

SONY®

Community Place™ Conductor 2.0 **チュートリアル**



警告

ソフトウェア製品は、安全のための注意事項を守らないと、健康を害することがあります。

この取扱説明書には、健康を守るための重要な注意事項と製品の取り扱いかたを示してあります。この取扱説明書をよくお読みのうえ、製品をお使いください。お読みになったあとは、いつでも見られるところに必ず保管してください。

下記の注意事項をよくお読みください。製品全般の注意事項が記されています。

警告表示の意味

インストールガイドおよび製品では、次のような表示をしています。表示の内容をよく理解してから本文をお読みください。



この表示の注意事項を守らないと、健康を害するおそれがあります。

この行為を禁止する記号



禁止



下記の注意を守らないと
健康を害するおそれがあります。



禁止

・ディスプレイ画面を長時間継続して見ない

ディスプレイなどの画面を長時間継続して見続けると、目が疲れたり、視力が低下するおそれがあります。ディスプレイ画面を見続けて体の一部に不快感や痛みを感じたときは、すぐにコンピューターの使用をやめて休息してください。万一、休息しても不快感や痛みがとれないときは、医師の診察を受けてください。



禁止

・キーボードを使いすぎない

キーボードやマウスなどを長時間継続して使用すると、腕や手首が痛くなったりすることがあります。キーボードやマウスなどを使用中、体の一部に不快感や痛みを感じたときは、すぐにコンピューターの使用をやめて休息してください。万一、休息しても不快感や痛みがとれないときは、医師の診察を受けてください。

Copyright © 1998 Sony Corporation. All rights reserved.
Community Place™ はソニー株式会社の商標です。

このソフトウェアの使用および複製は、契約書の条項で許容されている範囲でのみ許可されています。本書の記述内容は、予告なく変更する可能性があり、また、製品仕様またはソニー株式会社の義務遂行の確約を表すものではありません。ソニー株式会社の書面による許可なく、この出版物の一部または全部を再生産、転送、転写、保管、または他言語へ翻訳することは、その形態に関係なく、一切禁じられています。

Windows®、Windows NT® は米国 Microsoft Corporation の米国及びその他の国における登録商標です。

Java™、JDK™ は、米国およびその他の国における米国サン・マイクロシステムズ社の商標です。

Netscape Navigator™ は米国 Netscape Communications Corporation の商標です。
その他、記載されている会社名、製品名は、各社の商標または登録商標です。

目次

序章	1
コンテンツの作成方法	1
第1章 Cone をクリックしてライトを ON/OFF する	6
Cone (Geometry ノード) の配置	6
TouchSensor (Sensor ノード) の配置	7
PointLight (Common ノード) の配置	8
Route 対象のノードに名前を付ける	9
Route の設定	11
動作確認	12
第2章 Cone の上にマウスカースルが来たら Cone の色を変える	14
Route 対象のノードに名前を付ける	15
Script ノードの作成	15
Script ノードに対応するプログラムの記述 (デバッグコードのみ)	17
Script ノードへの Route の設定	18
動作確認	19
Script ノードに対応するプログラムの記述 (色を設定する)	20
Material ノードへの Route の設定	21
動作確認	23
第3章 Cone をクリックして音を鳴らす	25
Sound ノードの配置	25
動作確認	26
AudioClip ノードの設定と Route の設定	29
動作確認	30
第4章 ビジュアル表示 Route 編集	31
Route 対象のノードに名前を付ける	31
Route の設定	33
動作確認	34

第5章 キーフレームアニメーション	35
Cone (Geometry ノード) の配置	35
TimeSensor (Sensor ノード) の配置と名前付け	36
アニメーション編集モード	37
移動 : 1 番目のキーフレームを作成	38
移動 : 2 番目のキーフレームを作成	40
動作確認	41
アニメーションの速度を変える	41
回転 : 1 番目のキーフレームを作成	42
回転 : 2 番目のキーフレームを作成	44
動作確認	45
第6章 PROTO を使う	46
プリミティブ (Cylinder) を組み合わせて物体 (テーブル) を作成	47
PROTO の作成	48
テーブルの複製 (インスタンスの作成)	50
PROTO 内部のフィールドを外部的に見せる (1)	51
PROTO 内部のフィールドを外部的に見せる (2)	54
インスタンスのフィールドを変更	54
第7章 応用的なキーフレームアニメーション	57
色の变化 : 1 番目のキーフレームを作成	57
色の变化 : 2 番目のキーフレームを作成	61
動作確認	62
変形 : 1 番目のキーフレームを作成	62
変形 : 2 番目のキーフレームを作成	65
動作確認	66
TouchSensor (Sensor ノード) の配置	66
Route の作成	67
動作確認	68
参考文献	69

序章

このチュートリアルでは、Community Place™ Conductor 2.0（以下、Conductor と呼びます。）を使って簡単なコンテンツを作成する例について説明します。VRML97 を知らなくてもこのチュートリアルを試すことはできますが、VRML97 についての知識があるならば、さらに理解しやすいものとなるでしょう。もし、VRML97 について詳しくない人は、ヘルプに VRML97 の規格を添付してありますので、こちらを読んでください（ただし、現在のところ英語版のみです）。

本文中のイタリック文字（例 *color*）は、VRML のフィールド名を、ボールド文字（例 **value=32**）は、プログラムの一部を示しています。詳しくは、巻末に挙げる参考文献をご覧ください。

■ コンテンツの作成方法

図 1 に示すように、Conductor を使用しないで VRML97 のコンテンツを作成する場合は大きく 2 つのステップにわけることができます。

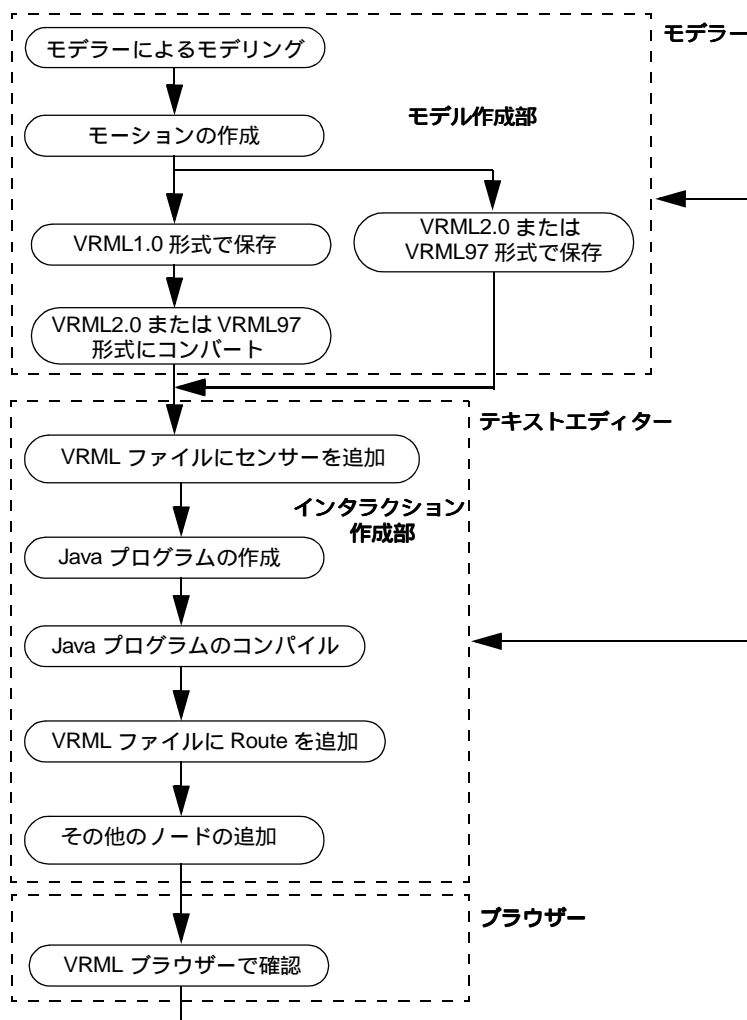


図 1 VRML97 による一般的なコンテンツ作成

- **ステップ 1**

最初のステップはモデルを作成する部分です。ここでは 3D モデラーを使用してモデルの作成を行います。またモーションやセンサーがつけられるモデラーを使った場合（例えば、AutoDesk の 3D Studio Max など）には、ここでモーションやセンサーも作成しておきます。そして、最後に VRML97 形式にファイルを変換します。

- **ステップ 2**

次のステップはインタラク션을追加する部分です。ここではテキストエディターを使って VRML ファイルにセンサーを追加したり、Script ノードを追加します。もし、Java を使ってスクリプトを作成した場合にはコンパイルも必要です。そして、VRML ファイルに ROUTE ノードを追加し、VRML97 準拠のブラウザ（例えば、ソニーの Community Place™ Browser）を使用して動作を確認します。

次に、Conductor を使った場合のコンテンツ作成方法を図 2 に示します。基本的には、従来のステップ 2 と同じ作業が 1 つのツールで行うことができるようになっています。

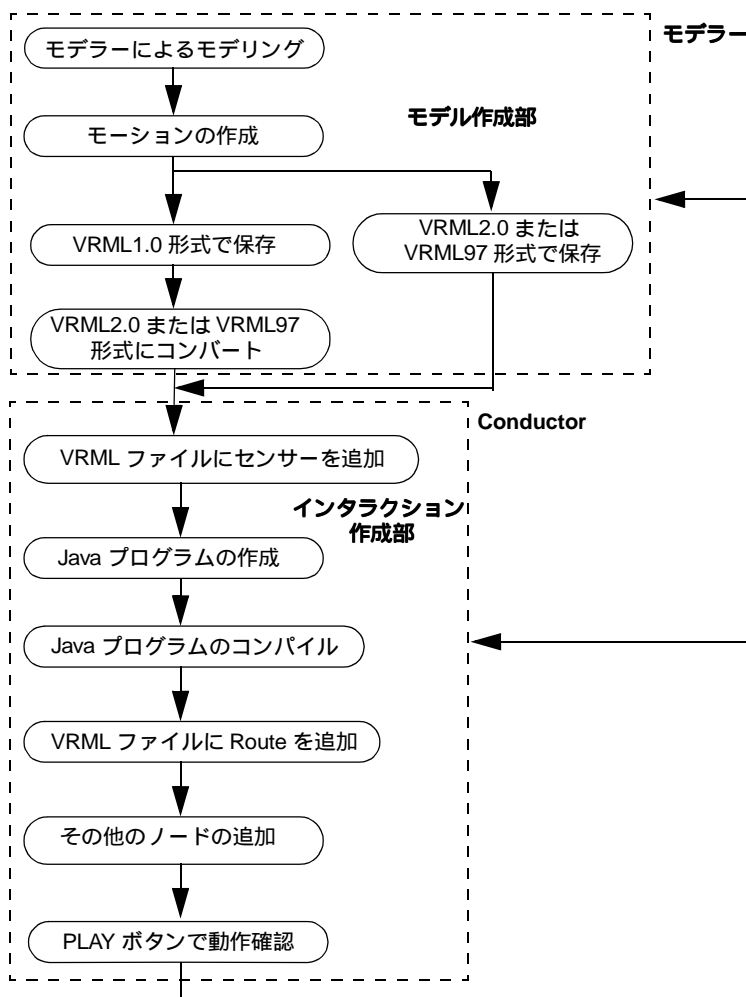



図 2 Conductor を使った場合のコンテンツ作成

Conductor では、センサーの追加や Route の編集などをビジュアルに行うことができます。また、Script Expert 機能により対話形式でスクリプトのテンプレートを作成する機能もあります(これについては本チュートリアルで解説します)。動作を確認したい時には Play ボタン  を押します。編集画面がそのままブラウザの画面となり、動作の確認が行えるようになります。

以上のように、Conductor を使うことにより VRML97 のコンテンツ作成時間を大幅に短縮することが可能になります。

以下の章では、Conductor を使用して簡単なコンテンツを作成する例について解説します。

(注) チュートリアルでは 3D/2D View ウィンドウにグリッドラインを表示していません。グリッドラインの表示、非表示切り替えは、Option メニュー Environment で表示される環境設定ダイアログボックスで行うことができます。

第1章 Cone をクリックしてライトを ON/OFF する

まずは、このストーリーを実現してみましょう。全体の流れは以下のようになります。

- Cone (Geometry ノード) の配置
- TouchSensor (Sensor ノード) の配置
- PointLight (Common ノード) の配置
- Route 対象のノードに名前を付ける
- Route の設定
- 動作確認

以下に各ステップについて説明します。

■ Cone (Geometry ノード) の配置

まず、ワールド内にオブジェクトを配置し全体を構築していきます。

Conductor を起動してメニューから File New を選択し、空のワールドを作成します。

次に、Cone を配置するには、Main ウィンドウの Geometry タブから Cone を選択し (マウスでクリックするとアイコンが黄色になります)、さらに 3D Perspective View (以下、3D View ウィンドウ) 上でクリックします。すると図3に示されるように、Cone が目の前に作成されます。

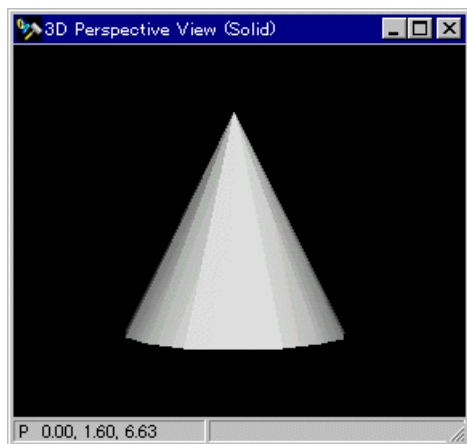


図3 Cone を配置した様子

(注) 配置できるオブジェクトは、Main ウィンドウのツールバーにある基本ノードや、Library ウィンドウで提供されるオブジェクトなどです。また、他のモデリングソフトで作成したオブジェクトをVRML97 ファイル形式に変換した後、読み込むこともできます。

■ TouchSensor (Sensor ノード) の配置

オブジェクトに動きをつけたり、ユーザの操作に反応するようにするためには Sensor ノードを配置しなければなりません。Sensor ノードとはある一定の条件を検出し、その変化をイベントとして他のノードに伝える働きをもつノードのことです。ここでは、マウスによる操作を検出する TouchSensor を配置します。

