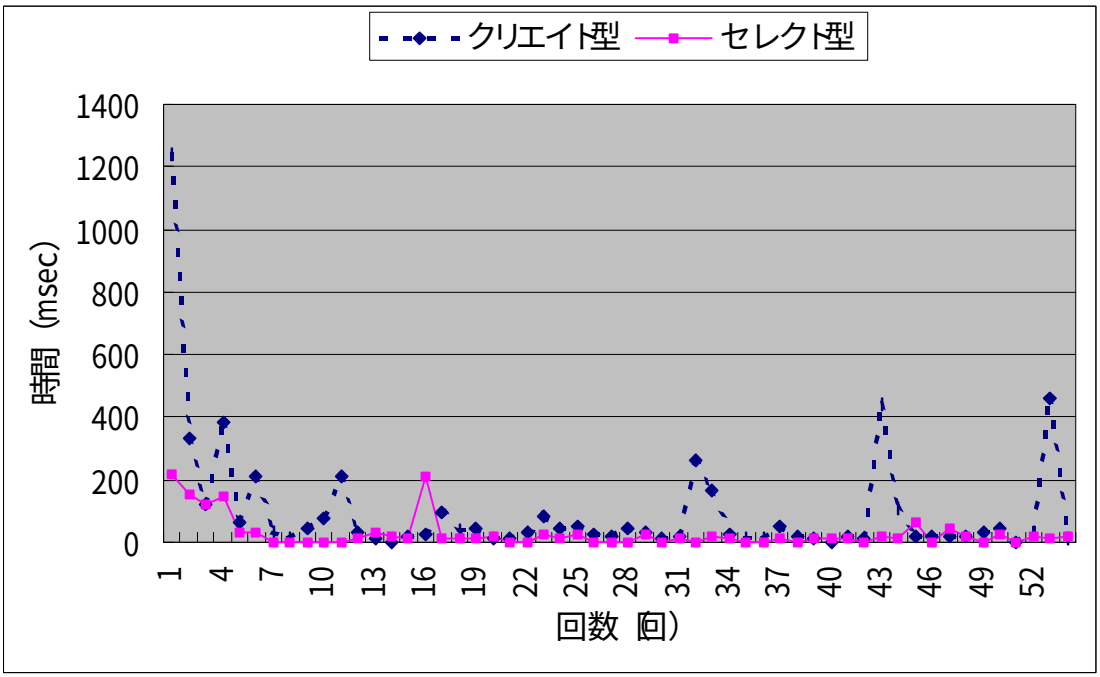


図-4.3.4　メモリ 96M バイト時の処理時間

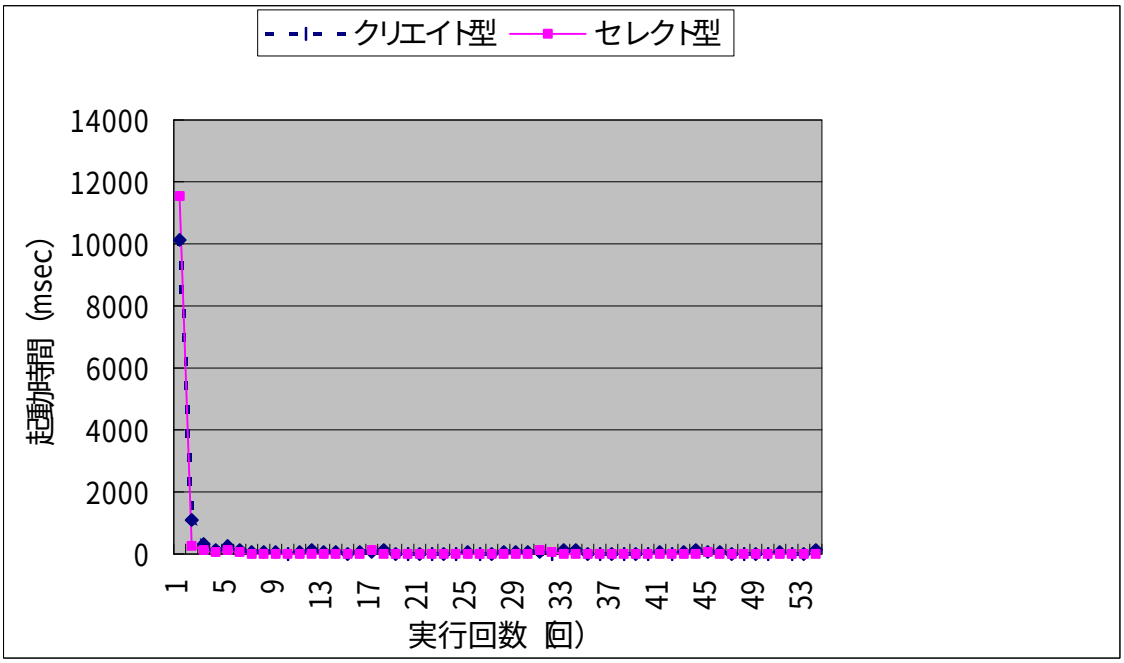


図-4.3.5　メモリ 160M バイト時の起動時間と処理時間

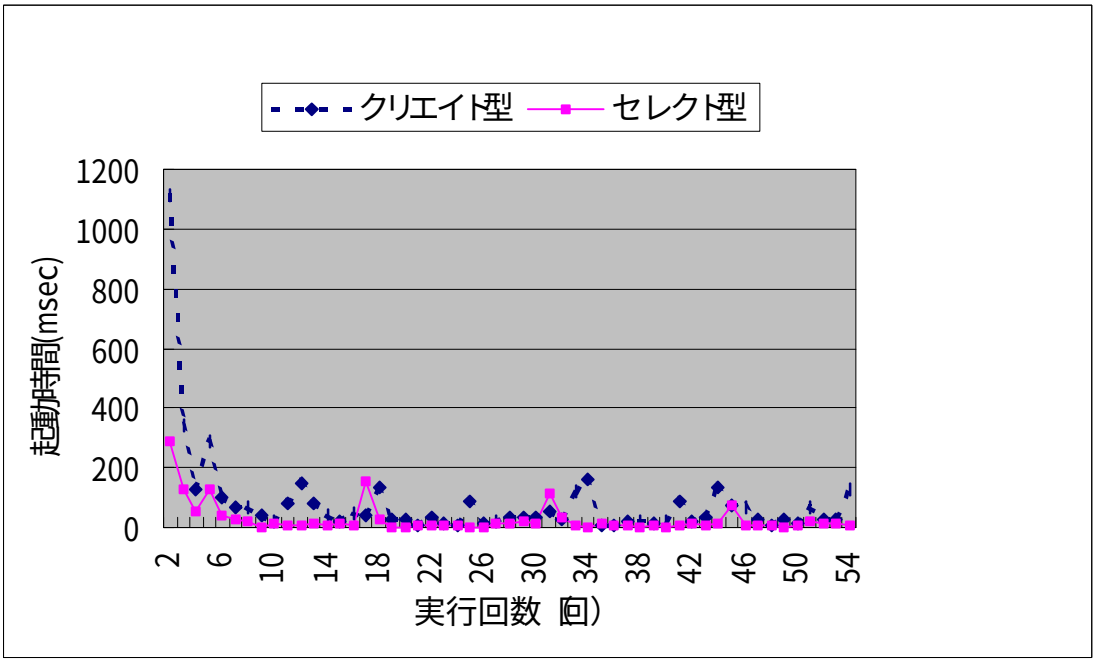


図-4.3.6　メモリ 160M バイト時の処理時間

メインメモリを増やしての計測の結果，メモリを増やしても速度は速くならなかった．これは，JavaVM の使用できるメモリの最大値がデフォルトで 64M となっているため，それを越えるメモリを増やしても使用されないためと思われる．

4.3.3 Java 3D 1.1 で HotSpot を使用した場合の測定

ここで，特定の回数で処理が多くなったことの原因として考えられるガーベッジコレクションを効率よく行わせて処理をスムーズにするために，JavaVM として HotSpot(Java HotSpot Performance Engin)を使用して測定を行うことにする．HotSpot は高速化のための機構特徴を 3 つ持ち，1 つ目は Java2 に含まれている標準の JavaVM よりも高速かつ効率的なガーベッジコレクト機能をサポートしている．2 つ目は，プログラムの実行時にそのボトルネックを検出し,ネーティブ･コンパイルを実行する．3 つ目は，大容量のメモリ空間やマルチプロセッサを有効に使うためのスレッド･チューニングの機能を備えている．

