VRML 実習マニュアル

version 1.00
2003 年 5 月 3 日（土）

1　VRML についての基礎知識

1.1　Web3D

ウェブというものは、さまざまな種類のデータから構成されています。テキストを始めとして、静止画、動画、音声、アーカイブ、ベクターグラフィックスなど、きわめて多種多様です。そしてさらに、ウェブを構成するデータの種類のひとつとして、「Web3D」と呼ばれるものもあります。

Web3D は、3 次元グラフィックスの一種です。しかしそれは、ただ単に眺めることができるだけのグラフィックスではありません。それは、3 次元の仮想的な世界になっていて、ユーザーは、その世界の中を自由に動き回ることができます。そしてさらに、Web3D では、時間とともに変化していくような世界を作ることもできますし、その中にあるものをユーザーが自由に操作することができるような世界を作ることも可能です。

1.2　VRML とは何か

Web3D を作るためのデータ形式としては、さまざまなものが使われているのですが、それらのうちのひとつに、「VRML」と呼ばれるものがあります（ちなみに、VRML というのは、Virtual Reality Modeling Language という言葉の頭字語です）。

VRML は、ISO(International Standardization Organization) と IEC(International Electrotechnical Commission) という二つの公的な機関によって標準規格として定められたデータ形式です。VRML の規格書は、

Web3D Consortium　`http://www.web3d.org/`

というサイトで公開されています。

なお、VRML は、次期バージョンから「X3D」という名前に変わる予定になっています。

1.3　VRML 文書

VRML は、言語の一種です。このことは、VRML というデータ形式を持つ文書、すなわち VRML 文書は、XHTML や CSS などの文書と同じように、人間が書いたり読んだりすることのできるテキストだということを意味しています。したがって、VRML 文書を書くために特別なソフトを準備する必要はありません。つまり、テキストエディターさえあれば、VRML 文書を書くことができるということです。

それでは、実際に VRML 文書を書いて、それをブラウザーに表示させてみましょう。何らかのテキストエディターを使って、次の VRML 文書を入力して、それを `shape.wrl` というファイル名で保存してください。

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 1 1 }
    }
    geometry Cylinder {
        radius 4
        height 3
    }
}
```

なお、VRML 文書を格納するファイルの名前には、普通、このように .wrl という拡張子を付けます。

次に、この VRML 文書をウェブのブラウザーで開いてみてください。ブラウザーに VRML のプラグインがインストールされているならば、ブラウザーのウィンドウの中に水色の円柱が表示されるはずです。

VRML のプラグインがインストールされていないために VRML 文書が表示できないという場合は、プラグインをダウンロードしてインストールしてください。たとえば、

Cosmo Software　http://www.cai.com/cosmo/

というサイトから、Cosmo Player という VRML のプラグインをダウンロードすることができます。

1.4 基本的な文法

VRML 文書は、先頭に「ヘッダー」(header) と呼ばれる行があって、その下に「文」(statement) と呼ばれるものが何個か並ぶ、という構造になっています。

VRML の現在のバージョンでは、ヘッダーは、

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

と書くことになっています。V2.0 というのは VRML の現在のバージョン番号で、utf8 というのは文字コードの名前です。

文にはいくつかの種類があるのですが、それらのうちでもっとも重要なのは、「ノード文」と呼ばれる文です。

VRML によって作られる世界は、「ノード」(node) と呼ばれるさまざまな要素から構成されます。「ノード文」(node statement) というのは、それぞれのノードについて記述している文のことです。

ノード文は、基本的には、

ノード型名 { ノード本体要素 ··· }

というように書きます。「ノード型名」(node type name) というのは、ノードの種類を識別する名前のことです。たとえば、先ほどの VRML 文書の中には、Shape、Appearance、Material、Cylinder という 4 種類のノード型名が書かれています。このように、ノード型名は、かならず英字の大文字で始まります。

ノードはさまざまな属性を持っていて、それぞれの属性は「フィールド」(field) と呼ばれます。フィールドは、「フィールド名」(field name) と呼ばれる名前によって識別されます。たとえば、先ほどの VRML 文書の中には、appearance、material、diffuseColor、geometry、radius、height という 6 種類のフィールド名が書かれています。このように、フィールド名は、かならず英字の小文字で始まります。

フィールドに値を与えたいときは、ノード文の中に、「ノード本体要素」(node body element) と呼ばれるものを書きます。ノード本体要素は、

フィールド名 フィールド値

という構文になっています。このようなものを書くことによって、「フィールド値」(field value) のところに書かれたものが、フィールド名で指定されたフィールドに値として与えられます。

なお、先ほどの VRML 文書にも例があるように、フィールド値としてさらにノード文を書く、という場合もあります。つまり、ノード文というのは、ノード文の中にノード文があるという入れ子の構造を持つ場合があるということです。

1.5 物体の作り方

VRML では、物体は、Shape というノードによって作られます。物体を作るときには、Shape ノードが持っている appearance と geometry という二つのフィールドに値を与える必要があります。

geometry というのは、物体の形状を指定するフィールドです。物体の形状は、形状をあらわすノードを使って指定します。そのようなノードにはさまざまな種類があって、先ほどの VRML 文書では、 Cylinder というノードが使われています。

Cylinder は、形状として円柱を指定するためのノードです。 radius というフィールドに底面の半径を指定して、 height というフィールドに高さを指定します。たとえば、

```
Cylinder {
    radius 1.2
    height 5.3
}
```

というノード文を書くことによって、底面の半径が 1.2 で、高さが 5.3 であるような円柱を指定することができます。ちなみに、VRML の長さの単位はメートルですので、この場合の 1.2 や 5.3 という数値は、1.2 メートルや 5.3 メートルという長さをあらわしています。

appearance というのは、物体の視覚的な属性を指定するフィールドです。このフィールドに与える値は、 Appearance というノードです。

Appearance ノードは material というフィールドを持っていて、そのフィールドに Material というノードを値として与えることによって、物体の色を指定することができます。

Material ノードは diffuseColor というフィールドを持っていて、そのフィールドに、色をあらわす数値の列を値として与えることによって、物体の色が指定されます。

色は、赤、緑、青という光の三原色の強さをあらわす 3 個の数値の列で指定します。それぞれの数値の範囲は、0 から 1 までです。たとえば、

```
1 1 0
```

という数値の列は、黄色をあらわしています。

1.6 ホワイトスペース

空白、タブ、改行などの文字を総称して、「ホワイトスペース」(white space) と呼びます。VRML の文を書くとき、名前や数値や括弧の前後にはホワイトスペースを何個でも好きなだけ挿入することができて、それらのホワイトスペースによって文の意味が変化することはありません。ですから、

```
Material { diffuseColor 0.8 0.7 0.3 }
```

というノード文は、

```
Material {
    diffuseColor 0.8 0.7 0.3
}
```

と書いたとしても、同じ意味になります。

なお、名前または数値が連続する場合は、それらのあいだにかならず 1 個以上のホワイトスペースを挿入する必要があります。

1.7 注釈

VRML 文書のような、プログラムによって処理されるテキストを書くとき、そのテキストを処理するプログラムに対してではなくて、そのテキストを読む人間に対して何かを伝えたい、ということがしばしばあります。そのような場合には、テキストの中に「注釈」と呼ばれる文字列を書きます。

「注釈」(comment) というのは、プログラムによって処理されるテキストの中に含まれる、そのテキストを処理するプログラムから無視される文字列のことです。テキストの中に注釈を書く場合には、そのための文法にしたがう必要があります。

VRML 文書の中に注釈を記入したい場合は、シャープ (#) という文字を書きます。そうすると、その文字から次の改行までが注釈とみなされることになります。たとえば、VRML 文書の中に、

```
Material {
    diffuseColor 1 0.5 0   # orange
}
```

というノード文を書いたとすると、VRML 文書を処理するプログラムは、その中に含まれている、