ある物体をクリックしたことを検出したい場合には、その物体の上位にある Transform に対して TouchSensor を取り付けます。もし、この Transform の下位に複数の物体が存在する場合、その全ての物体に対して TouchSensor が動作するようになります。

まず、Scene Graph ウィンドウ上で、Cone の上位にある Transform をクリックし、選択状態にしてください。さらに、Main ウィンドウから Sensor タブ

をクリックし、図 4 のように左から 3 番目にある TouchSensor を選択した後に、3D View ウィンドウ上でクリックします。

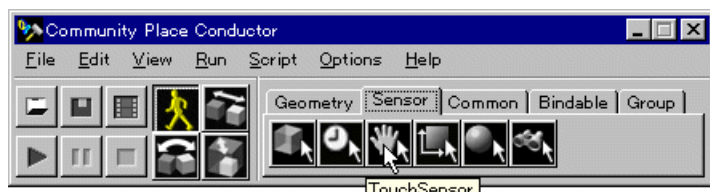


図 4 TouchSensor を選択する

これで Cone の上位の Transform に TouchSensor が付きました。図 5 に示すように、Scene Graph ウィンドウで Transform に TouchSensor がついていることが確認できます。

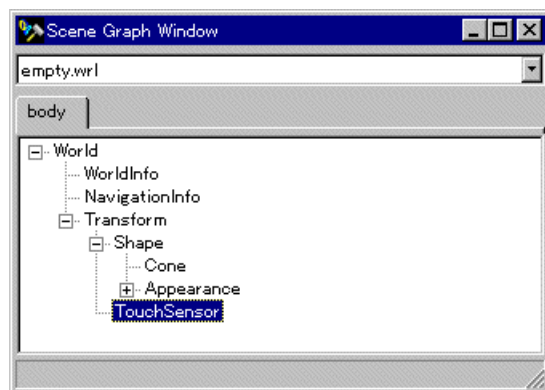


図 5 Scene Graph ウィンドウで TouchSensor を確認する

(注) この例では、Cone という単純なオブジェクトに対して TouchSensor を付加しましたが、複雑なオブジェクトに対して TouchSensor を付加することも可能です。

■ PointLight (Common ノード) の配置

VRML97 では Sensor ノードで検出したイベントを他のノードに伝達し、イベントを受けたノードが状態を変更するという流れで動作を記述します。

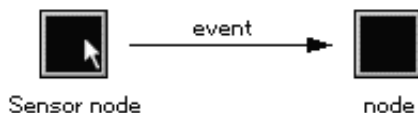


図6 処理の流れ

そこで、イベントを受けるノードとしてライトを配置しましょう。ライトは Common タブにあります。そのままライトを置くと、Cone と同じ場所に生成されてしまうので、あらかじめ視点を移動しておきます。3D View ウィンドウをアクティブにしており、下矢印キー で少し視点をひいておきます。この状態で PointLight を生成して見てください。生成すると、図7のように Cone が明るくなるはずです。

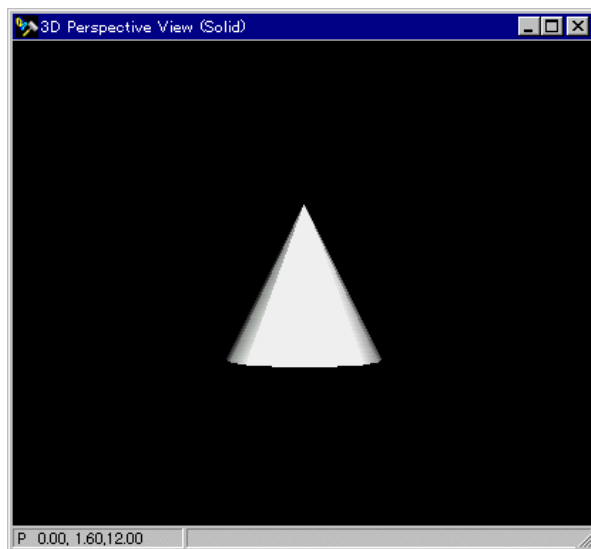


図7 PointLight を配置したところ (Cone が明るくなった)

■ Route 対象のノードに名前を付ける

次に、TouchSensor から PointLight へ Route の設定を行うわけですが、Route の設定を行うためにはまずノードに名前(DEF 名)を付けなければなりません。

Sensor ノードから他のノードにイベントを伝達するためには、両者を結び付けなければなりません。この結び付けのことを Route と呼びます。

DEF 名は Attribute ウィンドウでつけることができます。Attribute ウィンドウは選択されているオブジェクトの属性を変更するためのウィンドウです。Scene Graph ウィンドウから TouchSensor を選択し、Attribute ウィンドウで DEF 名を付けてみましょう。ここでは図 8 に示すように TS1 という名前を付けることにします。DEF 名の所で TS1 とタイプし、Enter キーを押します。この時、Scene Graph ウィンドウの Touch Sensor のノード名の後には、[] で囲まれた DEF 名 "[TS1]" が追加されます。

(注) DEF 名は DEF 名編集エリアに入力後、Enter キーを押すことで確定されます。

同様に、Scene Graph ウィンドウから PointLight を選択し、PL1 という名前を付けてください。また、ライトの初期状態を off にしておきましょう。これは、Attribute ウィンドウで PointLight の on フィールドをダブルクリックすることによりできます。(FALSE になります)

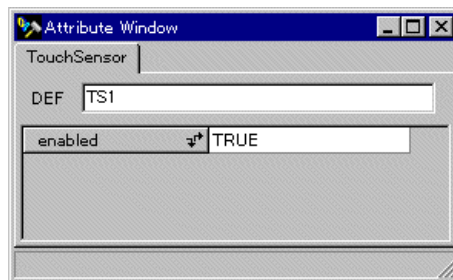


図 8 TouchSensor に名前をつけたところ

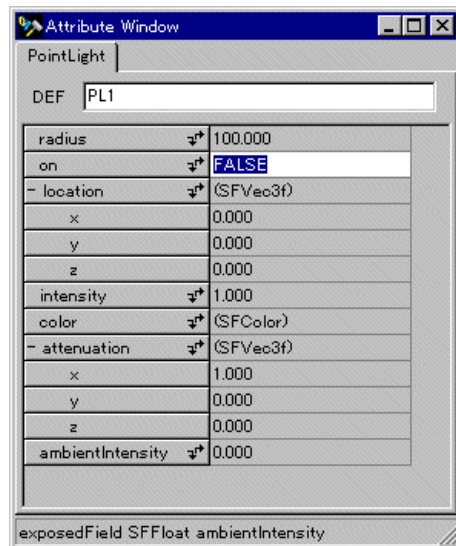


図 9 PointLight にも名前をつけライトを消したところ

■ Route の設定

これでいよいよ Route の設定を行うことができます。TouchSensor ノード (TS1) で発生したイベントを PointLight ノード (PL1) に伝達するように Route の設定を行います。

Conductor でこの設定を行うには、Route ウィンドウを使います。Main ウィンドウのメニューから View Route を選んで Route ウィンドウを表示させます。Route ウィンドウでは、リスト表示とビジュアル表示という 2 種類の方法を用いることができます。操作としてはビジュアル表示 Route 編集の方が簡単なのですが、リスト表示 Route 編集は VRML における Route の仕組みを理解するためにも有用ですので、最初はリスト表示 Route 編集を用いて説明を行います。ビジュアル表示 Route 編集については後ほど第 4 章で説明することになります。

始めに、Route ウィンドウ左下の 3 つのスイッチのうち一番左側のものを押して、リスト表示編集モードにしておきます。リスト表示編集モードになると Route ウィンドウは図 10 のようになります。Route ウィンドウ上段の Node name となっている所がイベントを出力するノード名、下段の Node name と

なっているところがイベントを入力するノード名です。課題では、"TouchSensor をクリックして..." となっているので、出力側の nodeName は TS1 を、また eventOut は isActive を選択します。すると、右側に SFBool という型が表示されます。これは、isActive で渡されるイベントの型が SFBool であることを示しています。次に入力側の設定をします。Node name には PL1 を、eventIn には on を選びます。

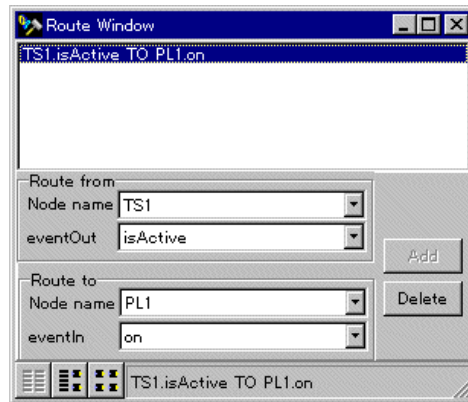



図 10 TouchSensor から PointLight へ Route する


選択が終了したら、Add ボタンを押します。これにより、Route が追加され「TS1.isActive TO PL1.on」が表示されます。

■ 動作確認

Conductor では、Play ボタンを押すことでブラウザをエミュレーションすることができます。これにより、VRML ブラウザを起動せずに編集画面中でそのままワールドの動作を確認することができます。また、Stop ボタンを押すことで通常の編集モードに戻ることができます。

それでは確認してみましょう。Main ウィンドウの左下にある Play ボタン  を押してみてください。Play ボタンを押すと、今までの編集状態からブラウザのエミュレーションモードへと移行します。Cone をクリックしてみてください。マウスを押している間はライトが ON になり、離すと OFF になります。

ここでのワールドの動きは次のようになります。Cone をクリックするとそこに付属している TouchSensor ノードがそれを検出しイベントを発生します。このイベントは Route の設定により PointLight ノードに伝えられます。PointLight ノードはこれを受け、状態を変えて点灯させます。マウスボタンを離れたときにも同じ流れでイベントが伝達され、ライトが消えます。

Stop ボタン  を押すと通常の編集モードに戻ります。

ファイルを保存したい場合には、File Save As メニューでファイル名を指定してください。作成したファイルは Community Place™ Browser でも見ることができます。

このように簡単な場合でしたら、ノードと Route を駆使することによりプログラミングせずに動作を追加することができます。

次の章ではもう少し複雑な場合について解説します。

第 2 章 Cone の上にマウスカーソルが来たら Cone の色を変える

前章の例は Sensor の発生したイベントを直接他のオブジェクトへ伝達するものですが、VRML97 では Script ノードを介在させることにより、さらに複雑な動きを実現することができます。

例えば「Cone の上にマウスカーソルが来たら、Cone の色を変える」場合について考えてみます。VRML97 ではオブジェクトの色は Material 属性で指定することができます。Material にはいくつかフィールドがありますが、*diffuseColor* を設定することでオブジェクトの色を変えることができます。そのためには、イベントにより *diffuseColor* を外部から設定しなければなりません。このような場合、Script ノードを使うことで *diffuseColor* を設定することができます。

全体の流れは以下ようになります。 は「第 1 章 Cone をクリックしてライトを ON/OFF する」(6 ページ)と同じなので、説明は省略します。Conductor を再起動し、Cone や TouchSensor を配置しておいてください。また、TouchSensor には TS1 という名前もつけておいてください。

- Cone (Geometry ノード) の配置

- TouchSensor (Sensor ノード) の配置

- Route 対象のノードに名前を付ける

- Script ノードの作成

- Script ノードに対応するプログラムの記述 (デバッグコードのみ)

- Script ノードへの Route の設定

- 動作確認

- Material ノードへの Route の設定

- 動作確認

以下に各ステップについて説明します。

■ Route 対象のノードに名前を付ける

Route を設定するには最初に名前をつける必要がありました。前章の例では、TouchSensor や PointLight に名前をつけましたが、今回は Material に名前をつける必要があります。

それでは作業を行ってみましょう。Material は VRML97 では Appearance ノードに含まれるので、Conductor でも同様に Material は Appearance ウィンドウに存在します。Main ウィンドウのメニューから View Appearance を選ぶと Appearance ウィンドウが表示されます。この Appearance ウィンドウは選択されているオブジェクトの Appearance (Material, Texture などが含まれます) を編集するためのウィンドウです。まず Cone を選択しておき、次に Appearance ウィンドウの material タブで CONE1_MAT という名前を付けます。

(注) DEF 名は DEF 名編集エリアに入力後、Enter キーを押すことで確定されます。

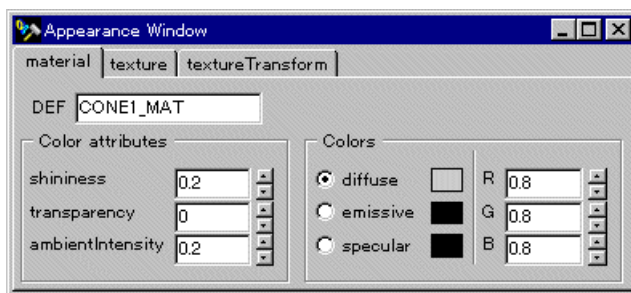


図 11 Material に名前をつける

■ Script ノードの作成

VRML97 では、外部のスクリプトプログラムとインターフェースするための Script ノードと呼ばれるノードがあります。

この Script ノードと外部スクリプトプログラム (Conductor では Java を使います) を使うことにより、複雑な動作を記述することができるようになります。しかしながら、Script ノードはインターフェースの役割をもつことから外部スクリプトプログラムと密接な関係があり、また複数の入出力が定義可能なため少しわかりづらいものになっています。

(注) Script を使うためには JDK1.1.x をインストールする必要があります。

そこで、Conductor には Script Expert と呼ばれる Script を簡単に作成するための機能があります。これは Script の入出力を定義することで、スクリプトプログラムのテンプレートを自動的に出力してくれるものです。また、Attribute ウィンドウで入出力のフィールドを編集することも可能です。

それでは Script ノードを作ってみます。Conductor ウィンドウのメニューから Script Expert を選択してみてください。Script ノードを作成するためのダイアログが開きます。いちばん上の DEF は VRML 上での Script ノードの名前です。今までのノードと同様、Script ノードへ Route を設定するにはここで名前をつけておく必要があります。名前は、SC1 としておきましょう。次の Class Name は Java 側でのクラス名です。とりあえず、SC1 としておきます。さらに Script ノードの入出力を定義します。TouchSensor からのイベントを入力したいので、eventIn 側の field type を SFBool とします(これは、TouchSensor からのイベントが SFBool という型で渡されるからです)。また field name を "inBool" とします。