以下に HotSpot を使用した場合の測定結果をメモリの量が異なる場合で分けて示す．

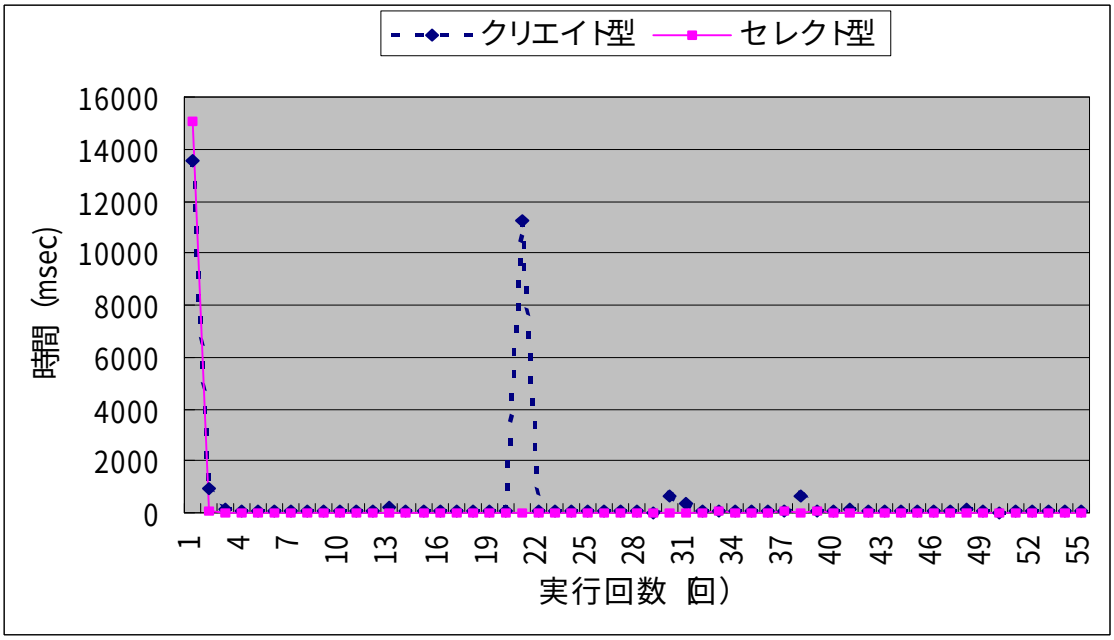


図-4.3.7　メモリ 96M バイト時 HotSpot 使用した場合の起動時間と処理時間

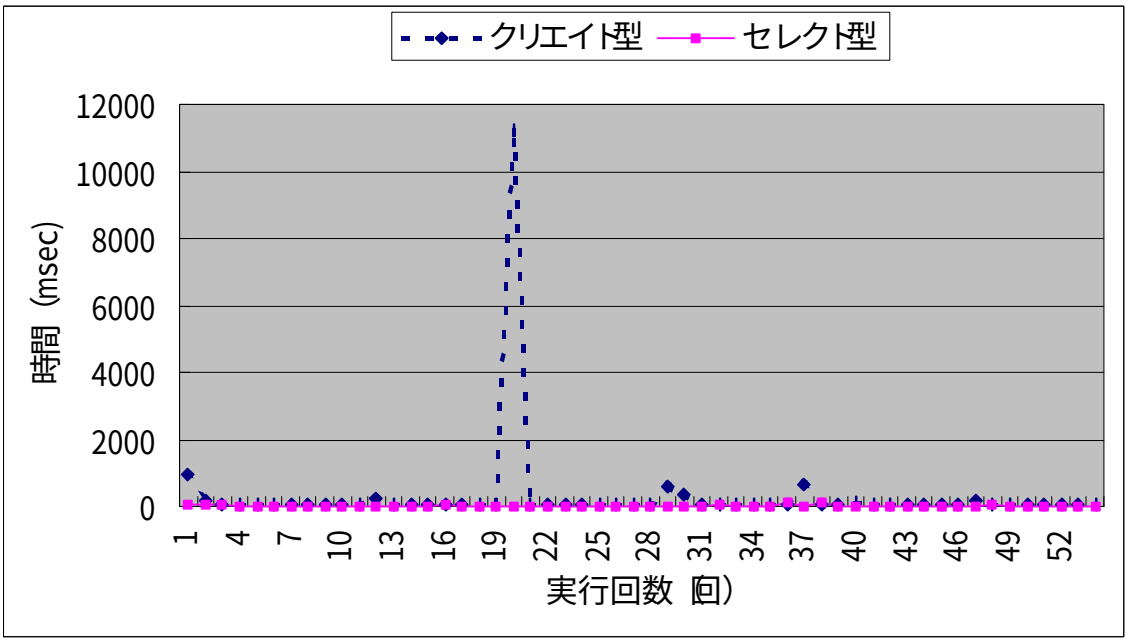


図-4.3.8　メモリ 96M バイト時 HotSpot 使用した場合の処理時間

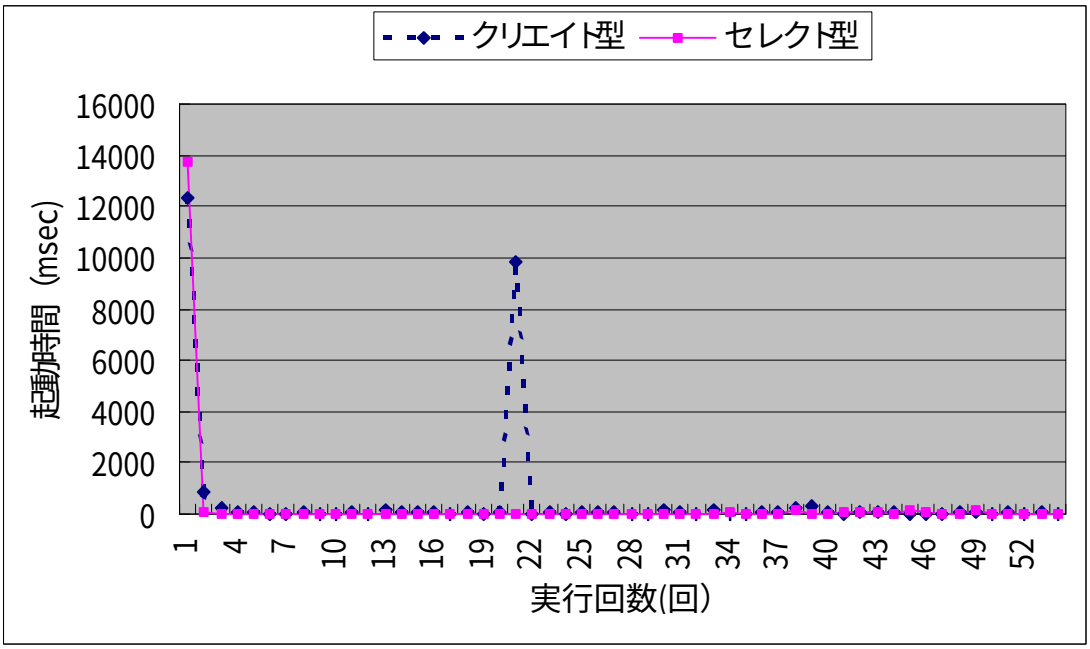


図-4.3.9　メモリ 160M バイト時 HotSpot 使用した場合の起動時間と処理時間

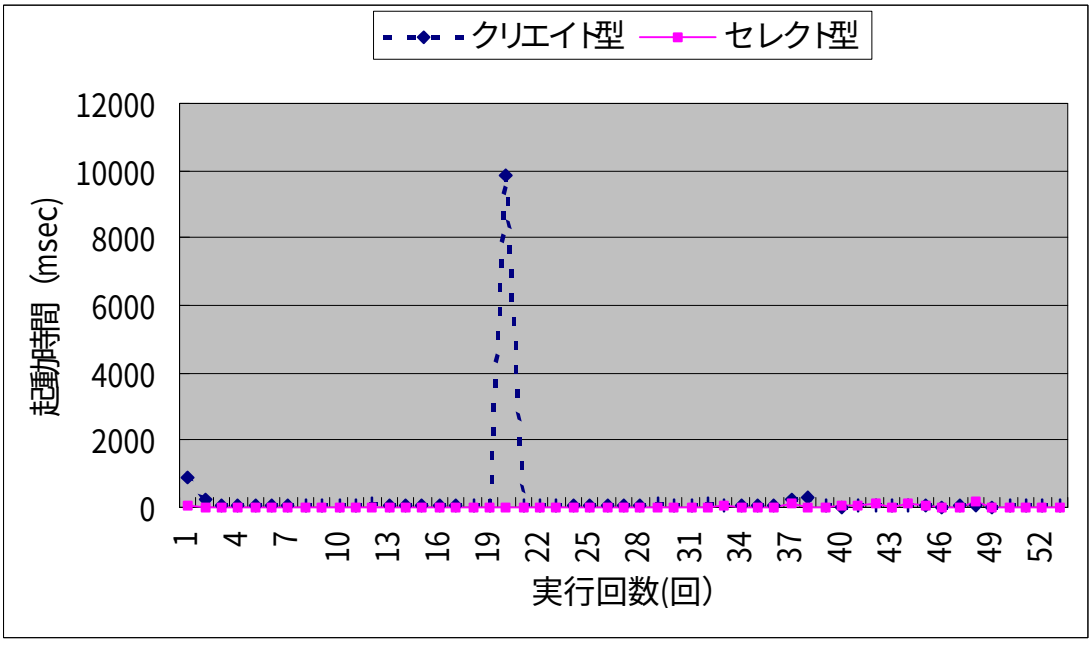


図-4.3.10　メモリ 160M バイト時 HotSpot 使用した場合の処理時間

図-4.3.7~10 より HotSpot を使用した結果，全体的に処理の時間が遅くなり，また，特定の回数で処理時間が以上に大きくなるところが発生した.

これは, HotSpot によってガーベッジコレクションのタイミングが変わったが, HotSpot はサーバ･プログラムの性能向上に力点を置いた環境でありクライアント･サイドではその能力を発揮できず，処理に時間がかかってしまった可能性が考えられる.

4.3.4 Java 3D 1.1.3 にした場合の測定

HotSpot を入れるにあたって，今まで使用していた Java3D 1.1 の後にリリースされた Java3D 1.1.1 以降では, 高速化とメモリ消費量の削減がはかられているということがわかり，その違いについて調べるため，最新版の Java3D 1.1.3 を入れ，JavaVM を使用した場合と HotSpot を使用した場合についてそれぞれ計測を行った.

以下にその 2 種類の結果を示す.

