バーチャルリアリティ

バーチャルリアリティ（仮想現実感）とは

- 人間がもついくつかの感覚器官に人工的に作り出した刺激を与えることで，実際には存在しない空間や物体を知覚させる．

バーチャルリアリティの構成要素

- 存在感（Presence）
- 会話性（Interactivity）
- 自律性（Autonomy）

現在のバーチャルリアリティ

- 視覚（リアルタイム３次元グラフィックス，立体視表示）
- 聴覚（ステレオ，ドルビーサラウンド，バーチャル音場）
- 触覚（フォースフィードバック，手触りディスプレイ）
- 前庭感覚（テーマパークのアトラクション，操縦訓練用シミュレータ）
- 味覚・嗅覚？

バーチャルリアリティ空間の記述

- VR Toolkit
- World Toolkit
- VRML
- Java 3D

VRML (Virtual Reality Modeling Language)

- バーチャルリアリティ空間を記述するための言語．形状データフォーマットの一種．
- 物体の特別な振る舞や現実空間とのインタフェース等を記述するために，JavaScript や Java などの別のプログラミング言語で VRML を制御するメカニズム（EAI）を用意している．

Java 3D

- プログラミング言語 Java の中にバーチャルリアリティ空間を記述するための「クラス」を VRML に沿って実装したもの．
- もとがプログラミング言語なので，物体の特別な振る舞や現実空間とのインタフェース等を容易に記述できる．

VRML という教材について

- コンピュータで立体形状を取り扱うには，形状をコンピュータのデータ（形状データ）として記述する必要がある．
- 形状データの書式の種類は無数にある（アプリケーション毎？）．デファクトスタンダード（事実上の標準）としては DXF などがあるが，VRML はより普遍的で理解しやすい．

VRML ファイルの書式

ヘッダ

VRML ファイルの先頭には、以下の１行を置きます。

```
#VRML V2.0 utf8
```

この後に、少なくともひとつの Shape ノードが含まれている必要があります。V2.0 は VRML の文法のバージョン、utf8 は使用している文字コードを示します（バージョン 1.0 の VRML ファイルは `#VRML V1.0 ascii` から始まりますが、これは 2.0 とは互換性がありません）。

VRML ファイルの書式

```
#VRML V2.0 utf8
ノード
ノード
```

1. 先頭に `#VRML V2.0 utf8` という１行を置く。
2. 以降にノードを列挙する。
3. １行目以外にある # より右側の部分は無視される（コメント）。

ノード

ノードは VRML の基本単位です。 VRML ファイルはいくつかのノードの集まりとして記述します。

ノードの書式

```
ノード名 {
        フィールド
        フィールド
        ..
}
```

1. ノードはノード名と、それに続く { ... } からなる。
2. { ... } 内にはいくつかのフィールドを置く。
3. ノード名は大文字で始まる（例：Shape）。
4. VRML ファイルには少なくとも１つの Shape ノードが必要。

フィールド

ノード名に続く { ... } 内には、ノードに関する詳細な情報を記述します。これをフィールドと言います。ノードの種類ごとにいくつかのフィールドが定義されています。

フィールドの書式

```
フィールド名 フィールド値 フィールド値 ..
```

1. フィールド名は小文字で始まる（例：geometry）。
2. 各フィールドにはフィールド値（数値・文字列またはノード）を設定する。
3. フィールドを省略するとデフォルト値が採用される。

VRML ファイルの例（陰影付けされない球）

```
#VRML V2.0 utf8
Shape {
        geometry Sphere {}        # 球が１個
}
```

図形ノード

Shape ノード

物体のノードです。 geometry フィールドに形状を指定します。 appearance フィールドには色などの属性情報を指定します。

Shape ノードの書式

```
Shape {
        geometry 形状
        appearance 見かけ
}
```

1. appearance フィールドには物体の色などの見かけの情報を指定する。これには Appearance ノードを使用する。
2. geometry フィールドに物体の形状を指定する。ここには以下のノードが指定できる。
3. Box（箱），Cone（円錐），Cylinder（円柱），Sphere（球），Text（文字），Extrusion（押し出し），ElevationGrid（地形等），IndexedFaceSet（ポリゴン集合），IndexedLineSet（線分集合），PointSet（点集合）

陰影付けされた球

```
#VRML V2.0 utf8
Shape {
        geometry Sphere {}
        appearance Appearance {
                material Material {}
        }
}
```

単純な形状ノード

Sphere ノード

Shape ノードの geometry フィールドに指定する、球の形状のノードです。

Sphere ノードの書式

```
Sphere {
        radius 半径
}
```

1. radius フィールドに半径を指定する。
2. radius フィールドを省略したとき、半径は 1（デフォルト）になる。
3. 原点は球の中心にある。

陰影付けされた半径２の球

```
#VRML V2.0 utf8
Shape {
        geometry Sphere {
                radius 2
        }
        appearance Appearance {
                material Material {}
        }
}
```

Box ノード

Shape ノードの geometry フィールドに指定する、直方体の形状のノードです。

Box ノードの書式

```
Box {
        size x y z
}
```

1. size フィールドに x 軸方向、y 軸方向、z 軸方向の長さを指定する。
2. size フィールドを省略したときは１辺の長さが 1 の立方体になる。
3. 原点は直方体の中心にある。

直方体

```
#VRML V2.0 utf8
Shape {
        geometry Box {
                size 2 3 4
        }
        appearance Appearance {
                material Material {}
        }
}
```

Cone ノード

Shape ノードの geometry フィールドに指定する、円錐の形状のノードです。

Cone ノードの書式

```
Cone {
        bottomRadius 底面の半径
        height 高さ
        bottom 底面の有無
        side 側面の有無
}
```

1. bottomRadius フィールドに底面の半径を指定する。
2. height フィールドに高さを指定する。
3. bottom フィールドが TRUE なら底面を付ける。FALSE なら付けない。デフォルトは TRUE。
4. side フィールドが TRUE なら側面を付ける。FALSE なら付けない。デフォルトは TRUE。

円錐

```
        #VRML V2.0 utf8
        Shape {
```

```
                geometry Cone {
                        bottomRadius 2.5
                        height 4
                }
                appearance Appearance {
                        material Material {}
                }
        }
```

Cylinder ノード

Shape ノードの geometry フィールドに指定する、円筒の形状のノードです。

Cylinder ノードの書式

```
Cylinder {
        radius 半径
        height 高さ
        bottom 底面の有無
        top 上面の有無
        side 側面の有無
}
```

1. radius フィールドに半径を指定する。
2. height フィールドに高さを指定する。
3. bottom フィールドが TRUE なら底面を付ける。FALSE なら付けない。デフォルトは TRUE。
4. top フィールドが TRUE なら上面を付ける。FALSE なら付けない。デフォルトは TRUE。
5. side フィールドが TRUE なら側面を付ける。FALSE なら付けない。デフォルトは TRUE。

円柱

```
#VRML V2.0 utf8
Shape {
        geometry Cylinder {
                radius 2.5
                height 4
        }
        appearance Appearance {
                material Material {}
        }
}
```

外観の設定

Appearance ノード

Shape ノードの appearance フィールドに指定する、物体の色などの情報を指定するノードです。

Appearance ノードの書式

```
Appearance {
        material 材質の指定
        texture 貼り付けるテクスチャの指定
        textureTransform テクスチャの張り付け位置
}
```

1. material フィールドには物体の表面の材質パラメータを指定する。これには Material ノードを使用する。
2. texture フィールドには物体表面に貼り付ける別の画像を指定する。貼り付ける画像の指定には ImageTexture ノード、MovieTexture ノード、あるいは PixelTexutre ノードを使用する。デフォルトでは何も貼り付けない。
3. textureTransform フィールドには、texture フィールドで指定した画像の位置やスケールを調整する座標変換を、TextureTransform ノードで指定する。デフォルトは無変換。

Material ノード

物体表面の材質パラメータを指定します。Appearance ノードの material フィールドで使用します。

Material ノードの書式

```
Material {
        ambientIntensity 環境光の反射率
        diffuseColor r g b
        specularColor r g b
        shininess 輝き
        emissiveColor r g b
        transparency 透明度
}
```

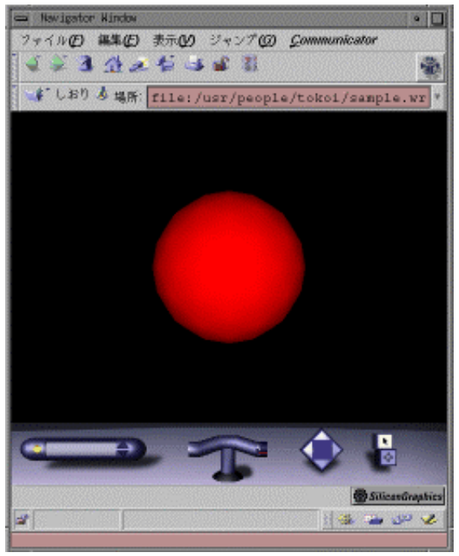
1. ambientIntensity フィールドは環境光（光源からの光の当たっていない部分の明るさ）に対する反射率を指定する。これが 0 だと陰の部分が真っ暗になる。
2. diffuseColor フィールドは拡散反射率を光の３原色（赤:r, 緑:g, 青:b、いずれも 0〜1）で指定する。これが物体の色になる。
3. specularColor フィールドには鏡面反射率を３原色（赤:r, 緑:g, 青:b、いずれも 0〜1）で指定する。これは光源の光が物体表面でそのまま反射して見える部分（ハイライト）の色になる。
4. shininess フィールドは輝き具合、すなわちハイライトの「強さ」を、0 〜１ の間で指定する。この値が大きくなるにつれてハイライトが鋭くなり、光沢が増す。
5. emissiveColor の３つの値（赤:r, 緑:g, 青:b、いずれも 0〜1）を ０ ０ ０ より大きくすると、この物体自体がその色で発光する。
6. transparency フィールドは透明度を指定する。 0 で不透明になり、1 で透明になる。デフォルトは 0。

拡散反射率の赤成分だけを１

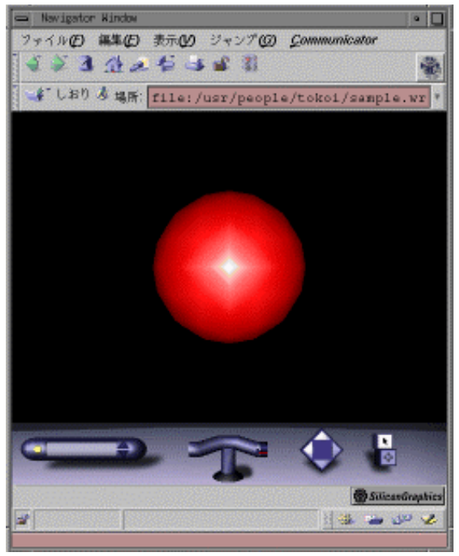
```
#VRML V2.0 utf8
Shape {
        geometry Sphere {
                radius 2
        }
        appearance Appearance {
                material Material {
                        diffuseColor １ ０ ０   # 拡散反射は赤
                }
        }
}
```

鏡面反射率の全成分を１

```
#VRML V2.0 utf8
Shape {
        geometry Sphere {
                radius 2
        }
        appearance Appearance {
                material Material {
                        diffuseColor 1 0 0    # 拡散反射は赤
                        specularColor 1 1 1   # 鏡面反射は白
                }
        }
}
```



拡散反射率の赤成分だけを１



鏡面反射率の全成分を１

演習

テキストエディタとして、この授業では「EmEditor」というものを使っていますが、「メモ帳」等各自使い慣れたものを使用していただいて結構です。

ただし、ファイルの保存の際に拡張子を指定する必要があるので、あらかじめファイルの拡張子を表示するようにしておいてください（下記）。

フォルダのツールメニューからフォルダオプションを選んで表示のタブをクリックし、登録されているファイルの拡張子は表示しないのチェックを外して OK をクリックしてください。

それでは、テキストエディタを起動して、太字のところを打ち込んでください。

球をひとつ作ってみます。 これを sample1.wrl というファイル名でデスクトップ（あるいは適当なフォルダ）に保存してください。 ファイルには渦巻状のアイコンが付くと思います。これをダブル栗句してみてください。 この球には陰影が付いていないので、円盤のようにしか見えません。	**#VRML V2.0 utf8** **Shape {** ** geometry Sphere {}** **}**