出力側は Material の色をコントロールしたいので、SFColor を選択します。field name は "outColor" としておきましょう。図 12 に各フィールドを埋め込んだ状態を示します。

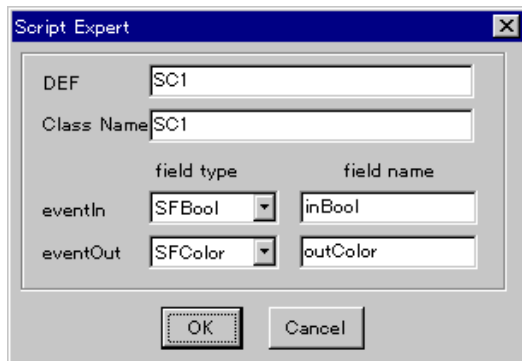


図 12 Script Expert を使って Script ノードを作成する

これで Script ノードに必要な項目が設定できたので、OK ボタンを押します。すると Script ノードが生成され、さらに Script ノードに対応付けられるプログラムのテンプレート(ひな型)が生成されます。

■ Script ノードに対応するプログラムの記述（デバッグコードのみ）

新しく定義したスクリプトが編集できるようになりました。ところが、このテンプレートは入出力の基本的な部分が定義されているだけで他には何もありません。すぐに色を変更するプログラムを書いてもよいのですが、まずはこのプログラムが本当に動くのかを確認するために、Java Console ウィンドウに変数の値を表示させるデバッグコードを記述します。

エディタの画面をよく見てみましょう。さきほど定義した出力側のフィールド名が `private` 変数 (`private SFCOLOR m_outColor` の部分です) として定義されています。同時に、`initialize()` メソッド内でこの変数を初期化するコードも埋め込まれています。`_inBoolCB()` というメソッドは、`inBool` フィールドにイベントが入力された場合にコールされるメソッドです。センサーからの入力によって動作するプログラムはこのメソッド内に記述します。図 13 に示すように、入力の値を見るための、以下のデバッグコードを `_inBoolCB()` メソッド内に追加してください。

```
System.out.println("_inBoolCB() called: " +  
ev.getValue());
```

これで、入力されたイベントの値がわかります。追加したらコンパイルしておきます。右ボタンをクリックして、ポップアップメニューから `Compile` を選択します。コンパイル中のメッセージは `Message` ウィンドウに表示されます。正常に終了すれば、`Done` となります。コンパイルエラーが表示された場合にはタイプミスなどがないか確認して、修正した後再度コンパイルしてください。



図 13 スクリプトにデバッグコードを追加する

■ Script ノードへの Route の設定

Script ノードを動作させるためにも Route は必要です。ここでは、TouchSensor で発生したイベントを Script ノードに Route させます。色を変えるのはその後で行います。

それでは作業を行ってみましょう。Cone の上にマウスがきたら Script ノードにイベントを渡す必要があります。TouchSensor が管理するオブジェクトの上にマウスが来たとき発生するイベントは `isOver` です。またこれを受け取る側の Script ノードのフィールドを、さきほど Script Expert で指定した `inBool` にします。Route を追加すると、Route ウィンドウは図 14 に示すような状態になるはずです。



図 14 TouchSensor から Script へ Route の設定をする

■ 動作確認

それでは実行してみましょう。スクリプト側の出力メッセージを表示するために、Conductor ウィンドウの View Java Console メニューで Java console ウィンドウを表示しておきます。次に、Play ボタンを押します。マウスを動かしてみましょう。図 15 に示すように、Cone の上にマウスカーソルが来ると true というメッセージが表示され、Cone から離れると false というメッセージが表示されるはずです。

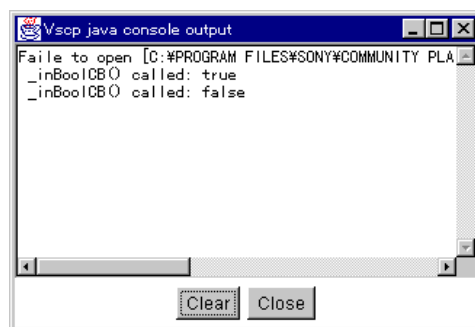


図 15 Java Console ウィンドウでデバッグメッセージを確認する

ここでのワールドの動きは次のようになります。Cone の上にマウスカーソルがくるとそこに付属している TouchSensor ノードがそれを検出しイベントを発生します。このイベントは Route の設定により Script ノードに伝えられます。Script ノードはこれを受け、メソッドを実行します。メソッド中の `System.out.println()` が実行されると、Java Console ウィンドウに結果が表示されます。マウスカーソルが Cone から離れた時にも同じ流れでイベントが伝えられ、Java Console ウィンドウに結果が表示されます。

■ Script ノードに対応するプログラムの記述（色を設定する）

これで、無事 Script ノードにイベントが渡されていることが確認できました。次は色を変えるコードを書いてみます。色を変えるためには、出力側のフィールド（SC1.java の中では `m_outColor` という変数名です）に値を書き込み、それを Material に Route します。以下のコードを `_inBoolCB()` に加えてください。

```
float r[] = {1.0f, 0.0f, 0.0f};
float g[] = {0.0f, 1.0f, 0.0f};
if (ev.getValue()) m_outColor.setValue(r);
else m_outColor.setValue(g);
```

これにより、Cone にマウスカーソル入ってきたとき（`ev.getValue()` は `true` になります）に Script から赤が出力され、マウスカーソルが出ていったとき（`ev.getValue()` は `false` になります）に Script から緑が出力されるようになります。

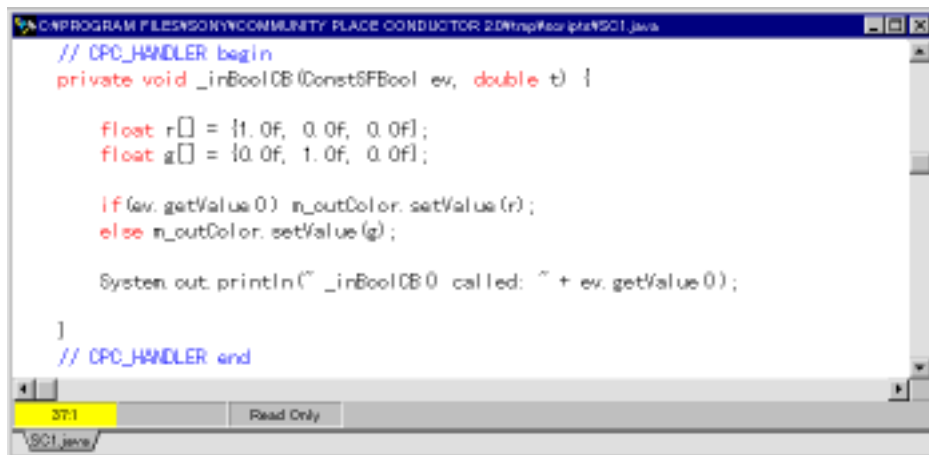


図 16 色を変えるコードを埋め込んだところ

■ Material ノードへの Route の設定

実際に Cone の色を変えるためには、Script ノードから Material ノードへ Route する必要があります。

下図のように、Route ウィンドウで SC1 の *outColor* を CONE1_MAT の *diffuseColor* に Route してみてください。

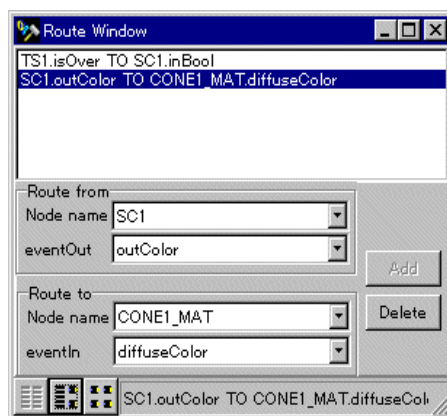


図 17 Script ノードから Material ノードへ Route する

これにより、先に設定した Route と合わせて、TouchSensor ノード (TS1) から Script ノード (SC1) へイベントが伝搬され、さらにその Script ノードから Material ノード (CONE1_MAT) にイベントが伝搬されるというイベントの流れが完成しました。

■ 動作確認

さあ、実行してみましょう。Play ボタンを押して、カーソルを Cone の上に持ってきてみてください。Cone の色が赤くなったはずです。カーソルを Cone から離してみてください。今度は緑色になったはずです。

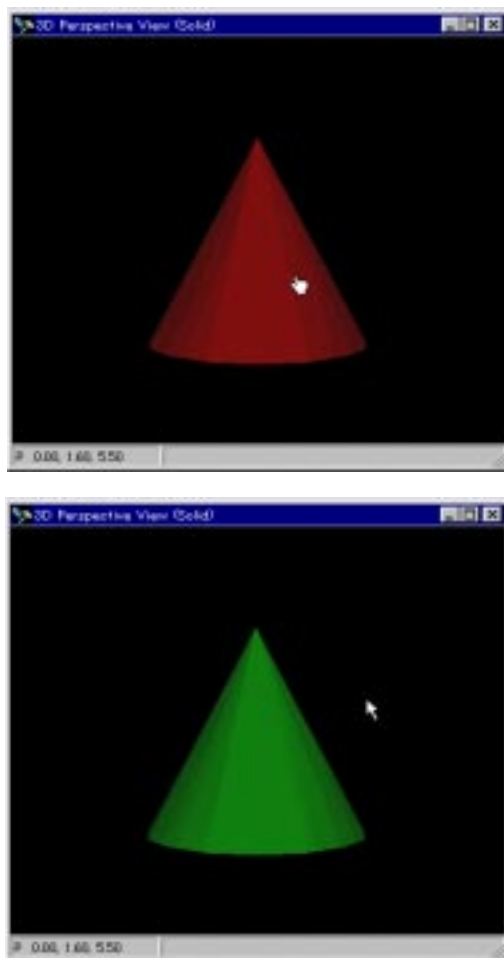


図 18 Cone の上にマウスカーソルがくると赤くなり（上）、出て行くと緑（下）になる様子

ここでのワールドの動きは次のようになります。つまり、Cone の上にマウスカーソルがくるとそこに付属している TouchSensor ノードがそれを検出しイベントを発生します。このイベントは Route の設定により Script ノードに伝えられます。Script ノードはこれを受け、メソッドを実行します。メソッド中では、入力イベント (true) を受け取ると出力のフィールドに赤を設定するようになっています。出力フィールドに書き込まれた色は、さらに Route により Cone の Material にセットされて実際に Cone の色が赤くなるのです。同様に、マウスカーソルが Cone から離れた時は TouchSensor からイベント (false) が伝達され、これにより Script ノードのメソッド中では出力のフィールドに緑を設定し、この色が Route されて Cone を緑色に変化させます。

第3章 Cone をクリックして音を鳴らす

今度は音を鳴らしてみます。VRML97 では、Sound ノードを使って音を鳴らすことができます。また、Conductor ではサウンドライブラリからドラッグ&ドロップすることで、ワールドに Sound ノードを追加することもできます。

全体の流れは以下ようになります。はじめは、Sound ノードのデフォルトの動作について確認します。その後属性の変更や Route の設定を行い、クリックして音を鳴らすことを確認してみます。は「第1章 Cone をクリックしてライトを ON/OFF する」(6 ページ)と同じなので、説明は省略します。あらかじめ Conductor を再起動し、Cone や TouchSensor を配置しておいてください。また、TouchSensor には TS1 という名前もつけておいてください。

Cone (Geometry ノード) の配置

TouchSensor (Sensor ノード) の配置

Sound ノードの配置

動作確認

AudioClip ノードの設定と Route の設定

動作確認

以下に各ステップについて説明します。

■ Sound ノードの配置

音を鳴らすためには Sound ノードを配置する必要があります。Conductor では、Library ウィンドウ (Sound タブをクリックします) にあるサウンドライブラリをワールドにドラッグ&ドロップすることでサウンドノードを追加します。

まず、3D View ウィンドウで Cone が見えていることを確認しておきます。Libray ウィンドウのサウンドライブラリから aki6.wav を選択し、3D View ウィンドウにドロップします。これで Sound ノードがワールドに追加されました。Scene Graph ウィンドウのツリーをクリックしてみると、図 19 のように Sound ノードが追加されていることがわかります。

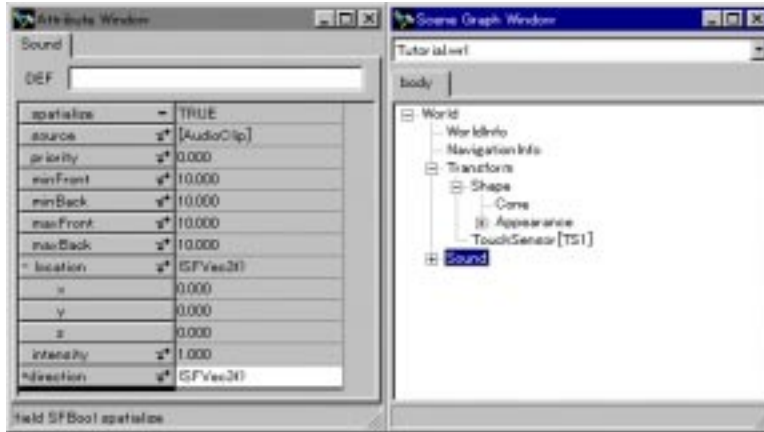


図 19 ライブラリからサウンドをドロップしたところ

Sound ノードでは *maxBack*, *maxFront* 等のフィールドによりサウンドの領域指定を行うことができます。デフォルトの状態（ライブラリからドロップされた時の状態）では、*maxBack* と *maxFront* の値は 10 に設定されており、この領域内に入ると音が聞こえるようになります。

■ 動作確認

サウンドの確認は Play ボタンで行います。この時点では、視点はサウンドの領域の境界に位置するため音は鳴りません。そこで、Cone に少し近づいてサウンドの領域内に入ってみてください。鈴虫の鳴く音が聞こえて来るはず。3D View ウィンドウでマウスを使ってナビゲーションすると、場所に応じて音の大きさや左右のパンが変化します。これは、デフォルトの状態では Sound ノードの *spatialize* フィールドが TRUE になっているからです。

Stop ボタンを押して編集モードに戻り、サウンドの領域を見えます。まず、Scene Graph ウィンドウで Sound を選択します。すると、サウンドの領域を示すバウンディングボックスが現れます。全体が見えるように少し Cone から遠ざかりましょう。