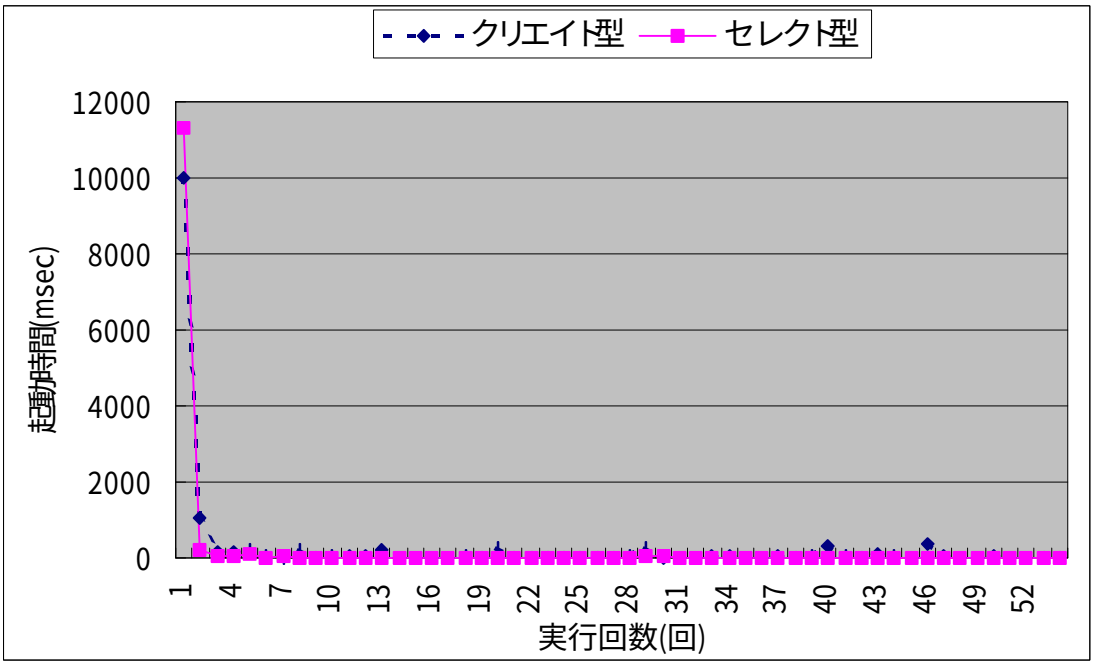


図-4.3.11 メモリ 160M バイト時 Java3D 1.1.3 にして
JavaVM を使用した場合の起動時間と処理時間

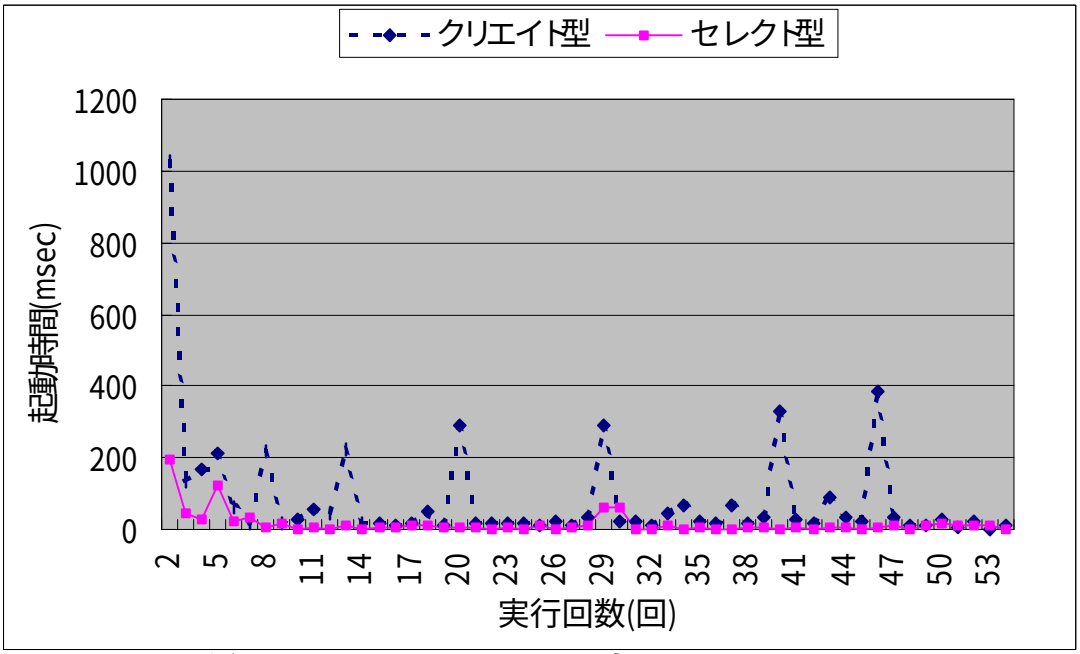


図-4.3.12 メモリ 160M バイト時 Java3D 1.1.3 にして
JavaVM を使用した場合の処理時間

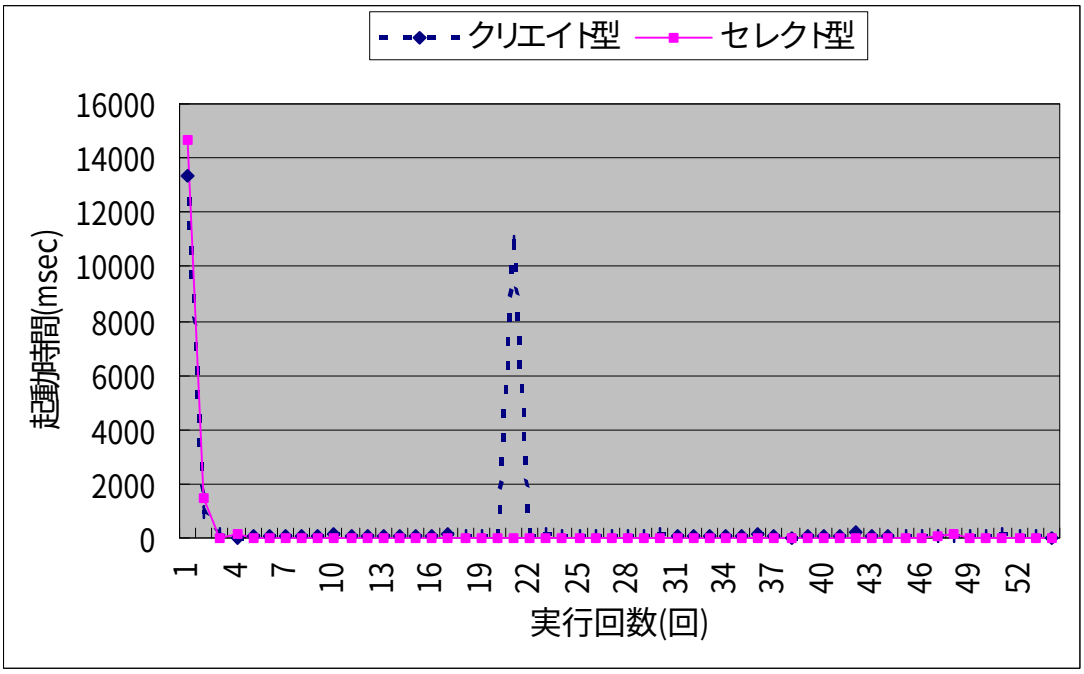


図-4.3.13 メモリ 160M バイト時 Java3D 1.1.3 にして HotSpot を使用した場合の起動時間と処理時間

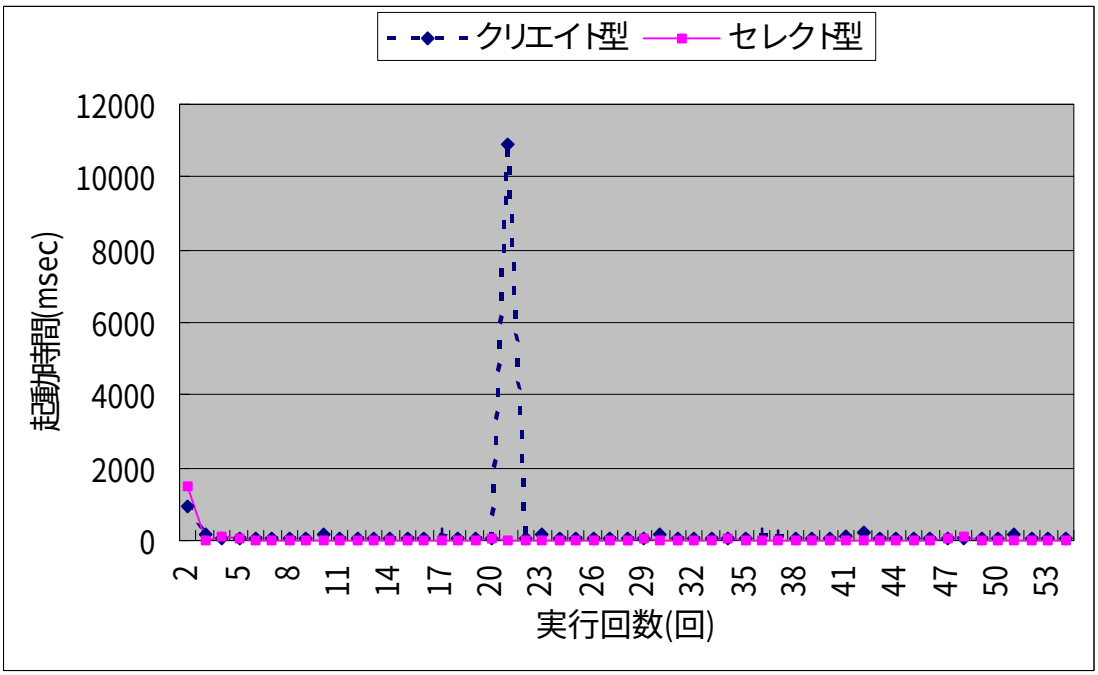


図-4.3.14 メモリ 160M バイト時 Java3D 1.1.3 にして JavaVM を使用した場合の処理時間

図 4.3.11~14 より Java3D 1.1.3 バージョンアップによる大きな変化は見られなかった.

5 結 論

クリエイト型，セレクト型による表示切り替えを比較することにより以下のことが言える．

1）クリエイト型は，周期的に処理速度が遅くなる場合がある．
切り替えの都度オブジェクトが書き換えられるため，メモリの使用率が高くなり，それがある一定の量になると動作が追いつかず遅くなると考えられる．

2)セレクト型は，全体的に書き換え速度が安定している．
これは最初から用意されているオブジェクトが切り替えられる構造のため，実行時に時間を必要とする処理が少ないためと考えられる．ただし，オブジェクトが最初から多数用意されており，プログラムの起動時間はクリエイト型よりも遅くなり，メモリの使用率も多くなる．

3)クリエイト型はセレクト型と比較してソースの量が少なくなる
セレクト型はクリエイト型に比べ，切り離しを行うことができる BranchGroup をオブジェクトの数だけ用意する必要があり，オブジェクトが多くなるにしたがって BranchGroup の量が増えてしまう．それに対し，クリエイト型はオブジェクトをぶら下げるための BranchGroup を最低限用意するだけでよいので，ソースの量は少なくなる．

4) 64Mバイト以上のメモリ量にオブジェクトの書き換え速度は依存していない
これはセレクト型，クリエイト型ともに共通しており，書き換え速度を上げるためには CPU の処理速度を上げることが有効であると考えられる．

以上のことから結論を導くと次の結論を得る．

1)セレクト型を使った開発は，長時間安定した処理を行いたい場合に効果的である．
ただし，定義するオブジェクトの量が多くなるために，プログラムの容量が増え，開発に手間と時間がかかる．

2)クリエイト型を使った開発は，短時間に速い処理を行いたい場合に適している．
ただし，ボタンを押した際の処理が多いため，不安定になる．1 度に多くのメモリ領域を使うような処理を行う動作には適さないといえる．

3)64M バイト以上のメモリ容量に書き換え速度は依存していないため，それ以上のメモリを搭載しても，書き換え速度の向上は期待できない．

4)クリエイト型，セレクト型，を使った開発の場合，これらの性質をうまく利用した開発が必要である．

参考文献

[1] H.Sowizral and K.Rushforth and M.Deering, *The Java3D API Specification*, [竹内里佳(訳), The Java3D API 仕様, アスキー, 1999].

[2] M.Chan and S.Griffith and A.Iasi, *1001 Java Programmer's Tips*, [舟木将彦(訳), Java プログラミング 1001Tips, オーム社, 1997].

[3]鈴木哲哉, だれでもできる Java アプレット, オーエス出版, 1998.

[4]太田篤史ほか, JFC&メディア API 入門, *JAVA PRESS*, Vol 7, pp.6-43, 技術評論社, 1999.

[5]高橋要ほか, 白帯 Java プログラマーのための Java 用語徹底解説, *JavaWorld*, 1999 年 7 月号, pp.168-169, IDG コミュニケーションズ, 1999.

[6]大内義朝, 飛田幹司, 新田実, JUN, 突撃!!3D グラフィック, ジャパンミックス, 1997.

[7]Atsushi, Atsushi's Homepage Java3D Tips, http://www.ipc-tokai.or.jp/~atusi/.

[8]あばぱげ, ABA Games, http://www.asahi-net.or.jp/~cs8k-cyu/.

[9]Y. Andoh, Computer Graphics Domain, http://tech.webcity.ne.jp/~andoh/.

[10]えんどうやすゆき, Java Friendly Society, http://www.javaopen.org/jfriends/.

付　録

本研究で使用したソースプログラムを示す

(1)空間に ColorCube を表示させるためのソースプログラム

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class Cube extends Applet {

    public Cube() {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe=new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );

        add( universe.getCanvas(),"Center" );

        BranchGroup root=new BranchGroup();

        Transform3D translation=new Transform3D();
        translation.setTranslation( new Vector3d( 1.0, 4.0, 2.0 ) );

        TransformGroup transform=new TransformGroup( translation );
        root.addChild( transform );

        Shape3D shape = new ColorCube( 0.3 );
        root.addChild( shape );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph (root );

    }

    public static void main( String[] args ) {
        new MainFrame( new Cube(), 512, 512 );
    }
}
```

(2)空間にダイヤを縦長にした物体を表示させるソースプログラム

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.geometry.*;

public class Daia extends Applet {

    public Daia() {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );

        add( universe.getCanvas(),"Center" );
```

```
        BranchGroup root = new BranchGroup();
        BoundingSphere bounds = new BoundingSphere();

        Transform3D translation = new Transform3D();
        translation.setTranslation( new Vector3d(1.0,4.0,2.0) );

        TransformGroup transform=new TransformGroup( translation );

        float[] vertexs = {
              0.3f, 0.0f, 0.0f, 0.0f, 0.0f, 0.3f, 0.0f,-0.6f, 0.0f,
              0.0f, 0.0f, 0.3f,-0.3f, 0.0f, 0.0f, 0.0f,-0.6f, 0.0f,
             -0.3f, 0.0f, 0.0f, 0.0f, 0.0f,-0.3f, 0.0f,-0.6f, 0.0f,
              0.0f, 0.0f,-0.3f, 0.3f, 0.0f, 0.0f, 0.0f,-0.6f, 0.0f,
              0.0f, 0.0f, 0.3f, 0.3f, 0.0f, 0.0f, 0.0f, 0.6f, 0.0f,
             -0.3f, 0.0f, 0.0f, 0.0f, 0.0f, 0.3f, 0.0f, 0.6f, 0.0f,
              0.0f, 0.0f,-0.3f,-0.3f, 0.0f, 0.0f, 0.0f, 0.6f, 0.0f,
              0.3f, 0.0f, 0.0f, 0.0f, 0.0f,-0.3f, 0.0f, 0.6f, 0.0f
             };

        float[] colors = {
              0.5f, 0.2f, 1.0f, 0.2f, 0.5f, 1.0f, 0.2f, 0.5f, 1.0f,
              0.2f, 0.5f, 1.0f, 0.5f, 0.2f, 1.0f, 0.5f, 0.2f, 1.0f,
              0.5f, 0.2f, 1.0f, 0.2f, 0.5f, 1.0f, 0.2f, 0.5f, 1.0f,
              0.2f, 0.5f, 1.0f, 0.5f, 0.2f, 1.0f, 0.5f, 0.2f, 1.0f,
              0.2f, 0.5f, 1.0f, 0.5f, 0.2f, 1.0f, 0.5f, 0.2f, 1.0f,
              0.5f, 0.2f, 1.0f, 0.2f, 0.5f, 1.0f, 0.2f, 0.5f, 1.0f,
              0.2f, 0.5f, 1.0f, 0.5f, 0.2f, 1.0f, 0.5f, 0.2f, 1.0f,
              0.5f, 0.2f, 1.0f, 0.2f, 0.5f, 1.0f, 0.2f, 0.5f, 1.0f
             };

        TriangleArray geometry=new TriangleArray(
              24,TriangleArray.COORDINATES | TriangleArray.COLOR_3 );

        geometry.setCoordinates( 0,vertexs );          //頂点座標の設定
        geometry.setColors( 0,colors );                //頂点の色の設定

        Shape3D shape=new Shape3D( geometry );

        root.addChild( transform );
        root.addChild( shape );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );

    }

    public static void main( String[] args ) {
        new MainFrame( new Daia(), 512, 512 );
    }
}
```

(3)ダイヤを縦長にのばした図形のワイヤーフレーム仕様

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class Daia2 extends Applet {

      public Daia2() {

        Canvas3D canvas = new Canvas3D(null);
        SimpleUniverse universe=new SimpleUniverse(canvas);

        setLayout(new BorderLayout());

        add( universe.getCanvas(),"Center");

        BranchGroup root=new BranchGroup();
        BoundingSphere bounds = new BoundingSphere();
```

```
        Transform3D translation=new Transform3D();

        translation.setTranslation(new Vector3d(1.0,4.0,2.0));

        TransformGroup transform=new TransformGroup(translation);

        float[] vertexs = {
              0.3f, 0.0f, 0.0f, 0.0f, 0.3f, 0.0f,-0.6f, 0.0f,
              0.0f, 0.0f, 0.3f,-0.3f, 0.0f, 0.0f, 0.0f,-0.6f, 0.0f,
             -0.3f, 0.0f, 0.0f, 0.0f, 0.0f,-0.3f, 0.0f,-0.6f, 0.0f,
              0.0f, 0.0f,-0.3f, 0.3f, 0.0f, 0.0f, 0.0f,-0.6f, 0.0f,
              0.0f, 0.0f, 0.3f, 0.3f, 0.0f, 0.0f, 0.0f, 0.6f, 0.0f,
             -0.3f, 0.0f, 0.0f, 0.0f, 0.0f, 0.3f, 0.0f, 0.6f, 0.0f,
              0.0f, 0.0f,-0.3f,-0.3f, 0.0f, 0.0f, 0.0f, 0.6f, 0.0f,
              0.3f, 0.0f, 0.0f, 0.0f, 0.0f,-0.3f, 0.0f, 0.6f, 0.0f
        };