これに陰影をつけてみましょう。太字の部分を追加してください。
 	
追加できたら上書き保存して、Web ブラウザを「更新」してください。
	#VRML V2.0 utf8
	Shape {
	  geometry Sphere {}
	  appearance Appearance {
	  ** material Material {**
	  ** }**
	  }
	}

球の色を赤に変えてみましょう。	#VRML V2.0 utf8
	Shape {
	  geometry Sphere {}
	  appearance Appearance {
	  material Material {
	  diffuseColor 1 0 0
	  }
	  }
	}

球の半径を２にしてみましょう。	#VRML V2.0 utf8
	Shape {
	  geometry Sphere { **radius 2** }
	  appearance Appearance {
	  material Material {
	  diffuseColor 1 0 0
	  }
	  }
	}

ナビゲーション方法の設定（視点の設定）

NavigationInfo ノード

```
NavigatonInfo {
        avatarSize      [radius, height, knee]
        headlight       TRUE/FALSE
        speed           速度
        type            ナビゲーションの方法
        visibilityLimit 遠地点の距離
}
```

1. アバター（観測者の仮身）は円柱で表現され、半径、高さ、および地面からの高さ（ひざの高さ）で表現する。デフォルトは [0.25, 1.6, 0.75] であり、だいたい人間の大きさである。この大きさはアバターが仮想空間上で行動するときの制約になる。たとえば、この円柱の直径より狭い隙間は通れないとか、ひざの高さより高いものを乗り越えるには難儀するとか。
2. headlight はアバターから視線方向に向けられた、指向性を持った光源である。デフォルトは ON。
3. speed フィールドにはナビゲーション時の平行移動の、ブラウザのデフォルトの速度に対する係数を指定する。1.0 でデフォルトの速度、2.0 でその２倍。
4. type フィールドにはナビゲーションの方法する。ブラウザによって指定できる方法が異なるが、最低限 ”WALK”, “FLY”, “EXAMINE”, “NONE” はサポートされている。
5. visibilityLimit フィールドの値より遠くにある物体は表示されない。この値が 0.0（デフォルト）なら、無限遠まで表示される。

NavigationInfo ノードの使用例

```
NavigationInfo { type [“WALK”, “EXAMINE”] }
```

グループ化ノード

Group ノード

```
Group {
        bboxCenter      x y z
        bboxSize                x y z
        children                [コドモ，コドモ，...]
}
```

1. children フィールドに指定した複数の Shape ノードや Group ノードをまとめて、ひとつのノードとして表現する。
2. bboxCenter および bboxSize は、それぞれ外接箱の中心とサイズを指定する。内包するすべてのノードを囲う外接箱を指定すれば、画像表示の速度を向上できる。
3. bboxSize のデフォルトは –1 –1 –1 で、これは外接箱が設定されていないことを示す。
4. VRML ファイルのトップレベルのノードとして使われる。

Group ノードの使用例

```
#VRML V1.0 utf8
Group {
        children [
                Shape { ... }
                Shape { ... }
                Group { ... }
        ]
}
```


Transform ノード

```
Transform {
        translation             x y z
        rotation                x y z r
        scale                   x y z
        scaleOrientation        x y z r
        bboxCenter              外接箱の中心
        bboxSize                外接箱サイズ
        children                [コドモ，コドモ，...]
}
```

1. Group ノード同様 children フィールドに指定した複数の Shape ノードや Group ノードをまとめて一つのノードとして表現するが、その際に内包するすべてのノードに対する変換を指定する。
2. translation フィールドには、children フィールドに指定したノードを移動する位置を指定する。
3. rotation フィールドには、children フィールドに指定したノードの回転を指定する。x y z には回転軸ベクトル、r には回転角を与える。
4. scale フィールドには、children フィールドに指定したノードの、x 軸、y 軸、z 軸方向の拡大率を指定する。x y z > 0。
5. scaleOrientation は、scale フィールドによる拡大縮小を行う「前」の、children フィールドに指定したノードの回転を指定する。この回転により、任意の軸方向の拡大縮小が行える。x y z は回転軸ベクトル、r は回転角。

Transform ノードの使用例

```
Transform {
        translation 5 0 0
        children [ Shape { ... } ]
}
```

Billboard ノード

```
Billboard {
        axisOfRotation          x y z
        bboxCenter              外接箱の中心
        bboxSize                外接箱のサイズ
        children                [コドモ，コドモ，... ]
}
```

1. Billboard ノードは Transform ノードの変形で、children に指定したノードの（ローカル座標系の）Z 軸が常に観測者の方向を向くように調整する。
2. axisOfRotation フィールドには、観測者の方向に向けるための回転の軸ベクトルを指定する。デフォルトは 0 1 0（y 軸中心）。

Billboard ノードの使用例

```
Billboard {
        children [ Shape { geometry Text { "For SALE" } } ]
}
```

文字の表示

Text ノード

```
Text {
        string                  ["文字列". "文字列", ...]
        fontStyle               字体
        length                  [長さ, 長さ, ... ]
        maxExtent               最大長
}
```

1. string フィールドには表示する文字列を指定する。複数指定したときは改行される。
2. fontStyle フィールドには使用する字体を FontStyle ノードを使って指定する。
3. length フィールドにはそれぞれの文字列の長さを指定する。デフォルトは 0（指定なし）。
4. maxExtent フィールドには、表示する文字がこの長さを超えないようにする（この長さを超える文字列がこの長さに圧縮される）。デフォルトは 0（制限なし）。

Text ノードの使用例

```
Text { string ["How", "Are", "you?"] }
```

テクスチャマッピング

ImageTexture ノード

```
ImageTexture {
        url                     画像ファイルの url
        repeatS                 画像空間の S 軸方向の繰り返し
        repeatT                 画像空間の T 軸方向の繰り返し
}
```

1. Appearance ノードで用いるテクスチャファイルを指定する。画像ファイルの url は、このノードを含む VRML ファイルからの相対 url あるいは http://... からはじまる絶対 url。画像ファイルには GIF, JPEG などが指定できる。
2. repeatS および repeatT フィールドが TRUE なら貼り付けるテクスチャのサイズが貼り付ける場所より小さかったときに、テクスチャを繰り返し表示する。デフォルトは TRUE。

ImageTexture ノードの使用例

```
Appearance {
        texture ImageTexture { url "image.gif" }
}
```

TextureTransform ノード

```
TextureTransform {
        center                  x y
        rotation                r
        scale                   x y
        translation             x y
}
```


1. center フィールドは rotation フィールドと scale フィールドが適用されるテクスチャ空間中の中心位置を指定する。
2. rotation, scale および translation フィールドは、それぞれテクスチャの回転、拡大縮小と平行移動を指定する。scale はテクスチャ「座標系」の拡大縮小を行うので、scale 2 1 とすれば、テクスチャ自体のサイズは S 軸方向に 2 分の 1 になり、同じテクスチャが 2 回表示される。

背景の設定

Background ノード

```
Background {
        skyColor                [r g b, r g b, ...]
        skyAngle                [角度, 角度, ...]
        groundColor             [r g b, r g b, ...]
        groundAngle             [角度, 角度, ...]
        backUrl                 url
        frontUrl                url
        leftUrl                 url
        rightUrl                url
        topUrl                  url
        bottomUrl               url
}
```

1. skyColor フィールドには空の色を指定する。複数の色を指定し、隣り合う色の間隔（角度）を skyAngle に指定すれば、グラデーションを付けることができる。
2. groundColor フィールドには地面の色を指定する。複数の色を指定し、隣り合う色の間隔（角度）を skyAngle に指定すれば、グラデーションを付けることができる。
3. backUrl, frontUrl, rightUrl, leftUrl, topUrl, bottomUrl はそれぞれ z 軸の正の方向、z 軸の負の方向、x 軸の正の方向、x 軸の負の方向、y 軸の正の方向、y 軸の負の方向の無限遠に置いた平面に貼り付ける画像の url を指定する。

天空と大地を単色に設定する

```
Background {
        skyColor [ 0.5 0.9 0.9, 0.5 0.9 0.9 ]
        skyAngle [ 1.5708 ]
        groundColor [ 0.5 0.4 0.2, 0.5 0.4 0.2 ]
        skyAngle [ 1.5708 ]
}
```

六方にテクスチャ（画像）をマッピングする

```
BackGround {
        backUrl "back.gif"
        frontUrl "back.gif"
        rightUrl "back.gif"
        leftUrl "back.gif"
        topUrl "back.gif"
        bottomUrl "back.gif"
}
```

複雑な形状ノード

Extrusion ノード

Shape ノードの geometry フィールドに指定する、押し出し形状のノードです。

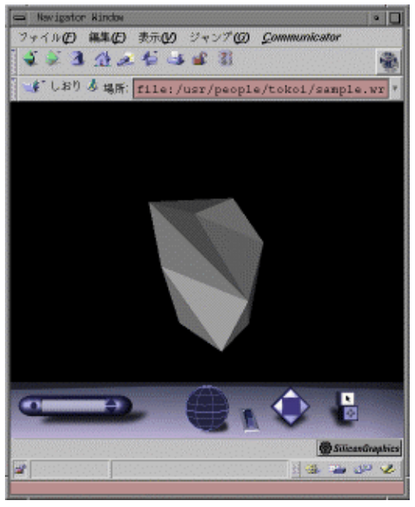
Extrusion ノードの書式

```
Extrusion {
        creaseAngle         スムーズシェーディングするときの限界角
        ccw                 頂点の順序が反時計回りか否か
        convex              面がすべて凸多角形か否か
        solid               閉じた形状か否か
        beginCap            最初の断面のフタの有無
        endCap              最後の断面のフタの有無
        crossSection        [断面の座標 0，断面の座標 1，.. ]
        scale               [断面 0 の拡大率，断面 1 の拡大率，.. ]
        orientation         [断面 0 の向き，断面 1 の向き，.. ]
        spine               [回転軸の座標 0，回転軸の座標 1，.. ]
}
```

1. 隣接する面同士がなす角度が creaseAngle フィールドの値よりも小さければ、この2つの面の間で輝度が補間されてスムーズシェーディングされる。デフォルトは 0（スムーズシェーディングしない）。
2. ccw フィールドが TRUE なら、多角形の両方ののうち、頂点の順序が反時計回り（左回り）に見える側の面を表として扱う。FALSE なら時計回り（右回り）に見える側を表として扱う。デフォルトは TRUE。
3. convex フィールドが TRUE なら、すべての面が凸多角形であると仮定して処理する。これを指定すると表示に要する時間を短縮できる可能性があるが、凹多角形が含まれていると正確に表示されない場合がある。デフォルトは TRUE。
4. solid フィールドが TRUE なら、物体を閉じた形状と仮定して処理する。開いた形状の場合に見える、視点に対して裏を向いている面は表示されない。デフォルトは TRUE。
5. beginCap および endCap フィールドが TRUE なら、それぞれ最初の断面と最後の断面にフタをする。デフォルトは TRUE。
6. crossSection フィールドには断面の（xz 平面上の）頂点の位置を指定する。scale フィールドには各断面の x 方向および z 方向の拡大率を指定する。
7. orientation フィールドには各断面の回転を、回転軸と回転角で指定する。
8. spine フィールドには回転軸の節点（断面と交差する点）の座標を指定する。

押し出し

```
#VRML V2.0 utf8
Shape {
        geometry Extrusion {
                crossSection [
                        0 1, 1 -1, -1 -1, 0 1   # 断面は３角形
                ]
                orientation [
                        0 1 0 0,                # 第１断面はそのまま
                        0 1 0 0.2,              # 第２断面は 0.2 よじる
                        0 0 1 0.5               # 第３断面は 0.5 傾ける
                ]
                scale [
                        1 1, 1.5 1.5, 1 1
                ]
                spine [
                        0 0 0, 0 2 0, 1 4 1
                ]
        }
        appearance Appearance {
                material Material {}
        }
}
```