```
# orange
```

という部分を注釈とみなして無視することになります。

1.8 デフォルト値

ノードが持っているそれぞれのフィールドは、「デフォルト値」と呼ばれる値を持っています。「デフォルト値」(default value) というのは、ノード本体要素によって値が与えられなかった場合に採用される値のことです。

たとえば、Cylinder ノードの場合、radius フィールドのデフォルト値は 1 で、height フィールドのデフォルト値は 2 です。ですから、

```
Cylinder {}
```

というノード文は、底面の半径が 1 で高さが 2 であるような円柱を意味しています。

また、Material ノードの diffuseColor フィールドのデフォルト値は、

```
0.8 0.8 0.8
```

です。ちなみに、これは明るい灰色をあらわしています。

VRML 文書の例 default.wrl

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

1.9 ノードの名前

ひとつの世界の中で、まったく同一のノードを何回も使う場合は、そのノードに名前を付けることによって、同じノード文を何回も書くという手間を省くことができます。

ノードに名前を付けたいときは、

DEF [名前] [ノード型名] { [ノード本体要素] ··· }

という形のノード文を書きます。そうすると、そのノード文があらわしているノードに、DEF の右側に書かれている名前が与えられます。たとえば、

```
DEF Orange Appearance {
    material Material { diffuseColor 1 0.5 0 }
}
```

というノード文を書くことによって、物体をオレンジ色にする Appearance ノードに、Orange という名前を与えることができます。

ノードの名前を使いたいときは、

USE [名前]

という形のものを書きます。たとえば、

```
Shape {
    appearance USE Orange
    geometry Sphere {}
```

```
}
```

というノード文を書いたとすると、その appearance フィールドには、 Orange という名前で参照される Appearance ノードが、値として与えられます。

VRML 文書の例 defuse.wrl

```
#VRML V2.0 utf8
DEF Orange Appearance {
    material Material { diffuseColor 1 0.5 0 }
}
Shape {
    appearance USE Orange
    geometry Sphere {}
}
Shape {
    appearance USE Orange
    geometry Cylinder {
        radius 0.3
        height 7
    }
}
```

2 物体

2.1 基本的な形状

物体の形状を指定するためには、物体の形状をあらわすノードを使う必要があります。前の節で登場した cylinder は、円柱という形状をあらわすノードです。

基本的な形状をあらわすノードは、 cylinder 以外にもいくつかあります。たとえば、直方体を指定する Box 、球を指定する Sphere 、円錐を指定する Cone などです。

直方体を指定する Box ノードは、直方体の大きさを指定する size というフィールドを持っています。このフィールドには、

size 幅 高さ 奥行

というように、3 個の数値の列を値として与えます。たとえば、

```
Box { size 5 8 3 }
```

というノード文は、幅が 5 で、高さが 8 で、奥行が 3 であるような直方体を意味しています。なお、 size フィールドのデフォルト値は、幅と高さと奥行がいずれも 2 です。

VRML 文書の例 box.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 1 0 }
    }
    geometry Box { size 1.2 0.8 1.8 }
}
```

球を指定する Sphere ノードは、 radius というフィールドを持っています。そのフィールドに値として数値を与えると、それが球の半径になります。たとえば、

```
Sphere { radius 0.7 }
```

というノード文は、半径が 0.7 の球を意味しています。なお、 radius フィールドのデフォルト値は 1 です。

VRML 文書の例 sphere.wrl

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

```
        appearance Appearance {
            material Material { diffuseColor 0 0.5 1 }
        }
        geometry Sphere { radius 3 }
    }
```

円錐を指定する Cone ノードは、円錐の底面の半径を指定する bottomRadius というフィールドと、円錐の高さを指定する height というフィールドを持っています。たとえば、

```
Cone {
    bottomRadius 1.2
    height       0.8
}
```

というノード文は、底面の半径が 1.2 で、高さが 0.8 であるような円錐を意味しています。なお、bottomRadius フィールドのデフォルト値は 1 で、height フィールドのデフォルト値は 2 です。

VRML 文書の例 cone.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 1 0 0.3 }
    }
    geometry Cone {
        bottomRadius 0.8
        height       3
    }
}
```

2.2 テキスト

Text というノードを使うことによって、テキストを形状として指定する、ということができます。

Text ノードは、string というフィールドを持っていて、そのフィールドに値としてテキストを与えることによって、そのテキストを形状にすることができます。

フィールドに値としてテキストを与えたいときは、そのテキストを二重引用符 (") で囲んだものを書きます。たとえば、

```
Text { string "namako" }
```

というノード文は、namako というテキストの形状を意味しています。

VRML 文書の例 text.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 1 0.6 }
    }
    geometry Text { string "I am a text." }
}
```

複数の行から構成されるテキストを形状にしたいときは、string フィールドに、テキストから構成される多重値を与えます。「多重値」(multiple value) というのは、複数の値から構成される値のことです。ちなみに、多重値ではない単独の値は、「単一値」(single value) と呼ばれます。

多重値は、角括弧 ([]) の中に単一値を並べて書くことによって記述されます。たとえば、

```
[ "namako" "hitode" "isoginchaku" "umiushi" "uni" ]
```

というのは、5 個のテキストから構成される多重値になります。多重値を構成するそれぞれの単一値は、コンマで区切ってもかまいませんので、上の例は、

```
[ "namako", "hitode", "isoginchaku", "umiushi", "uni" ]
```

と書くこともできます。

VRML 文書の例 multext.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0.6 0 1 }
    }
    geometry Text {
        string [ "first line" "second line" "third line" ]
    }
}
```

Text ノードは、fontStyle というフィールドを持っていて、そのフィールドに値を与えることによって、テキストのフォントファミリーやフォントスタイルや文字の大きさを指定することができます。

fontStyle フィールドに与える値は、FontStyle というノードです。FontStyle ノードは、フォントファミリーを指定する family、フォントスタイルを指定する style、文字の大きさを指定する size、というようなフィールドを持っています。

family フィールドには、フォントファミリーの名前を二重引用符で囲んだものを値として与えます。使うことのできるフォントファミリーはブラウザーに依存しますが、すべてのブラウザーは、最低限、SERIF、SANS、TYPEWRITER、という 3 種類のフォントファミリーをサポートしなければならないことになっています。なお、family フィールドのデフォルト値は "SERIF" です。

style フィールドには、フォントスタイルの名前を二重引用符で囲んだものを値として与えます。PLAIN で標準のスタイル、BOLD で太字、ITALIC で斜体、BOLDITALIC で太字の斜体を指定することができます。なお、style フィールドのデフォルト値は "PLAIN" です。

size フィールドには、文字の大きさ（ベースラインからの高さ）を指定します。デフォルト値は 1 です。

VRML 文書の例 font.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0.3 1 0.7 }
    }
    geometry Text {
        string "SANS, ITALIC, 1.2"
        fontStyle FontStyle {
            family "SANS"
            style  "ITALIC"
            size   1.2
        }
    }
}
```

2.3 発光する物体

前の節で説明したように、物体の色は、Material というノードが持っている diffuseColor というフィールドに、色をあらわす値を与えることによって指定するわけですが、Material ノードは、物体の色を指定するためのフィールドを、diffuseColor のほかにもうひとつ持っています。それは、emissiveColor というフィールドです。

diffuseColor というのが物体の反射光の色を指定するためのフィールドであるのに対して、emissiveColor というのは、物体が自分で出す光の色を指定するためのフィールドです。色をあらわす値をこのフィールドに与えると、物体はその色で発光することになります。

VRML 文書の例 emissive.wrl

```
#VRML V2.0 utf8
NavigationInfo { headlight FALSE }
Shape {
```

```
    appearance Appearance {
        material Material { emissiveColor 1 0.6 0 }
    }
    geometry Cylinder {}
}
```

ところで、この VRML 文書の中では、NavigationInfo という、まだ説明していないノードが使われていますので、それについて説明しておくことにしましょう。

ユーザーは、「ヘッドライト」(headlight) と呼ばれる、前方を照らすライトを持っています。ヘッドライトは、ブラウザーを操作することによって点灯したり消灯したりすることができて、デフォルトでは点灯している状態になっています。

ヘッドライトは、ノード文を書くことによって、最初から消えている状態にすることも可能です。それが、

```
NavigationInfo { headlight FALSE }
```