        TriangleArray geometry=new TriangleArray(
             24,TriangleArray.COORDINATES | TriangleArray.COLOR_3);

        Background background = new Background(
                                        new Color3f( 0.4f, 0.6f, 0.8f ) );

        background.setApplicationBounds(bounds);
        root.addChild(background);

        PolygonAttributes attr=new PolygonAttributes();
        attr.setPolygonMode(PolygonAttributes.POLYGON_LINE);
        Appearance appearance=new Appearance();
        appearance.setPolygonAttributes(attr);

        geometry.setCoordinates(0,vertexs);             //頂点座標の設定

        Shape3D shape=new Shape3D(geometry,appearance);

        root.addChild( shape );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph(root);

    }

    public static void main( String[] args ) {
    new MainFrame(new Daia2(), 512, 512);
    }
}
```

(4)円柱を表示させるためのソースプログラム

```
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;


public class Cylinder1 extends Applet {

    public Cylinder1 (){

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );
        add( universe.getCanvas(), "Center" );

        BranchGroup root = new BranchGroup();

        TransformGroup objtrans = new TransformGroup();

        Transform3D trans = new Transform3D();
        trans.setScale( 0.4 );
```

```
        objtrans.setTransform( trans );
        root.addChild( objtrans );

        BoundingSphere bounds = new BoundingSphere();

        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

        //シリンダーを作成
        Cylinder Cylinder = new Cylinder();
        objtrans.addChild( Cylinder );

        // 背景を設定する

        TextureLoader bgTexture = new TextureLoader(
                                    "../images/sora.gif", this );
        Background bg = new Background( bgTexture.getImage() );
        bg.setApplicationBounds( bounds );
        root.addChild( bg );

        // 回転させるノードの作成
        MouseRotate behavior1 = new MouseRotate( objtrans );
        objtrans.addChild( behavior1 );
        behavior1.setSchedulingBounds( bounds );

        // ズームさせるノードの作成
        MouseZoom behavior2 = new MouseZoom( objtrans );
        objtrans.addChild(behavior2);
        behavior2.setSchedulingBounds( bounds );

        //2 つのライトでそれを照らす
        Color3f lColor1 = new Color3f( 0.7f, 0.7f, 0.7f );//図形の色
        Color3f lColor2 = new Color3f( 0.9f, 0.9f, 0.9f );//右側のライトの色
        Vector3f lDir1  = new Vector3f(-1.0f, -1.0f, -1.0f);//ライトの強さ
        Vector3f lDir2  = new Vector3f(-1.0f, -1.0f, -1.0f);//影の強さ

        DirectionalLight lgt1 = new DirectionalLight( lColor1, lDir1 );
        DirectionalLight lgt2 = new DirectionalLight( lColor2, lDir2 );
        lgt1.setInfluencingBounds( bounds );
        lgt2.setInfluencingBounds( bounds );
        objtrans.addChild( lgt1 );
        objtrans.addChild( lgt2 );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );
    }

    public static void main( String[] args ) {
        new MainFrame( new Cylinder1(), 512, 512 );
  }
}
```

(5)円柱を表示させるためのプログラムソース色つきライト

```
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;


public class Cylinder4 extends Applet {

    public Cylinder4 (){

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );
        add( universe.getCanvas(), "Center" );
```

```
        BranchGroup root = new BranchGroup();

        TransformGroup objtrans = new TransformGroup();

        Transform3D trans = new Transform3D();
        trans.setScale( 0.4 );
        objtrans.setTransform( trans );
        root.addChild( objtrans );

        BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);

        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

        Appearance ap = new Appearance();
        Color3f aColor   = new Color3f(7.1f, 7.1f, 7.1f);
        Color3f eColor   = new Color3f(5.0f, 5.0f, 5.0f);
        Color3f dColor   = new Color3f(6.8f, 6.8f, 7.8f);
        Color3f sColor   = new Color3f(5.0f, 5.0f, 5.0f);

        Material m = new Material(aColor, eColor, dColor, sColor, 80.0f);
        m.setLightingEnable(true);
        ap.setMaterial(m);

        //シリンダーを作成
        Cylinder Cylinder = new Cylinder();
        objtrans.addChild( Cylinder );

        // 回転させるノードの作成
        MouseRotate behavior1 = new MouseRotate( objtrans );
        objtrans.addChild( behavior1 );
        behavior1.setSchedulingBounds( bounds );

        // ズームさせるノードの作成
        MouseZoom behavior2 = new MouseZoom( objtrans );
        objtrans.addChild(behavior2);
        behavior2.setSchedulingBounds( bounds );

        //2 つのライトでそれを照らす
        Color3f lColor1 = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
        Color3f lColor2 = new Color3f( 0.7f, 0.0f, 0.0f );//右側のライトの色
        Vector3f lDir1   = new Vector3f(-1.0f, -1.0f, -1.0f);//ライトの強さ
        Vector3f lDir2   = new Vector3f( 0.0f,   0.0f, -1.0f);//影の強さ

        DirectionalLight lgt1 = new DirectionalLight( lColor1, lDir1 );
        DirectionalLight lgt2 = new DirectionalLight( lColor2, lDir2 );
        lgt1.setInfluencingBounds( bounds );
        lgt2.setInfluencingBounds( bounds );
        objtrans.addChild( lgt1 );
        objtrans.addChild( lgt2 );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );
    }

    public static void main( String[] args ) {
        new MainFrame( new Cylinder4(), 512, 512 );
  }
}
```

(6)円柱を表示させるためのプログラムソース色つきライト

```
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;


public class Cylinder4 extends Applet {

    public Cylinder4 (){
```

```
        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );
        add( universe.getCanvas(), "Center" );

        BranchGroup root = new BranchGroup();

        TransformGroup objtrans = new TransformGroup();

        Transform3D trans = new Transform3D();
        trans.setScale( 0.4 );
        objtrans.setTransform( trans );
        root.addChild( objtrans );

        BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);

        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

        Appearance ap = new Appearance();
        Color3f aColor   = new Color3f(7.1f, 7.1f, 7.1f);
        Color3f eColor   = new Color3f(5.0f, 5.0f, 5.0f);
        Color3f dColor   = new Color3f(6.8f, 6.8f, 7.8f);
        Color3f sColor   = new Color3f(5.0f, 5.0f, 5.0f);

        Material m = new Material(aColor, eColor, dColor, sColor, 80.0f);
        m.setLightingEnable(true);
        ap.setMaterial(m);

        //シリンダーを作成
        Cylinder Cylinder = new Cylinder();
        objtrans.addChild( Cylinder );

        // 回転させるノードの作成
        MouseRotate behavior1 = new MouseRotate( objtrans );
        objtrans.addChild( behavior1 );
        behavior1.setSchedulingBounds( bounds );

        // ズームさせるノードの作成
        MouseZoom behavior2 = new MouseZoom( objtrans );
        objtrans.addChild(behavior2);
        behavior2.setSchedulingBounds( bounds );

        //2 つのライトでそれを照らす
        Color3f lColor1 = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
        Color3f lColor2 = new Color3f( 0.7f, 0.0f, 0.0f );//右側のライトの色
        Vector3f lDir1   = new Vector3f(-1.0f, -1.0f, -1.0f);//ライトの強さ
        Vector3f lDir2   = new Vector3f( 0.0f,   0.0f, -1.0f);//影の強さ

        DirectionalLight lgt1 = new DirectionalLight( lColor1, lDir1 );
        DirectionalLight lgt2 = new DirectionalLight( lColor2, lDir2 );
        lgt1.setInfluencingBounds( bounds );
        lgt2.setInfluencingBounds( bounds );
        objtrans.addChild( lgt1 );
        objtrans.addChild( lgt2 );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );
    }

    public static void main( String[] args ) {
        new MainFrame( new Cylinder4(), 512, 512 );
  }
}
```

(7) 背景ライト付きの球を表示するソースプログラム

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
```

```
import javax.media.j3d.TransparencyAttributes;
import javax.media.j3d.BranchGroup;

public class Orb extends Applet {

    public Orb() {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );

        add( universe.getCanvas(),"Center" );

        BranchGroup root=new BranchGroup();

        Background background = new Background(
                                    new Color3f( 0.4f, 0.6f, 0.8f ) );

        BoundingSphere bounds = new BoundingSphere(
                                    new Point3d( 0.0,0.0,0.0 ),100.0 );

        background.setApplicationBounds( bounds );
        root.addChild( background );

        DirectionalLight light = new DirectionalLight();
        light.setDirection( new Vector3f( -0.3f, -0.5f, -1.0f ) );
        light.setInfluencingBounds( bounds );
        root.addChild( light );

        Primitive primitive = new Sphere( 0.4f );
        root.addChild( primitive );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );

    }

    public static void main( String[] args ) {
        new MainFrame( new Orb(), 512, 512 );
    }
}
```

(8)背景ライト付きの球を緑に着色したソースプログラム

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import javax.media.j3d.TransparencyAttributes;
import javax.media.j3d.BranchGroup;

public class Orb2 extends Applet {

    public Orb2() {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );

        add( universe.getCanvas(),"Center" );

        BranchGroup root=new BranchGroup();

        Background background = new Background(
                                    new Color3f( 0.4f, 0.6f, 0.8f ) );

        BoundingSphere bounds = new BoundingSphere(
                                    new Point3d( 0.0,0.0,0.0 ),100.0 );

        background.setApplicationBounds( bounds );
        root.addChild( background );
```

```
        DirectionalLight light = new DirectionalLight();
        light.setDirection( new Vector3f( -0.3f, -0.5f, -1.0f ) );
        light.setInfluencingBounds( bounds );
        root.addChild( light );

        Appearance appearance = new Appearance();

        Material mt = new Material(
                                new Color3f( 0.0f, 0.4f, 0.0f ),
                                new Color3f( 0.0f, 0.6f, 0.0f ),
                                new Color3f( 0.0f, 0.8f, 0.0f ),
                                new Color3f( 0.0f, 1.0f, 0.0f ),
                                2.0f );
        appearance.setMaterial( mt );

        Primitive primitive = new Sphere( 0.4f );
        primitive.setAppearance( appearance );
        root.addChild( primitive );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );

    }

    public static void main( String[] args ) {
        new MainFrame( new Orb2(), 512, 512 );
    }
}
```

(9)球のワイヤフレームを表示させるソースプログラム

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import javax.media.j3d.TransparencyAttributes;
import javax.media.j3d.BranchGroup;

public class Orb3 extends Applet {

    public Orb3() {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );

        add( universe.getCanvas(),"Center" );

        BranchGroup root = new BranchGroup();

        Background background = new Background(
                                    new Color3f( 0.4f, 0.6f, 0.8f ) );
        BoundingSphere bounds = new BoundingSphere(
                                    new Point3d( 0.0,0.0,0.0 ),100.0 );

        background.setApplicationBounds( bounds );
        root.addChild( background );

        PolygonAttributes attr = new PolygonAttributes();
        attr.setPolygonMode( PolygonAttributes.POLYGON_LINE );  // ワイヤーフレームで表示

        Appearance appearance = new Appearance();
        appearance.setPolygonAttributes( attr );

        Primitive primitive = new Sphere( 0.4f,appearance );

        root.addChild(primitive);

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph(root);
```

```
    }

    public static void main( String[] args ) {
       new MainFrame( new Orb3(), 512, 512 );
    }
}
```

(10)半透明の物体を表示させるソースプログラム

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class Orb4 extends Applet {

    public Orb4() {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe=new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );

        add( universe.getCanvas(),"Center" );

        BranchGroup root = new BranchGroup();

        Background background = new Background(
                                    new Color3f(0.4f,0.6f,0.8f) );
        BoundingSphere bounds = new BoundingSphere();

        background.setApplicationBounds( bounds );
        root.addChild( background );

        DirectionalLight light = new DirectionalLight();
        light.setDirection( new Vector3f( -0.3f, -0.5f, -1.0f ) );
        light.setInfluencingBounds( bounds );
        root.addChild( light );

        Primitive primitive=new Sphere( 0.5f );

        root.addChild( primitive );

        Appearance ap=new Appearance();

        TransparencyAttributes ta = new TransparencyAttributes(
                            TransparencyAttributes.BLENDED, 0.5f );

        ap.setTransparencyAttributes( ta );
        primitive.setAppearance( ap );

        //中心の物体

        Primitive primitive2 = new Sphere( 0.2f );

        root.addChild( primitive2 );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );

    }

    public static void main( String[] args ) {
        new MainFrame( new Orb4(), 512, 512 );
    }
}
```

(11)半透明の物体を表示させるソースプログラムその2 ワイヤーフレーム