ElevationGrid ノード

Shape ノードの geometry フィールドに指定する、地形などを表現するための形状ノードです。

ElevationGrid ノードの書式

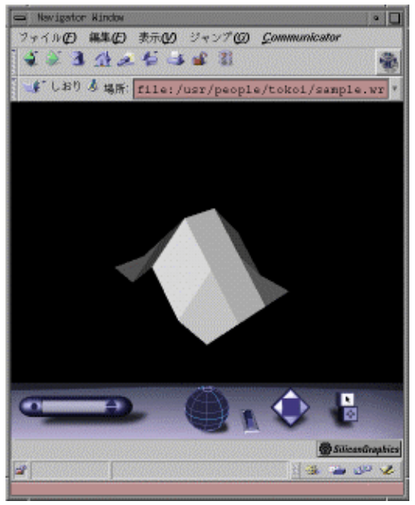
```
ElevationGrid {
        creaseAngle         スムーズシェーディングするときの限界角
        ccw                 頂点の順序が反時計回りか否か
        convex              面がすべて凸多角形か否か
        solid               閉じた形状か否か
        colorPerVertex      頂点ごとに色を与えるか否か
        normalPerVertex     頂点ごとに法線を与えるか否か
        color               各面／頂点の色
        normal              各面／頂点の法線ベクトル
        texCoord            テクスチャの座標
        xDimension          x 方向の格子点数
        zDimension          z 方向の格子点数
        xSpacing            x 方向の格子点間隔
        zSpacing            z 方向の格子点間隔
        height              [格子点 0 の高さ，格子点 1 の高さ，.. ]
}
```

1. creaseAngle，ccw，convex，solid フィールドは Extrusion と同じ。
2. colorPerVertex フィールドが TRUE なら頂点ごとに色を指定する。FALSE なら面ごとに色を指定する。デフォルトは TRUE。
3. normalPerVertex フィールドが TRUE なら頂点ごとに法線ベクトルを指定する。FALSE なら面ごとに法線ベクトルを指定する。デフォルトは TRUE。
4. color フィールドには頂点／面に与える色を指定する。これには Color ノードを使用する。

5. normal フィールドには頂点／面に与える法線ベクトルを指定する。これには Normal ノードを使用する。
6. texCoord フィールドにはテクスチャの座標を指定する。これには TextureCoordinate ノードを使用する。
7. xDimension および zDimensions フィールドに、それぞれ x 方向と z 方向の格子点の数を指定する。
8. xSpacing および zSpacing フィールドに、それぞれ x 方向と z 方向の格子点の間隔を指定する。
9. height フィールドにそれぞれの格子点の高さを指定する。

地形

```
#VRML V2.0 utf8
Shape {
        geometry ElevationGrid {
                xDimension 4
                zDimension 5
                xSpacing 1
                zSpacing 1
                height [
                        0, 0, 0, 0,
                        0, 1, 1, 0,
                        1, 2, 2, 1,
                        0, 1, 1, 0,
                        0, 0, 0, 0
                ]
        }
        appearance Appearance {
                material Material {}
        }
}
```



IndexedFaceSet ノード

Shape ノードの geometry フィールドに指定する、任意の多面体形状（ポリゴン）を表現するための形状ノードです。

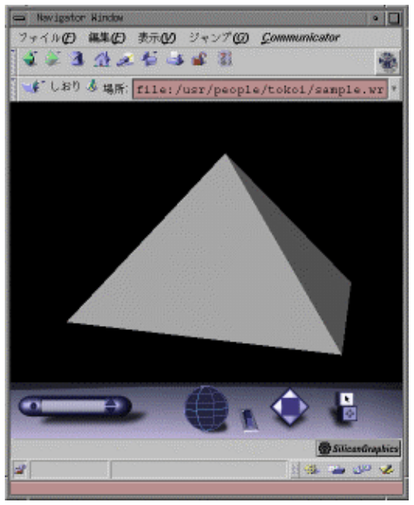
IndexedFaceSet ノードの書式

```
IndexedFaceSet {
        creaseAngle             スムーズシェーディングするときの限界角
        ccw                     頂点の順序が反時計回りか否か
        convex                  面がすべて凸多角形か否か
        solid                   閉じた形状か否か
        colorPerVertex          頂点ごとに色を与えるか否か
        normalPerVertex         頂点ごとに法線を与えるか否か
        color                   各面／頂点の色
        normal                  各面／頂点の法線ベクトル
        texCoord                テクスチャの座
        colorIndex              色データの指標
        normalIndex             法線ベクトルデータの指標
        texCoordIndex           テクスチャ座標データの指標
        coord                   座標データ
        coordIndex              面データ
}
```

1. creaseAngle, ccw, convex, solid フィールドは Extrusion と同じ。
2. colorPerVertex, normalPerVertex, color, normal, texCood フィールドは ElevationGrid と同じ。
3. colorIndex, normalIndex, texCoordIndex フィールドは、色／法線ベクトル／テクスチャの座標を、それぞれのデータの指標（番号）で指定する場合に用いる。同じデータが繰り返し現れるような場合は、データ量を削減できる。
4. coord フィールドに座標データを指定する。これには Coordinate ノードを用いる。
5. coordIndex フィールドに面データを指定する。面データは面（多角形）を構成する頂点の座標を、座標データの指標で列挙する。面と面の区切りに -1 を置く。

任意の多面体形状

```
#VRML V2.0 utf8
Shape {
        geometry IndexedFaceSet {
                coord Coordinate {
                        point [
                                0  4  0,
                                -3  0 -3,
                                -3  0  3,
                                3  0  3,
                                3  0 -3
                        ]
                }
                coordIndex [
                        0, 1, 2, -1,
                        0, 2, 3, -1,
                        0, 3, 4, -1,
                        0, 4, 1, -1,
                        4, 3, 2, 1, -1
                ]
        }
        appearance Appearance {
                material Material {}
        }
}
```



IndexedLineSet ノード

Shape ノードの geometry フィールドに指定する、任意の線図形を表現するための形状ノードです。

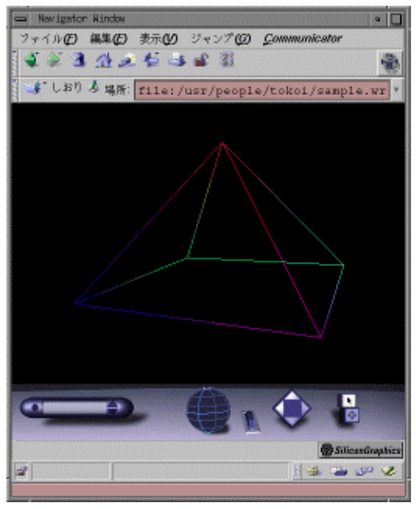
IndexedLineSet ノードの書式

```
IndexedFaceSet {
        colorPerVertex          頂点ごとに色を与えるか否か
        color                   各面／頂点の色
        colorIndex              色データの指標
        coord                   座標データ
        coordIndex              線分（折れ線）データ
}
```

1. colorPerVertex フィールドが TRUE なら頂点ごとに色を指定する。FALSE なら線分ごとに色を指定する。デフォルトは TRUE。
2. color フィールドには頂点／線分に与える色を指定する。これには Color ノードを使用する。
3. colorIndex フィールドは、頂点／線分の色を、色データの指標（番号）で指定する場合に用いる。同じ色が繰り返し現れるような場合は、データ量を削減できる。
4. coord フィールドに座標データを指定する。これには Coordinate ノードを用いる。
5. coordIndex フィールドに線分データを指定する。線分データは線分（折れ線）の端点（節点）の座標を、座標データの指標で列挙する。折れ線と折れ線の区切りに -1 を置く。

線図形（端点ごとに色を指定）

```
#VRML V2.0 utf8
Shape {
        geometry IndexedLineSet {
                coord Coordinate {
                        point [
                                0  4  0,
                               -3  0 -3,
                               -3  0  3,
                                3  0  3,
                                3  0 -3
                        ]
                }
                coordIndex [
                        0, 1, -1,
                        0, 2, -1,
                        0, 3, -1,
                        0, 4, -1,
                        4, 3, 2, 1, 4, -1
                ]
                color Color {
                        color [
                                1 0 0,
                                0 1 0,
                                0 0 1,
                                1 0 1,
                                0 1 1
                        ]
                }
        }
}
```

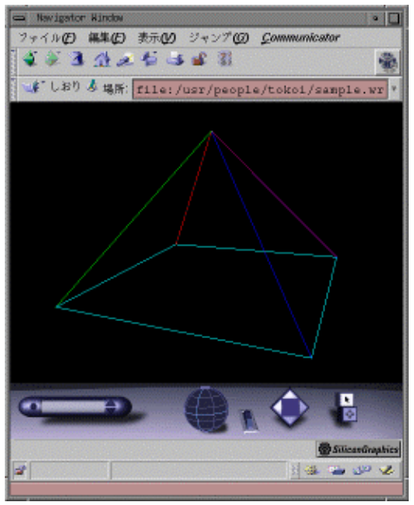


線図形（線ごとに色を指定）

```
        #VRML V2.0 utf8
        Shape {
        geometry IndexedLineSet {
                coord Coordinate {
                        point [
                                0  4  0,
```



```
                                -3  0 -3,
                                -3  0  3,
                                 3  0  3,
                                 3  0 -3
                        ]
                }
                coordIndex [
                        0, 1, -1,
                        0, 2, -1,
                        0, 3, -1,
                        0, 4, -1,
                        4, 3, 2, 1, 4, -1
                ]
                color Color {
                        color [
                                1 0 0,
                                0 1 0,
                                0 0 1,
                                1 0 1,
                                0 1 1
                        ]
                }
        }
}
```



PointSet ノード

Shape ノードの geometry フィールドに指定する、任意の点集合を表現するための形状ノードです。

PointSet ノードの書式

```
PointSet {
        color 点の色
        coord 座標データ
}
```

1. color フィールドには点に与える色を指定する。これには Color ノードを使用する。
2. coord フィールドに点の座標データを指定する。これには Coordinate ノードを用いる。

頂点位置、法線、色

Coordinate ノード

IndexedFaceSet ノード、IndexedLineSet ノード、PointSet ノードの coord フィールドに指定する、頂点の座標値を指定するためのノードです。

Coordinate ノードの書式

```
Coordinate {
        point [ x y z, x y z, ... ]
}
```

Normal ノード

IndexedFaceSet ノード、ElevationGrid ノードの normal フィールドに指定する、面あるいは頂点の法線ベクトルを指定するためのノードです。

Normal ノードの書式

```
Normal {
                vector [ x y z, x y z, ... ]
}
```

Color ノード

IndexedFaceSet ノード、IndexedLineSet ノード、ElevationGrid ノード、PointSet ノードの color フィールドに指定する、頂点の色を指定するためのノードです。

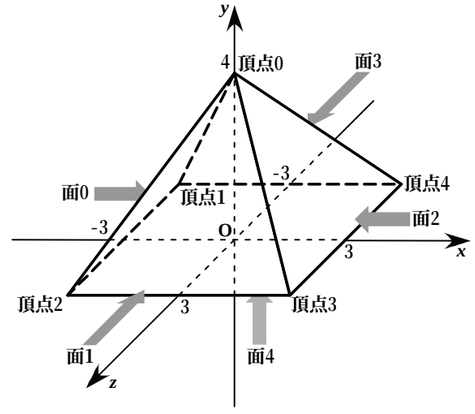
Color ノードの書式

```
Color {
        color [ r g b, r g b, ... ]
}
```


立体形状の表現

球や直方体などの単純な形状の組み合わせるだけでも様々な立体形状を表現することが可能だが、頂点や面といった細かな形状情報を用いれば、より一般的な立体形状を表現することが可能になる。

右の図形（５面体）は、a, b, c, d, e の５つの頂点と、５つの面から構成されている。



頂点	x	y	z
頂点 0	0	4	0
頂点 1	-3	0	-3
頂点 2	-3	0	3
頂点 3	3	0	3
頂点 4	3	0	-3

この頂点の座標データを、Coordinate ノードを用いて表現すると、以下のようになる。

```
Coordinate {
    point [ 0 4 0, -3 0 -3, -3 0 3, 3 0 3, 3 0 -3 ]
}
```

右上の５面体の各面は、これらの頂点を以下のように結んで定義される。頂点の順序は、その面を「表」から見たときに「**左回り**」になるようにする。

面	頂点			
面 0	頂点 0	頂点 1	頂点 2	
面 1	頂点 0	頂点 2	頂点 3	
面 2	頂点 0	頂点 3	頂点 4	
面 3	頂点 0	頂点 4	頂点 1	
面 4	頂点 4	頂点 3	頂点 2	頂点 1