という、上の VRML 文書の中に書かれているノード文なのです。

2.4 テクスチャー

Appearance ノードは、material というフィールドだけではなく、texture というフィールドも持っています。このフィールドに値を与えることによって、物体にテクスチャーを指定することができます。

texture フィールドに与える値は、テクスチャーを指定するいくつかのノードのうちのどれかひとつです。テクスチャーとして画像を指定したいときは、ImageTexture というノードを使います。

ImageTexture ノードは、url というフィールドを持っていて、このフィールドに画像の URL を値として与えると、その画像が物体のテクスチャーになります。なお、URL は、二重引用符で囲む必要があります。

テクスチャーとして使うことのできる画像の形式は、ブラウザーに依存しますが、すべてのブラウザーは、最低限、JPEG と PNG という 2 種類の形式をサポートしないといけないことになっています（GIF についても、サポートが推奨されています)。

VRML 文書の例 image.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material {}
        texture ImageTexture { url "image.jpg" }
    }
    geometry Cylinder {}
}
```

3 座標系

3.1 VRML の座標系

空間の中にある点の位置は、普通、「座標系」(coordinate system) と呼ばれるものを使うことによって記述されます。VRML でも、3 次元空間の中での位置を記述するために、やはり座標系を使います。VRML の座標系は、x 軸、y 軸、z 軸と呼ばれる、方向を持った 3 本の直線から構成されます。これらの 3 本の直線は、「原点」(origin) と呼ばれる点で互いに直角に交わっています。

x 軸、y 軸、z 軸のそれぞれの上の位置は、その位置が、原点から見て直線が向いている方向（プラスの方向）にある場合は、原点からの距離であらわされ、それとは逆の方向（マイナスの方向）にある場合は、原点からの距離をマイナスにしたものによってあらわされます。

そして、空間の中にある点の位置は、x 軸上の位置、y 軸上の位置、z 軸上の位置をこの順序で並べてできる 3 個の数値の列によって記述されます。このような、位置をあらわしている数値の列は、「座標」(coordinates) と呼ばれます。

3 次元空間の座標系には、「右手系」(right-handed) と「左手系」(left-handed) と呼ばれる 2 種類のものがあって、VRML は右手系の座標系を採用しています。

右手系と左手系とのあいだの相違点は、x 軸、y 軸、z 軸のそれぞれがどちらの方向を向いているか、という点にあります。右手系の座標系は、右手の親指を x 軸、人差し指を y 軸、中指を z 軸だとみなしたときに、それぞれの指がプラスの方向を向くような座標系で、それに対して、左手について同じことが成り立つような座標系が左手系です。

3.2 座標系の変換

VRML で使われる座標系は、必要に応じて、移動や拡大や回転などの変更を加えることができるようになっています。そのような変更のことを、座標系の「変換」(transformation) と呼びます。

Shape ノードが物体を作るときの位置と向きは、座標系の上で固定されています。ですから、座標系を変換しなければ、位置や向きを指定して物体を作るということができません。

座標系を変換して物体を作りたいときは、Transform というノードを使います。このノードが持っている children というフィールドに、物体を作るノードを値として与えると、その物体は、変換された座標系を使って作られます。

どのように座標系を変換するのかというのは、Transform ノードが持っている、translation、rotation、scale、というフィールドを使って指定します。

座標系を移動させたいときは、translation というフィールドに、移動をあらわす値を与えます。その値は、

$x\ y\ z$

というように、3 個の数値の列で記述します。x、y、z という数値は、x 軸、y 軸、z 軸のそれぞれの方向への移動距離をあらわします。たとえば、

```
7 0.6 -8
```

というのは、x 軸の方向へ 7、y 軸の方向へ 0.6、z 軸の方向へ -8 だけ座標系を移動させるという意味になります。

VRML 文書の例 transla.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 0 1 } # blue
    }
    geometry Sphere {}
}
Transform {
    translation 5 2 0
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 1 0 0 } # red
        }
        geometry Sphere {}
    }
}
```

座標系を回転させたいときは、rotation というフィールドに、回転をあらわす値を与えます。その値は、

$x\ y\ z\ \theta$

というように、4 個の数値の列で記述します。x、y、z という数値は、回転軸をあらわしています。原点から見て、x、y、z であらわされる位置の方向が、回転軸のプラスの方向になります。そして、θ は、回転の角度をあらわしています（単位はラジアンです）。回転の方向は、親指が

プラスの方向を向くように右手で回転軸を握ったときに、親指以外の指が指し示す方向です。たとえば、

```
0 1 0 0.52
```

というのは、y 軸を回転軸にして、0.52 ラジアン（約 30 度）だけ座標系を回転させるという意味になります。

VRML 文書の例 rotation.wrl

```
#VRML V2.0 utf8
DEF Needle Cone {
    bottomRadius 0.5
    height       7
}
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 0 1 } # blue
    }
    geometry USE Needle
}
Transform {
    rotation 0 0 1 0.785
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 1 0 0 } # red
        }
        geometry USE Needle
    }
}
```

座標系を拡大したいときは、scale というフィールドに、拡大をあらわす値を与えます。その値は、

x y z

というように、3 個の数値の列で記述します。x、y、z という数値は、x 軸、y 軸、z 軸のそれぞれの方向への拡大率をあらわします。たとえば、

```
0.8 0.3 1.5
```

というのは、座標系を、x 軸の方向へ 0.8 倍、y 軸の方向へ 0.3 倍、z 軸の方向へ 1.5 倍だけ拡大するという意味です。

VRML 文書の例 scale.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 0 1 } # blue
    }
    geometry Sphere {}
}
Transform {
    scale 5 0.3 0.3
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 1 0 0 } # red
        }
        geometry Sphere {}
    }
}
```

children フィールドには、1 個だけではなく何個でも好きなだけ、物体を作るノードを値として与えることができます。そうすると、それらのノードがあらわしているすべての物体は、同一の座標系で作られることになります。

2 個以上のノードを children フィールドに与えたいときは、多重値を書きます。つまり、角

括弧で囲んだ中に、2 個以上のノード文を書けばいいわけです。

VRML 文書の例 mushape.wrl

```
#VRML V2.0 utf8
DEF Aqua Appearance {
    material Material { diffuseColor 0 1 1 }
}
Shape {
    appearance USE Aqua
    geometry Sphere {}
}
Transform {
    translation 3 1 2
    children [
        Shape {
            appearance USE Aqua
            geometry Cylinder {}
        }
        Shape {
            appearance USE Aqua
            geometry Box { size 3 0.2 3 }
        }
    ]
}
```

3.3 ユーザーの位置と方向

ブラウザーで VRML 文書を開いた直後のユーザーの位置は、その VRML 文書の中で何も指定されていなければ、デフォルトの位置になっています。それは、原点から z 軸のプラスの方向へ 10 メートルだけ進んだ位置です。

ユーザーが向いている方向も、VRML 文書の中で指定されていなければデフォルトの方向になっていて、それは、z 軸のマイナスの方向が前方で、y 軸のプラスの方向が上で、x 軸のプラスの方向が左です。

ブラウザーで VRML 文書を開いた直後のユーザーの位置と方向は、Viewpoint というノードを使うことによって、あらかじめ指定しておくことができます。

ユーザーの位置を指定したいときは、Viewpoint ノードが持っている position というフィールドに、位置をあらわす値を与えます。たとえば、

```
Viewpoint { position 3 0.8 -7 }
```

というノード文は、原点から、x 軸の方向へ 3、y 軸の方向へ 0.8、z 軸の方向へ -7 だけ移動した位置をユーザーの初期位置にする、ということをあらわしています。

ユーザーの方向を指定したいときは、Viewpoint ノードが持っている orientation というフィールドに、回転をあらわす値（つまり回転軸と回転角度）を与えます。そうすると、ユーザーは、デフォルトの方向から回転した方向を向くことになります。たとえば、

```
Viewpoint { orientation 0 1 0 3.14 }
```

というノード文は、デフォルトの方向から、y 軸を回転軸にして 3.14 ラジアン（約 180 度）だけ回転した方向（つまり z 軸の方向）をユーザーが向くようにする、ということをあらわしています。

VRML 文書の例 vpoint.wrl