```
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import java.applet.Applet;
import java.awt.BorderLayout;
```

```
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class Orb5 extends Applet {

    public Orb5() {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe=new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );

        add( universe.getCanvas(),"Center" );

        BranchGroup root = new BranchGroup();

        Background background = new Background(
                                    new Color3f(0.4f,0.6f,0.8f) );
        BoundingSphere bounds = new BoundingSphere();

        background.setApplicationBounds( bounds );
        root.addChild( background );

        DirectionalLight light = new DirectionalLight();
        light.setDirection( new Vector3f( -0.3f, -0.5f, -1.0f ) );
        light.setInfluencingBounds( bounds );
        root.addChild( light );

        Primitive primitive=new Sphere( 0.5f );

        root.addChild( primitive );

        Appearance ap=new Appearance();

        TransparencyAttributes ta = new TransparencyAttributes(
                            TransparencyAttributes.BLENDED, 0.5f );

        ap.setTransparencyAttributes( ta );
        primitive.setAppearance( ap );

        //中心の物体
        PolygonAttributes attr = new PolygonAttributes();
        attr.setPolygonMode( PolygonAttributes.POLYGON_LINE );

        Appearance appearance = new Appearance();
        appearance.setPolygonAttributes( attr );

        Primitive primitive2 = new Sphere( 0.2f,appearance );

        root.addChild( primitive2 );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );

    }

    public static void main( String[] args ) {
        new MainFrame( new Orb5(), 512, 512 );
    }
}
```

(12)円柱を表示させるためのプログラムソース NISE缶詰'

```
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
```

```
import javax.vecmath.*;

public class Cylinder2 extends Applet {

    public Cylinder2 ( String[] args ) {

        Canvas3D canvas = new Canvas3D( null );
        SimpleUniverse universe = new SimpleUniverse( canvas );

        setLayout( new BorderLayout() );
        add( universe.getCanvas(), "Center" );

        BranchGroup root = new BranchGroup();

        TransformGroup objtrans = new TransformGroup();

        Transform3D trans = new Transform3D();
        trans.setScale( 0.4 );
        objtrans.setTransform( trans );
        root.addChild( objtrans );

        BoundingSphere bounds = new BoundingSphere();

        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        objtrans.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

        Appearance ap = new Appearance();
        Color3f aColor   = new Color3f(0.1f, 0.1f, 0.1f);
        Color3f eColor   = new Color3f(0.0f, 0.0f, 0.0f);
        Color3f dColor   = new Color3f(0.8f, 0.8f, 0.8f);
        Color3f sColor   = new Color3f(1.0f, 1.0f, 1.0f);

        Material m = new Material(aColor, eColor, dColor, sColor, 80.0f);
        m.setLightingEnable(true);
        ap.setMaterial(m);

        //シリンダーを生成する
        Cylinder CylinderObj = new Cylinder(1.0f, 2.0f,
                    Cylinder.GENERATE_TEXTURE_COORDS |
                    Cylinder.GENERATE_NORMALS, ap);
        // シーングラフを作り出す
        objtrans.addChild( CylinderObj );

        if ( args.length > 0 ) {
            TextureLoader tex = new TextureLoader(
                                args[0], new String( "RGB" ), this );
        if ( tex != null )
                                ap.setTexture(tex.getTexture());
        } else {
            TextureLoader tex = new TextureLoader(
                                    new String("../images/title.gif"),
                                    new String("RGB"), this);
        if (tex != null)
            ap.setTexture(tex.getTexture());
        }

        // 背景を設定する

        Background bg
           = new Background( new Color3f( 0.4f, 0.6f, 0.8f ) );
        bg.setApplicationBounds( bounds );
        root.addChild( bg );

        // 回転させるノードの作成
        MouseRotate behavior1 = new MouseRotate( objtrans );
        objtrans.addChild( behavior1 );
        behavior1.setSchedulingBounds( bounds );

        // ズームさせるノードの作成
        MouseZoom behavior2 = new MouseZoom( objtrans );
        objtrans.addChild(behavior2);
        behavior2.setSchedulingBounds( bounds );

        //2 つのライトでそれを照らす
        Color3f lColor1 = new Color3f( 0.7f, 0.7f, 0.7f );//図形の色
        Color3f lColor2 = new Color3f( 0.9f, 0.9f, 0.9f );//右側のライトの色
        Vector3f lDir1   = new Vector3f(-1.0f, -1.0f, -1.0f);//ライトの強さ
        Vector3f lDir2   = new Vector3f(-1.0f, -1.0f, -1.0f);//影の強さ
```

```
        DirectionalLight lgt1 = new DirectionalLight( lColor1, lDir1 );
        DirectionalLight lgt2 = new DirectionalLight( lColor2, lDir2 );
        lgt1.setInfluencingBounds( bounds );
        lgt2.setInfluencingBounds( bounds );
        objtrans.addChild( lgt1 );
        objtrans.addChild( lgt2 );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );
    }

    public static void main( String[] args ) {
        new MainFrame( new Cylinder2( args ), 512, 512 );
    }
}
```

(13)テクスチャを貼った球

```
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import com.sun.j3d.utils.applet.MainFrame;
import java.awt.*;

public class Ease3 extends Applet {

    Color3f bg1color = new Color3f( 0.0f,0.0f,0.3f );
    Canvas3D canvas = new Canvas3D( null );
    SimpleUniverse universe = new SimpleUniverse( canvas );
    BorderLayout bl = new BorderLayout();
    Background bg = new Background( bg1color );
    BranchGroup root = new BranchGroup();

    Point3d point = new Point3d( 0.0,0.0,0.0 );
    BoundingSphere bounds = new BoundingSphere( point,200.0 );

    TransformGroup transform = new TransformGroup();

    Primitive primitive = new Sphere( 0.5f, Sphere.GENERATE_NORMALS |
                                      Sphere.GENERATE_TEXTURE_COORDS, 45 );

    Color3f lColorA = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
    Color3f lColorB = new Color3f( -0.7f, 0.0f, 0.0f );//右側のライトの色
    Vector3f lDirA  = new Vector3f( -8.0f, -8.0f, -8.0f );//ライトの強さ
    Vector3f lDirB  = new Vector3f( 0.0f, 0.0f, 0.0f );//影の強さ
    DirectionalLight lgtA = new DirectionalLight( lColorA, lDirA );
    DirectionalLight lgtB = new DirectionalLight( lColorB, lDirB );

    MouseRotate behaviorA = new MouseRotate( transform );
    MouseTranslate behaviorB = new MouseTranslate( transform );
    MouseZoom behaviorC = new MouseZoom( transform );


    public Ease3( String[] args ) {

        setLayout( bl );
        add( universe.getCanvas(),"Center" );


        bg.setApplicationBounds( bounds );
        root.addChild( bg );

        transform.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        transform.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

        root.addChild( transform );
        transform.addChild( primitive );
```

```
        Appearance ap = primitive.getAppearance();

                if ( args.length > 0 ) {
            TextureLoader tex = new TextureLoader( args[0],
                                                   new String( "RGB" ), this );
        if ( tex != null )ap.setTexture( tex.getTexture() );
        }

        else {
            TextureLoader tex = new TextureLoader( new String
                                                   ( "../images/earth.jpg" ),
                                                   new String( "RGB" ), this    );
        if ( tex != null )ap.setTexture( tex.getTexture() );
        }

        //2 つのライトでそれを照らす
        lgtA.setInfluencingBounds( bounds );
        lgtB.setInfluencingBounds( bounds );
        root.addChild( lgtA );
        root.addChild( lgtB );

        transform.addChild( behaviorA );
        behaviorA.setSchedulingBounds( bounds );

        transform.addChild( behaviorB );
        behaviorB.setSchedulingBounds( bounds );

        transform.addChild( behaviorC );
        behaviorC.setSchedulingBounds( bounds );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root );

    }

  public static void main( String[] args ) {
    new MainFrame( new Ease3( args ),512,512 );
  }
}
```

(14)テクスチャを貼って作った球が回転するプログラム背景変更機能付き

```
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import com.sun.j3d.utils.applet.MainFrame;
import java.awt.*;

public class Ease extends Applet implements ItemListener{

    public static float bg0R = 50.0f;
    public static float bg0G = 50.0f;
    public static float bg0B = 50.0f;

    public static float bg1R = 50.0f;
    public static float bg1G = 50.0f;
    public static float bg1B = 50.0f;

    public static float bg2R = 0.0f;
    public static float bg2G = 0.0f;
    public static float bg2B = 0.0f;

    public static float bg3R = 50.0f;
    public static float bg3G = 0.0f;
    public static float bg3B = 0.0f;
```

```
public static float bg4R = 0.0f;
public static float bg4G = 50.0f;
public static float bg4B = 0.0f;

public static float bg5R = 0.0f;
public static float bg5G = 0.0f;
public static float bg5B = 50.0f;

CheckboxGroup Box1 = new CheckboxGroup();

Checkbox box1 = new Checkbox( "白",Box1,true );
Checkbox box2 = new Checkbox( "黒",Box1,false );
Checkbox box3 = new Checkbox( "赤",Box1,false );
Checkbox box4 = new Checkbox( "緑",Box1,false );
Checkbox box5 = new Checkbox( "青",Box1,false );

Panel p = new Panel();//ボタンを配置するためのパネルの設定

Color3f bg1color = new Color3f( bg1R,bg1G,bg1B );
Color3f bg2color = new Color3f( bg2R,bg2G,bg3B );
Color3f bg3color = new Color3f( bg3R,bg3G,bg3B );
Color3f bg4color = new Color3f( bg4R,bg4G,bg4B );
Color3f bg5color = new Color3f( bg5R,bg5G,bg5B );

Canvas3D canvas = new Canvas3D( null );
SimpleUniverse universe = new SimpleUniverse( canvas );
BorderLayout bl = new BorderLayout();
GridLayout gl = new GridLayout(5,1);

BranchGroup root1 = new BranchGroup();
BranchGroup root2 = new BranchGroup();
BranchGroup root3 = new BranchGroup();
BranchGroup root4 = new BranchGroup();
BranchGroup root5 = new BranchGroup();

Point3d point = new Point3d( 0.0,0.0,0.0 );
BoundingSphere bounds = new BoundingSphere( point,200.0 );

Background bg1 = new Background( bg1color );
Background bg2 = new Background( bg2color );
Background bg3 = new Background( bg3color );
Background bg4 = new Background( bg4color );
Background bg5 = new Background( bg5color );

TransformGroup transform1 = new TransformGroup();
TransformGroup transform2 = new TransformGroup();
TransformGroup transform3 = new TransformGroup();
TransformGroup transform4 = new TransformGroup();
TransformGroup transform5 = new TransformGroup();

Alpha alpha = new Alpha(-1, Alpha.INCREASING_ENABLE,
                            0,2000,0,0);
Interpolator interpolator1 = new RotationInterpolator( alpha,transform1 );
Interpolator interpolator2 = new RotationInterpolator( alpha,transform2 );
Interpolator interpolator3 = new RotationInterpolator( alpha,transform3 );
Interpolator interpolator4 = new RotationInterpolator( alpha,transform4 );
Interpolator interpolator5 = new RotationInterpolator( alpha,transform5 );

Primitive primitive1 = new Sphere( 0.5f, Sphere.GENERATE_NORMALS |
                                   Sphere.GENERATE_TEXTURE_COORDS, 45 );
Primitive primitive2 = new Sphere( 0.5f, Sphere.GENERATE_NORMALS |
                                   Sphere.GENERATE_TEXTURE_COORDS, 45 );
Primitive primitive3 = new Sphere( 0.5f, Sphere.GENERATE_NORMALS |
                                   Sphere.GENERATE_TEXTURE_COORDS, 45 );
Primitive primitive4 = new Sphere( 0.5f, Sphere.GENERATE_NORMALS |
                                   Sphere.GENERATE_TEXTURE_COORDS, 45 );
Primitive primitive5 = new Sphere( 0.5f, Sphere.GENERATE_NORMALS |
                                   Sphere.GENERATE_TEXTURE_COORDS, 45 );

Color3f lColorA1 = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
Color3f lColorB1 = new Color3f( -0.7f, 0.0f, 0.0f );//右側のライトの色
Vector3f lDirA1  = new Vector3f( -8.0f, -8.0f, -8.0f );//ライトの強さ
Vector3f lDirB1  = new Vector3f( 0.0f, 0.0f, 0.0f );//影の強さ
DirectionalLight lgtA1 = new DirectionalLight( lColorA1, lDirA1 );
DirectionalLight lgtB1 = new DirectionalLight( lColorB1, lDirB1 );