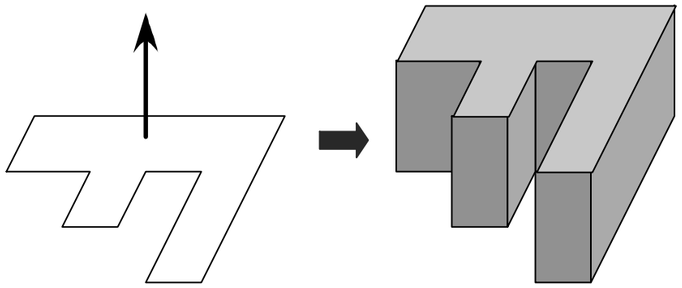
右上の５面体を作成するには、IndexedFaceSet という**形状ノード**を用いる。この coord フィールドに Coordinate ノードを指定し、coordinateIndex フィールドに Coordinate ノードの各座標値の**インデックス番号**を列挙する。“-1” は**「面データの区切り」**である。

```
IndexedFaceSet {
    coord Coordinate {
        point [0 4 0, -3 0 -3, -3 0 3, 3 0 3, 3 0 -3 ]
    }
    coordIndex [ 0, 1, 2, -1,   0, 2, 3, -1,   0, 3, 4, -1,   0, 4, 1, -1,   4, 3, 2, 1, -1]
}
```

掃引による形状の表現

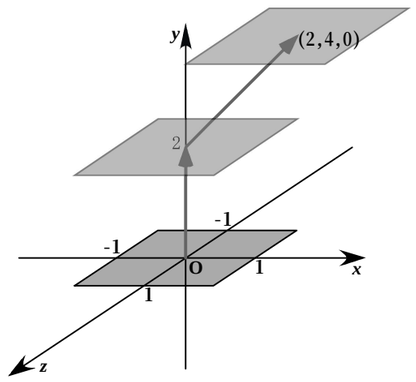
IndexedFaceSet ノードのように、頂点の座標値や面の情報（これらは物体の内部と外部の隔てる境界なので**境界情報**とも呼ばれる）を用いた形状記述は、柔軟でどのような形状でも表現することが可能だが、形状の定義が煩雑でデータ量も多くなりがちである。

掃引とは、ある平面形状を指定した経路（パス）に沿って移動させたとき、その平面形状が通過する空間領域を用いて立体形状を表現する方法である。この方法は、（境界情報を用いるより）少ないデータ量で、かなり柔軟に様々な形状を表現できる。



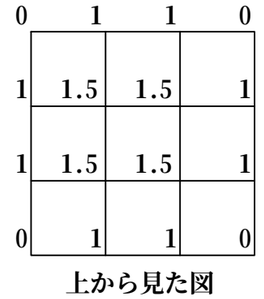
Extrusion ノードは、このような掃引図形の表現に用いる。

1. 右図のような正方形の断面形状（cross section）を、(0, 2, 0) に平行移動した後、更に (2, 4, 0) に平行移動した掃引形状を表現する場合を考える。
2. crossSection フィールドには、断面形状の頂点座標（２次元）を、左回りで列挙する。
3. spine フィールドに、この断面形状を平行移動する位置を設定する。
4. scale フィールドには、各断面形状の拡大率を設定する。
5. orientation フィールドには、各断面形状の回転軸と回転角を指定する。



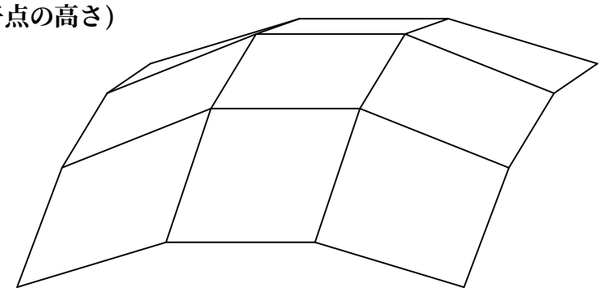

```
Extrusion {
    crossSection [1 1, 1 -1, -1 -1, -1 1]
    spine [0 0 0, 0 2 0, 2 4 0]
    scale [1 1, 1 1, 1 1]
    orientation [0 1 0 0, 0 1 0 0, 0 1 0 0]
}
```


課題１

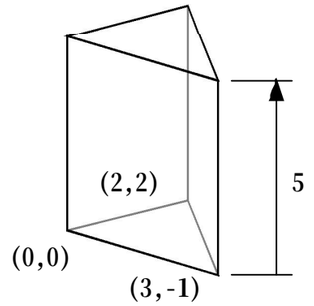


上から見た図

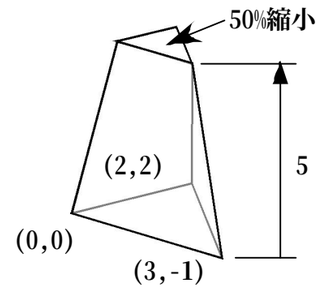
(格子点の高さ)



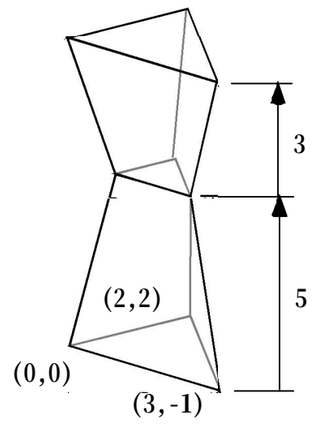
課題２(a)



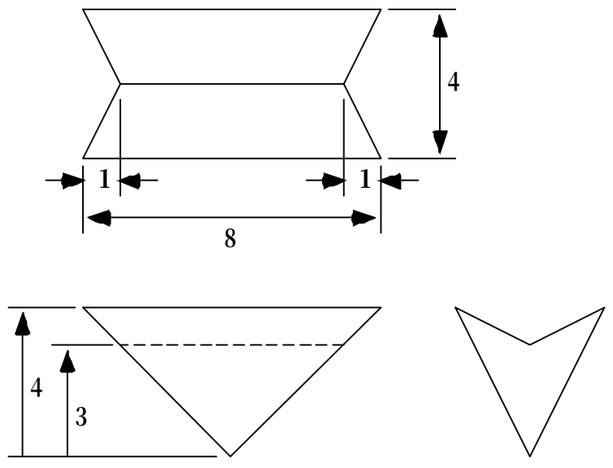
課題２(b)



課題２(c)



課題３



光源ノード

概要

ＶＲＭＬシーン中に光源を設定します。光源には平行光線（DirectionalLight）、点光源（SpotLight）、スポットライト（SpotLight）、およびヘッドライトがあります。デフォルト（何も設定しない状態）では、ヘッドライト（視点位置から放射される光）だけが存在します。

DirectionalLight

光を一定の方向に放射する光源です。光の放射方向を指定します。

PointLight

空間中の１点から全方向に光を放射する光源です。光源の位置や減衰率を指定します。

SpotLight

空間中の１点から特定の方向に光を放射する光源です。光源の位置と光の放射方向、減衰率、および光の広がり角度を設定します。

DirectionalLight ノード

書式

```
DirectionalLight {
        direction           0 0 -1              # 光の放射方向
        color               1 1 1               # 光源の色
        intensity           1                   # 光源の強さ
        on                  TRUE                # 光源の on/off
        ambientIntensity    0                   # 環境光の強さ
}
```

フィールド

direction: 光の放射方向ベクトルを指定します。単位ベクトルである必要はありません。デフォルト値（0 0 -1）では、光の方向はＺ軸の反対方向を向いています。
color: 光源の色を RGB 比（それぞれ 0〜1 の値）で指定します。デフォルト値（1 1 1）では、光源は白色です。
intensity: 光源の強度（明るさ）を 0〜1 の値で指定します。デフォルト値（1）は、最大の明るさです。
on: 光源の on/off を制御します。TRUE で on（点灯）、FALSE で off（消灯）します。デフォルトは TRUE（on）です。
ambientIntensity: この光源による、環境光の強度を（明るさ）を 0〜1 の値で指定します。環境光は、他の物体表面で反射して届いた光（間接光）など、この光源から直接物体表面に届く光（直接光）以外の成分をまとめたもので、直接光が当らない影の部分の明るさになります。

例

```
#VRML V2.0 utf8
DirectionalLight {
        direction 1 0 0      # X 軸の正の方向に向いた光
        color 0 1 0          # 色は緑
        ambientIntensity 0.2# 環境光強度
}
Shape {
        geometry Sphere {}
        appearance Appearance {
                material Material {}
        }
}
```


PointLight ノード

書式

```
PointLight {
        location            0 0 0               # 光源の位置
        color               1 1 1               # 光源の色
        intensity           1                   # 光源の強さ
        on                  TRUE                # 光源の on/off
        ambientIntensity    0                   # 環境光の強さ
        attenuation         1 0 0               # 光の減衰率
        radius              100                 # 光の到達半径
}
```

フィールド

location: 光源の位置を指定します。デフォルト値（0 0 0）では、光源は原点位置にあります。
color: 光源の色を RGB 比（それぞれ 0〜1 の値）で指定します。デフォルト値（1 1 1）では、光源は白色です。
intensity: 光源の強度（明るさ）を 0〜1 の値で指定します。デフォルト値（1）は、最大の明るさです。
on: 光源の on/off を制御します。TRUE で on（点灯）、FALSE で off（消灯）します。デフォルトは TRUE（on）です。
ambientIntensity: この光源による、環境光の強度を（明るさ）を 0〜1 の値で指定します。環境光は、他の物体表面で反射して届いた光（間接光）など、この光源から直接物体表面に届く光（直接光）以外の成分をまとめたもので、直接光が当らない影の部分の明るさになります。
attenuation: 光の減衰率を指定します。点光源の場合、光源から距離 r 離れた位置での明るさは、物理的には $1/r^2$ になりますが、これをそのまま用いると減衰が早すぎて使いにくい（少し離れただけですぐに真っ暗になる）ものになります。attenuation フィールドの３つのパラメータを左からそれぞれ a_1, a_2, a_3 とすると、明るさは $1/(a_1+a_2*r+a_3*r^2)$ になります。デフォルト値の(1 0 0)では、光は減衰しません。
radius: この光源の光が到達する範囲を指定します。デフォルト値は 100 です。

例

```
#VRML V2.0 utf8
PointLight {
        location 0 0 2       # 光源の位置はＺ軸上の点(0 0 2)
        color 0 1 0          # 色は緑
        attenuation 1 1 0    # 距離に反比例して減衰
}
Shape {
        geometry Sphere {}
        appearance Appearance {
                material Material {}
        }
}
```

SpotLight ノード

書式

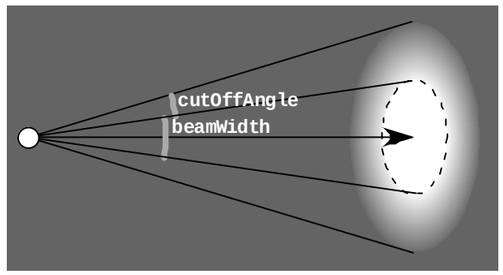
```
SpotLight {
        direction           0 0 -1              # 光の方向
        location            0 0 0               # 光源の位置
        color               1 1 1               # 光源の色
        intensity           1                   # 光源の強さ
        on                  TRUE                # 光源の on/off
        ambientIntensity    0                   # 環境光の強さ
        attenuation         1 0 0               # 光の減衰率
        radius              100                 # 光の到達半径
        beamWidth           1.570796            #
        cutOffAngle         0.785398            #
}
```

フィールド

direction: 光の放射方向ベクトルを指定します。単位ベクトルである必要はありません。デフォルト値（0 0 –1）では、光の方向はＺ軸の反対方向を向いています。
location: 光源の位置を指定します。デフォルト値（0 0 0）では、光源は原点位置にあります。
color: 光源の色を RGB 比（それぞれ 0〜1 の値）で指定します。デフォルト値（1 1 1）では、光源は白色です。
intensity: 光源の強度（明るさ）を 0〜1 の値で指定します。デフォルト値（1）は、最大の明るさです。
on: 光源の on/off を制御します。TRUE で on（点灯）、FALSE で off（消灯）します。デフォルト値は TRUE（on）です。
ambientIntensity: この光源による、環境光の強度を（明るさ）を 0〜1 の値で指定します。環境光は、他の物体表面で反射して届いた光（間接光）など、この光源から直接物体表面に届く光（直接光）以外の成分をまとめたもので、直接光が当らない影の部分の明るさになります。
attenuation: 光の減衰率を指定します。点光源の場合、光源から距離 r 離れた位置での明るさは、物理的には $1/r^2$ になりますが、これをそのまま用いると減衰が早すぎて使いにくい（少し離れただけですぐに真っ暗になる）ものになります。attenuation フィールドの３つのパラメータを左からそれぞれ a_1,a_2,a_3 とすると、明るさは $1/(a_1+a_2*r+a_3*r^2)$ になります。デフォルト値の(1 0 0)では、光は減衰しません。
radius: この光源の光が到達する範囲です。デフォルト値は 100。
beamWidth: 光の強度の最大値で光が広がる角度です。デフォルト値はπ/2。
cutOffAngle: 光が完全に遮断される角度です。デフォルト値はπ/4。