また、Main ウィンドウの View メニューから 2D/3D View Top View を選択し、上からも見てみます。こちらのウィンドウでも全体が見えるように視点

を移動します。(Shift + マウス左ボタンで画面の下方向にドラッグします)
図 20 にサウンド領域を表示している画面を示します。

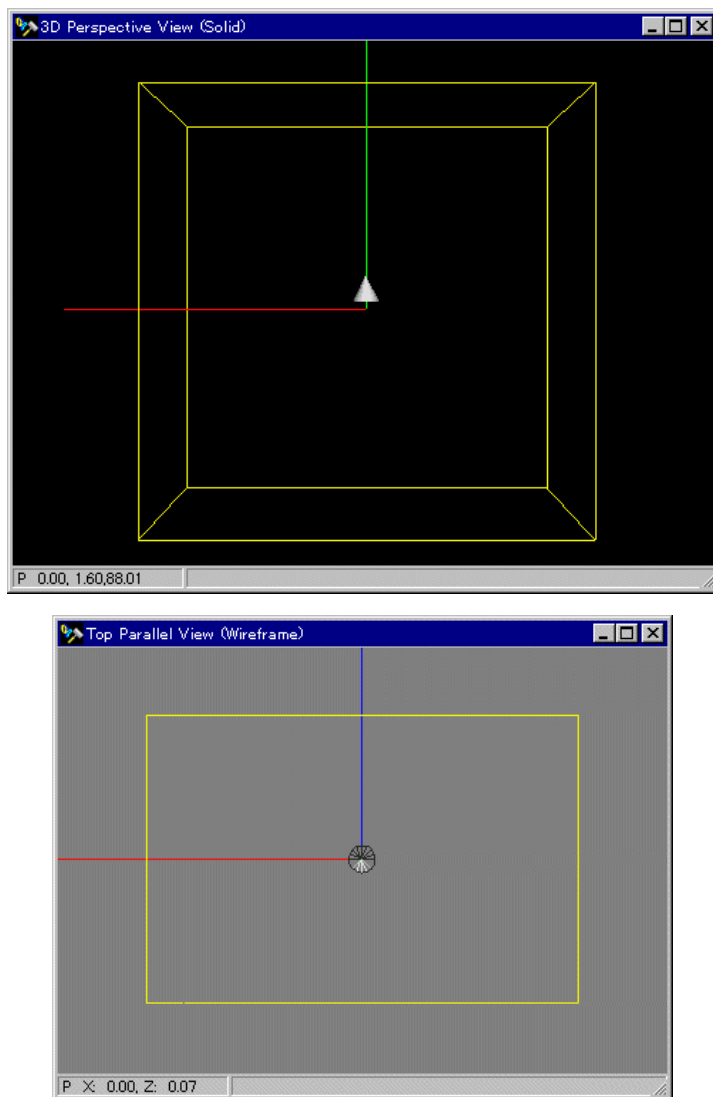


図 20 サウンドの領域を表示したところ

(注) 実際のサウンド領域は楕円体ですが、画面上は直方体で表示してあります。

Play ボタンを押してから 3D View ウィンドウ内を移動した場合、このサウンドの領域に入ると音が聞こえ、またこの領域から出ると音が聞こえなくなります。

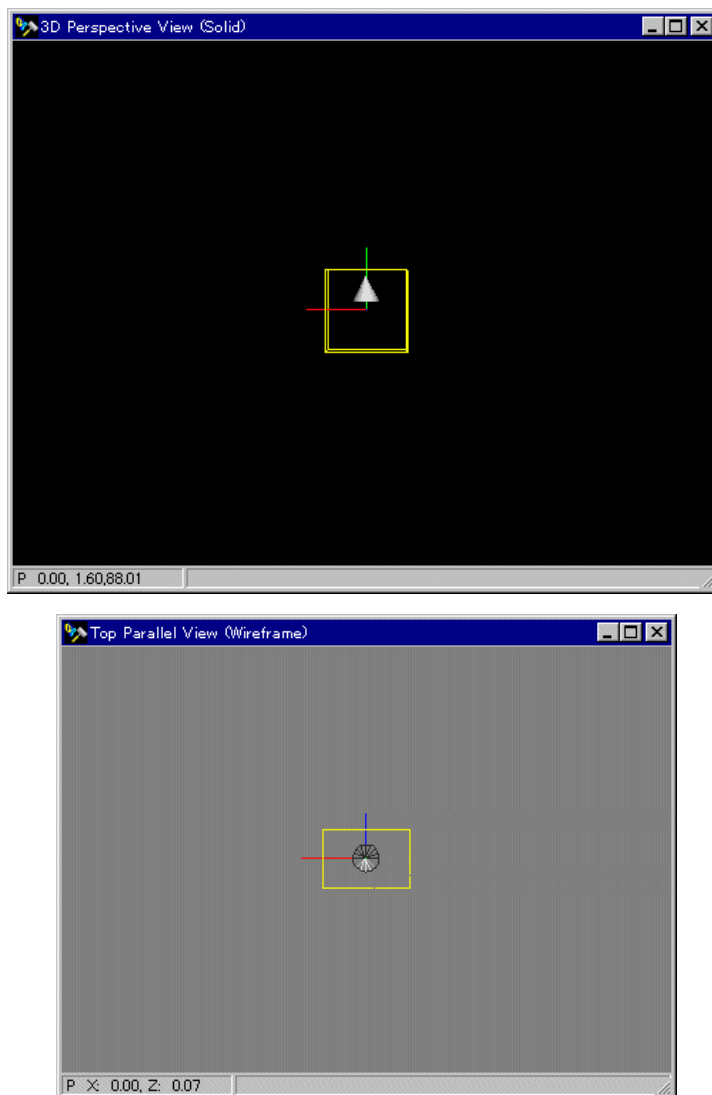


図 21 マウスによりサウンドの領域を小さくしたところ
(この作業を行う必要はありません)

(注) これらの領域は Attribute ウィンドウで数値指定することができますが、Main ウィンドウでマウスモードを切り替えることにより、マウスを使ってこの領域を移動・回転・スケールしたりすることもできます。(ただし、マウスによるスケールは等方スケールとなります)

■ AudioClip ノードの設定と Route の設定

サウンドをドロップした状態では、サウンドの領域内に入ると音が自動的に鳴りました。これは、サウンドノードについている AudioClip ノードの $startTime=0$, $stopTime=0$, $loop=TRUE$ に関係があります。VRML97 では、 $startTime$ は開始時刻を表し、 $stopTime$ は終了時刻を表します。

$startTime \geq stopTime$ の時は $stopTime$ が無効として扱われるので $loop=TRUE$ であれば音を繰り返し鳴らすので永遠に音が鳴り続けることになります。しかし、今度はクリックするまでは音をならさないように変更する必要があります。このためには、 $startTime=-1$ とします。

またクリックしたら音が鳴るようにするためには、 $startTime$ にクリックした時刻を代入するようにします。このためには、まず AudioClip に名前をつけておく必要があります。AudioClip ノードに AC1 という名前をつけ、TouchSensor (TS1) の $touchTime$ から AudioClip ノードの $startTime$ に Route してみましょう。

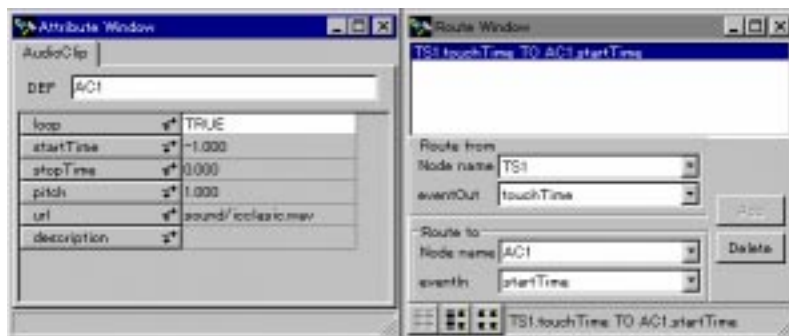


図 22 AudioClip の属性変更と TouchSensor から AudioClip への Route

■ 動作確認

それでは、動作を確認してみましょう。Play ボタンを押して Cone をクリックしてみます。音が鳴りはじめたはずです。

ここでのワールドの動きは次のようになります。Cone をクリックするとそれに付属している TouchSensor から *touchTime* というイベントが発生します。このイベントによってクリックした時刻 (00:00:00 GMT Jan 1 1970 からの相対時間) が AudioClip の *startTime* に渡されます。*startTime* がセットされると音が鳴り始めます。また、AudioClip の *loop* フィールドが TRUE になっているのでこの音は繰り返し再生しつづけます。

ただし、このままでは音を止めることはできません。音を止めたい場合には、Script ノードを使ってプログラムを書く必要があります。

第4章 ビジュアル表示 Route 編集

Route ウィンドウでは、リスト表示とビジュアル表示という2種類の編集方法を用いることができます。第1章から第3章まではリスト表示 Route 編集を用いてきましたが、本章ではビジュアル表示 Route 編集を説明します。リスト表示 Route 編集を通じて VRML における Route の仕組みが理解できるので少なくとも一度は試して頂きたいのですが、それに慣れたらビジュアル表示 Route 編集に移行すると良いでしょう。ビジュアル表示 Route 編集ではノード間の関係が視覚的に分かりやすく表示されるため、多くのノードが存在する場合でも編集が行いやすくなっています。

本章では第1章と同じ内容を、ビジュアル表示 Route 編集を用いて実現します。全体の流れは以下ようになります。は「第1章 Cone をクリックしてライトを ON/OFF する」(6 ページ)と同じなので、説明は省略します。

- Cone (Geometry ノード) の配置
- TouchSensor (Sensor ノード) の配置
- PointLight (Common ノード) の配置
- Route 対象のノードに名前を付ける
- Route の設定
- 動作確認

■ Route 対象のノードに名前を付ける

リスト表示 Route 編集では、Attribute ウィンドウを用いてノードに名前を付け、次にその名前を用いて Route ウィンドウ上で Route の作成を行います。しかしビジュアル表示 Route 編集では、こういった作業は Attribute ウィンドウを用いなくても Route ウィンドウ内で、ドラッグ&ドロップ操作によって視覚的かつ直観的に行うことができます。

まず、Route ウィンドウをビジュアル表示編集モードにしておきます。編集モードは同ウィンドウ左下の3つのスイッチにより切り替えることができます。ここでは一番右側のスイッチを押し、アイコン表示状態にします。そして Scene Graph ウィンドウ上の PointLight を Route ウィンドウにドラッグ&ドロップします。

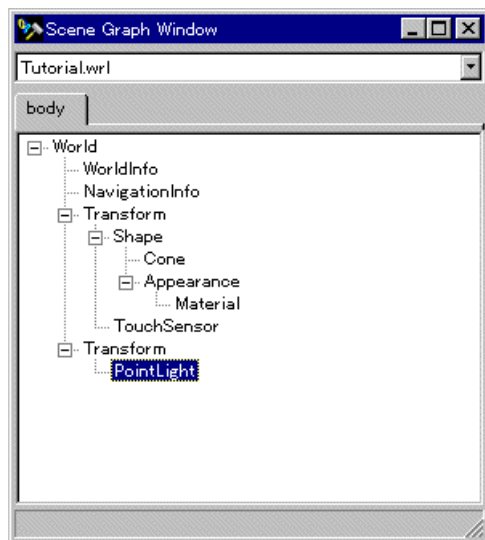


図 23 Scene Graph 上の PointLight を Route ウィンドウにドラッグする

すると PointLight のアイコンが生成され、自動的に PointLight_00 という名前が付けられます。もちろん名前を編集して、好きな名前にすることもできます。

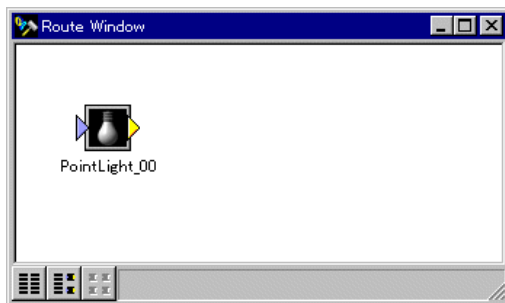


図 24 PointLight に対して自動的に名前が付けられた

■ Route の設定

次に Route を作成します。それには、イベントを発生するノード(TouchSensor など)を、Route ウィンドウにあるイベントを受け付けるノード (PointLight など) 上にドラッグ&ドロップします。

ここでは Scene Graph ウィンドウ上の TouchSensor を Route ウィンドウまでドラッグして、先ほど生成された PointLight のアイコンの上にドロップします。

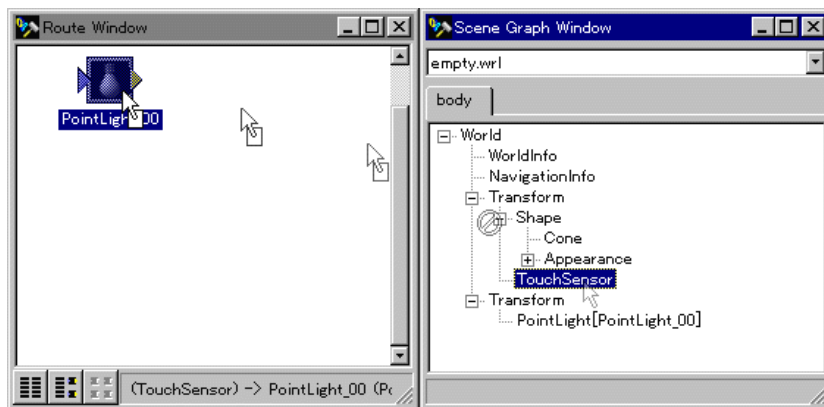


図 25 Scene Graph の TouchSensor を Route ウィンドウ内の PointLight のアイコンの上にドラッグ&ドロップする

Route を選択するためのメニューが表示されるので、「(TouchSensor).isActive TO PointLight_00.on」を選択します。

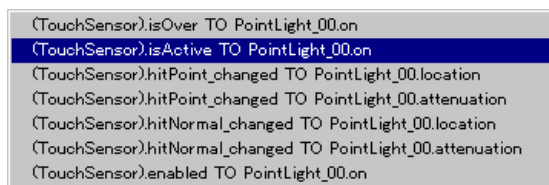


図 26 Route を選択するためのポップアップメニュー

すると Route ウィンドウ上には TouchSensor のアイコンとともに、TouchSensor から PointLight への Route が生成されます。