```
#VRML V2.0 utf8
Viewpoint {
    position 0 4 7
    orientation 1 0 0 -0.52
}
Shape{
    appearance Appearance {
        material Material { diffuseColor 0 0.5 1 }
```

```
        }
        geometry Cylinder {}
    }
```

4 光源

4.1 光源の基礎

VRML の世界の中には、「光源」(light source) と呼ばれるものを何個でも好きなだけ作ることができます。光源というのは光を出すもののことで、VRML の世界の中にある物体は、光源が出す光によって照らし出されることになります。VRML の世界の中にいるユーザーは「ヘッドライト」と呼ばれるものを持っているわけですが、このヘッドライトというのも光源のひとつです。

光源を作りたいときは、「光源ノード」(light source node) と呼ばれる、`DirectionalLight`、`PointLight`、`SpotLight` という 3 種類のノードのうちのいずれかを使います。

すべての光源ノードは、`intensity` というフィールドと `color` というフィールドを持っています。

`intensity` は、光源が出す光の明るさを指定するフィールドです。このフィールドには、0 から 1 までの数値を値として与えます。光は、このフィールドの値が 0 に近いほど暗くなり、1 に近いほど明るくなります。デフォルト値は 1 です。

`color` は、光源が出す光の色を指定するフィールドです。物体の色を指定する場合と同じように、このフィールドには、赤、緑、青という光の三原色の強さをあらわす 3 個の数値を値として与えます。デフォルト値は白色です。

4.2 指向性光源

平行な光で物体を照らし出す光源のことを「指向性光源」(directional light source) と呼びます。これは、無限に遠い位置にある光源だと考えることができます。

指向性光源は、`DirectionalLight` というノードを使うことによって作ります。このノードは、光の方向を指定する `direction` というフィールドを持っています。光の方向は、

x y z

というように、3 個の数値の列で指定します。原点から見て、x、y、z であらわされる位置の方向が、光の方向になります。たとえば、

```
DirectionalLight { direction 0 -1 0 }
```

というノード文を書くことによって、真上から真下へ向かう光を出す指向性光源を作ることができます。`direction` フィールドのデフォルト値は、z 軸のマイナスの方向です。

VRML 文書の例 `dilight.wrl`

```
#VRML V2.0 utf8
NavigationInfo { headlight FALSE }
DirectionalLight { direction -1 0 0 }
Shape{
    appearance Appearance {
        material Material { diffuseColor 1 1 0.3 }
    }
    geometry Sphere {}
}
```

4.3 点光源

自分の周囲のあらゆる方向に向かって光を出す光源のことを「点光源」(point light source) と呼びます。

点光源は、PointLight というノードを使うことによって作ります。このノードは、光源の位置を指定する location というフィールドを持っています。光源の位置は、座標を使って指定します。たとえば、

```
PointLight { location 5 0 0 }
```

というノード文を書くことによって、原点から x 軸の方向へ 5 だけ移動した位置に点光源を作ることができます。location フィールドのデフォルト値は、原点です。

VRML 文書の例 polight.wrl

```
#VRML V2.0 utf8
DEF LightGreen Appearance {
    material Material { diffuseColor 0.6 1 0.6 }
}
NavigationInfo { headlight FALSE }
PointLight { location 0 0 3 }
Transform {
    translation -3 0 0
    children Shape {
        appearance USE LightGreen
        geometry Cylinder {
            radius 2
            height 4
        }
    }
}
Transform {
    translation 3 0 0
    children Shape {
        appearance USE LightGreen
        geometry Sphere { radius 2 }
    }
}
```

4.4 スポットライト

特定の方向に向かって円錐状に光を出す光源のことを「スポットライト」(spotlight) と呼びます。

スポットライトは、SpotLight というノードを使うことによって作ります。このノードは、光源の位置を指定する location、光を出す方向を指定する direction、光を出す方向の範囲を指定する cutOffAngle などのフィールドを持っています。cutOffAngle には、値として、direction で指定された方向と、範囲の境界となる方向との差をあらわす角度を、ラジアンで与えます。たとえば、

```
SpotLight {
    location     -20 0 0
    direction    1 0 0
    cutOffAngle  0.17
}
```

というノード文を書くことによって、原点から x 軸のマイナスの方向へ 20 だけ移動した位置に、x 軸の方向へ光を出すスポットライトを作ることができます。そして、このスポットライトは、x 軸から 0.17 ラジアン（約 10 度）だけ離れた方向まで光を出すことになります。

location フィールドのデフォルト値は原点で、direction フィールドのデフォルト値は z 軸のマイナスの方向で、cutOffAngle フィールドのデフォルト値は 0.785398 ラジアン（約 45 度）です。

VRML 文書の例 splight.wrl

```
#VRML V2.0 utf8
NavigationInfo { headlight FALSE }
SpotLight {
```

```
    location     0 10 0
    direction    0 -1 0
    cutOffAngle  0.35
}
DEF Kyuu Shape {
    appearance Appearance {
        material Material { diffuseColor 1 0.6 0 }
    }
    geometry Sphere { radius 2 }
}
Transform {
    translation -3 0 0
    children USE Kyuu
}
Transform {
    translation 3 0 0
    children USE Kyuu
}
```

5 アンカー

5.1 ハイパーテキスト

ウェブを構成するそれぞれのページは、直線的な順序で並べられているのではなくて、「リンク」(link) と呼ばれる関連付けによって網の目のような形に編成されています。そのような、リンクをたどっていくことによって自由な順序で見ていくことのできるデータは、「ハイパーテキスト」(hypertext) と呼ばれます。

リンクというのは、データの一部分を出発点にして、そこから別のデータへ行くことのできる通路のようなものだと考えることができます。データの一部分で、リンクの出発点となっているもののことを、「アンカー」(anchor) と呼びます。ウェブのブラウザーは、アンカーがポインティングデバイスでクリックされた場合、そのアンカーから出発したリンクの先にあるデータを画面に表示します。

アンカーは、リンクの先にあるデータを参照するための情報を持っている必要があります。ウェブの場合、その情報は、URL(uniform resource locator) と呼ばれる形式で記述されます。URL の仕様は、IETF(Internet Engineering Task Force) という組織によって規格化されていて、RFC1738 という文書で規定されています。

5.2 VRML 文書のアンカー

ウェブを構成するデータの形式のうちのいくつかは、アンカーを作るための手段をもっています。VRML も、アンカーを作る手段を持っているデータ形式のひとつです。

VRML で記述された世界の中にアンカーを作りたいときは、`Anchor` というノードを使います。このノードは、`url` と `children` というフィールドを持っています。`url` に値として URL を与え、`children` に `Shape` ノードを値として与えると、その `Shape` ノードによって作られた物体が、URL で指定されたデータをリンク先とするアンカーになります。たとえば、

```
Anchor {
    url "kurage.htm"
    children Shape {
        appearance USE Green
        geometry Cylinder {}
    }
}
```

というノード文を書いたとすると、この中の `Shape` ノードによって作られた円柱が、`kurage.htm` をリンク先とするアンカーになります（この例のように、`url` フィールドに与える URL は、二重引用符で囲む必要があります）。

次の二つの VRML 文書は、リンクによって相互に関連付けられていますので、それぞれの中の物体をポインティングデバイスでクリックすると、表示が相手側に切り換わります。

VRML 文書の例 worlda.wrl

```
#VRML V2.0 utf8
DEF SkyBlue Appearance {
    material Material { diffuseColor 0 0.6 1 }
}
Anchor {
    url "worldb.wrl"
    children Transform {
        translation -1.5 0 0
        children [
            Transform {
                translation -1 0.3 0
                children Shape {
                    appearance USE SkyBlue
                    geometry Box { size 0.8 0.8 0.8 }
                }
            }
            Shape {
                appearance USE SkyBlue
                geometry Text { string "worldb.wrl" }
            }
        ]
    }
}
```

VRML 文書の例 worldb.wrl