例

```
#VRML V2.0 utf8
SpotLight {
        location 0 0 2
        direction 0 0 -1
        color 0 1 0
        attenuation 1 1 0
        beamWidth 1
        cutOffAngle 0.2
        ambientIntensity 0.2
}
Shape {
        geometry Sphere {}
        appearance Appearance {
                material Material {}
        }
}
```



ヘッドライト

ヘッドライトは視点の位置にあり、光の方向は視線の方向を向いた平行光線の光源です。この光源に対応したノードはありません。ヘッドライトの点灯・消灯には、NavigationInfo ノードを使います。

例

```
NavigationInfo {
        headlight FALSE             # ヘッドライトを消す
}
```


視点ノード

概要

ＶＲＭＬシーン中の視点の位置と方向を設定するには Viewpoint ノードを用います。視点の位置はブラウザの操作によって自由に変更できますが（ナビゲーション）、Viewpoint ノードによって、その初期値を与えることができます。また、ナビゲーションを禁止（NavigationInfo ノードの type フィールドに “NONE” を指定）し、複数の視点の位置を切り替えることによって、視点の移動によるアニメーションを表示する場合にも使用されます。

Viewpoint ノード

書式

```
Viewpoint {
        position        0 0 10              # 視点の位置
        orientation     0 1 0 0             # 視線の方向
        fieldOfView     0.785398            # 視野角
        jump            TRUE                # 視点の移動の可否
        description     ""                  # 視点の説明
}
```

フィールド

position: 光源の位置を指定します。デフォルト値は（0 0 10）です。
orientation: 視線の方向を、回転軸と回転角で指定します。デフォルト値では視線はＺ軸の負の方向（0 0 –1 の方向）を向いており、これを回転させて視線の向きを変更します。
fieldOfView: 視点の視野角です。カメラの画角に相当します。この値が大きいほどワイドレンズになり、小さいほど望遠レンズになります。デフォルト値はπ/4 です。
jump: 視点の移動の可否を指定します。デフォルト値は TRUE（視点の移動可）で、ブラウザの操作によって複数の視点を渡り歩くことができます。
description: 視点に名前をつけます。デフォルト値は空文字列です。視点を移動するときなどに、この名前を指定します。

例

```
#VRML V2.0 utf8
Viewpoint {
        posision 5 5 5              # 視点の位置
        orientation 1 0 1 –1.5      # 視線の方向
        descriptin “position1”      # この視点の名前
}
Viewpoint {
        posision -5 5 5             # 視点の位置
        orientation 1 0 -1 –1.5     # 視線の方向
        descriptin “position2”      # この視点の名前
}
Shape {
        geometry Box {
                size 2 3 4
        }
        appearance Appearance {
                material Material {}
        }
}
```

ユーザ定義ノード

概要

DEF はノードに名前をつけます。USE は名前をつけたノードを、その名前で再度呼び出します。これを同じ形状のノードが複数ある場合に、データ量を節約できます。また、あるノードから送出されたイベント（後述）を別のノードに送る場合にも、この名前を使用します。他に DEF より一般的なユーザ定義ノードを定義できる PROTO というものもあります（後述）。

書式

```
DEF ユーザ定義ノード名 ノード          # ノードに“ユーザ定義ノード名”を付ける
USE ユーザ定義ノード名                 # “ユーザ定義ノード名”のノードを呼び出す
```

例

```
#VRML V2.0 utf8
# ４輪の台車
Shape {
	geometry Box {
		size 6 3.6 2
	}
	appearance Appearance {
		material Material {}
	}
}
Transform {
	translation -2 2 1
	children [
		DEF WHEEL Shape {
			geometry Cylinder {
				radius 1
				height 0.4
			}
			appearance Appearance {
				material Material {
					diffuseColor 0.8 0.5 0.2
				}
			}
		}
	]
}
Transform {
	translation 2 2 1
	children [ USE WHEEL ]
}
Transform {
	translation -2 -2 1
	children [ USE WHEEL ]
}
Transform {
	translation 2 -2 1
	children [ USE WHEEL ]
}
```


演習

太字のところを打ち込んでください。各ステップごとに「更新」して、画面で表示を確認してください。

まず、箱をひとつ置きます。ちょっと背を高めにしておきます。 できたら.wrl という拡張子を付けたファイル名で保存し、Web ブラウザで見てみましょう。	**#VRML V2.0 utf8** **Shape {** **　geometry Box { size 1 3 1.5 }** **　appearance Appearance {** **　　material Material {** **　　diffuseColor 1 1 1** **　　}** **　}** **}**

この箱の Shape ノードに名前を付けましょう。名前は何でも構わないのですが、とりあえず building とでもしましょう。	#VRML V2.0 utf8
	DEF building Shape {
	　geometry Box { size 1 3 1.5 }
	　appearance Appearance {
	　　material Material {
	　　diffuseColor 1 1 1
	　　}
	　}
	}

名前を付けたノードのコピーを作ります。	#VRML V2.0 utf8
	DEF building Shape {
	　geometry Box { size 1 3 1.5 }
	　appearance Appearance {
	　　material Material {
	　　diffuseColor 1 1 1
	　　}
	　}
	}
	
	USE building

このままだと元のノードと同じ場所にコピーを置いてしまうことになるので、コピーの位置を少しずらします。	#VRML V2.0 utf8
	DEF building Shape {
	　geometry Box { size 1 3 1.5 }
	　appearance Appearance {
	　　material Material {
	　　diffuseColor 1 1 1
	　　}
	　}
	}
	
	Transform {
	**　translation 2 0 0**
	**　children [** USE building **]**
	}

もうひとつコピーを作りましょう。	#VRML V2.0 utf8
	DEF building Shape {
	　geometry Box { size 1 3 1.5 }
	　appearance Appearance {
	　　material Material {
	　　diffuseColor 1 1 1
	　　}
	　}
	}
	
	Transform {
	　translation 2 0 0
	　children [USE building]

	} **Transform {** **translation -2 0 0** **children [USE building]** **}**

これらをひとつのノードにグループ化してしまいましょう。	#VRML V2.0 utf8
	Group {
	children [
	DEF building Shape {
	geometry Box { size 1 3 1.5 }
	appearance Appearance {
	material Material {
	diffuseColor 1 1 1
	}
	}
	}
	
	Transform {
	translation 2 0 0
	children [USE building]
	}
	
	Transform {
	translation -2 0 0
	children [USE building]
	}
	]
	}

このグループにも名前を付けます。やはり名前は何でもいいんですが、ここでは安直に buildings とでもしましょう。	#VRML V2.0 utf8
	DEF buildings Group {
	children [
	DEF building Shape {
	geometry Box { size 1 3 1.5 }
	appearance Appearance {
	material Material {
	diffuseColor 1 1 1
	}
	}
	}
	
	Transform {
	translation 2 0 0
	children [USE building]
	}
	
	Transform {
	translation -2 0 0
	children [USE building]
	}
	]
	}

このグループのコピーを作ります。先ほどと同じように、コピーは元のノードから少し離れたところに移動します。	#VRML V2.0 utf8
	DEF buildings Group {
	children [
	DEF building Shape {
	geometry Box { size 1 3 1.5 }
	appearance Appearance {
	material Material {
	diffuseColor 1 1 1

	}
	}
	}
	
	Transform {
	translation 2 0 0
	children [USE building]
	}
	
	Transform {
	translation -2 0 0
	children [USE building] }
	]
	}
	
	Transform {
	translation 0 0 2.5
	children [USE buildings]
	}

それでは、ヘッドライトを消してみましょう。
 	
VRML ファイルが長くなってきたので、途中は省略することにします。
	#VRML V2.0 utf8
	DEF buildings Group {
	
	（中略）
	
	}
	
	Transform {
	translation 0 0 2.5
	children [USE buildings]
	}
	
	NavigationInfo {
	headlight FALSE
	}

ヘッドライトを消してしまうと、光源がひとつも無くなってしまうので、真っ暗になってしまいます。そこで、太陽光のような役割を果たす平行光線の光源をひとつ置きます。	#VRML V2.0 utf8
	
	（中略）
	
	NavigationInfo {
	headlight FALSE
	}
	
	DirectionalLight {
	direction -4 -5 -3
	}

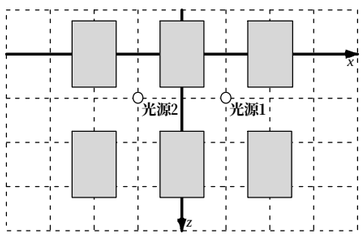
このままだと光があたっていない部分が真っ黒のままなので、環境光（その光源の光が直接当たらない部分に届く光）の強さを設定します。	#VRML V2.0 utf8
	
	（中略）
	
	DirectionalLight {
	direction -4 -5 -3
	ambientIntensity 0.5
	}

さらに、スポットライトを使ってビルをライトアップしてみましょう。まず、正面から赤い光を当ててみます。	#VRML V2.0 utf8 （中略） DirectionalLight { direction -4 -5 -3 ambientIntensity 0.5

	} **SpotLight {** **location 0 0 5** **direction 0 0 –1** **color 1 0.2 0** **}**

大半の VRML ブラウザは、物体が別の物体に影を落とすという処理ができないため、奥のほうの箱にまでスポットライトが届いています。これをごまかす？ために、光源の減衰率を指定してみてください（スポットライトの到達範囲 radius を指定しても効かないみたい）。	#VRML V2.0 utf8
	
	（中略）
	
	SpotLight {
	location 0 0 5
	direction 0 0 –1
	color 1 0.2 0
	attenuation 0 1 0
	}

「交差点」のところに点光源を置いてみましょう。PointLight ノードにも location フィールドがあるので、これを用いて位置を指定できますが、ここでは Transform ノードを用います。ついでに、この光源に名前を付けておきましょう。	#VRML V2.0 utf8
	
	（中略）
	
	SpotLight {
	location 0 0 5
	direction 0 0 –1
	color 1 0.2 0
	attenuation 0 1 0
	}
	
	Transform {
	translation 1 0 1
	children [
	DEF streetlight PointLight {
	color 1 1 0.3
	attenuation 0 1 0
	}
	]
	}

もう一方の交差点にも同じ点光源を置きます。
 
 	
	#VRML V2.0 utf8
	
	（中略）
	
	Transform {
	translation 1 0 1
	children [
	DEF streetlight PointLight {
	color 1 1 0.3
	attenuation 0 1 0
	}
	]
	}
	
	Transform {
	translation –1 0 1
	children [USE streetlight]
	}

今度は視点の位置を設定してみましょう。視点が設定できたら、視点を移動してみてください。	#VRML V2.0 utf8 （中略） Transform {

	translation –1 0 1 children [USE streetlight] } **Viewpoint {** **position 1 –1.3 1.5** **orientation 0 1 0 1.5708** **}**

視点の画角を設定します。初期値は 0.785398（約π/4=45°）ですから、これを 60° に広げてみます。遠くが少し小さくなったと思います。	#VRML V2.0 utf8
	
	（中略）
	
	Viewpoint {
	position 1 –1.3 1.5
	orientation 0 1 0 1.5708
	fieldOfView 1.0472
	}

視点をもうひとつ追加してみましょう。２つの視点を区別するために、それぞれに「説明」を付けます。
 	
できたらブラウザの表示を「更新」して、VRML ブラウザの左下にある視点切り替えのボタンをクリックしてみてください。
 
 	
	#VRML V2.0 utf8
	
	（中略）
	
	Viewpoint {
	position 1 –1.3 1.5
	orientation 0 1 0 1.5708
	fieldOfView 1.0472
	description "position1"
	}
	