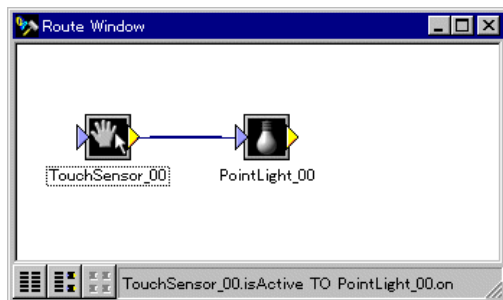


図 27 Route が生成された

これで Route の作成は終了です。なお、Route ウィンドウの表示モードを切り替えてみると、生成された Route が第 1 章と同様のものであるということが分かります。

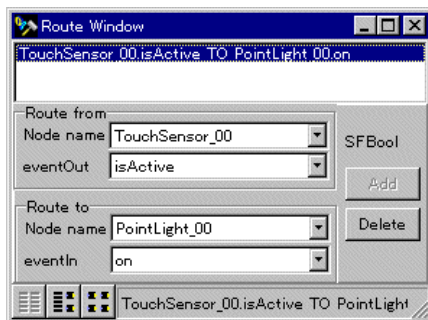


図 28 第 1 章と同様の方式で Route を表示したところ

■ 動作確認

あとは動作確認を残すのみです。Main ウィンドウの Play ボタンを押してみてください。Cone をクリックすると、ライトのスイッチが入って明るく照らされるはずです。

第5章 キーフレームアニメーション

VRML97では、TimeSensor ノードや Interpolator ノードを組み合わせることによって、キーフレームアニメーションを作ることができます。キーフレームアニメーションでは、ポイントとなる幾つかのキーフレームを用意しておくことで、キーフレームとキーフレームの間は Interpolator ノードが自動的にフレームの補間を行います。このため、比較的楽に滑らかなアニメーションを作成することができます。全体の流れは以下のようになります。

Cone (Geometry ノード) の配置

TimeSensor (Sensor ノード) の配置と名前付け

アニメーション編集モード

移動：1 番目のキーフレームを作成

移動：2 番目のキーフレームを作成

動作確認

アニメーションの速度を変える

回転：1 番目のキーフレームを作成

回転：2 番目のキーフレームを作成

動作確認

以下、各ステップについて説明します。

■ Cone (Geometry ノード) の配置

まずはオブジェクトを配置します。Conductor を起動してメニューから File New を選択し、空の世界を作成します。そして Cone を置きます。

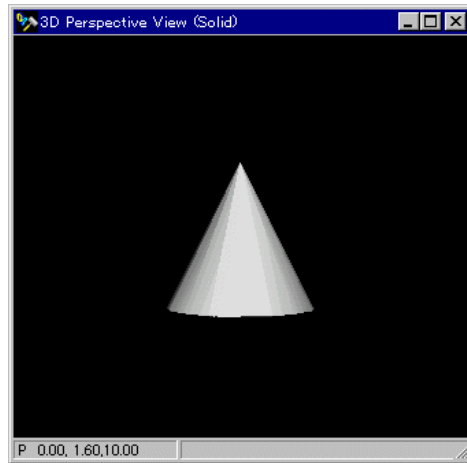


図 29 Cone を配置する

■ TimeSensor (Sensor ノード) の配置と名前付け

次に、アニメーションのための TimeSensor を配置します。アニメーションでは時間の経過とともにオブジェクトが変化していきますが、この時間の経過をイベントとして発生させるために TimeSensor を用います。

3D View ウィンドウでオブジェクトの無い部分をクリックするか、Scene Graph ウィンドウにて World を選択状態にしておきます。次に Main ウィンドウから Sensor タブをクリックし、TimeSensor アイコンを選択した後に、3D View ウィンドウ内をクリックします。これで TimeSensor が配置できました。なお、この TimeSensor には名前を付けておく必要があります。TimeSensor を Scene Graph ウィンドウから Route ウィンドウへとドラッグして、名前を付けます。ここでは「TimeSensor_00」という名前が自動的に付けられました。

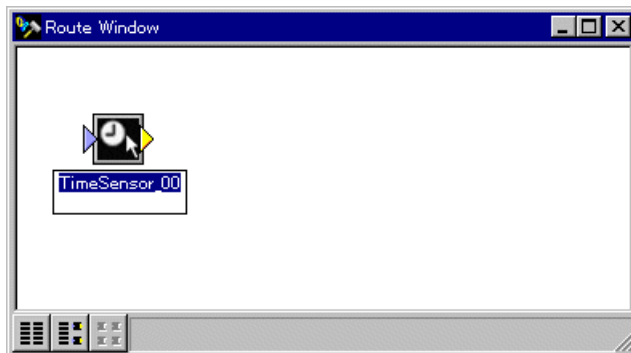



図 30 Route ウィンドウを用いて TimeSensor に名前を付ける

■ アニメーション編集モード

いよいよキーフレームの作成に入ります。まずはMainウィンドウのKeyframe Editor ボタン  を押します。すると図 31 に示す Keyframe Editor ウィンドウが開きます。Keyframe Editor ウィンドウでは、キーフレームの生成や選択、アニメーションの動作確認などを行います。

さて、始めに幾つかの設定を行っておく必要があります。まず、左上の Source コンボボックスにおいて、「TimeSensor_00.fraction」を選択します。これは、先ほどの TimeSensor をアニメーションのタイマーとして用いるための設定です。次に、その下方にある「Loop」チェックボックスをチェックしておきます。これはアニメーションを繰り返し再生するための設定です。



図 31 Keyframe Editor ウィンドウにて幾つかの設定を行う

■移動：1 番目のキーフレームを作成

それでは1 番目のキーフレームを作成します。

まず3D View ウィンドウを右クリックすると現れるポップアップメニューにおいて、MouseMode Move を選択します。これでオブジェクトをマウスで動かすことが出来るようになるので、Cone を動かしてウィンドウの左上に持って行きます。

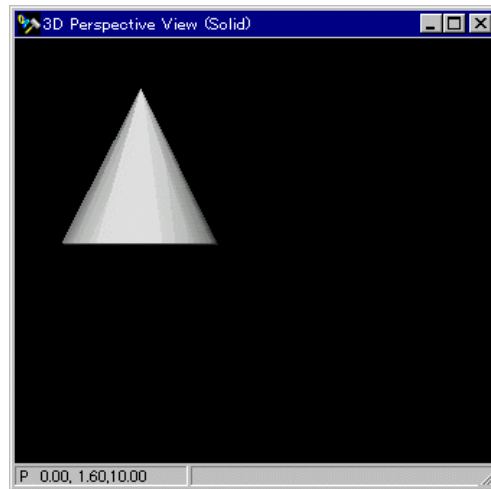


図 32 Cone を左上に移動させる

このときの Keyframe Editor ウィンドウは図 33 のような状態になります。目盛りの下部に表示されているのはキーフレームが登録されている位置を示すグラフです。今作成したキーフレームは、目盛りの0 の位置に登録されました。キーフレームが登録されている箇所には太い縦棒が表示されており、これをクリックすることによってそのキーフレームを編集することができます。通常のアニメーションでは、目盛りの0 から1 までの間にキーフレームを登録します。

Attribute ウィンドウで Interpolator ノードから Route されている Transform ノードの *translation* の値を直接設定することで、キーフレームを設定することも可能です。

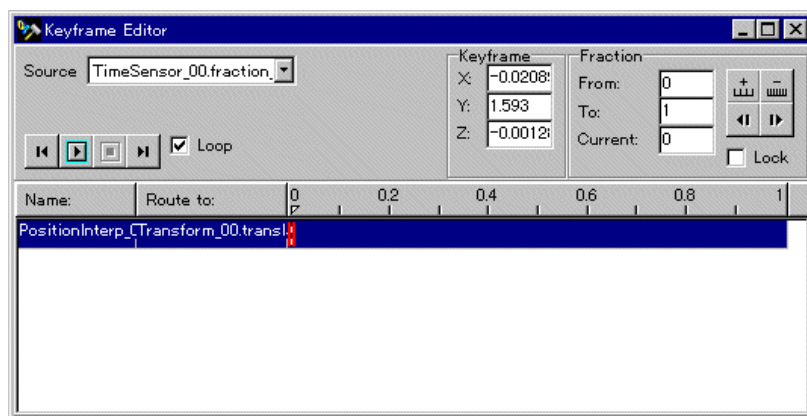


図 33 キーフレーム登録時の Keyframe Editor ウィンドウの状態

一方、Route ウィンドウは図 34 のようになります。自動的に Interpolator ノード（ここでは PositionInterpolator ノード）が生成され、また TimeSensor PositionInterpolator Transform という Route も生成されています。

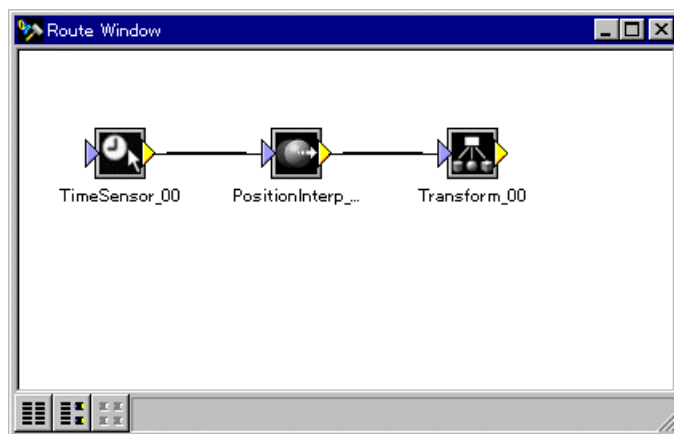


図 34 Interpolator ノードや Route が新たに生成された

Interpolator ノードは、登録されたキーフレームを基に、TimeSensor から受け取った時間情報に対応したフレームを補間処理によって生成し、Transform ノードを変更する役割を担っています。

■移動：2番目のキーフレームを作成

続いて2番目のキーフレームを作成します。

目盛り1の下あたりをクリックして、点線で示されるロケーターがその位置に来るようにします。そして同じ場所を右クリックして、ポップアップメニューにて New Keyframe を実行すれば、新しいキーフレームが生成されます。新たな太い縦棒が表示されて、先程の1番目のキーフレームとの間に太い横棒が表示されることを確認してください。

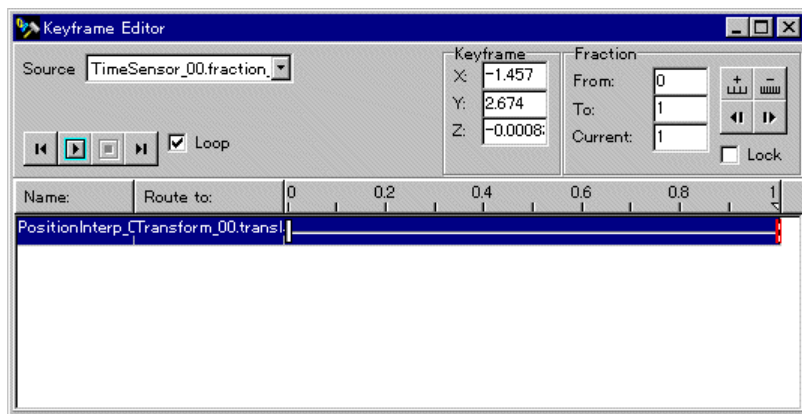


図 35 Keyframe Editor ウィンドウにて2番目のフレームを作成する

次に 3D View ウィンドウ上で Cone を右下に移動させます。これで2番目のキーフレームが出来ました。

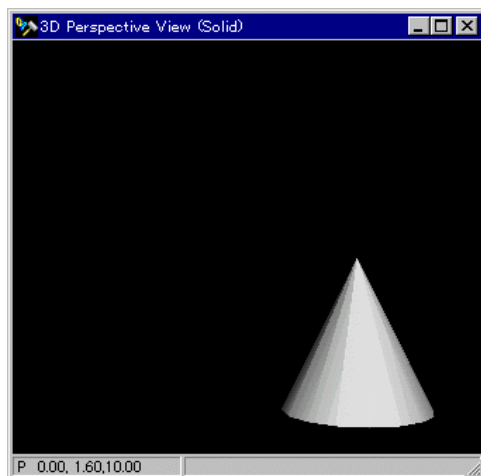


図 36 Cone を右下に移動させれば 2 番目のキーフレームは完成

■ 動作確認

ではアニメーションを実行してみましょう。Keyframe Editor ウィンドウを押すか、Keyframe Editor ウィンドウを閉じて Main ウィンドウ上の Play ボタンを押します（注 1）。するとアニメーションが実行されて、Cone がウィンドウの左上から右下に向かって移動するというアニメーションが繰り返されます。もしも上手く動かないようであれば、Keyframe Editor ウィンドウにて Loop チェックボックスがチェックされているか、あるいは各キーフレームが正しく作成されているか、といったことを確認してみてください。

（注 1）Keyframe Editor ウィンドウを表示している間は、Main ウィンドウの Play ボタンは使えません。

（注 2）ワールド全体の動作を確認するためには、Keyframe Editor を終了して、Main ウィンドウの Play ボタンを押してください。

■ アニメーションの速度を変える

アニメーションの速度を変えるには、TimeSensor ノードの *cycleInterval* フィールドを変更します。先ほどのアニメーションでは多少速度が速いようですので、遅くしてみることにしましょう。まずは TimeSensor を Route ウィンドウ

または Scene Graph ウィンドウにて選択します。そして Attribute ウィンドウにて *cycleInterval* を調整します。ここではアニメーション全体の所要時間を秒単位で入力します。先ほどは 1 秒になっていたのですが、今度は 4 秒にしてみました。この値を大きくするとアニメーションは遅く、小さくすると速くなります。



図 37 TimeSensor の *cycleInterval* フィールドでアニメーションの速度を変更する

変更が済んだら、動作を確認してみてください。アニメーションの速度が遅くなっているはずです。なお、フレームレート（秒当たりの描画回数）は実行環境における描画速度に依存しますので、*cycleInterval* に同じ値を設定していても、実行環境によってアニメーションのスムーズさは異なります。