Color3f lColorA2 = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
Color3f lColorB2 = new Color3f( -0.7f, 0.0f, 0.0f );//右側のライトの色
Vector3f lDirA2  = new Vector3f( -8.0f, -8.0f, -8.0f );//ライトの強さ
Vector3f lDirB2  = new Vector3f( 0.0f, 0.0f, 0.0f );//影の強さ
```

```
DirectionalLight lgtA2 = new DirectionalLight( lColorA2, lDirA2 );
DirectionalLight lgtB2 = new DirectionalLight( lColorB2, lDirB2 );

Color3f lColorA3 = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
Color3f lColorB3 = new Color3f( -0.7f, 0.0f, 0.0f );//右側のライトの色
Vector3f lDirA3    = new Vector3f( -8.0f, -8.0f, -8.0f );//ライトの強さ
Vector3f lDirB3    = new Vector3f( 0.0f, 0.0f, 0.0f );//影の強さ
DirectionalLight lgtA3 = new DirectionalLight( lColorA3, lDirA3 );
DirectionalLight lgtB3 = new DirectionalLight( lColorB3, lDirB3 );

Color3f lColorA4 = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
Color3f lColorB4 = new Color3f( -0.7f, 0.0f, 0.0f );//右側のライトの色
Vector3f lDirA4    = new Vector3f( -8.0f, -8.0f, -8.0f );//ライトの強さ
Vector3f lDirB4    = new Vector3f( 0.0f, 0.0f, 0.0f );//影の強さ
DirectionalLight lgtA4 = new DirectionalLight( lColorA4, lDirA4 );
DirectionalLight lgtB4 = new DirectionalLight( lColorB4, lDirB4 );

Color3f lColorA5 = new Color3f( 0.1f, 0.8f, 1.2f );//図形の色
Color3f lColorB5 = new Color3f( -0.7f, 0.0f, 0.0f );//右側のライトの色
Vector3f lDirA5    = new Vector3f( -8.0f, -8.0f, -8.0f );//ライトの強さ
Vector3f lDirB5    = new Vector3f( 0.0f, 0.0f, 0.0f );//影の強さ
DirectionalLight lgtA5 = new DirectionalLight( lColorA5, lDirA5 );
DirectionalLight lgtB5 = new DirectionalLight( lColorB5, lDirB5 );

public Ease( String[] args ) {

    setLayout( bl );
    add( universe.getCanvas(),"Center" );

    bg1.setApplicationBounds( bounds );
    root1.addChild( bg1 );
    bg2.setApplicationBounds( bounds );
    root2.addChild( bg2 );
    bg3.setApplicationBounds( bounds );
    root3.addChild( bg3 );
    bg4.setApplicationBounds( bounds );
    root4.addChild( bg4 );
    bg5.setApplicationBounds( bounds );
    root5.addChild( bg5 );

    transform1.addChild( interpolator1 );
    interpolator1.setSchedulingBounds( bounds );
    transform2.addChild( interpolator2 );
    interpolator2.setSchedulingBounds( bounds );
    transform3.addChild( interpolator3 );
    interpolator3.setSchedulingBounds( bounds );
    transform4.addChild( interpolator4 );
    interpolator4.setSchedulingBounds( bounds );
    transform5.addChild( interpolator5 );
    interpolator5.setSchedulingBounds( bounds );


    // パネルの設定とそこへのボタンの配置

    p.setLayout(gl);
    p.add( box1 );
    p.add( box2 );
    p.add( box3 );
    p.add( box4 );
    p.add( box5 );
    add("West",p);
    box1.addItemListener( this );
    box2.addItemListener( this );
    box3.addItemListener( this );
    box4.addItemListener( this );
    box5.addItemListener( this );

    root1.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
    root1.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
    root1.setCapability( BranchGroup.ALLOW_DETACH );

    root2.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
    root2.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
    root2.setCapability( BranchGroup.ALLOW_DETACH );

    root3.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
    root3.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
    root3.setCapability( BranchGroup.ALLOW_DETACH );

    root4.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
```

```
root4.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
root4.setCapability( BranchGroup.ALLOW_DETACH );

root5.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
root5.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
root5.setCapability( BranchGroup.ALLOW_DETACH );

transform1.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
transform1.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

root1.addChild( transform1 );
transform1.addChild( primitive1 );

Appearance ap1 = primitive1.getAppearance();

          if ( args.length > 0 ) {
     TextureLoader tex1 = new TextureLoader( args[0],
                                        new String( "RGB" ), this );
if ( tex1 != null )ap1.setTexture( tex1.getTexture() );
}

else {
     TextureLoader tex1 = new TextureLoader( new String
                                        ( "../images/earth.jpg" ),
                                        new String( "RGB" ), this    );
if ( tex1 != null )ap1.setTexture( tex1.getTexture() );
}

//2 つのライトでそれを照らす
lgtA1.setInfluencingBounds( bounds );
lgtB1.setInfluencingBounds( bounds );
root1.addChild( lgtA1 );
root1.addChild( lgtB1 );

transform2.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
transform2.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );


root2.addChild( transform2 );
transform2.addChild( primitive2 );

Appearance ap2 = primitive2.getAppearance();

if ( args.length > 0 ) {
     TextureLoader tex2 = new TextureLoader( args[0],
                                        new String( "RGB" ), this );
if ( tex2 != null )ap2.setTexture( tex2.getTexture() );
}

else {
     TextureLoader tex2 = new TextureLoader( new String
                                        ( "../images/earth.jpg" ),
                                        new String( "RGB" ), this );
if ( tex2 != null) ap2.setTexture( tex2.getTexture() );
}

//2 つのライトでそれを照らす
lgtA2.setInfluencingBounds( bounds );
lgtB2.setInfluencingBounds( bounds );
root2.addChild( lgtA2 );
root2.addChild( lgtB2 );

transform3.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
transform3.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

root3.addChild( transform3 );
transform3.addChild( primitive3 );

Appearance ap3 = primitive3.getAppearance();

if ( args.length > 0 ) {
     TextureLoader tex3 = new TextureLoader( args[0],
                                        new String( "RGB" ), this );
if ( tex3 != null )ap3.setTexture( tex3.getTexture() );
}

else {
     TextureLoader tex3 = new TextureLoader( new String
                                        ( "../images/earth.jpg" ),
                                        new String( "RGB" ), this );
```

```
        if ( tex3 != null )ap3.setTexture( tex3.getTexture() );
        }

        //2 つのライトでそれを照らす
        lgtA3.setInfluencingBounds( bounds );
        lgtB3.setInfluencingBounds( bounds );
        root3.addChild( lgtA3 );
        root3.addChild( lgtB3 );

        transform4.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        transform4.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

        root4.addChild( transform4 );
        transform4.addChild( primitive4 );

        Appearance ap4 = primitive4.getAppearance();

        if ( args.length > 0 ) {
            TextureLoader tex4 = new TextureLoader( args[0],
                                                    new String( "RGB" ), this );
        if ( tex4 != null )ap4.setTexture( tex4.getTexture() );
        }

        else {
            TextureLoader tex4 = new TextureLoader( new String
                                                    ( "../images/earth.jpg" ),
                                                    new String( "RGB" ), this );
        if ( tex4 != null )ap4.setTexture( tex4.getTexture() );
        }

        //2 つのライトでそれを照らす
        lgtA4.setInfluencingBounds( bounds );
        lgtB4.setInfluencingBounds( bounds );
        root4.addChild( lgtA4 );
        root4.addChild( lgtB4 );

        transform5.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        transform5.setCapability( TransformGroup.ALLOW_TRANSFORM_READ );

        root5.addChild( transform5 );
        transform5.addChild( primitive5 );

        Appearance ap5 = primitive5.getAppearance();

        if ( args.length > 0 ) {
            TextureLoader tex5 = new TextureLoader( args[0],
                                                    new String( "RGB" ), this );
        if ( tex5 != null )ap5.setTexture( tex5.getTexture() );
        }

        else {
            TextureLoader tex5 = new TextureLoader( new String
                                                    ( "../images/earth.jpg" ),
                                                    new String( "RGB" ), this );
        if ( tex5 != null )ap5.setTexture( tex5.getTexture() );
        }

        //2 つのライトでそれを照らす
        lgtA5.setInfluencingBounds( bounds );
        lgtB5.setInfluencingBounds( bounds );
        root5.addChild( lgtA5 );
        root5.addChild( lgtB5 );

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( root1 );

    }

    // ボタン操作の処理
    public void itemStateChanged( ItemEvent e ) {

        if( box1.getState() ) {
            root2.detach();
            root3.detach();
            root4.detach();
            root5.detach();

            universe.addBranchGraph( root1 );
        }
```

```
            if( box2.getState() ) {
                root1.detach();
                root3.detach();
                root4.detach();
                root5.detach();

                universe.addBranchGraph( root2 );
            }

            if( box3.getState() ) {
                root1.detach();
                root2.detach();
                root4.detach();
                root5.detach();

                universe.addBranchGraph( root3 );
            }

            if( box4.getState() ) {
                root1.detach();
                root2.detach();
                root3.detach();
                root5.detach();

                universe.addBranchGraph( root4 );
            }

            if( box5.getState() ) {
                root1.detach();
                root2.detach();
                root3.detach();
                root4.detach();

                universe.addBranchGraph( root5 );
            }

        }

       public static void main( String[] args ) {
         new MainFrame( new Ease( args ), 512, 512 );
       }
}
```

(15)物体を切り替えるソースプログラム

```
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;

import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

import java.applet.*;
import java.awt.*;

public class Cube2 extends Applet implements ItemListener {

      CheckboxGroup Bor = new CheckboxGroup();
      Canvas3D canvas = new Canvas3D( null );
      SimpleUniverse universe = new SimpleUniverse( canvas );
      BorderLayout bl = new BorderLayout();
      Panel p = new Panel();
      Vector3d ve = new Vector3d( 1.0,4.0,2.0 );

      Checkbox boxA = new Checkbox( "ColorCube(0.2)",Bor,true );
      Checkbox boxB = new Checkbox( "ColorCube(0.3)",Bor,false );
      Checkbox boxC = new Checkbox( "ColorCube(0.4)",Bor,false );

      TransformGroup transform1 = new TransformGroup();
      TransformGroup transform2 = new TransformGroup();
      TransformGroup transform3 = new TransformGroup();
```

```
Transform3D translation1 = new Transform3D();
Transform3D translation2 = new Transform3D();
Transform3D translation3 = new Transform3D();

BranchGroup root1 = new BranchGroup();
BranchGroup root2 = new BranchGroup();
BranchGroup root3 = new BranchGroup();

Shape3D shape1 = new ColorCube( 0.1 );
Shape3D shape2 = new ColorCube( 0.3 );
Shape3D shape3 = new ColorCube( 0.5 );

public Cube2() {

    root1.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
    root1.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
    root1.setCapability( BranchGroup.ALLOW_DETACH );

    root2.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
    root2.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
    root2.setCapability( BranchGroup.ALLOW_DETACH );

    root3.setCapability( BranchGroup.ALLOW_CHILDREN_EXTEND );
    root3.setCapability( BranchGroup.ALLOW_CHILDREN_WRITE );
    root3.setCapability( BranchGroup.ALLOW_DETACH );

    setLayout( bl );
    add( universe.getCanvas(),"Center" );

    p.add( boxA );
    p.add( boxB );
    p.add( boxC );
    add( "North",p );

    boxA.addItemListener(this);
    boxB.addItemListener(this);
    boxC.addItemListener(this);

    translation1.setTranslation( ve );
    translation2.setTranslation( ve );
    translation3.setTranslation( ve );

    root1.addChild( transform1 );
    root2.addChild( transform2 );
    root3.addChild( transform3 );

    root1.addChild( shape1 );
    root2.addChild( shape2 );
    root3.addChild( shape3 );

    universe.getViewingPlatform().setNominalViewingTransform();
    universe.addBranchGraph( root1 );

}

public void itemStateChanged( ItemEvent e ) {

    if( boxA.getState() ) {
        root2.detach();
        root3.detach();
        universe.addBranchGraph( root1 );
    }

    if( boxB.getState() ) {
        root1.detach();
        root3.detach();
        universe.addBranchGraph( root2 );
    }

    if( boxC.getState() ) {
        root1.detach();
        root2.detach();
        universe.addBranchGraph( root3 );
    }

}

public static void main( String[] args ) {
    new MainFrame( new Cube2(), 512, 512 );
```

```
    }
}
```

(16)クリエイト型の顔表示プログラムソース (本体)

```
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.image.TextureLoader;
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.*;

public class FaceViewer extends Applet implements ItemListener{
                                                    //implements ItemListener は，ラジオボタンを使用する時に必要となる