	Viewpoint {
	position -1 –1.3 1.5
	orientation 0 1 0 1.5708
	fieldOfView 1.0472
	description "position2"
	}

視点をあと２つ追加してみます。できたらブラウザを更新して、視点を切り替えてみてください。	#VRML V2.0 utf8
	
	（中略）
	
	Viewpoint {
	position -1 –1.3 1.5
	orientation 0 1 0 1.5708
	fieldOfView 1.0472
	description "position2"
	}
	
	Viewpoint {
	position -1 –1.3 3.5
	orientation 0 1 0 -1.5708
	fieldOfView 1.0472
	description "position3"
	}
	
	Viewpoint {
	position 1 –1.3 3.5
	orientation 0 1 0 -1.5708
	fieldOfView 1.0472
	description "position4"
	}

センサノード

概要

ＶＲＭＬシーンの中に特定の状況が発生したときに、そのことを知らせる「イベント」を発生します。

Anchor

マウスでクリックしたオブジェクト（のグループ）に結びつけられた URL に移動します。

TouchSensor

マウスがオブジェクトに触れたことを知らせます。

SphereSensor

マウスの動きを特定の点を中心にしたオブジェクトの回転角に変換します。

CylinderSensor

マウスの動きを特定の軸を中心にしたオブジェクトの回転角に変換します。

PlaneSensor

マウスの動きを xy 平面上の位置に変換します。

ProximitySensor

特定の領域内に視点が入ったことを知らせます。

VisibilitySensor

特定の領域が視点から見えるようになったことを知らせます。

Collision

視点（アバター）がオブジェクトと衝突したことを知らせます。これはグループノードです。

Anchor ノード

書式

```
Anchor {
	url		リンクする URL
	parameter	リンク先に渡す追加情報
	description	説明
	children	URL を結びつけるオブジェクト
	bboxCenter	外接箱の中心位置
	bboxSize	外接箱の大きさ
}
```

例

```
#VRML V2.0 utf8
Anchor {
	url “another.wrl”	# 他の VRML ファイル。HTML ファイルなども指定可。
	description “Link to another VRML file”
	children [
		Shape {
			geometry Sphere {}
			appearance Appearance {
				material Material {}
			}
		}
	]
}
```


TouchSensor ノード

書式

```
TouchSensor {
	enabled		使用可(TRUE)/使用不可(FALSE)
}
```

送出イベント

```
hitPoint_changed	マウスがある場所の三次元位置
hitNormal_changed	マウスがある場所の法線ベクトル
hitTexCoord_changed	マウスがある場所のテクスチャ座標
isActive		マウスでオブジェクトをクリックしている間 TRUE
isOver			マウスがオブジェクトの上に来たら TRUE
touchTime		マウスでオブジェクトをクリックした時間
```

例

```
#VRML V2.0 utf8
Group {
	children [
		DEF TS TouchSensor {}
		Shape {
			geometry Sphere {}
			appearance Appearance {
				material Material {}
			}
		}
	]
}
DEF SL SpotLight {
	direction 0 0 -1
	position 0 0 2
	on FALSE
}
ROUTE TS.isActive TO SL.set_on
```

SphereSensor ノード

書式

```
Sphere {
	enabled		使用可(TRUE)/使用不可(FALSE)
	autoOffset	回転角を保存(TRUE)/保存しない(FALSE)
	offset		x y z r （回転軸ベクトルと回転角）
}
```

送出イベント

```
isActive		マウスでオブジェクトをクリックしている間 TRUE
rotation_changed	オブジェクトの回転角
trackpoint_changed	オブジェクトを回転させているときの１マウスの位置
```

例

```
#VRML V2.0 utf8
Group {
	children [
		DEF SS SphereSensor {}
		DEF HAKO Transform {
			children [
				Shape {
					geometry Box {}
					appearance Appearance {
						material Material {}
					}
				}
			]
		}
```

```
        ]
    }
    ROUTE SS.rotation_changed TO HAKO.set_rotation
```

CylinderSensor ノード

書式

```
    CylinderSensor {
        enabled       使用可(TRUE)/使用不可(FALSE)
        diskAngle     円柱を上・下面から操作するか側面から操作するか切り替える角度
        maxAngle      回転角の上限
        minAngle      回転角の下限
        autoOffset    回転角を保存(TRUE)/保存しない(FALSE)
        offset        回転角
    }
```

送出イベント

```
    isActive            マウスでオブジェクトをクリッしている間 TRUE
    rotation_changed    オブジェクトの回転角
    trackpoint_changed  オブジェクトを回転させているときの１マウスの位置
```

例

```
    #VRML V2.0 utf8
    Group {
        children [
            DEF CS CylinderSensor {}
            DEF HAKO Transform {
                children [
                    Shape {
                        geometry Box {}
                        appearance Appearance {
                            material Material {}
                    }
                ]
            }
        ]
    }
    ROUTE CS.rotation_changed TO HAKO.set_rotation
```

PlaneSensor ノード

書式

```
    PlaneSensor {
        enabled       使用可(TRUE)/使用不可(FALSE)
        maxPosition   位置の上限
        minPosition   位置の下限
        autoOffset    位置を保存(TRUE)/保存しない(FALSE)
        offset        位置
    }
```

送出イベント

```
    isActive            マウスでオブジェクトをクリッしている間 TRUE
    translation_changed オブジェクトの位置
    trackpoint_changed  オブジェクトを移動させているときの１マウスの位置
```

例

```
    #VRML V2.0 utf8
    Transform {                 # Billboard にすれば常に視点側を向く
        rotation 1 0 0 0.5  # 今は PlaneSensor を傾けてみる
        children [
            DEF PS PlaneSensor {}
            DEF HAKO Transform {
                children [
                    Shape {
                        geometry Box {}
                        appearance Appearance {
                            material Material {}
                    }
                ]
            }
        ]
    }
    ROUTE PS.translation_changed TO HAKO.set_translation
```


ProximitySensor ノード

書式

```
    ProximitySensor {
        enabled       使用可(TRUE)/使用不可(FALSE)
        center        領域の中心位置
        size          領域の大きさ
    }
```

送出イベント

```
    isActive            視点が領域内に入ったら TRUE
    enterTime           視点が領域内に入った時間
    exitTime            視点が領域外に出た時間
    position_changed    領域内での視点の位置
    orientation_changed 領域内での視点の向き
```

例

```
    #VRML V2.0 utf8
    DEF PS ProximitySensor {
        center 0 0 0
        size 2 2 2
    }
    Group {
        children [
            DEF SS SphereSensor {}
            DEF HAKO Transform {
                children [
                    Shape {
                        geometry Box {}
                        appearance Appearance {
                            material Material {}
                    }
                ]
            }
        ]
    }
    # 領域内に入ったときだけ SphereSensor を有効にする
    ROUTE PS.isActive TO SS.enabled
    ROUTE SS.rotation_changed TO HAKO.set_rotation
```

VisibilitySensor ノード

書式

```
    VisibilitySensor {
        enabled       使用可(TRUE)/使用不可(FALSE)
        center        領域の中心位置
        size          領域の大きさ
    }
```

送出イベント

```
    isActive            視点が領域内に入ったら TRUE
    enterTime           視点が領域内に入った時間
    exitTime            視点が領域外に出た時間
```

例

```
#VRML V2.0 utf8
DEF VS ProximitySensor {
	center 0 0 0
	size 2 2 2
}
Group {
	children [
		DEF SS SphereSensor {}
		DEF HAKO Transform {
			children [
				Shape {
					geometry Box {}
					appearance Appearance {
						material Material {}
					}
				}
			]
		}
	]
}
# 領域が見えるときだけ SphereSensor を有効にする
ROUTE VS.isActive TO SS.enabled
ROUTE SS.rotation_changed TO HAKO.set_rotation
```

Collision ノード

書式

```
Collision {
	collide		衝突検知する(TRUE)/しない(FALSE)
	children	衝突検出するオブジェクト
	proxy		衝突検出するが表示しないオブジェクト
	bboxCenter	外接箱の中心位置
	bboxSize	外接箱の大きさ
}
```

送出イベント

```
	collideTime		衝突した時間
```

例

```
#VRML V2.0 utf8
Collision {
	children [
		Shape {
			geometry Box {
				size 5 5 1
			}
			appearance Appearance {
				material Material {}
			}
		}
	]
}
```


演習

太字のところを打ち込んでください。各ステップごとに「更新」して、画面で表示を確認してください。

球をひとつ作ってみます。 できたら.wrlという拡張子を付けたファイル名で保存し、Web ブラウザで見てみましょう。	**#VRML V2.0 utf8** **Shape {** **　geometry Sphere {}** **　appearance Appearance {** **　　material Material {** **　　　diffuseColor 1 0 0** **　　}** **　}** **}**

この球を Anchor ノードの子供にします。
 	
Anchor ノードの url フィールドに、どこかの Web ページの URL を指定してみましょう。
 	
ブラウザを「更新」して、球をクリックしてみましょう。
	#VRML V2.0 utf8
	Anchor {
	**　url "http://www.yahoo.co.jp/"**
	**　description "Yahoo! Japan"**
	**　children [**
	　　Shape {
	　　　geometry Sphere {}
	　　　appearance Appearance {
	　　　　material Material {
	　　　　　diffuseColor 1 0 0
	　　　　}
	　　　}
	　　}
	**　]**
	}

球をもうひとつ追加します。この球は最初の球と重ならないように、右にずらします。 既に入力している VRML ファイルの最後に、右の内容を追加してください。	**Transform {** **　translation 3 0 0** **　children [** **　　Shape {** **　　　geometry Sphere {}** **　　　appearance Appearance {** **　　　　material Material {** **　　　　　diffuseColor 0 0 1** **　　　　}** **　　　}** **　　}** **　]** **}**

スポットライトでこの球を照らします。	Transform {
	　translation 3 0 0
	　children [
	　　Shape {
	　　　geometry Sphere {}
	　　　appearance Appearance {
	　　　　material Material {
	　　　　　diffuseColor 0 0 1
	　　　　}
	　　　}
	　　}
	　]
	}
	
	SpotLight {
	**　location 3 0 3**
	**　direction 0 0 -1**
	}

でも、このスポットライトは消してしまいます。

```
Transform {
    translation 3 0 0
    children [
        Shape {
            geometry Sphere {}
            appearance Appearance {
                material Material {
                    diffuseColor 0 0 1
                }
            }
        }
    ]
}

SpotLight {
    location 3 0 3
    direction 0 0 –1
    on FALSE
}
```

次に、球と同じグループ（Transform ノードはグループ化ノードのひとつ）に、TouchSensor ノードを追加します。

この TouchSensor ノードには名前を付けておきます。ここでは TS としておきます。

同時に、SpotLight ノードにも名前を付けておきます。こちらは SL としておきます。

```
Transform {
    translation 3 0 0
    children [
        DEF TS TouchSensor {}
        Shape {
            geometry Sphere {}
            appearance Appearance {
                material Material {
                    diffuseColor 0 0 1
                }
            }
        }
    ]
}

DEF SL SpotLight {
    location 3 0 3
    direction 0 0 –1
    on FALSE
}
```

TouchSensor ノードは、同じグループに属する形状ノードがクリックされたときに、isActive というイベントに対して TRUE を送出します。

これを SpotLight の on フィールドにセットするために、ROUTE コマンドを追加します。

これによって、右側の青い球をクリックしたときに、スポットライトが点灯するようなります。

```
Transform {
    translation 3 0 0
    children [
        DEF TS TouchSensor {}
        Shape {
            geometry Sphere {}
            appearance Appearance {
                material Material {
                    diffuseColor 0 0 1
                }
            }
        }
    ]
}

DEF SL SpotLight {
    location 3 0 3
    direction 0 0 –1
    on FALSE
}

ROUTE TS.isActive TO SL.set_on
```


TouchSensor ノードの isActive イベントの代わりに isOver イベントを用いると、マウスポインタが青い球の上を通過するだけで（クリックしなくても）、スポットライトが点灯するようになります。

isActive を isOver に書き換えてみてください。

```
Transform {
    translation 3 0 0
    children [
        DEF TS TouchSensor {}
        Shape {
            geometry Sphere {}
            appearance Appearance {
                material Material {
                    diffuseColor 0 0 1
                }
            }
        }
    ]
}

DEF SL SpotLight {
    location 3 0 3
    direction 0 0 –1
    on FALSE
}

ROUTE TS.isOver TO SL.set_on
```