■ 回転：1 番目のキーフレームを作成

先ほどのアニメーションは Cone が移動するだけでしたが、今度は回転も加えてみることにしましょう。

まずは Keyframe Editor ウィンドウの右側のグラフにて 1 番目のキーフレームを選択します。それから 3D View ウィンドウにて右クリックを行い、ポップアップメニューから MouseMode Rotate を選択しておきます。すると Cone を回転させられるようになったので、適当に Cone を回転させます。これで 1 番目のフレームは出来上がりです。

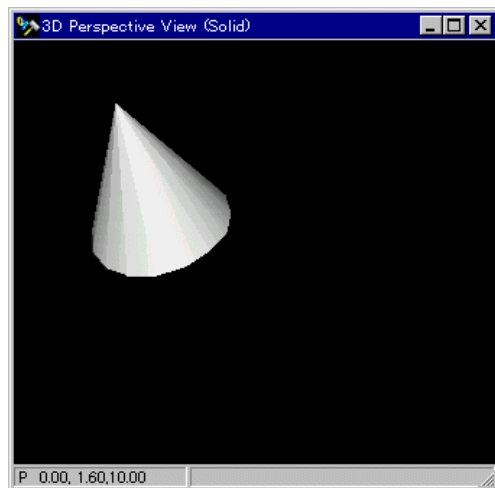


図 38 Cone を回転させて 1 番目のキーフレームを作成

すると Keyframe Editor ウィンドウには新たに OrientationInterpolator が追加され、PositionInterpolator の 1 番目のキーフレームに対応する位置に新たなキーフレームが生成されます。OrientationInterpolator は回転の補間を行うための Interpolator です。このように様々な動作を伴うアニメーションでは、複数の Interpolator を併用することになります。

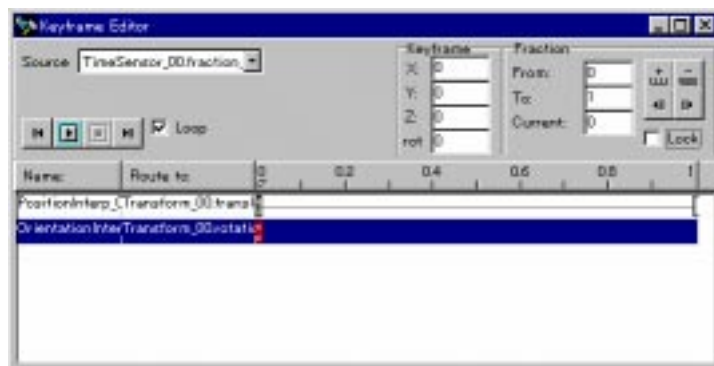


図 39 回転のための OrientationInterpolator が自動的に追加される

また、Route ウィンドウにも OrientationInterpolator および新しい Route が自動的に追加されます。

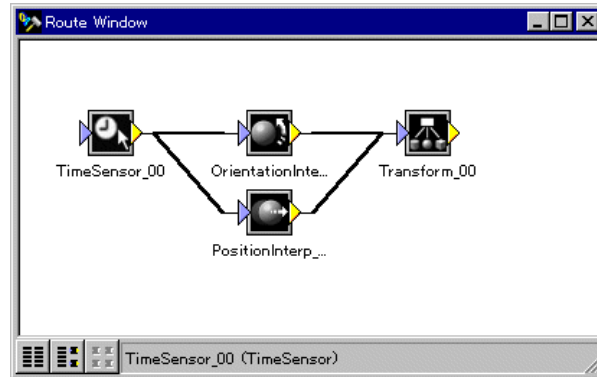


図 40 Route ウィンドウでもアイコンや Route が自動的に生成される

■ 回転：2 番目のキーフレームを作成

1 番目のキーフレームと同じ要領にて、Keyframe Editor ウィンドウの右側のグラフを用いて、目盛りの 1 のところにある 2 番目のキーフレームを選択します。そして 3D View ウィンドウにて Cone を回転させます。

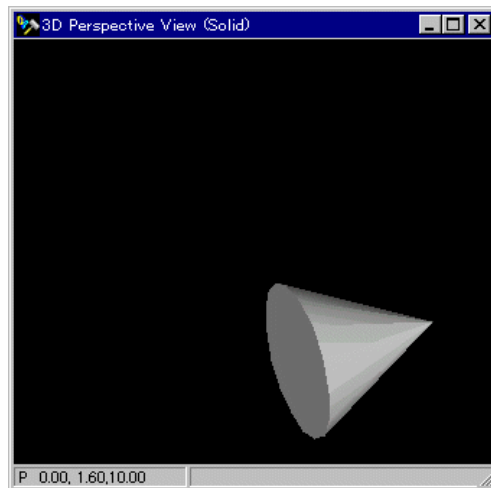


図 41 Cone を回転させて 2 番目のキーフレームを作成

また Keyframe Editor ウィンドウを見ると、新たなキーフレームが追加されていることが分かります。これで回転についてもアニメーションが完成しました。

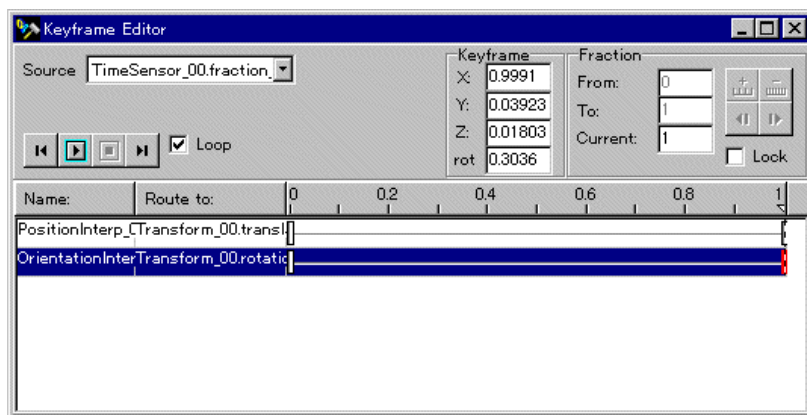


図 42 Keyframe Editor ウィンドウにて新たなキーフレームの生成を確認

■ 動作確認

それではアニメーションを実行してみましょう。41 ページの動作確認と同様に、Keyframe Editor ウィンドウの Play ボタンを押すか、Keyframe Editor ウィンドウを閉じて Main ウィンドウ上の Play ボタンを押します。すると、Cone がウィンドウの左上から右下に向かって回転しながら移動するというアニメーションが繰り返されます。

移動と回転だけではなく、スケールや色の変化といった動作も組み込んで、より複雑なアニメーションを実現することもできます。それには、各キーフレーム編集時に、物体を移動させたり回転させたりする代わりに、Attribute ウィンドウの color や scale といった exposedField を変更します。具体的な操作については「第 7 章 応用的なキーフレームアニメーション」(57 ページ) を参照して下さい。

第6章 PROTO を使う

複数のプリミティブを組み合わせて複雑な物体、例えばテーブルを作ったときのことを考えてみてください。作成中のシーンにおいて、テーブルが1つだけではなく、多くのテーブルを使うような場合、個々のテーブルについてプリミティブから組み立てることは避けて、初めに作ったテーブルを複製して再利用したいところです。また、これらのテーブルが1つ1つ微妙に色や形が異なっていたらどうでしょうか。ただ複製するだけでは済まなくなりますが、多少の違いはあってもどれもテーブルには相違ありません。そこで初めに作ったテーブルをひな形として登録しておき、複製して個々の違いに関しては簡単な操作で変更可能であれば、便利かつ、効率的です。

VRML97ではPROTOという機構が追加されており、このような「ひな形」を定義することが可能になっています。一度ひな形を定義してしまえば、それはVRML97組み込みのノード（ConeやCylinderなど）と同様に、複製したり再配置したりすることができます。また、ただ単に物体の構成要素をグループ化するだけではなく、構成要素のアトリビュートのうちの特定のフィールドを、新しく作った物体のアトリビュートとして登録することができます。この機能を用いれば、例えば「青いテーブル」「赤いテーブル」といったバリエーションを簡単に作り出せます。その上、構成要素の複数のアトリビュートを新たな単一のアトリビュートとして登録することにより、一度に変更することもできます。例えばテーブルの脚を構成するパーツが複数ある場合にも、単一のフィールドを変更することによって全てのパーツの色を変化させるといったことが可能です。なお、オブジェクト指向プログラミングをご存知の方ならば、PROTOはクラスに近いものであり、その複製とはインスタンスを作ることである、ということにお気づきになることでしょう。

さて、本章ではテーブルを作成する例を通じて、PROTO機能の使い方を説明します。全体の流れは以下のようになります。

- プリミティブ（Cylinder）を組み合わせて物体（テーブル）を作成
- PROTOの作成
- テーブルの複製（インスタンスの作成）
- PROTO内部のフィールドを外部に見せる（1）
- PROTO内部のフィールドを外部に見せる（2）
- インスタンスのフィールドを変更

以下、各ステップについて説明します。

■ プリミティブ (Cylinder) を組み合わせて物体 (テーブル) を作成

まずはプリミティブを配置して、テーブルを作ります。

Conductor を起動してメニューから File → New を選択し、空の世界を作成します。そして Cylinder (円柱) を配置します。Cylinder を 1 つ置いては移動させ、合計 3 つの Cylinder を用いてテーブル状に配置します。Cylinder の形を変えるには、3D View ウィンドウにて Cylinder をクリックしておき、Attribute ウィンドウにて *radius* と *height* を調整します。



図 43 Cylinder を変形しつつ配置してテーブルを作る

Cylinder を置く際には Scene Graph ウィンドウ上で Cylinder の親ノードとなっている Transform ノードをクリックしてから置くようにして、全ての Cylinder が共通の Transform ノードの子ノードになるようにします。全て配置してしまってから、Scene Graph ウィンドウ上でカット&ペーストやドラッグを行い、ノードの位置を調整しても構いません。また、全ての Cylinder が共通の Transform ノードの子ノードでありさえすれば、細部はこの図 44 と異なっても構いません。

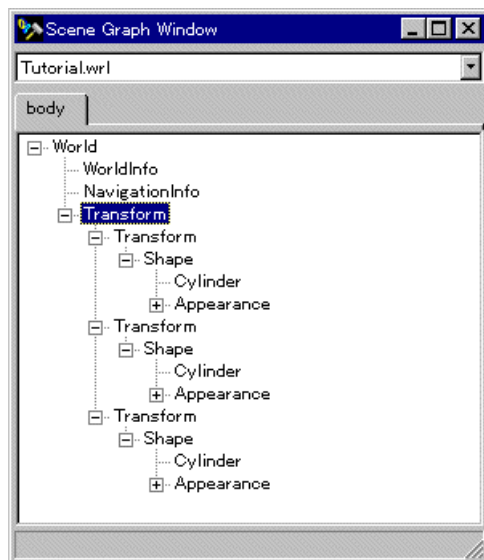


図 44 テーブル作成後の Scene Graph ウィンドウ

■ PROTO の作成

次は作ったテーブルを PROTO として定義します。

Scene Graph ウィンドウ上で3つのCylinderの共通の親ノードである Transform ノードを右クリックし、現れたポップアップメニューにて CreatePROTO を選択します。すると入力ダイアログが表示されるので、PROTO 名として「Table」を入力します。

(注) CreatePROTO により PROTO を生成する際に、それまでの作業に対する Undo ができなくなるという Warning が表示されます。

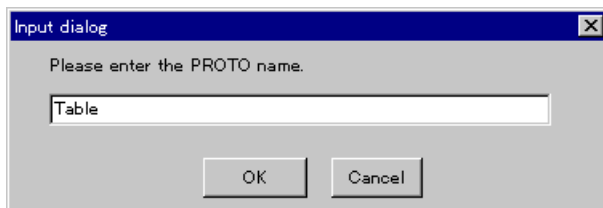


図 45 ダイアログにて PROTO 名を入力

すると先の Transform ノードの内容がまとめられて、1 つの新たなユーザ定義ノード「Table」になります。これで PROTO 定義は完了しました。

(注) CreatePROTO により PROTO を生成すると、自動的に大文字表記による DEF 名も付けられます。

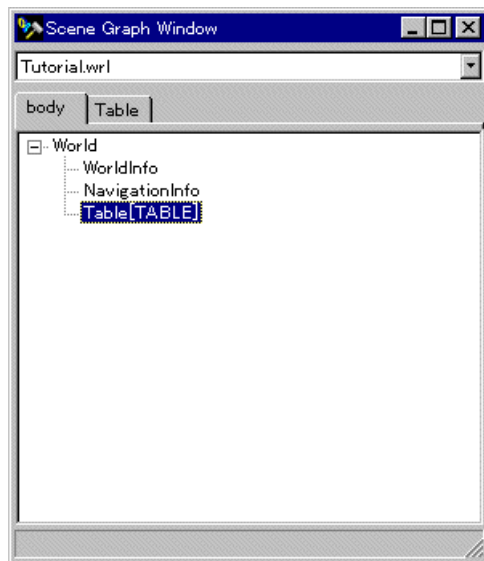


図 46 PROTO 定義が行われた後の Scene Graph ウィンドウ

■ テーブルの複製（インスタンスの作成）

それでは作成したテーブルを複製してみることにしましょう。

まず、先のテーブルを左上に動かしておきます。



図 47 テーブルを移動させておく

Scene Graph ウィンドウにて Table の親ノードとなっている Transform を選択しておき、右クリックしてポップアップメニューにて Add ProtoInstance Table を選択します。すると元のテーブルと同じ位置に新しいテーブルの複製が生成されますので、Scene Graph ウィンドウにて複製の方を選択した後に、3D View ウィンドウにて図 48 のように移動させます。このように一度 PROTO 定義をしておけば、幾つでも複製を生成して、それぞれを移動・回転させることができます。



図 48 新しいテーブルを配置

■ PROTO 内部のフィールドを外部に見せる (1)

続いて、テーブルの 1 つ 1 つのインスタンスの色を簡単に变化させられるようにしましょう。そのためには Table 内部のノードのフィールドを Table というノードのフィールドとして外部に見えるように登録します。ここでは、テーブルの上板の部分と、軸および脚の部分を分けて、それぞれの Cylinder の色を指定するフィールドを Table のフィールドとして登録します。このように、PROTO 機能を用いて作り出した新しいノードの内部の複数のノードのフィールドの中から必要なフィールドを選び、新しいノードのフィールドとして登録することによって、1 つ 1 つのインスタンスの色や形状といった属性を変更できるようになります。