```
#VRML V2.0 utf8
DEF Orange Appearance {
    material Material { diffuseColor 1 0.6 0 }
}
Anchor {
    url "worlda.wrl"
    children Transform {
        translation -1.8 1 0
        children [
            Transform {
                translation -1 0.3 0
                children Shape {
                    appearance USE Orange
                    geometry Sphere { radius 0.5 }
                }
            }
            Shape {
                appearance USE Orange
                geometry Text { string "worlda.wrl" }
            }
        ]
    }
}
Anchor {
    url "http://www.paradise.ac.jp/"
    children Transform {
        translation -1.8 -1 0
        children [
            Transform {
                translation -1 0.3 0
                children Shape {
                    appearance USE Orange
                    geometry Sphere { radius 0.5 }
                }
            }
```

```
                Shape {
                    appearance USE Orange
                    geometry Text { string "Paradise College" }
                }
            ]
        }
    }
```

6 センサー

6.1 センサーの基礎

VRML の世界の中には、「センサー」(sensor) と呼ばれるものを作ることができます。センサーというのは、何らかの変化を検出するもののことです。センサーを使うことによって、ユーザーによる操作に反応するような世界を作ることができます。

センサーは、「センサーノード」(sensor node) と呼ばれる、 TouchSensor 、 PlaneSensor 、 CylinderSensor 、 SphereSendor 、 ProximitySensor 、 TimeSensor などのノードを使うことによって作ることができます。

大多数のセンサーノードは、それによって作られたセンサーを物体に取り付けて使う、という使い方を想定しています。センサーが取り付けられた物体を、ユーザーがマウスなどのポインティングデバイスを使って操作した場合、センサーはその操作を検出することになります。

物体にセンサーを取り付けたいときは、複数のノードをグループにまとめる機能を持つノードを使って、物体を作るノードとセンサーノードとをひとつのグループにまとめます。ノードのグループを作ることのできるノードの例としては、 Transform があります。 Tranform が持っている children というフィールドに対して、複数のノードから構成される多重値を与えると、それらのノードはひとつのグループを形成することになります。

ノードのグループを作る機能を持つノードとしては、 Transform のほかに Group というノードもあります。 Group というのは、座標系を変換する機能を Transform から取り除いたものだと考えることができます。座標系を変換する必要がない場合は、 Transform ではなく Group を使うほうがいいでしょう。

センサーが何らかの変化を検出すると、センサーノードが持っているフィールドのいくつかは、「イベント」(event) と呼ばれる、検出した変化を知らせるメッセージをノードの外へ送り出します。イベントを送り出すフィールドは、「eventOut」と呼ばれます。

フィールドには、送られてきたイベントを受け取って自分の値を変化させることができるものと、そうでないものとがあります。イベントを受け取って値を変化させることのできるフィールドは、「eventIn」と呼ばれます。

イベントをフィールドからフィールドへ送るためには、それらのフィールドを「ルート」(route) と呼ばれるもので接続する必要があります。ルートを作りたいときは、「ルート文」(route statement) と呼ばれる文を書きます。ルート文というのは、

ROUTE ノード名 1 . eventOut 名 TO ノード名 2 . eventIn 名

という構文を持つ文で、 TO の左側で指定された eventOut と、 TO の右側で指定された eventIn とを接続する、ということをあらわしています。たとえば、

```
ROUTE SENSOR.isActive TO LIGHT.on
```

というルート文は、 SENSOR というノードが持っている isActive という eventOut と、 LIGHT というノードが持っている on という eventIn とを接続する、という意味になります。

ルート文は、VRML 文書の末尾に書くというのが普通ですが、末尾ではなく途中に書いてもかまいませんし、そのほうが読みやすいならばノード文の中に書いてもかまいません。

6.2 TouchSensor

ポインティングデバイスを使って物体に触れたり、あるいはポインティングデバイスのボタンを押したりしたときに、何らかの反応が生じるようにしたいときは、TouchSensor というセンサーノードを使います。

TouchSensor は、isOver と isActive という eventOut を持っています。

TouchSensor が取り付けられた物体にポインティングデバイスで触れると、isOver は TRUE という真偽値のイベントを送り出します。そしてポインティングデバイスが物体から離れると、isOver は FALSE という真偽値のイベントを送り出します。

また、ポインティングデバイスが物体に触れているときに、そのポインティングデバイスのボタンを押すと、isActive は TRUE を送り出します。そしてボタンを離すと、FALSE を送り出します。

光源ノードは、光源の ON/OFF を指定する on というフィールドを持っています。このフィールドに値として TRUE を与えると光源が ON になり、FALSE を与えると OFF になります。このフィールドは eventIn ですので、真偽値のイベントを送ることによって、その値を変化させることができます。

VRML 文書の例 touch.wrl

```
#VRML V2.0 utf8
NavigationInfo { headlight FALSE }
Transform {
    translation 0 0 -10
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 1 1 1 }
        }
        geometry Sphere { radius 6 }
    }
}
Transform {
    translation -5 0 0
    children [
        DEF RED_LIGHT PointLight {
            color 1 0 0
            on FALSE
        }
        DEF RED_SWITCH TouchSensor {}
        Shape {
            appearance Appearance {
                material Material { emissiveColor 1 0 0 }
            }
            geometry Box { size 1 1 1 }
        }
    ]
    ROUTE RED_SWITCH.isActive TO RED_LIGHT.on
}
Transform {
    translation 5 0 0
    children [
        DEF BLUE_LIGHT PointLight {
            color 0 0 1
            on FALSE
        }
        DEF BLUE_SWITCH TouchSensor {}
        Shape {
            appearance Appearance {
                material Material { emissiveColor 0 0 1 }
            }
            geometry Box { size 1 1 1 }
        }
    ]
    ROUTE BLUE_SWITCH.isActive TO BLUE_LIGHT.on
```

```
}
```

6.3 PlaneSensor

ポインティングデバイスのドラッグによって、物体や光源を平面の上で移動させたいときは、PlaneSensor というセンサーノードを使います。

PlaneSensor は、座標のイベントを送り出す translation という eventOut を持っています。この eventOut と、Transform ノードの translation とをルートで接続すると、その Transform ノードの中で作られた物体や光源は、センサーから送られてきた座標のとおりに移動することになります。

PlaneSensor を作るときには、minPosition と maxPosition というフィールドに値を与える必要があります。これらは移動の範囲を指定するフィールドで、x 座標と y 座標の二つの数値から構成される列を値として与えます。たとえば、

```
DEF PS PlaneSensor {
    minPosition -8 -7
    maxPosition 4 3
}
```

というノード文を書いたとすると、このセンサーは、x 軸方向には -8 から 4 まで、y 軸方向には -7 から 3 までの移動を検出することになります。

PlaneSensor が検出する移動は、xy 平面の上に固定されています。それ以外の平面での移動を検出したいときは、Transform を使って座標系を回転させる必要があります。

VRML 文書の例 plane.wrl

```
#VRML V2.0 utf8
Transform {
    translation 0 0 -20
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 1 1 1 }
        }
        geometry Box { size 21 21 1 }
    }
}
Transform {
    translation 0 0 -19
    children [
        DEF PS PlaneSensor {
            minPosition -10 -10
            maxPosition 10 10
        }
        DEF HAKO Transform {
            children Shape {
                appearance Appearance {
                    material Material { diffuseColor 0 0 1 }
                }
                geometry Box { size 1 1 1 }
            }
        }
    ]
    ROUTE PS.translation TO HAKO.translation
}
```

6.4 CylinderSensor

ポインティングデバイスのドラッグによって、物体や光源を特定の回転軸で回転させたいときは、CylinderSensor というセンサーノードを使います。

CylinderSensor は、回転のイベントを送り出す rotation という eventOut を持っています。

この eventOut と、Transform ノードの rotation とをルートで接続すると、その Transform ノードの中で作られた物体や光源は、センサーから送られてきたイベントのとおりに回転することになります。

CylinderSensor が検出する回転の回転軸は、y 軸に固定されています。それ以外の回転軸での回転を検出したいときは、Transform を使って座標系を回転させる必要があります。

VRML 文書の例 cylisen.wrl