今度は箱をひとつ追加します。この箱は左にずらします。

既に入力している VRML ファイルの最後に、右の内容を追加してください。

```
Transform {
    translation -3 0 0
    children [
        Shape {
            geometry Box {}
            appearance Appearance {
                material Material {
                    diffuseColor 0 1 0
                }
            }
        }
    ]
}
```

これを、さらに Transform ノードの子供にします。

このとき、この Transform ノードには、translation フィールドや rotation フィールドを指定しません。

したがって、この Transform ノードは、図形に対して何の座標変換も行いません。

そのかわり、この Transform ノードには名前を付けておきます。ここでは SP という名前にします。

```
DEF SP Transform {
    children [
        Transform {
            translation -3 0 0
            children [
                Shape {
                    geometry Box {}
                    appearance Appearance {
                        material Material {
                            diffuseColor 0 1 0
                        }
                    }
                }
            ]
        }
    ]
}
```

そして、この箱と同じグループに PlaneSensor {} ノードを追加します。この PlaneSensor ノードには PS という名前を付けておきます。

PlaneSensor ノードは、同じグループに属する形状ノードをドラッグしたときに、translation_changed

```
DEF SP Transform {
    children [
        DEF PS PlaneSensor {}
        Transform {
            translation -3 0 0
            children [
                Shape {
                    geometry Box {}
                    appearance Appearance {
```

たときに、translation_changed というイベントに対してその形状ノードの位置を送出します。
 	
ROUTE コマンドを使って、これを Transform ノードの translation フィールドにセットします。
 	
これによって、左側の緑の箱の位置をマウスで移動できるようになります。
	material Material {
	diffuseColor 0 1 0
	}
	}
	}
	]
	}
	]
	}
	
	ROUTE PS.translation_changed TO SP.set_translation

SphereSensor ノードは同じグループに属する形状ノードをドラッグしたときに、rotation_changed というイベントに対してその形状ノードの回転角を送出します。
 	
ROUTE コマンドでは、これを Transform ノードの rotation フィールドにセットします。
 	
つまり、この場合は箱を回転させることができます。
 	
PlaneSensor を SphereSensor に、translation_changed を rotation_changed に、set_translation を set_rotation に書き換えてください。
	DEF SP Transform {
	children [
	DEF PS **SphereSensor** {}
	Transform {
	translation -3 0 0
	children [
	Shape {
	geometry Box {}
	appearance Appearance {
	material Material {
	diffuseColor 0 1 0
	}
	}
	}
	]
	}
	]
	}
	
	ROUTE PS.**rotation**_changed TO SP.set_**rotation**

この場合は SphereSensor ノードと同じグループの Transform ノードに対してイベントを送るようにしないと、回転の中心がずれてしまいます。DEF SP を移動してください	DEF SP Transform {
	children [
	DEF PS SphereSensor {}
	DEF SP Transform {
	translation -3 0 0
	children [
	Shape {
	（以下略）

なお、CylinderSensor も回転角を送出しますが、軸は固定されます。
 	
SphereSensor を CylinderSensor に書き換えてください。
	Transform {
	children [
	DEF PS **CylinderSensor** {}
	DEF SP Transform {
	translation -3 0 0
	children [
	Shape {
	（以下略）

緑の箱と同じグループに円柱を追加します。円柱が緑の箱に「刺さる」ように回転・移動します。
 	
こうすると、緑の箱の回転といっしょに円柱も回転するようになります。
	Transform {
	children [
	DEF PS CylinderSensor {}
	DEF SP Transform {
	translation -3 0 0
	children [
	Shape {
	geometry Box {}
	appearance Appearance {
	material Material {
	diffuseColor 0 1 0
	}
	}

	} **Transform {** **rotation 0 0 1 1.5708** **translation 3 0 0** **children [** **Shape {** **geometry Cylinder {** **radius 0.3** **height 2** **}** **appearance Appearance {** **material Material {}** **}** **}** **]** **}**] }] } ROUTE PS.rotation_changed TO SP.set_rotation

そこで、この円柱の位置を設定している Transform ノードに生をつけます。ここでは CY とします。
 	
また、この円柱と同じグループに SphereSensor ノードを追加します。これにも SS という名前を付けておきます。
 	
繰り返しますが、DEF でつける名前は任意です。この場合の CY や SS も特別な名称ではなく、これにこだわる必要はありません。
 	
そして、ROUTE コマンドを使って SS で発生した回転のイベントを CY に送ります。
 	
こうすると、緑の箱を回転させれば円柱もいっしょに回転しますが、円柱だけを回転させることもできるようになります。
	Transform {
	children [
	DEF PS CylinderSensor {}
	DEF SP Transform {
	translation -3 0 0
	children [
	Shape {
	geometry Box {}
	appearance Appearance {
	material Material {
	diffuseColor 0 1 0
	}
	}
	}
	DEF CY Transform {
	rotation 0 0 1 1.5708
	translation 3 0 0
	children [
	DEF SS SphereSensor {}
	Shape {
	geometry Cylinder {
	radius 0.3
	height 2
	}
	appearance Appearance {
	material Material {}
	}
	}
	]
	}
	]
	}
	]
	}
	
	ROUTE PS.rotation_changed TO SP.set_rotation
	ROUTE SS.rotation_changed TO CY.set_rotation

アニメーション

概要

TimeSensor によって時間経過のイベント（０～１）を発生させ、それを用いて位置や角度などを線形補間し、アニメーションを実現します。

TimeSensor

時間の経過とともにイベントを生成します。

ScalarInterpolator

radius フィールドのような、単独の値を補間します。

PositionInterpolator

位置を補間します。Transform ノードの translation フィールドなどを変化させるのに使います。

OrientationInterpolator

回転角を補間します。Transform ノードの rotation フィールドなどを変化させるのに使います。

ColorInterpolator

色を補間します。Material ノードの diffuseColor フィールドなどを変化させるのに使います。

CoordinateInterpolator

座標値を補間します。Coordinate ノードの point フィールドを変化させるのに使います。

NormalInterpolator

法線ベクトルを補間します。Normal ノードの vector フィールドを変化させるのに使います。

TimeSensor ノード

書式

```
TimeSensor {
        enabled         動作(TRUE)/停止(FALSE)
        cycleInterval   イベントを発生させる時間（長さ）
        loop            繰り返す(TRUE)/繰り返さない(FALSE)
        startTime       開始時間
        stopTime        終了時間
}
```

出力イベント

```
        isActive                TimeSensor が動いていれば TRUE
        fraction_changed        経過時間
        cycleTime               現在サイクルの開始時間
        time                    現在時刻
```

ScalarInterpolator ノード

書式

```
ScalarInterpolator {
        key             キー
        keyValue        キー値
}
```

出力イベント

```
        value_changed           変化した値
```


例

```
#VRML V2.0 utf8
DEF TIMER TimeSensor {
        cycleInterval 2
}
DEF SI ScalarInterpolator {
        key             [0.0 0.5 1.0]
        keyValue        [1.0 2.0 1.0] # 最初１で１秒後に２になり２秒後に１に戻る
}
Group {
        children [
                DEF TS TouchSensor {}
                DEF SP Shape {
                        geometry Sphere {}
                        appearance Appearance {
                                material Material {}
                        }
                }
        ]
}
ROUTE TS.touchTime TO TIMER.startTime           # クリックでタイマ起動
ROUTE TIMER.fraction_changed TO SI.set_fraction# タイマの経過時間を SI に
ROUTE SI.value_changed to SP.set_radius         # SI による補間値を radius
```

PositionInterpolator ノード

書式

```
PositionInterpolator {
        key             キー
        keyValue        キー値
}
```

出力イベント

```
        value_changed           変化した値
```

例

```
#VRML V2.0 utf8
DEF TIMER TimeSensor {
        loop TRUE
        cycleInterval 2
}
DEF PI PositionInterpolator {
        key     [0.0 0.25 0.5 0.75 1.0]
        keyValue [
                 1.0    0.0    0.0,
                 0.0    0.0    1.0,
                -1.0    0.0    0.0,
                 0.0    0.0   -1.0,
                 1.0    0.0    0.0
        ]
}
DEF PLANET Transform {
        translation 1.0 0.0 0.0
        children [
                Shape {
                        geometry Sphere {}
                        appearance Appearance {
                                material Material {}
                        }
                }
        ]
}
ROUTE TIMER.fraction_changed TO PI.set_fraction
ROUTE PI.value_changed TO PLANET.set_translation
```

OrientationInterpolator ノード

書式

```
OrientationInterpolator {
	key		キー
	keyValue	キー値
}
```

出力イベント

```
value_changed		変化した値
```

例

```
#VRML V2.0 utf8
DEF TIMER TimeSensor {
	cycleInterval 3
	loop TRUE
}
DEF OI OrientationInterpolator {
	key	[0.0 0.3333 0.6667 1.0]
	keyValue [
		0.0	1.0	0.0	0.0,
		0.0	1.0	0.0	2.0944,
		0.0	1.0	0.0	4.1888,
		0.0	1.0	0.0	0.0
	]
}
DEF DICE Transform {
	translation 1.0 0.0 0.0
	children [
		Shape {
			geometry Box {}
			appearance Appearance {
				material Material {}
			}
		}
	]
}
ROUTE TIMER.fraction_changed TO OI.set_fraction
ROUTE OI.value_changed TO DICE.set_translation
```

CoordinateInterpolator ノード

書式

```
CoordinateInterpolator {
	key		キー
	keyValue	キー値
}
```

出力イベント

```
value_changed		変化した値
```

例

```
#VRML V2.0 utf8
DEF TIMER TimeSensor {
	cycleInterval 3
	loop TRUE
}
DEF CI CoordinateInterpolator {
	key	[0.0 0.5 1.0]
	keyValue [
		# key:0 の時の頂点位置
		 0.0	2.0	0.0,
		 2.0	0.0	2.0,
		 2.0	0.0	-2.0,
```



```
		 -2.0	0.0	-2.0,
		 -2.0	0.0	2.0,

		# key:0.5 の時の頂点位置
		 0.0	4.0	0.0,
		 1.0	0.0	1.0,
		 1.0	0.0	-1.0,
		 -1.0	0.0	-1.0,
		 -1.0	0.0	1.0,

		# key:1 の時の頂点位置
		 0.0	2.0	0.0,
		 2.0	0.0	2.0,
		 2.0	0.0	-2.0,
		 -2.0	0.0	-2.0,
		 -2.0	0.0	2.0,
	]
}
Shape {
	geometry IndexedFaceSet {
		coord DEF PYRAMID Coordinate {
			point [
				# key:0 の時の頂点位置
				 0.0	2.0	0.0,
				 2.0	0.0	2.0,
				 2.0	0.0	-2.0,
				 -2.0	0.0	-2.0,
				 -2.0	0.0	2.0,
			]
		}
		coordIndex [
			0 1 2 0 -1
			0 2 3 0 -1
			0 3 4 0 -1
			0 4 1 0 -1
			4 3 2 1 4 -1
		]
	}
	appearance Appearance {
		material Material {}
	}
}
ROUTE TIMER.fraction_changed TO CI.set_fraction
ROUTE CI.value_changed TO PYRAMID.set_point
```

演習

太字のところを打ち込んでください。各ステップごとに「更新」して、画面で表示を確認してください。

箱をひとつ作ってみます。この箱は Transform ノードの子供にします。フィールドは指定しないでおきます。

この Transform ノードに名前を付けておきます。ここでは HAKO としておきます。

できたら.wrl という拡張子を付けたファイル名で保存し、Web ブラウザで見てみましょう。

```
#VRML V2.0 utf8
DEF HAKO Transform {
    children [
        Shape {
            geometry Box {}
            appearance Appearance {
                material Material {
                    diffuseColor 0.2 0.8 0.4
                }
            }
        }
    ]
}
```

これに TimeSensor ノードを追加し、JIKAN という名前を付けておきます。

loop フィールドを TRUE にして、この TimeSensor を動かしっぱなしにしておきます。

しかし、TimeSensor ノードを追加しただけでは何も起こりません。

```
#VRML V2.0 utf8
DEF HAKO Transform {
（中略）
}

DEF JIKAN TimeSensor {
    loop TRUE
}
```