Scene Graph ウィンドウにて Table タブをクリックします。すると図 49 のように Table の内部構造が表示されます。

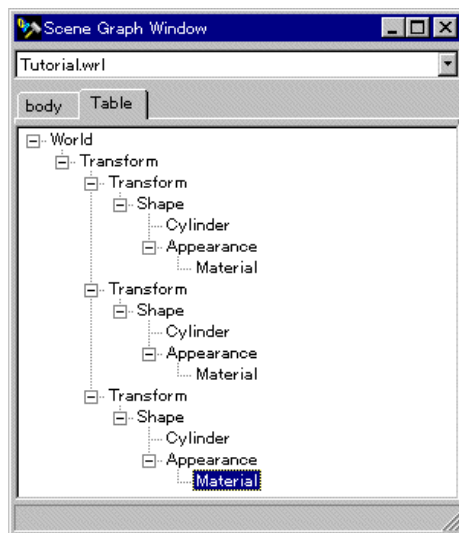


図 49 Scene Graph ウィンドウにて Table の内部構造を見る

上板の Material をクリックし、Attribute ウィンドウを開きます。対応する Cylinder ノードをクリックすると、3D View ウィンドウ上でオブジェクトを囲むバウンディングボックスが表示された選択状態になるため、上板の Material かどうかが分かります。

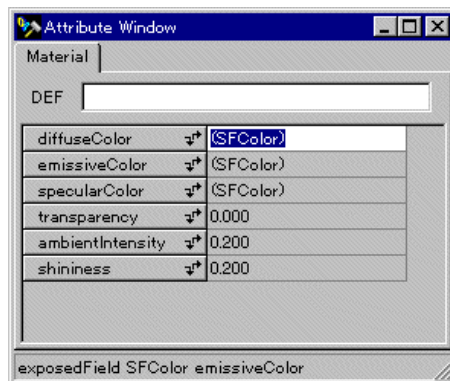


図 50 テーブルの上板の Material の属性

Attribute ウィンドウにて *diffuseColor* を右クリックし、ポップアップメニューから map with IS new を選択します。Add field ダイアログが表示されるので、図 51 に示すように field name の項に「BoardColor」と入力します。usage は exposedField、field type は SFColor で、共にデフォルトで構いません。

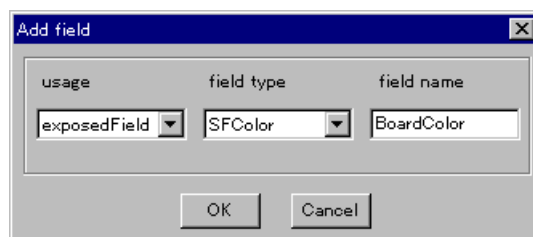


図 51 Add field ダイアログにてフィールド名を入力

Add field ダイアログにて OK を押すと、Attribute ウィンドウの *diffuseColor* フィールドには「IS BoardColor」と表示され、登録が成功したことが確認できます。これで、テーブルの上板にあたる Cylinder の *diffuseColor* フィールドは、新しく作成した Table の BoardColor フィールドとして見えるようになり、操作することができるようになりました。すなわち、Table の BoardColor フィールドを操作することにより、テーブルの上板の色を変更することが可能になりました。

これでテーブルの上板の色については登録が完了しました。

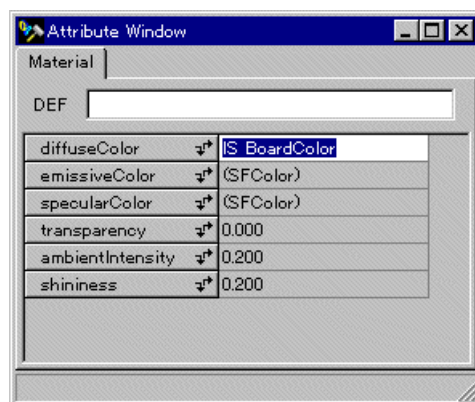


図 52 登録後の Attribute ウィンドウ

■ PROTO 内部のフィールドを外部的に見せる (2)

引き続き、今度はテーブルの軸および脚の部分の色のフィールドを Table のフィールドとして登録します。軸と脚、2つのフィールドを、Table の同一のフィールドにマッピングするのがポイントです。

Scene Graph ウィンドウにて、テーブルの軸となる Cylinder の Material をクリックして Attribute ウィンドウを開き、*diffuseColor* を右クリックします。ポップアップメニューから map with IS new を選択し、Add field ダイアログにて field name として「LegColor」と入力します。これは「PROTO 内部のフィールドを外部的に見せる (1)」(51 ページ)と全く同じ要領です。

最後にテーブルの脚となる Cylinder の Material をクリックして Attribute ウィンドウを開き、*diffuseColor* を右クリックします。ポップアップメニューから map with IS LegColor を選択します。これで、テーブルの軸と脚の色のフィールドが Table の LegColor というフィールドとして登録されます。

■ インスタンスのフィールドを変更

登録したフィールドを用いて、テーブルの色を変更してみましょう。

Scene Graph ウィンドウの body タブをクリックし、Table ノードを選択します(どちらの Table でも構いません)。すると Attribute ウィンドウに「BoardColor」と「LegColor」という2つのフィールドが現れるはずです。これらのフィールドを編集すると、PROTO 内部の対応するフィールドに値が設定されます。この仕組みによって、あたかも Table というノードにこれらのフィールドが存在するように扱うことができます。

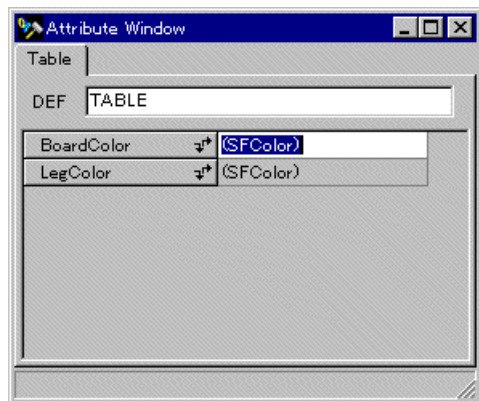


図 53 Table ノードの Attribute ウィンドウに登録したフィールドが現れる

これらのフィールドをダブルクリックして、ColorMap ダイアログにて色を指定してみてください。すると、それぞれのテーブルごとに好きな色を着けることができます。



図 54 それぞれのテーブルごとに好きな色を着けることができる

このように、*PROTO* 機能を用いることによってユーザが独自のノードを定義することができるようになります。

第7章 応用的なキーフレームアニメーション

「第5章 キーフレームアニメーション」(35 ページ)では基礎的なキーフレームアニメーションとして、Cone が回転しつつ移動するアニメーションを作成しました。本章ではより進んだアニメーションの手法として、アトリビュートの `exposedField` を変更することによって色やスケールを変化させる方法や、`TouchSensor` を用いてアニメーションを起動させる方法などを説明します。全体の流れは以下ようになります。 は「第5章 キーフレームアニメーション」(35 ページ)と同じなので、説明は省略します。

Cone (Geometry ノード) の配置

TimeSensor (Sensor ノード) の配置と名前付け

アニメーション編集モード

色の变化 : 1 番目のキーフレームを作成

色の变化 : 2 番目のキーフレームを作成

動作確認

変形 : 1 番目のキーフレームを作成

変形 : 2 番目のキーフレームを作成

動作確認

TouchSensor (Sensor ノード) の配置

Route の作成

動作確認

以下、各ステップについて説明します。

■ 色の变化 : 1 番目のキーフレームを作成

本章では時間とともに物体 (Cone) の色がだんだんと変化していくアニメーションを作ります。Cone の色を設定するには Attribute ウィンドウを使います。まず Scene Graph ウィンドウにて Cone の Material を選択しておき、Attribute ウィンドウに表示された項目の中にある `diffuseColor` フィールドをダブルクリックします。すると図 55 に示す ColorMap ダイアログが表示されます。

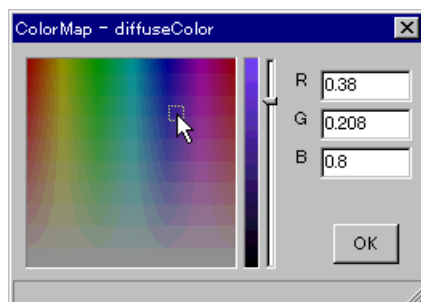
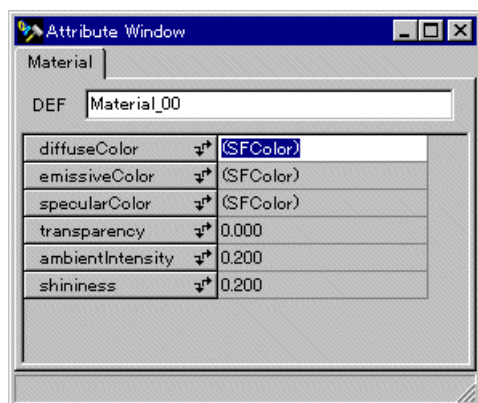


図 55 *diffuseColor* フィールド (上) とダブルクリック後に表示される ColorMap ダイアログ (下)

ここでは青系の色を設定しました。

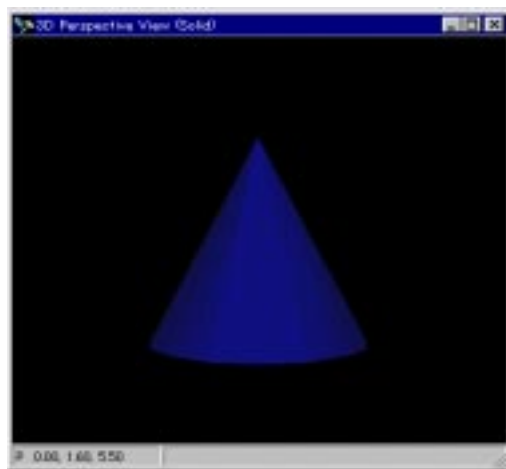


図 56 1 番目のキーフレームでは青系の色を設定した

Keyframe Editor ウィンドウには新たに ColorInterpolator が追加されます。また、ウィンドウ中央上部の Keyframe の欄には選択した色が表示されます。Keyframe の欄には Interpolator の種類に応じて様々な情報が表示されるので、キーフレームを編集する際のヒントにすると良いでしょう。

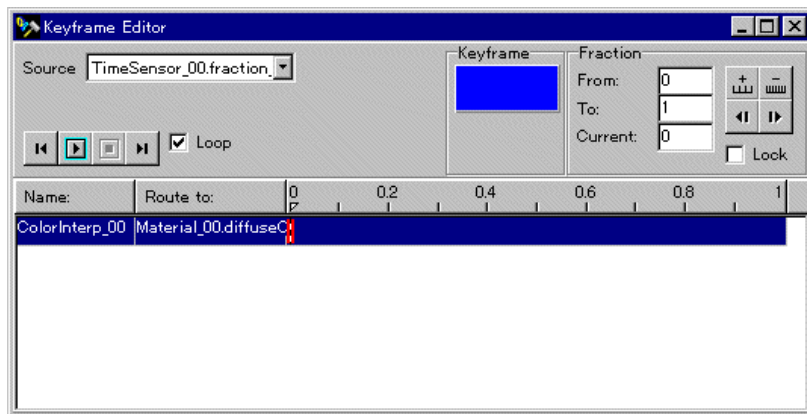


図 57 キーフレーム登録後の Keyframe Editor ウィンドウ

一方、Route ウィンドウを見ると、ColorInterpolator のアイコン、Material のアイコン、そして新たな Route が生成されています。ここで ColorInterpolator は、TimeSensor から時間情報を受け取り、それに対応した色情報を生成して、Material に対して渡しています。このように Interpolator ノードには用途に応じて PositionInterpolator / OrientationInterpolator / ColorInterpolator などの様々な種類があり、それぞれのアニメーションに関して補間処理を行います。

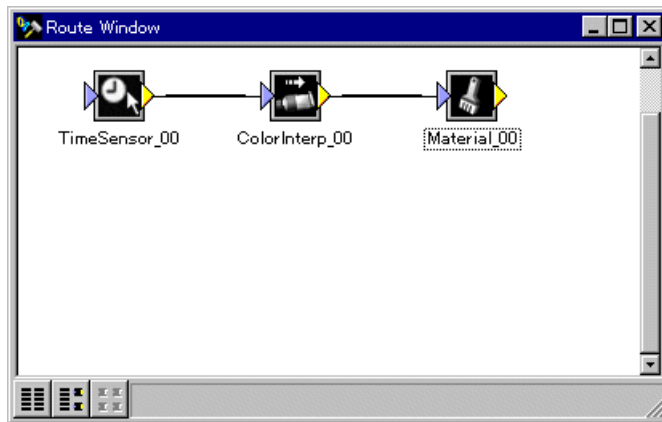



図 58 Route ウィンドウではアイコンや Route が自動的に生成されている

このように、アニメーション編集モードにおいて Attribute ウィンドウを操作すると、Interpolator ノードや Route が全て自動的に作成され、様々な種類のアニメーションを行うことができます。今回はフィールドとして *diffuseColor* を使っていますが、他のフィールドを使うことも可能です。ただし、使用可能なフィールドは `exposedField` だけであることに注意してください。`exposedField` はノードの外部から操作できるフィールドです。`exposedField` でなければ Interpolator ノード側から操作することができないので、アニメーションに使うことは不可能なのです。なおアトリビュートウィンドウ中でフィールド名の後ろに  と示されているのが `exposedField` です。

■ 色の変化：2 番目のキーフレームを作成

これで 1 番目のキーフレームは完成しましたので、続いて 2 番目のキーフレームを作成します。まずは第 5 章と同じ要領で、Keyframe Editor ウィンドウの右側にあるグラフにおいて、目盛りの 1 の下あたりをクリックして、点線で示されるロケーターがその位置に来るようにします。そして Attribute ウィンドウの *diffuseColor* フィールドをダブルクリックし、ColorMap ダイアログにて赤系の色を設定します（図 59）。