    //ボタン部品の作成
      CheckboxGroup BorW = new CheckboxGroup();
      CheckboxGroup BorW2 = new CheckboxGroup();
      Checkbox boxB,boxW,boxY,boxN;
      Checkbox box1,box2,box3;
            Canvas3D canvas=new Canvas3D(null);
            SimpleUniverse universe=new SimpleUniverse(canvas);

      BranchGroup root=new BranchGroup();
      BranchGroup bg1=new BranchGroup();
      BranchGroup bg2=new BranchGroup();
      BranchGroup bg3=new BranchGroup();
      BranchGroup branchFace1=new BranchGroup();

      TransformGroup transform = new TransformGroup();
      Transform3D trans = new Transform3D();
      Transform3D faceTrans = new Transform3D();

    // コンストラクタ
      public FaceViewer() {

    // Virtual World   --空間の構築--


            setLayout(new BorderLayout());
          add( universe.getCanvas(), "Center" );

            root.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
            root.setCapability(BranchGroup.ALLOW_DETACH);
            root.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

            branchFace1.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);       // オブジェクトの追加を許可
            branchFace1.setCapability(BranchGroup.ALLOW_DETACH);
            branchFace1.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);


            bg1.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
            bg1.setCapability(BranchGroup.ALLOW_DETACH);
            bg1.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

            bg2.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
            bg2.setCapability(BranchGroup.ALLOW_DETACH);
            bg2.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

            bg3.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
            bg3.setCapability(BranchGroup.ALLOW_DETACH);
            bg3.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
```

```
        transform.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);          // オブジェクトの追加を許可
        transform.setCapability(BranchGroup.ALLOW_DETACH);
        transform.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);


    // Scene Graph

        BoundingSphere bounds = new BoundingSphere();
        bounds.setRadius( 10.0 );

     // 顔などを動かせるようにするための設定

        transform.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        transform.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        root.addChild(transform);

     // パネルの設定とそこへのボタンの配置
        Panel p = new Panel();          //ボタンを配置するためのパネルの設定
        p.setLayout(new GridLayout(4,1));

        p.add(boxY = new Checkbox("Yello",BorW,false));
        p.add(boxW = new Checkbox("Black",BorW,false));
        p.add(boxB = new Checkbox("White",BorW,false));
        p.add(boxN = new Checkbox("Nothing",BorW,true));
                                        //"表示文章",グループ名,
                                        // true=最初にチェックが入っている場所. false は入っていない

        add("West",p);
                                        //他に South East North Center  がある

        boxY.setForeground(Color.white);
        boxW.setForeground(Color.white);
        boxB.setForeground(Color.white);
        boxN.setForeground(Color.white);

        boxY.addItemListener(this);
        boxW.addItemListener(this);
        boxB.addItemListener(this);
        boxN.addItemListener(this);
        p.setBackground(Color.black);



        Panel q = new Panel();          //ボタンを配置するためのパネルの設定

        q.add(box1 = new Checkbox("BG1",BorW2,true));
        q.add(box2 = new Checkbox("BG2",BorW2,false));
        q.add(box3 = new Checkbox("BG3",BorW2,false));
                                        //"表示文章",グループ名,
                                        // true=最初にチェックが入っている場所. false は入っていない

        add("South",q);
                                        //他に South East North Center  がある

        box1.setForeground(Color.black);
        box2.setForeground(Color.black);
        box3.setForeground(Color.black);

        box1.addItemListener(this);
        box2.addItemListener(this);
        box3.addItemListener(this);
        q.setBackground(Color.white);


     // Background --背景の設定--
        Background bg = new Background(new Color3f(0.4f,0.6f,0.8f));
       bg.setApplicationBounds( bounds );
       bg1.addChild( bg );

        TextureLoader bgTexture = new TextureLoader("../images/sora.gif", this);
        Background background2 = new Background(bgTexture.getImage());
        background2.setApplicationBounds(bounds);
        bg2.addChild( background2 );

        TextureLoader bgTexture3 = new TextureLoader("../images/bgwinter.jpg", this);
        Background background3 = new Background(bgTexture3.getImage());
        background3.setApplicationBounds(bounds);
        bg3.addChild(background3);
```

```
// Light --光源の設定
      DirectionalLight light = new DirectionalLight();
      light.setDirection( new Vector3f( -0.3f, -0.5f, -1.0f ) );
      light.setInfluencingBounds( bounds );
      root.addChild( light );


//Face クラスの呼び出しとそれに関する細かな設定
      faceTrans.setTranslation(
                  new Vector3d( 0.0, 0.0, 0.0 ) );
      Transform3D rotation = new Transform3D();
      rotation.rotY( -Math.PI/5.0 );
      faceTrans.mul( rotation );


// マウスによって回転させるためのノードの作成
        MouseRotate behavior = new MouseRotate(transform);
        transform.addChild(behavior);
        behavior.setSchedulingBounds(bounds);

// マウスによって移動させるためのノードの作成
        MouseTranslate behavior2 = new MouseTranslate(transform);
        transform.addChild(behavior2);
        behavior2.setSchedulingBounds(bounds);

// マウスによってズームさせるためのノードの作成 3
        MouseZoom behavior3 = new MouseZoom(transform);
        transform.addChild(behavior3);
        behavior3.setSchedulingBounds(bounds);


        root.addChild(bg1);
      universe.addBranchGraph( root );


// Observer --視点の設定--
      ViewingPlatform viewingPlatform
         = universe.getViewingPlatform();
      viewingPlatform.setNominalViewingTransform();
 }


// ボタン操作の処理
  public void itemStateChanged( ItemEvent e )


        if(boxY.getState()){
                  Controler(1);
        }


        else if(boxW.getState()){
                  Controler(2);
        }


        else if(boxB.getState()){
                  Controler(3);
        }

        else if(boxN.getState()){
            root.detach();
            branchFace1.detach();
            universe.addBranchGraph( root );
        }


        if(box1.getState()){
            bg2.detach();
            bg3.detach();
            bg1.detach();
            root.addChild( bg1 );
        }

        else if(box2.getState()){
            bg3.detach();
            bg1.detach();
            bg2.detach();
```

```
                root.addChild( bg2 );
            }

            else if(box3.getState()){
                bg1.detach();
                bg2.detach();
                bg3.detach();
                root.addChild( bg3 );
            }

        }

        public void Controler(int co) {
                root.detach();
                branchFace1.detach();
               FaceSetting face = new FaceSetting( 0.5 , co );
               face.setTransform( faceTrans );
               branchFace1=new BranchGroup();
                branchFace1.addChild( face );
                transform.addChild(branchFace1);
                universe.addBranchGraph( root );
        }

    // 実行確認のためのメソッド
        public static void main( String[] args ) {

                        new MainFrame(new FaceViewer(), 400, 400);

        }
}
```

(17)セレクト型の顔表示プログラムソース (本体)

```
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.image.TextureLoader;
import java.awt.Frame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.*;

public class FaceViewer extends Applet implements ItemListener{
                                                    //implements ItemListener は，ラジオボタンを使用す
る時に必要となる

    //ボタン部品の作成
      CheckboxGroup BorW = new CheckboxGroup();
      CheckboxGroup BorW2 = new CheckboxGroup();
      Checkbox boxB,boxW,boxY,boxN;
      Checkbox box1,box2,box3;
            Canvas3D canvas=new Canvas3D(null);
            SimpleUniverse universe=new SimpleUniverse(canvas);

      BranchGroup root=new BranchGroup();
      BranchGroup bg1=new BranchGroup();
      BranchGroup bg2=new BranchGroup();
      BranchGroup bg3=new BranchGroup();

      BranchGroup branchFace1=new BranchGroup();
      BranchGroup branchFace2=new BranchGroup();
      BranchGroup branchFace3=new BranchGroup();

      TransformGroup transform = new TransformGroup();
      Transform3D trans = new Transform3D();

    // コンストラクタ
      public FaceViewer() {

    // Virtual World   --空間の構築--
```

```
        setLayout(new BorderLayout());
      add( universe.getCanvas(), "Center" );

        root.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
        root.setCapability(BranchGroup.ALLOW_DETACH);
        root.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

        branchFace1.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);       // オブジェクトの追加を許可
        branchFace1.setCapability(BranchGroup.ALLOW_DETACH);
        branchFace1.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

        branchFace2.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);       // オブジェクトの追加を許可
        branchFace2.setCapability(BranchGroup.ALLOW_DETACH);
        branchFace2.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

        branchFace3.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);       // オブジェクトの追加を許可
        branchFace3.setCapability(BranchGroup.ALLOW_DETACH);
        branchFace3.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

        bg1.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
        bg1.setCapability(BranchGroup.ALLOW_DETACH);
        bg1.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

        bg2.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
        bg2.setCapability(BranchGroup.ALLOW_DETACH);
        bg2.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

        bg3.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);      // オブジェクトの追加を許可
        bg3.setCapability(BranchGroup.ALLOW_DETACH);
        bg3.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);

        transform.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);          // オブジェクトの追加を許可
        transform.setCapability(BranchGroup.ALLOW_DETACH);
        transform.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);


    // Scene Graph

        BoundingSphere bounds = new BoundingSphere();
        bounds.setRadius( 10.0 );

     // 顔などを動かせるようにするための設定

        transform.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        transform.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        root.addChild(transform);

     // パネルの設定とそこへのボタンの配置
        Panel p = new Panel();          //ボタンを配置するためのパネルの設定

        p.setLayout(new GridLayout(4,1));

        p.add(boxY = new Checkbox("Yello",BorW,false));
        p.add(boxW = new Checkbox("Black",BorW,false));
        p.add(boxB = new Checkbox("White",BorW,false));
        p.add(boxN = new Checkbox("Nothing",BorW,true));
                                        //"表示文章",グループ名,
                                        // true=最初にチェックが入っている場所. false は入っていない

        add("West",p);
                                        //他に South East North Center　がある

        boxY.setForeground(Color.white);
        boxW.setForeground(Color.white);
        boxB.setForeground(Color.white);
        boxN.setForeground(Color.white);

        boxY.addItemListener(this);
        boxW.addItemListener(this);
        boxB.addItemListener(this);
        boxN.addItemListener(this);
```

```
        p.setBackground(Color.black);

        Panel q = new Panel();          //ボタンを配置するためのパネルの設定

        q.add(box1 = new Checkbox("BG1",BorW2,true));
        q.add(box2 = new Checkbox("BG2",BorW2,false));
        q.add(box3 = new Checkbox("BG3",BorW2,false));
                                        //"表示文章",グループ名,
                                        // true=最初にチェックが入っている場所. false は入っていない

        add("South",q);
                                        //他に South East North Center　がある

        box1.setForeground(Color.black);
        box2.setForeground(Color.black);
        box3.setForeground(Color.black);

        box1.addItemListener(this);
        box2.addItemListener(this);
        box3.addItemListener(this);
        q.setBackground(Color.white);


// Background --背景の設定--
        Background bg = new Background(new Color3f(0.4f,0.6f,0.8f));
      bg.setApplicationBounds( bounds );
      bg1.addChild( bg );

        TextureLoader bgTexture = new TextureLoader("../images/sora.gif", this);
        Background background2 = new Background(bgTexture.getImage());
        background2.setApplicationBounds(bounds);
        bg2.addChild( background2 );

        TextureLoader bgTexture3 = new TextureLoader("../images/bgwinter.jpg", this);
        Background background3 = new Background(bgTexture3.getImage());
        background3.setApplicationBounds(bounds);
        bg3.addChild(background3);

// Light --光源の設定
      DirectionalLight light = new DirectionalLight();
      light.setDirection( new Vector3f( -0.3f, -0.5f, -1.0f ) );
      light.setInfluencingBounds( bounds );
      root.addChild( light );


//Face クラスの呼び出しとそれに関する細かな設定
      FaceSetting face = new FaceSetting( 0.5 , 1 );
      Transform3D faceTrans = new Transform3D();
      faceTrans.setTranslation(
                  new Vector3d( 0.0, 0.0, 0.0 ) );
      Transform3D rotation = new Transform3D();
      rotation.rotY( -Math.PI/5.0 );
      faceTrans.mul( rotation );
      face.setTransform( faceTrans );

      FaceSetting face2 = new FaceSetting( 0.5 ,2 );
      face2.setTransform( faceTrans );

      FaceSetting face3 = new FaceSetting( 0.5 ,3 );
      face3.setTransform( faceTrans );


// マウスによって回転させるためのノードの作成
        MouseRotate behavior = new MouseRotate(transform);
        transform.addChild(behavior);
        behavior.setSchedulingBounds(bounds);

// マウスによって移動させるためのノードの作成
        MouseTranslate behavior2 = new MouseTranslate(transform);
        transform.addChild(behavior2);
        behavior2.setSchedulingBounds(bounds);