さらに PotisonInterpolator（位置の補間子）ノードを追加します。

key フィールドには０～１の間の値を列挙します。

keyValue フィールドには、key フィールドの個々の値に対応した座標値を設定します。この例では、次のように対応しています。

```
key:0   → (0, 0, 0)
key:0.5 → (0, 2, 0)
key:1   → (0, 0, 0)
```

```
#VRML V2.0 utf8
DEF HAKO Transform {
（中略）
}

DEF JIKAN TimeSensor {
    loop TRUE
}

DEF KISEKI PositionInterpolator {
    key [ 0 0.5 1 ]
    keyValue [ 0 0 0, 0 2 0, 0 0 0 ]
}
```

TimeSensor ノード(JIKAN)で発生した fraction_changed というイベントを PositionInterpolator ノード(KISEKI)の fraction フィールドにセットしてやります。

そして、その結果として発生した PositionInterpolator ノード(KISEKI)の value_changed というイベントを Transform ノード(HAKO)の translation フィールドにセットしてやります。

箱は動き出したでしょうか？

```
#VRML V2.0 utf8
DEF HAKO Transform {
（中略）
}

DEF JIKAN TimeSensor {
    loop TRUE
}

DEF KISEKI PositionInterpolator {
    key [ 0 0.5 1 ]
    keyValue [ 0 0 0, 0 2 0, 0 0 0 ]
}

ROUTE JIKAN.fraction_changed TO KISEKI.set_fraction
ROUTE KISEKI.value_changed TO HAKO.set_translation
```


箱のスピードを遅くしてみます。

TimeSensor ノードに cycleInterval フィールドを指定して、１周期に４秒かかるようにしてみてください。

cycleInterval フィールドは、TimeSensor ノードから送出される fraction_chanaged イベントの値が、０→１に変化するのに要する時間を指定します。

cycleInterval フィールドを省略したときは１秒です。

```
#VRML V2.0 utf8
DEF HAKO Transform {
（中略）
}

DEF JIKAN TimeSensor {
    loop TRUE
    cycleInterval 4
}

DEF KISEKI PositionInterpolator {
    key [ 0 0.5 1 ]
    keyValue [ 0 0 0, 0 2 0, 0 0 0 ]
}

ROUTE JIKAN.fraction_changed TO KISEKI.set_fraction
ROUTE KISEKI.value_changed TO HAKO.set_translation
```

PositionInterpolator ノードの key フィールドに 0.25 と 0.75 という値を追加してください。

そして、これらの値に対応した座標地を、keyValue フィールドに追加してください。

今度はどういう動き方になったでしょうか。

```
#VRML V2.0 utf8
DEF HAKO Transform {
（中略）
}

DEF JIKAN TimeSensor {
    loop TRUE
    cycleInterval 4
}

DEF KISEKI PositionInterpolator {
    key [ 0 0.25 0.5 0.75 1 ]
    keyValue [ 0 0 0, 1 1 0, 0 2 0, -1 1 0, 0 0 0 ]
}

ROUTE JIKAN.fraction_changed TO KISEKI.set_fraction
ROUTE KISEKI.value_changed TO HAKO.set_translation
```

今度は TimeSensor ノードの loop フィールドを FALSE にしてみます。

箱の動きは止まるはずです。

```
#VRML V2.0 utf8
DEF HAKO Transform {
（中略）
}

DEF JIKAN TimeSensor {
    loop FALSE
    cycleInterval 4
}

DEF KISEKI PositionInterpolator {
    key [ 0 0.25 0.5 0.75 1 ]
    keyValue [ 0 0 0, 1 1 0, 0 2 0, -1 1 0, 0 0 0 ]
}

ROUTE JIKAN.fraction_changed TO KISEKI.set_fraction
ROUTE KISEKI.value_changed TO HAKO.set_translation
```

箱の左下に青い球を置いてみます。

この球には TouchSensor ノードを取り付け(すなわち Sphere ノードと同じ親の children フィールドに TouchSensor ノードを置く)、それに HAJIME という名前を付けます。

そして、この TouchSensor ノード(HAJIME)がアクティブになった(マウスでクリックした)時間のイベント touchTime を、TimeSensor ノード(JIKAN)の startTime に送ります。

TimeSensor ノード(JIKAN)は、loop フィールドが FALSE なので、最初は止まっています。

この startTime に、TouchSensor ノード(HAJIME)のイベントの発生時間を送りつけることで、イベントが派生した時間、すなわちクリックした時間から、TimeSensor ノードが動き始めます。

青い球をクリックしてみてください。

```
#VRML V2.0 utf8
DEF HAKO Transform {
 (中略)
}

DEF JIKAN TimeSensor {
   loop FALSE
   cycleInterval 4
}

DEF KISEKI PositionInterpolator {
   key [ 0 0.25 0.5 0.75 1 ]
   keyValue [ 0 0 0, 1 1 0, 0 2 0, -1 1 0, 0 0 0 ]
}

ROUTE JIKAN.fraction_changed TO KISEKI.set_fraction
ROUTE KISEKI.value_changed TO HAKO.set_translation

Transform { # スタート用の球
   translation -3 -3 0
   children [
      Shape {
         geometry Sphere { radius 0.5 }
         appearance Appearance {
            material Material {
               diffuseColor 0.2 0.2 0.8
            }
         }
      }
      DEF HAJIME TouchSensor {}
   ]
}

ROUTE HAJIME.touchTime TO JIKAN.startTime
```

止めることもできます。右下にストップ用のスイッチとして使う赤い球を追加します。

これにも TouchSensor ノードを取り付け、OWARI という名前を付けます。

この stopTime に、TouchSensor ノード(OWARI)のイベントの発生時間を送りつけることで、イベントが派生した時間、すなわちクリックした時間に、TimeSensor ノードが止まり、箱の動きが止まります。

```
#VRML V2.0 utf8
DEF HAKO Transform {
 (中略)
}

DEF JIKAN TimeSensor {
   loop FALSE
   cycleInterval 4
}

DEF KISEKI PositionInterpolator {
   key [ 0 0.25 0.5 0.75 1 ]
   keyValue [ 0 0 0, 1 1 0, 0 2 0, -1 1 0, 0 0 0 ]
}

ROUTE JIKAN.fraction_changed TO KISEKI.set_fraction
ROUTE KISEKI.value_changed TO HAKO.set_translation

Transform { # スタート用の球
 (中略)
}

ROUTE HAJIME.touchTime TO JIKAN.startTime

Transform { # ストップ用の球
   translation 3 -3 0
   children [
      Shape {
         geometry Sphere { radius 0.5 }
         appearance Appearance {
            material Material {
               diffuseColor 0.8 0.2 0.2
            }
         }
      }
      DEF OWARI TouchSensor {}
   ]
}

ROUTE OWARI.touchTime TO JIKAN.stopTime
```


今度は箱に回転を加えてみましょう。

OrientationInterpolator(方向の補間子)ノードを追加します。これに KAITEN という名前を付けておきます。

key フィールドには0～1の間の値を列挙します。

keyValue フィールドには、key フィールドの値に対応した回転軸と回転角を列挙します。この例では、次のように対応しています。

key:0 → (0, 0, 1), 0
key:0.5 → (0, 0, 1), π
key:1 → (0, 0, 1), 2π

回転はZ軸中心であり、TimeSensor の1周期でちょうど1回転するように回転角を設定しています(ただし、この例では回転方向が明らかではないので、回転角はもう少し刻みを細かく指定したほうがいいでしょう)。

```
#VRML V2.0 utf8
DEF HAKO Transform {
 (中略)
}

DEF JIKAN TimeSensor {
   loop FALSE
   cycleInterval 4
}

DEF KISEKI PositionInterpolator {
   key [ 0 0.25 0.5 0.75 1 ]
   keyValue [ 0 0 0, 1 1 0, 0 2 0, -1 1 0, 0 0 0 ]
}

ROUTE JIKAN.fraction_changed TO KISEKI.set_fraction
ROUTE KISEKI.value_changed TO HAKO.set_translation

Transform { # スタート用の球
 (中略)
}

ROUTE HAJIME.touchTime TO JIKAN.startTime

Transform { # ストップ用の球
 (中略)
}

ROUTE OWARI.touchTime TO JIKAN.stopTime

DEF KAITEN OrientationInterpolator {
   key [ 0 0.5 1 ]
   keyValue [ 0 0 1 0, 0 0 1 3.1416, 0 0 1 6.2832 ]
}

ROUTE JIKAN.fraction_changed TO KAITEN.set_fraction
ROUTE KAITEN.value_changed TO HAKO.set_rotation
```

IndexedFaceSet ノードは多面体の頂点を Coordinate ノードで指定します。

このCoordinateノードのpointフィールドに列挙している座標値を、CoordinateInterpolator ノードを使って補間してやれば、モーフィングのようなこともできます。

以前に作成した５面体（ピラミッド）を追加してみてください。ただし、ちょっと高さを下げておきます。

```
#VRML V2.0 utf8
DEF HAKO Transform {
 （中略）
}

 （中略）

DEF KAITEN OrientationInterpolator {
   key [ 0 0.5 1 ]
   keyValue [ 0 0 1 0, 0 0 1 3.1416, 0 0 1 6.2832 ]
}

ROUTE JIKAN.fraction_changed TO KAITEN.set_fraction
ROUTE KAITEN.value_changed TO HAKO.set_rotation

Transform {
   translation 0 -3 0
   children [
      Shape {
         geometry IndexedFaceSet {
            coord Coordinate {
               point [
                  0.0   2.0   0.0,
                  2.0   0.0   2.0,
                  2.0   0.0   -2.0,
                  -2.0  0.0   -2.0,
                  -2.0  0.0   2.0,
               ]
            }
            coordIndex [
               0 1 2 0 -1
               0 2 3 0 -1
               0 3 4 0 -1
               0 4 1 0 -1
               4 3 2 1 4 -1
            ]
         }
         appearance Appearance {
            material Material {}
         }
      }
   ]
}
```

この Coordinate ノードに名前を付けます。ここではPYRAMIDとします。

さらに、この Coordinate ノードの point フィールドに指定した座標をもとに CoordinateInterpolator の座標値を決めます。

CoordinateInterpolator の場合は key 値に対応する keyValue の値の数が、対応する IndexedFaceSet ノードの形状によって決まります。

```
#VRML V2.0 utf8
DEF HAKO Transform {
 （中略）
}

 （中略）

DEF KAITEN OrientationInterpolator {
   key [ 0 0.5 1 ]
   keyValue [ 0 0 1 0, 0 0 1 3.1416, 0 0 1 6.2832 ]
}

ROUTE JIKAN.fraction_changed TO KAITEN.set_fraction
ROUTE KAITEN.value_changed TO HAKO.set_rotation

Transform {
   translation 0 -3 0
   children [
      Shape {
         geometry IndexedFaceSet {
            coord DEF PYRAMID Coordinate {
               point [
                  0.0   2.0   0.0,
                  2.0   0.0   2.0,
                  2.0   0.0   -2.0,
                  -2.0  0.0   -2.0,
                  -2.0  0.0   2.0,
               ]
            }
            coordIndex [
               0 1 2 0 -1
               0 2 3 0 -1
               0 3 4 0 -1
               0 4 1 0 -1
               4 3 2 1 4 -1
            ]
         }
         appearance Appearance {
            material Material {}
         }
      }
   ]
}

DEF HENKEI CoordinateInterpolator {
   key[0.0 0.5 1.0]
   keyValue [

      # key:0 の時の頂点位置（もとの形）
       0.0   2.0   0.0,
       2.0   0.0   2.0,
       2.0   0.0   -2.0,
       -2.0  0.0   -2.0,
       -2.0  0.0   2.0,

      # key:0.5 の時の頂点位置
       0.0   4.0   0.0,
       1.0   0.0   1.0,
       1.0   0.0   -1.0,
       -1.0  0.0   -1.0,
       -1.0  0.0   1.0,

      # key:1 の時の頂点位置
       0.0   2.0   0.0,
       2.0   0.0   2.0,
       2.0   0.0   -2.0,
       -2.0  0.0   -2.0,
       -2.0  0.0   2.0,
   ]
}

ROUTE JIKAN.fraction_changed TO HENKEI.set_fraction
ROUTE HENKEI.value_changed TO PYRAMID.set_point
```