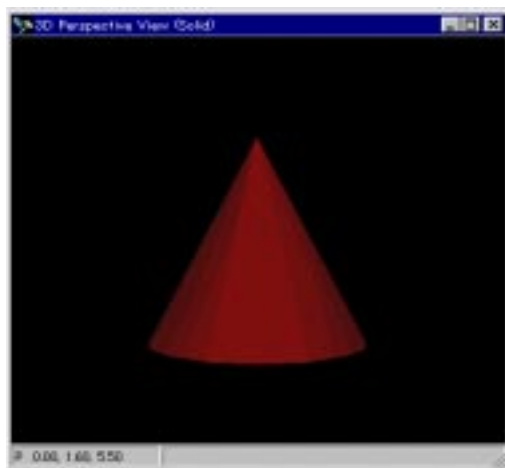


図 59 2 番目のキーフレームでは赤系の色を設定した

Keyframe Editor ウィンドウは図 60 のようになります。

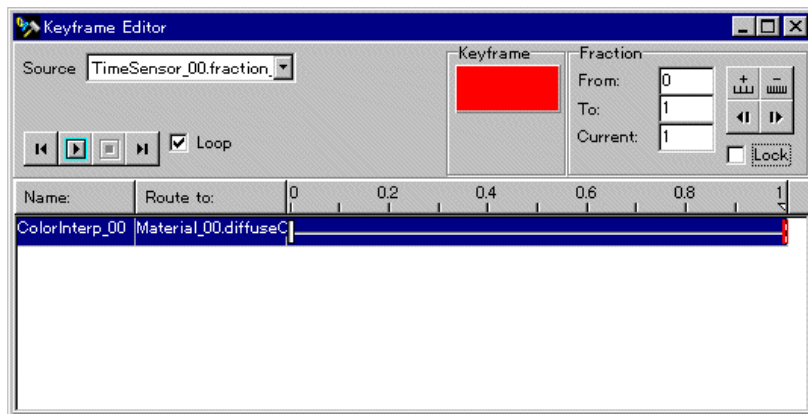


図 60 キーフレーン登録後の Keyframe Editor ウィンドウ

■ 動作確認

それではアニメーションを実行してみましょう。Keyframe Editor ウィンドウ、もしくは Main ウィンドウ上の Play ボタンを押します。すると、Cone が青から赤へと徐々に色を変化させていくアニメーションが繰り返されます。変化が速すぎて見にくい場合には、TimeSensor の *cycleInterval* フィールドを用いてアニメーションの速度を調整すると良いでしょう（「アニメーションの速度を変える」(41 ページ)を参照）。

■ 変形：1 番目のキーフレームを作成

このようにアニメーション編集モードにおいて Attribute ウィンドウを操作することにより様々な種類のアニメーションが作成できます。そこで今度は、色の変化とともに物体のスケールが変化するアニメーションを作ってみることにしましょう。

Keyframe Editor ウィンドウの右側のグラフにて 1 番目のキーフレームを選択します。それから Scene Graph ウィンドウにて Cone の親ノードになっている Transform ノードを選択し、Attribute ウィンドウを表示させます。そして Attribute ウィンドウにて *scale* フィールド（*exposedField* であることに注意）をクリックします。*scale* フィールドは物体の x,y,z 方向の拡大縮小率を設定

するフィールドです。クリックすると x,y,z それぞれの成分が表示されるので、これらを変更します。

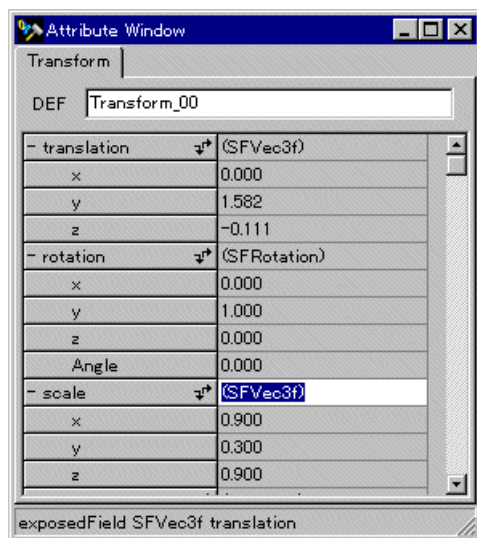


図 61 変更すべき scale フィールド

ここではやや偏平気味に変形してみました。

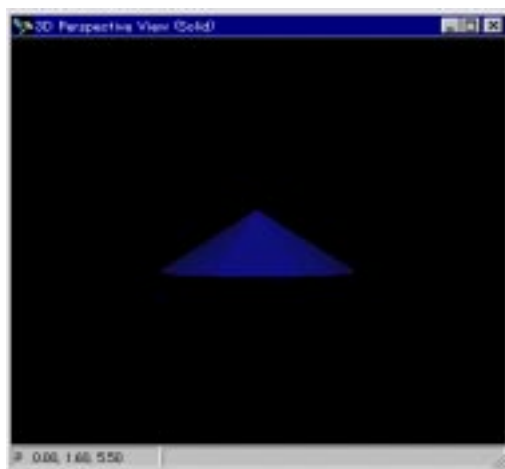


図 62 scale 変更後のやや偏平な Cone

Keyframe Editor ウィンドウを見ると、PositionInterpolator が追加されていることが分かります。

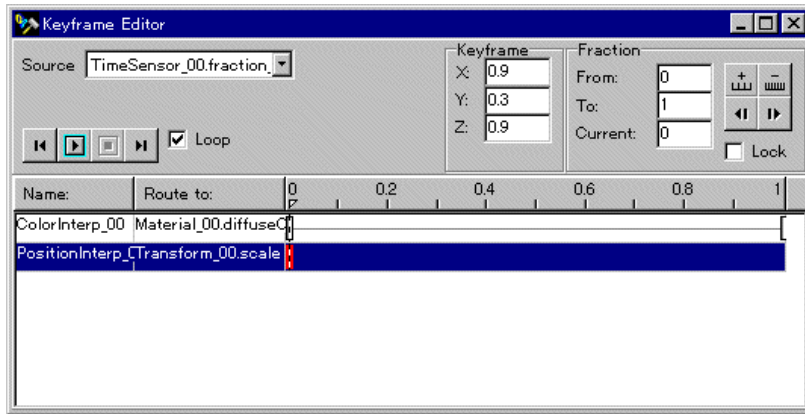


図 63 キーフレーム登録後の Keyframe Editor ウィンドウ

また、Route ウィンドウを見ると、PositionInterpolator ノードや Transform ノードのアイコン、そして Route が生成されています。

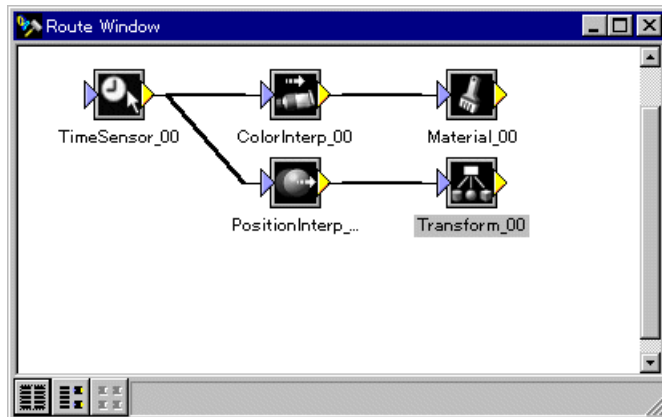


図 64 Route ウィンドウではアイコンや Route が自動的に生成されている

■変形：2番目のキーフレームを作成

続いて2番目のキーフレームを作成します。Keyframe Editor ウィンドウの右側にあるグラフにおいて、目盛りの1の下あたりをクリックして、点線で示されるロケーターがその位置に来るようにします。そして、Attribute ウィンドウ上もしくは Keyframe Editor ウィンドウの中央上部に表示された *scale* フィールドの内容を編集します。今度は Cone を図 65 のように細長くしてみました。

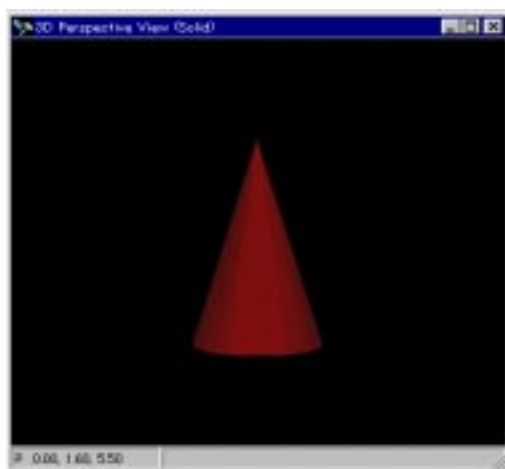


図 65 Cone を細長く変形させて2番目のキーフレームとする

Keyframe Editor ウィンドウは図 66 のようになります。

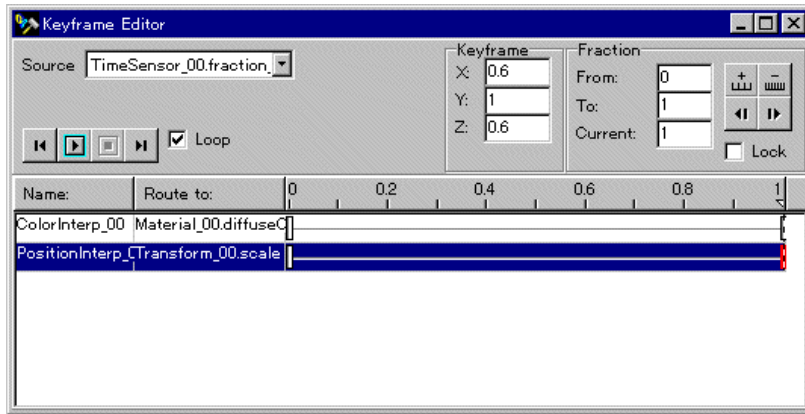


図 66 キーフレーム登録後の Keyframe Editor ウィンドウ

■ 動作確認

それではアニメーションを実行してみましょう。Keyframe Editor ウィンドウ、もしくは Main ウィンドウ上の Play ボタンを押します。すると、Cone が色を変化させつつ、偏平な状態から細長い状態へと伸びていくアニメーションが繰り返されます。

■ TouchSensor (Sensor ノード) の配置

最後に、スイッチに反応してアニメーションが起動するようにしてみましょう。これまで作成したアニメーションは単純な繰り返しでしたが、スイッチと組み合わせることによって様々な応用が広がります。例えば、スイッチを押すと開くドアなどを実現することができそうですね。

今回はConeをクリックしたときにアニメーションが起動するようにします。始めに Keyframe Editor ウィンドウにおいて、Loop チェックボックスを Off にしておいてください。続いて TouchSensor を配置します。図 67 に示すように Scene Graph ウィンドウにて Cone ノードの親ノードになっている Transform ノードを選択しておいて、TouchSensor を追加します。

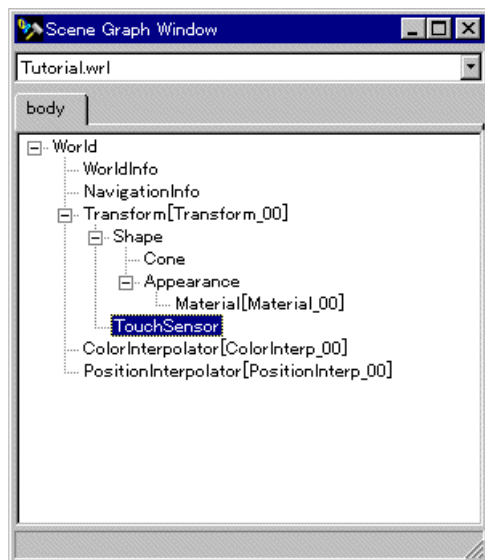


図 67 TouchSensor を Transform ノードの子ノードとして登録する

■ Route の作成

Scene Graph ウィンドウ上の TouchSensor を、Route ウィンドウ上の TimeSensor のアイコンにドラッグ&ドロップします。そして図 68 に示すポップアップメニューから「TouchSensor_00.touchTime TO TimeSensor_00.startTime」を選択します。これは、TouchSensor(TouchSensor_00)を押した時刻(*touchTime*)を TimeSensor(TimeSensor_00)の開始時刻(*startTime*)として設定するという意味です。これで、TouchSensor のクリックに反応して TimeSensor が起動し、アニメーションが始まるようになります。TouchSensor ではない他の Sensor ノードを使用する場合でも、Sensor においてイベントが起きた時刻を TimeSensor に伝達するという方法は同じです。

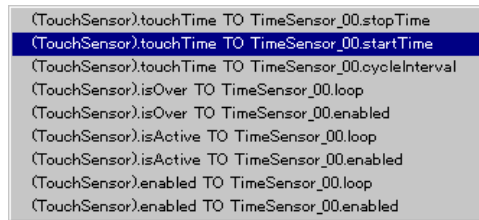


図 68 TouchSensor をドラッグした際に表示されるポップアップメニュー

Route ウィンドウには新たなアイコンや Route が生成され、図 69 のようになります。

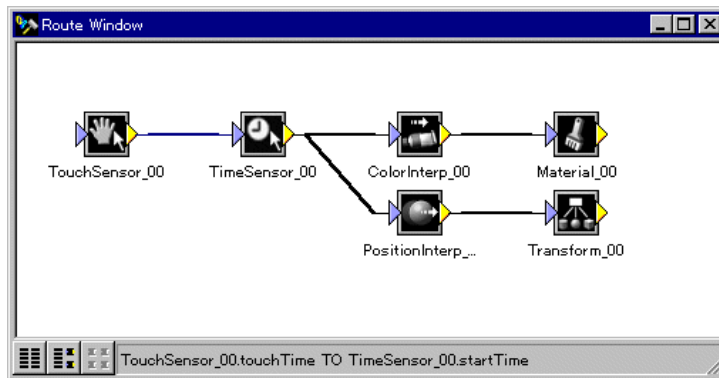


図 69 Route 作成後の Route ウィンドウ

■ 動作確認

それでは実行してみましょう。今回はアニメーションだけではなく TouchSensor も含んでいるので、Keyframe Editor ウィンドウ上の Play ボタンではなく、Keyframe Editor を閉じて Main ウィンドウ上の Play ボタンを押します。3D View ウィンドウにて Cone をクリックするたびに、Cone が色を変化させつつ、偏平な状態から細長い状態へと伸びていくアニメーションが行われます。

参考文献

Rodger Lea, Kouichi Matsuda, Ken Miyashita: "Java for 3D and VRML Worlds",
New Riders.