```
#VRML V2.0 utf8
Transform {
    translation 0 0 -10
    children DEF HAKO Transform {
        children Shape {
            appearance Appearance {
                material Material { diffuseColor 1 1 0 }
            }
            geometry Box { size 10 10 10 }
        }
    }
}
Transform {
    translation -2 -2 0
    rotation 1 0 0 -1.57
    children [
        DEF CS CylinderSensor {}
        DEF TSUMAMI Transform {
            children [
                Shape {
                    appearance Appearance {
                        material Material {
                            diffuseColor 0 1 1
                        }
                    }
                    geometry Cylinder {}
                }
                Transform {
                    translation 0 -1 1
                    children Shape {
                        appearance Appearance {
                            material Material {
                                diffuseColor 1 0 0
                            }
                        }
                        geometry Box { size 0.3 0.3 0.3 }
                    }
                }
            ]
        }
    ]
}
ROUTE CS.rotation TO TSUMAMI.rotation
ROUTE CS.rotation TO HAKO.rotation
```

6.5 SphereSensor

ポインティングデバイスのドラッグによって、物体や光源を、特定の位置を中心としてボールを転がすように回転させたいときは、SphereSensor というセンサーノードを使います。

SphereSensor も、CylinderSensor と同じように rotation という eventOut を持っています。SphereSensor の rotation と Transform の rotation とをルートで接続すると、Transform の中で作られた物体や光源は、センサーから送られてきたイベントのとおりに回転することになります。

SphereSensor が検出する回転の中心は、原点に固定されています。それ以外の位置を中心と

する回転を検出したいときは、Transformを使って座標系を移動させる必要があります。

VRML 文書の例 sphesen.wrl

```
#VRML V2.0 utf8
Transform {
    translation 0 0 -10
    children DEF HAKO Transform {
        children Shape {
            appearance Appearance {
                material Material { diffuseColor 1 1 0 }
            }
            geometry Box { size 10 10 10 }
        }
    }
}
Transform {
    translation -2 -2 0
    children [
        DEF SS SphereSensor {}
        DEF BALL Transform {
            children Shape {
                appearance Appearance {
                    material Material { diffuseColor 0 1 1 }
                }
                geometry Sphere {}
            }
        }
    ]
}
ROUTE SS.rotation TO BALL.rotation
ROUTE SS.rotation TO HAKO.rotation
```

6.6 ProximitySensor

ユーザーが特定の領域に入ったときとか、領域から出ていったときに何らかの反応が生じるようにしたいときは、ProximitySensorというセンサーノードを使います。

ProximitySensorは、物体に取り付けて使うのではなく、単独で使います。このセンサーは直方体の領域を持っていて、その領域に対するユーザーの出入りを検出します。領域の位置と大きさは、centerとsizeという二つのフィールドを使って指定します。centerには、領域の中心となる位置の座標を値として与えます。そしてsizeには、x 軸方向、y 軸方向、z 軸方向の長さ、つまり、幅、高さ、奥行を指定する 3 個の数値の列を値として与えます。たとえば、

```
DEF PS ProximitySensor {
    center  100 0 0
    size    20 20 20
}
```

というノード文を書くことによって、x 軸の方向に 100 だけ移動した位置を中心とする、一辺が 20 の立方体の領域を持つProximitySensorを作ることができます。

centerフィールドは、原点がデフォルト値になっていますので、そのままでかまわなければ指定する必要はありません。それに対して、sizeフィールドは、三つの方向の長さがすべてゼロというのがデフォルト値になっていますので、このフィールドには適切な値を与える必要があります。

ProximitySensorは、ユーザーの出入りのイベントを送り出すisActiveという eventOut を持っています。この eventOut は、ユーザーが領域に入ったときにTRUEを送り出し、領域から出ていったときにFALSEを送り出します。

VRML 文書の例 proximi.wrl

```
#VRML V2.0 utf8
NavigationInfo { headlight FALSE }
```

```
Transform {
    translation 0 0 -50
    children [
        DEF LIGHT PointLight {
            location  0 30 0
            on        FALSE
        }
        DEF PS ProximitySensor { size 30 10 30 }
        DEF HASHIRA Transform {
            translation 15 0 15
            children Shape {
                appearance Appearance {
                    material Material { emissiveColor 0 1 1 }
                }
                geometry Box { size 0.2 10 0.2 }
            }
        }
        Transform {
            translation -30 0 0
            children USE HASHIRA
        }
        Transform {
            translation 0 0 -30
            children USE HASHIRA
        }
        Transform {
            translation -30 0 -30
            children USE HASHIRA
        }
        Shape {
            appearance Appearance {
                material Material { diffuseColor 0 1 0 }
            }
            geometry Sphere {}
        }
    ]
    ROUTE PS.isActive TO LIGHT.on
}
```

ProximitySensor は、ユーザーの出入りだけではなく、領域の中でのユーザーの移動と方向転換も検出します。ProximitySensor が持っている position_changed という eventOut は、領域の中でユーザーが移動すると、その位置のイベントを送り出します。そして orientation_changed という eventOut は、領域の中でユーザーが方向を変えると、その回転のイベントを送り出します。

position_changed と orientation_changed を使うことによって、ダッシュボードのような、ユーザーが移動したり方向転換したりしてもユーザーとの位置関係が変化しないような物体を作ることができます。

VRML 文書の例 dashbd.wrl

```
#VRML V2.0 utf8
DEF PS ProximitySensor { size 1e25 1e25 1e25 }
Shape {
    appearance Appearance {
        material Material { diffuseColor 1 1 0 }
    }
    geometry Sphere {}
}
DEF DASHBOARD Transform {
    children Transform {
        translation 0 -0.23 -0.8
        children Shape {
            appearance Appearance {
                material Material { diffuseColor 0 1 0 }
            }
            geometry Box { size 0.6 0.1 0.01 }
```

```
            }
        }
    }
    ROUTE PS.position_changed TO DASHBOARD.translation
    ROUTE PS.orientation_changed TO DASHBOARD.rotation
```

7 アニメーション

7.1 アニメーションの基礎

VRML の世界の中に、ユーザーによる操作とは無関係に時間とともに変化するものを作る、ということも可能です。そのような自律的な変化は、「アニメーション」(animation) と呼ばれます。

VRML の世界の中にアニメーションを作るためには、まず第一に、時間の変化を検出するセンサーを作る必要があります。

時間の変化を検出するセンサーは、TimeSensor というセンサーノードを使うことによって作ることができます。このノードは、cycleInterval と loop というフィールドを持っています。

cycleInterval というフィールドには、アニメーションが始まってから終わるまでの時間（単位は秒）を値として与えます。デフォルト値は 1 秒です。

loop は、アニメーションを何回も繰り返すのか、1 回で終わりなのか、ということを指定するフィールドです。何回も繰り返すならば TRUE を、1 回で終わりならば FALSE を値として与えます。デフォルト値は FALSE です。

たとえば、

```
DEF TS TimeSensor {
    cycleInterval  20
    loop           TRUE
}
```

というノード文を書くことによって、20 秒間のアニメーションを何回も繰り返すためのセンサーを作ることができます。

TimeSensor は、franction_changed という eventOut を持っています。この eventOut は、cycleInterval で指定された時間の中で、0 から 1 までのあいだの数値のイベントを連続的に送り出します。

アニメーションを作るためには、TimeSensor のほかに、もうひとつ、「補間子」(interpolator) と呼ばれるものが必要になります。補間子というのは、数値のイベントを別のイベントに変換する機能を持っているもののことです。補間子は、「補間子ノード」(interpolator node) と呼ばれる、