// マウスによってズームさせるためのノードの作成 3
        MouseZoom behavior3 = new MouseZoom(transform);
        transform.addChild(behavior3);
        behavior3.setSchedulingBounds(bounds);


        branchFace1.addChild( face );// 球 (インデックス 0)
```

```
            branchFace2.addChild( face2 );              // 色付き立方体 (インデックス 1)
            branchFace3.addChild( face3 );
            root.addChild(bg1);
          universe.addBranchGraph( root );


      // Observer --視点の設定--
            ViewingPlatform viewingPlatform
                = universe.getViewingPlatform();
            viewingPlatform.setNominalViewingTransform();
       }


      // ボタン操作の処理
       public void itemStateChanged( ItemEvent e )

            if(boxY.getState()){
            long startTime = System.currentTimeMillis();
                  branchFace3.detach();
                  branchFace2.detach();
                  branchFace1.detach();
                 transform.addChild(branchFace1);
             long stopTime = System.currentTimeMillis();
//             System.out.println("Connection time_Play" + "¥t" + (stopTime - startTime) + "¥t msecs");
             System.out.println((stopTime - startTime));
            }


            else if(boxW.getState()){
            long startTime = System.currentTimeMillis();
                  branchFace1.detach();
                  branchFace3.detach();
                  branchFace2.detach();
                 transform.addChild(branchFace2);
             long stopTime = System.currentTimeMillis();
//             System.out.println("Connection time_Play" + "¥t" + (stopTime - startTime) + "¥t msecs");
             System.out.println((stopTime - startTime));
            }


            else if(boxB.getState()){
            long startTime = System.currentTimeMillis();
                  branchFace2.detach();
                  branchFace1.detach();
                  branchFace3.detach();
                 transform.addChild(branchFace3);
             long stopTime = System.currentTimeMillis();
//             System.out.println("Connection time_Play" + "¥t" + (stopTime - startTime) + "¥t msecs");
             System.out.println((stopTime - startTime));
            }

            else if(boxN.getState()){
                  branchFace2.detach();
                  branchFace3.detach();
                  branchFace1.detach();
            }


            if(box1.getState()){
                  bg2.detach();
                  bg3.detach();
                  bg1.detach();
                  root.addChild( bg1 );
            }

            else if(box2.getState()){
                  bg3.detach();
                  bg1.detach();
                  bg2.detach();
                  root.addChild( bg2 );
            }

            else if(box3.getState()){
                  bg1.detach();
                  bg2.detach();
                  bg3.detach();
                  root.addChild( bg3 );
            }
```

```
    }

  // 実行確認のためのメソッド
    public static void main( String[] args ) {
          long startTime = System.currentTimeMillis();
                  new MainFrame(new FaceViewer(), 400, 400);
           long stopTime = System.currentTimeMillis();
//             System.out.println("Connection  time_Start" + "¥t" + (stopTime - startTime) + "¥t
msecs");
           System.out.println((stopTime - startTime));

    }

}
```

(18)顔表示プログラム，顔の部品の位置設定を行うクラスのソース (16.17 共用)

```
import javax.media.j3d.*;
import javax.vecmath.*;

public class FaceSetting extends TransformGroup {

  /** コンストラクタ */
    public FaceSetting( double faceRadius,int n ) {



       if(n==1){
          FaceBase                    base                    =                    new
FaceBase( faceRadius ,0.2f,0.2f,0.2f,0.0f,0.0f,0.0f,0.7f,0.5f,0.5f,0.7f,0.5f,0.5f);
          addChild( base );
       }

       if(n==2){
          FaceBase base = new FaceBase( faceRadius ,0.0f, 0.0f, 0.0f,0.3333f, 0.15294f, 0.0f,0.3333f,
0.15294f, 0.0f,0.3333f, 0.15294f, 0.0f);
          addChild( base );
       }

       if(n==3){
          FaceBase                    base                    =                    new
FaceBase( faceRadius ,0.4f,0.3412f,0.258823f,0.6f,0.5412f,0.458823f,0.8f,0.7412f,0.658823f,1.0f,0.941
2f,0.858823f);
          addChild( base );
       }

          FaceEye rEye = new FaceEye( faceRadius*0.2 );
          Transform3D rEyeTrans = new Transform3D();
          rEyeTrans.setTranslation(
               new Vector3d( faceRadius*-0.35, faceRadius*-0.12,
                             faceRadius*0.85 ) );
          rEye.setTransform( rEyeTrans );
          addChild( rEye );

          FaceEye lEye = new FaceEye( faceRadius*0.2 );
          Transform3D lEyeTrans = new Transform3D();
          lEyeTrans.setTranslation(
               new Vector3d( faceRadius*0.35, faceRadius*-0.12,
                             faceRadius*0.85 ) );
          lEye.setTransform( lEyeTrans );
          addChild( lEye );

          FaceNose nose = new FaceNose( faceRadius*0.3 );
          Transform3D noseOrigTrans = new Transform3D();
          nose.getTransform( noseOrigTrans );
          Transform3D noseTrans = new Transform3D();
          noseTrans.setTranslation(
               new Vector3d( 0.0, 0.0, 0.0 ) );
          noseTrans.mul( noseOrigTrans );
          nose.setTransform( noseTrans );
          addChild( nose );

          FaceCap cap = new FaceCap( faceRadius*0.5 );
          Transform3D capOrigTrans = new Transform3D();
          cap.getTransform( capOrigTrans );
          Transform3D capTrans = new Transform3D();
          capTrans.setTranslation(
               new Vector3d( 0.0, faceRadius*0.20, 0.0 ) );
          capTrans.mul( capOrigTrans );
```

```
        cap.setTransform( capTrans );
        addChild( cap );


        FaceTop top = new FaceTop( faceRadius*0.6 );
        Transform3D topOrigTrans = new Transform3D();
        top.getTransform( topOrigTrans );
        Transform3D topTrans = new Transform3D();
        topTrans.setTranslation(
            new Vector3d( 0.0, faceRadius*0.09, faceRadius*0.42) );
        topTrans.mul( topOrigTrans );
        top.setTransform( topTrans );
        addChild( top );

        FaceMouth mouth = new FaceMouth( faceRadius*0.4 );
        Transform3D mouthOrigTrans = new Transform3D();
        mouth.getTransform( mouthOrigTrans );
        Transform3D mouthTrans = new Transform3D();
        mouthTrans.setTranslation(
            new Vector3d( 0.0, faceRadius*-0.6,
                          faceRadius*0.71 ) );
        Transform3D mouthRotation = new Transform3D();
        mouthRotation.rotX( Math.PI/8.0 );
        mouthTrans.mul( mouthRotation );
        mouthTrans.mul( mouthOrigTrans );
        mouth.setTransform( mouthTrans );
        addChild( mouth );
    }
}
```

(19)顔表示プログラム，顔の輪郭をコンストラクトするクラスのソース (16.17 共用)

```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class FaceBase extends TransformGroup {

  /** コンストラクタ */
    public FaceBase( double scale ,float a1,float a2,float a3,float b1,float b2,float b3,float c1,float
c2,float c3,float d1,float d2,float d3) {

        Sphere sphere = new Sphere( (float)scale );
        addChild( sphere );

        Appearance ap = new Appearance();
        sphere.setAppearance( ap );

        Material mt = new Material(
                            new Color3f( a1, a2, a3 ),
                            new Color3f( b1, b2, b3 ),
                            new Color3f( c1, c2, c3 ),
                            new Color3f( d1, d2, d3 ),
                            2.0f );
        ap.setMaterial( mt );

        Transform3D transform = new Transform3D();
        transform.setScale( new Vector3d( 1.0, 1.0, 1.0 ) );
        setTransform( transform );
    }
}
```

(20)顔表示プログラム，顔の目をコンストラクトするクラスのソース (16.17 共用)

```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class FaceEye extends TransformGroup {

  /** コンストラクタ */
    public FaceEye( double scale ) {

        Sphere sphere = new Sphere( (float)scale );
        addChild( sphere );

        Appearance ap = new Appearance();
        sphere.setAppearance( ap );

        Material mt = new Material(
```

```
                        new Color3f( 0.2f, 0.2f, 0.2f ),
                        new Color3f( 0.0f, 0.0f, 0.0f ),
                        new Color3f( 0.6f, 0.6f, 0.6f ),
                        new Color3f( 1.0f, 1.0f, 1.0f ),
                        128.0f );
        ap.setMaterial( mt );

        TransparencyAttributes ta = new TransparencyAttributes(
                        TransparencyAttributes.BLENDED, 0.75f );
        ap.setTransparencyAttributes( ta );

        Sphere sphere2 = new Sphere( (float)scale*0.5f );
        addChild( sphere2 );

        Appearance ap2 = new Appearance();
        sphere2.setAppearance( ap2 );

        Material mt2 = new Material(
                        new Color3f( 0.1f, 0.1f, 0.1f ),
                        new Color3f( 0.0f, 0.0f, 0.0f ),
                        new Color3f( 0.0f, 0.0f, 0.0f ),
                        new Color3f( 1.0f, 1.0f, 1.0f ),
                        128.0f );
        ap2.setMaterial( mt2 );
    }
}
```

(21)顔表示プログラム，顔の口をコンストラクトするクラスのソース (16.17 共用)

```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class FaceMouth extends TransformGroup {

  /** コンストラクタ */
    public FaceMouth( double scale ) {

        Sphere sphere = new Sphere( (float)scale );
        addChild( sphere );

        Appearance ap = new Appearance();
        sphere.setAppearance( ap );

        Material mt = new Material(
                        new Color3f( 0.2f, 0.2f, 0.2f ),
                        new Color3f( 0.0f, 0.0f, 0.0f ),
                        new Color3f( 0.5f, 0.0f, 0.1f ),
                        new Color3f( 0.5f, 0.0f, 0.1f ),
                        2.0f );
        ap.setMaterial( mt );

        Transform3D transform = new Transform3D();
        transform.setScale( new Vector3d( 1.0, 0.6, 0.2 ) );
        setTransform( transform );
    }
}
```

(22)顔表示プログラム，顔の鼻をコンストラクトするクラスのソース (16.17 共用)

```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class FaceNose extends TransformGroup {

  /** コンストラクタ */
    public FaceNose( double scale ) {

        Sphere sphere = new Sphere( (float)scale );
        addChild( sphere );

        Appearance ap = new Appearance();
        sphere.setAppearance( ap );

        Material mt = new Material(
                        new Color3f( 0.2f, 0.2f, 0.2f ),
                        new Color3f( 0.0f, 0.0f, 0.0f ),
                        new Color3f( 0.7f, 0.5f, 0.5f ),
                        new Color3f( 0.7f, 0.5f, 0.5f ),
```

```
                                    2.0f );
            ap.setMaterial( mt );

            Transform3D transform = new Transform3D();
            transform.setScale( new Vector3d( 1.2, 1.0, 1.0 ) );
            setTransform( transform );
        }
}
```

(23)顔表示プログラム，帽子をコンストラクトするクラスのソース (16.17 共用)

```
import com.sun.j3d.utils.image.TextureLoader;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;


public class FaceCap extends TransformGroup {

  /** コンストラクタ */
    public FaceCap( double scale ) {


        Sphere sphere = new Sphere( (float)scale );
        addChild( sphere );

        Appearance ap = new Appearance();
        sphere.setAppearance( ap );



        Material mt = new Material(
                                    new Color3f( 7.0f, 7.0f, 7.0f ),
                                    new Color3f( 7.0f, 7.0f, 7.0f ),
                                    new Color3f( 7.0f, 7.0f, 7.0f ),
                                    new Color3f( 7.0f, 7.0f, 7.0f ),
                                    2.0f );
        ap.setMaterial( mt );

        Transform3D transform = new Transform3D();
        transform.setScale( new Vector3d( 2.0, 2.0, 2.0 ) );
        setTransform( transform );
    }
}
```

(24)顔表示プログラム，帽子のつばをコンストラクトするクラスのソース (16.17 共用)

```
import com.sun.j3d.utils.image.TextureLoader;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.geometry.Cylinder;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;


public class FaceTop extends TransformGroup {

  /** コンストラクタ */
    public FaceTop( double scale ) {
```

```
        Sphere sphere = new Sphere( (float)scale );
        addChild( sphere );

        Appearance ap = new Appearance();
        sphere.setAppearance( ap );


        Material mt = new Material(
                                new Color3f( 0.1f, 0.1f, 0.1f ),
                                new Color3f( 0.0f, 0.0f, 0.0f ),
                                new Color3f( 0.0f, 0.0f, 0.0f ),
                                new Color3f( 0.0f, 0.0f, 0.0f ),
                                2.0f );
        ap.setMaterial( mt );


        Transform3D transform = new Transform3D();
        transform.setScale( new Vector3d( 1.7, 0.24, 1.9 ) );
        setTransform( transform );
    }
}
```