```
PositionInterpolator    OrientationInterpolator
ColorInterpolator       ScalarInterpolator
```

などのノードを使うことによって作ることができます。

補間子ノードは、set_fraction という eventIn と value_changed という eventOut を持っています。set_fraction に数値のイベントを送ると、そのイベントは何らかの別のイベントに変換されて、value_changed から送り出されます。ですから、TimeSensor の fraction_changed と補間子ノードの set_fraction とをルートで接続して、さらに、その補間子ノードの value_changed と別のノードのフィールドとをルートで接続すれば、アニメーションができることになります。

補間子ノードを使って数値のイベントを別のイベントに変換するためには、補間子ノードが持っている key と keyValue という二つのフィールドに、どういう変換をするのかということをあらわす値を与えておく必要があります。数値から構成される多重値を key に与えて、それと同じ個数の値から構成される多重値を keyValue に与えると、key に与えた数値が、それに対応する keyValue の値に変換されます。さらに、key に与えた多重値を構成する数値だけではなくて、それらの数値のあいだにある数値も、自動的に適切な値に変換されます。

7.2 移動のアニメーション

物体や光源が直線的に移動するアニメーションを作りたいときは、PositionInterpolator という補間子ノードを使います。この補間子ノードは、数値のイベントを位置のイベントに変換する補間子を作ります。たとえば、

```
DEF PI PositionInterpolator {
    key [ 0, 0.25, 0.5, 0.75, 1 ]
    keyValue [ 4 0 4, 4 0 -4, -4 0 -4, -4 0 4, 4 0 4 ]
}
```

というノード文を書くことによって、xz 平面の上で正方形を描いて移動するアニメーションを作るための補間子を作ることができます。

VRML 文書の例 posani.wrl

```
#VRML V2.0 utf8
DEF ENCHUU Cylinder {
    radius 0.6
    height 1
}
Transform {
    translation 4 0 0
    rotation 0 0 1 1.57
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 0 0 1 }
        }
        geometry USE ENCHUU
    }
}
Transform {
    translation -4 0 0
    rotation 0 0 1 1.57
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 0 1 0 }
        }
        geometry USE ENCHUU
    }
}
DEF KYUU Transform {
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 1 0 0 }
        }
        geometry Sphere { radius 0.5 }
    }
}
DEF TS TimeSensor {
    cycleInterval  10
    loop           TRUE
}
DEF PI PositionInterpolator {
    key [ 0, 0.7, 1 ]
    keyValue [ -3 0 0, 3 0 0, -3 0 0 ]
}
ROUTE TS.fraction_changed TO PI.set_fraction
ROUTE PI.value_changed TO KYUU.translation
```

7.3 回転のアニメーション

物体や光源が回転するアニメーションを作りたいときは、OrientationInterpolator という補間子ノードを使います。この補間子ノードは、数値のイベントを回転のイベントに変換する補

間子を作ります。たとえば、

```
DEF OI OrientationInterpolator {
    key [ 0, 0.5, 1 ]
    keyValue [ 1 0 0 0, 1 0 0 3.14, 1 0 0 6.28 ]
}
```

というノード文を書くことによって、x 軸を回転軸にしてプラスの方向に回転するアニメーションを作るための補間子を作ることができます。

VRML 文書の例 oriani.wrl

```
#VRML V2.0 utf8
DEF HAKO Transform {
    children [
        Shape {
            appearance Appearance {
                material Material { diffuseColor 1 1 0 }
            }
            geometry Box { size 4 4 4 }
        }
        Transform {
            translation 2 2 2
            children Shape {
                appearance Appearance {
                    material Material { diffuseColor 1 0 0 }
                }
                geometry Box { size 0.4 0.4 0.4 }
            }
        }
    ]
}
DEF TS1 TimeSensor {
    cycleInterval  20
    loop           TRUE
}
DEF OI1 OrientationInterpolator {
    key [ 0, 0.5, 1 ]
    keyValue [ 0 1 0 0, 0 1 0 -3.14, 0 1 0 -6.28 ]
}
ROUTE TS1.fraction_changed TO OI1.set_fraction
ROUTE OI1.value_changed TO HAKO.rotation
DEF KYUU Transform {
    children Transform {
        translation 0 0 5
        children Shape {
            appearance Appearance {
                material Material { diffuseColor 0 1 1 }
            }
            geometry Sphere { radius 0.6 }
        }
    }
}
DEF TS2 TimeSensor {
    cycleInterval  5
    loop           TRUE
}
DEF OI2 OrientationInterpolator {
    key [ 0, 0.5, 1 ]
    keyValue [ 0.4 1 0 0, 0.4 1 0 3.14, 0.4 1 0 6.28 ]
}
ROUTE TS2.fraction_changed TO OI2.set_fraction
ROUTE OI2.value_changed TO KYUU.rotation
```

7.4 色のアニメーション

物体や光源の色が変化するアニメーションを作りたいときは、ColorInterpolator という補間子ノードを使います。この補間子ノードは、数値のイベントを色のイベントに変換する補間子を作ります。たとえば、

```
DEF CI ColorInterpolator {
    key [ 0, 0.5, 1 ]
    keyValue [ 0 0 1, 0 1 1, 0 0 1 ]
}
```

というノード文を書くことによって、青色から水色に変化して、そののち青色に戻る、というアニメーションを作るための補間子を作ることができます。

VRML 文書の例 colani.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material DEF M Material {}
    }
    geometry Sphere { radius 3 }
}
DEF TS TimeSensor {
    cycleInterval  10
    loop           TRUE
}
DEF CI ColorInterpolator {
    key [ 0, 0.5, 1 ]
    keyValue [ 1 0 0, 0 1 0, 1 0 0 ]
}
ROUTE TS.fraction_changed TO CI.set_fraction
ROUTE CI.value_changed TO M.diffuseColor
```

7.5 スカラーのアニメーション

2 個以上の数値の列ではない単独の数値のことを「スカラー」(scalar) と呼びます。VRML では、たとえば光源が出す光の明るさのような、スカラーで指定されるものをアニメーションにすることも可能です。スカラーのアニメーションを作りたいときは、ScalarInterpolator という補間子ノードを使います。この補間子ノードは、数値のイベントを別の数値のイベントに変換する補間子を作ります。たとえば、

```
DEF SI ScalarInterpolator {
    key [ 0, 0.5, 1 ]
    keyValue [ 1, 0, 1 ]
}
```

というノード文を書くことによって、1 から 0 へ減少していって、そののち 0 から 1 へ増加していく、というアニメーションを作るための補間子を作ることができます。

VRML 文書の例 scalani.wrl

```
#VRML V2.0 utf8
NavigationInfo { headlight FALSE }
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 1 0 }
    }
    geometry Cylinder { radius 3 }
}
DEF DL DirectionalLight { direction -1 0 -1 }
DEF TS TimeSensor {
    cycleInterval  10
    loop           TRUE
}
```

```
DEF SI ScalarInterpolator {
    key [ 0, 0.5, 1 ]
    keyValue [ 0, 1, 0 ]
}
ROUTE TS.fraction_changed TO SI.set_fraction
ROUTE SI.value_changed TO DL.intensity
```

7.6 ユーザーによるアニメーションの開始

ところで、常に動きつづけるアニメーションではなくて、ユーザーによる何らかの操作によって開始されるアニメーションを作りたいときはどうすればいいのでしょうか。

TimeSensor は、startTime という eventIn を持っています。これは、センサーが動作を開始する時刻をイベントとして受け取る eventIn です。TimeSensor が持っている loop というフィールドの値が、デフォルト値の FALSE のままになっている場合、そのセンサーは、startTime に送られてきた時刻に、動作を開始します。

TouchSensor は、touchTime という eventOut を持っています。これは、ユーザーがポインティングデバイスで物体をクリックしたときに、そのときの時刻をイベントとして送り出す eventOut です。この eventOut と、TimeSensor の startTime とをルートで接続することによって、物体がクリックされたときに動作を開始するアニメーションを作ることができます。

ProximitySensor は、enterTime と exitTime という eventOut を持っています。enterTime はユーザーが領域の中に入った時刻を送り出す eventOut で、exitTime はユーザーが領域の外へ出ていった時刻を送り出す eventOut です。これらの eventOut を使うことによって、ユーザーの出入りによって開始されるアニメーションを作ることができます。

VRML 文書の例 startan.wrl

```
#VRML V2.0 utf8
Group {
    children [
        DEF TOS TouchSensor {}
        Shape {
            appearance Appearance {
                material DEF M Material {
                    diffuseColor 1 1 1
                }
            }
            geometry Cylinder {}
        }
    ]
}
DEF TS TimeSensor { cycleInterval 2 }
DEF CI ColorInterpolator {
    key      [ 0, 0.5, 1 ]
    keyValue [ 1 1 1, 1 0 0, 1 1 1 ]
}
ROUTE TOS.touchTime TO TS.startTime
ROUTE TS.fraction_changed TO CI.set_fraction
ROUTE CI.value_changed TO M.diffuseColor
```

8 スクリプト

8.1 VRML とスクリプト

VRML では、センサーと補間子を使うことによって、ユーザーとのあいだの相互作用を可能にしたり、アニメーションを作ったりすることができるわけですが、しかし、センサーと補間子だけでできる相互作用やアニメーションは、かなり単純なものに限定されます。それでは、もっと複雑な相互作用やアニメーションを実現するためには、どうすればいいのでしょうか。

VRML には、何らかのプログラミング言語で書かれたスクリプトをブラウザーに実行させる

ことができるという機能があります（使うことのできるプログラミング言語は、ブラウザーに依存します)。この機能を使うことによって、センサーと補間子だけではできないような、複雑な相互作用やアニメーションが可能になります。

ブラウザーにスクリプトを実行させたいときは、Script というノードを使います。これは、補間子と同じように、イベントを受け取って、それを別のイベントに変換した結果を送り出すノードです。イベントをどのように変換するかということは、何らかのプログラミング言語で記述されたスクリプトによって決定されます。

8.2 型

VRML では、フィールドに与える値の種類のことを、その値の「型」(type) と呼びます。型は、「型名」(type name) と呼ばれる名前によって識別されます。VRML には、次のような型があります。

SFBool	真偽値	SFVec2f	2 次元ベクトル
SFInt32	32 ビットの整数	SFVec3f	3 次元ベクトル
SFFloat	浮動小数点数	SFColor	色
SFString	文字列	SFRotation	回転
SFTime	時刻	SFImage	画像
SFNode	ノード		

これらの型は、すべて多重値ではない値の型です。多重値の型は、先頭の文字を S から M に変更した型名を持ちます。たとえば、文字列から構成される多重値の型をあらわす型名は、MFString です。ただし、 SFBool と SFImage から構成される多重値の型というのは存在しません。

JavaScript のスクリプトの中では、

new 型名 (式 , ···)

という形の式を書くことによって、VRML で扱うことのできるオブジェクトを生成することができます。たとえば、

```
new SFRotation(0,1,0,3.14)
```

という式を評価すると、y 軸を回転軸にして 3.14 ラジアンだけ回転するという回転のオブジェクトが値として得られます。

8.3 フィールドを作る記述

Script ノードは、ほかのノードとは違って、その中に何個でも好きなだけフィールドを作ることができます。Script ノードを作るノード文の中に、

eventIn 型名 名前

という形の記述を書くと、指定された型と名前を持つ eventIn が、 Script ノードの中に作られます。たとえば、

```
eventIn SFVec3f ichi
```

という記述を書くことによって、SFVec3f という型のイベントを受け取ることのできる ichi という eventIn を作ることができます。

同じように、eventOut も、

eventOut 型名 名前

という形の記述を書くことによって作ることができます。

8.4 スクリプトを指定するフィールド

Script ノードは、url というフィールドを持っています。これは、スクリプトそのものを指定するためのフィールドです。スクリプトが格納されているファイルの URL をこのフィールドに値として与えることによって、ブラウザーがそのスクリプトを実行することができるようになります。たとえば、kamenote.js というファイルに格納されている JavaScript で書かれたスクリプトをブラウザーに実行させたいならば、Script ノードを作るノード文の中に、

```
url "kamenote.js"
```

と書けばいいわけです。

JavaScript で書かれたスクリプトは、VRML 文書とは別のファイルに格納してもかまいませんが、スクリプトそのものを url フィールドに値として与えることもできます。そうしたいときは、

```
url "javascript:  ..."
```

というように、先頭に javascript: と書いて、そのうしろにスクリプトを書きます。スクリプトの途中に改行を入れてもかまいません。

Script ノードの中に作られた eventIn と同じ名前の関数を作ると、その関数は、イベントが eventIn に送られてきたときに呼び出されることになります。そしてその関数は、送られてきたイベントを引数として受け取ります。そして、関数が eventOut にオブジェクトを代入すると、その eventOut は、代入されたオブジェクトをイベントとして送り出します。

VRML 文書の例 touchco.wrl

```
#VRML V2.0 utf8
Group {
    children [
        DEF TOS TouchSensor {}
        Shape {
            appearance Appearance {
                material DEF M Material {
                    diffuseColor 1 1 1
                }
            }
            geometry Cylinder {}
        }
    ]
}
DEF TOUCHCOLOR Script {
    eventIn  SFBool  touch
    eventOut SFColor color
    url "javascript:
        function touch(isActive) {
            if (isActive)
                color = new SFColor(0,0.5,1);
            else
                color = new SFColor(1,1,1);
        }"
}
ROUTE TOS.isActive TO TOUCHCOLOR.touch
ROUTE TOUCHCOLOR.color TO M.diffuseColor
```

8.5 状態の保持

Script ノードの中に、eventIn でも eventOut でもないフィールドを作っておくことによって、何らかの状態をそのフィールドに保持させておくということができます。

eventIn でも eventOut でもないフィールドを作りたいときは、

field 型名 名前 初期値

という形の記述を書きます。たとえば、

```
field SFColor iro 1 0 0
```

という記述を書くことによって、iro という名前を持つ色のフィールドを作って、初期値として赤色を与えることができます。

VRML 文書の例 tcount.wrl

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 1 0 0 }
    }
    geometry DEF T Text {
        string "0"
        fontStyle FontStyle { size 4 }
    }
}
Transform {
    translation 0 -2 0
    children [
        DEF TOS TouchSensor {}
        Shape {
            appearance Appearance {
                material Material { diffuseColor 0 1 0 }
            }
            geometry Cylinder {}
        }
    ]
}
DEF TOUCHCOUNT Script {
    field    SFInt32  count 0
    eventIn  SFBool   touch
    eventOut MFString string
    url "javascript:
        function touch(isActive) {
            if (isActive == false) {
                count++;
                string = new MFString(count + '');
            }
        }"
}
ROUTE TOS.isActive TO TOUCHCOUNT.touch
ROUTE TOUCHCOUNT.string TO T.string
```

8.6 ノードの生成

VRML のブラウザーは、Browser という名前のオブジェクトを持っていて、この中には、いろいろと便利なメソッドが含まれています。

Browser に含まれている createVrmlFromString というメソッドは、引数として VRML の記述を文字列で受け取って、それをノードに変換した結果を返します。このメソッドを使うことによって、新しいノードを動的に生成することが可能になります。

スクリプトによって生成されたノードを世界に出現させたいときは、Group または Transform が持っている addChildren という eventIn を使います。この eventIn にイベントとしてノードを送ると、そのノードがグループに追加されて、世界の中に出現することになります。

VRML 文書の例 create.wrl

```
#VRML V2.0 utf8
DEF G Group {}
Transform {
    translation -4 0 0
    children [
```

```
        DEF TOS TouchSensor {}
        Shape {
            appearance Appearance {
                material Material { diffuseColor 0 1 0 }
            }
            geometry Cylinder {}
        }
    ]
}
DEF CREATESPHERE Script {
    field    SFFloat z 0
    eventIn  SFBool  touch
    eventOut MFNode  node
    url "javascript:
        function touch(isActive) {
            if (isActive == false) {
                node = Browser.createVrmlFromString(
                    'Transform {' +
                    '    translation 4 0 ' + z +
                    '    children Shape{' +
                    '        appearance Appearance {' +
                    '            material Material {' +
                    '                diffuseColor 0 1 1' +
                    '            }' +
                    '        }' +
                    '        geometry Sphere {}' +
                    '    }' +
                    '}');
                z -= 5.0;
            }
        }"
}
ROUTE TOS.isActive TO CREATESPHERE.touch
ROUTE CREATESPHERE.node TO G.addChildren
```