NAIST-IS-MT9651064

修士論文

汎用3次元ユーザインタフェースツールキットの開発

世利 至彦

1998年2月13日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科において
修士 (工学) 授与の要件として提出された修士論文である。

提出者： 世利 至彦

指導教官： 横矢 直和 教授
千原 國宏 教授
竹村 治雄 助教授

汎用3次元ユーザインタフェースツールキットの開発*

世利 至彦

内容梗概

近年，3次元コンピュータグラフィックスを利用した情報の可視化や，仮想物体のモデリングなど，3次元仮想環境を利用したアプリケーションが多数提案されている．しかし既存のグラフィックスライブラリのみで，これらの仮想環境を提供するアプリケーションを作成するには，多大な時間と労力が必要となる．このような背景から，仮想環境を利用したアプリケーションの作成を容易にするツールキットの開発が求められている．本論文では，仮想環境内に立体的なスイッチなどの3次元的なインタフェース (3DUI) 部品を提供し，かつ様々な入力装置や表示装置を統合的に扱うことができるツールキットの開発について論じる．具体的には，ツールキットが提供すべき機能，ツールキットの設計方針と特徴，実際の実装手法と使用例について述べる．

キーワード

ユーザインタフェース，ツールキット，仮想環境，オブジェクト指向プログラミング

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT9651064, 1998年2月13日.

Development of a 3D User Interface Tool-kit*

Yoshihiko Seri

Abstract

This paper describes a case study of designing and implementing a 3D user interface tool-kit. The tool-kit is designed for prototyping and constructing 3D user interface. The tool-kit provides 3D widgets and methods for direct manipulation of 3D primitives, and supports a variety of input and output devices for virtual reality systems. As a result of adopting such a tool-kit, programmers are able to save considerable labor and time to construct 3D user interface. This paper goes into details of the design policy, function and implementation of the tool-kit as well as some programming examples.

Keywords:

user interface, tool-kit, virtual environment, object oriented programming

*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT9651064, February 13, 1998.

目 次

1. はじめに **1**

2. 3DUI ツールキットの概要 **3**

2.1 3DUI の構成 3

2.2 3DUI ツールキットが満たすべき要件 5

3. 汎用 3DUI ツールキットの設計と試作 **7**

3.1 3DUI ツールキットの設計方針 7

3.2 オブジェクト指向設計の有効性 7

3.3 3DUI ツールキットの構成 10

3.4 インタフェース部 12

3.4.1 インタフェース部品のクラス構成 12

3.4.2 インタフェース部品の例 14

3.4.3 バインディング 18

3.4.4 インタフェース部品の新規作成 18

3.4.5 インタフェース部品の描画 19

3.5 入力装置制御部 21

3.5.1 3 次元位置計測装置 24

3.5.2 入力装置制御部の実装 24

3.6 表示装置制御部 25

3.6.1 表示装置と表示手法 25

3.6.2 表示装置制御部の実装 27

4. プログラム例 **29**

4.1 プログラムの概要 29

4.2 例 1 – プッシュボタンとダイアログボックス – 29

4.3 例 2 – *Small World* の作成 – 32

4.4 例 3 – 3 次元カーソル – 37

5. 考察 40

6. むすび 41

謝辞 43

参考文献 44

付録 I

A. 3次元ユーザインタフェースツールキット仕様書 I

A.1 ツールキットの概要 I
A.2 2D モードと 3D モード II
A.3 Button クラス II
A.4 Camera クラス IV
A.5 Constraint クラス VI
A.6 Entry クラス VII
A.7 Light クラス VIII
A.8 MenuButton クラス X
A.9 Object クラス XI
A.10 Primitive クラス XVII
A.11 PushButton クラス XXII
A.12 RadioButton クラス XXIII
A.13 Scaler クラス XXIV
A.14 Sensor クラス XXV
A.15 Switch クラス XXVIII
A.16 Text クラス XXIX
A.17 ToggleButton クラス XXXII
A.18 Viewer クラス XXXIII
A.19 World クラス XXXVII

B. パラメータファイル XL

C. プログラム例 **XLIII**

図 目 次

1	3DUI と 3DAP の関係	3
2	オブジェクト間で共通の関数インタフェース	9
3	3DUI ツールキットの構成	10
4	本ツールキットのクラス構成	13
5	プッシュボタン	14
6	トグルボタン	14
7	ラジオボタン	15
8	メニューボタン	16
9	ダイアログボックス	16
10	スケーラ	17
11	エントリ	17
12	関数をバインドする疑似コード	18
13	シーングラフの例	20
14	図 13 のシーングラフを構成する疑似コード	20
15	入力装置の構成例	21
16	3 次元位置入力装置	22
17	3 次元位置計測装置	22
18	3 次元位置入力装置と HMD を用いた作業風景	23
19	通常の CRT	26
20	Mediamask(OLYMPUS)	26
21	See-Through Vision(島津製作所)	26
22	HMD を用いたシステム構成例	27
23	プログラム例 1	30
24	プログラム実行例 1	31
25	昼の風景	32
26	水位上昇 (1)	33
27	水位上昇 (2)	33
28	夜の風景	34

29 3 次元カーソルの形状 37
30 プッシュボタンでの関数 Reaction III
31 ラジオボタンを含むシーングラフ XXIII
32 測定点と Sensor クラスでの座標変換の関係 XXVI

1. はじめに

近年，様々な分野で 3 次元コンピュータグラフィックス (3 次元 CG) を利用した 3 次元仮想環境が研究されるようになってきた．3 次元 CG によるアニメーションや入出力装置の開発をはじめ，情報の可視化，多人数参加型仮想環境，マンマシンインタフェースなどの研究や，建築，医学，芸術などの分野で仮想環境を提示するアプリケーション (3DAP) が多数提案されている [1, 2, 3, 4, 5, 6]．特に仮想環境の構築や仮想物体のモデリングに関する研究が盛んに行なわれており，様々なアプリケーションが開発されている [7, 8, 9]．3DAP のユーザは，3 次元 CG で構築された仮想環境の中を自由に移動したり，仮想環境内に存在する仮想オブジェクトとインタラクションを行なうことができる．例えば，仮想的な遊園地を作成して，ジェットコースターや観覧車に乗ったり，仮想的な美術館で絵画を観賞するなど，様々なアプリケーションが考えられる．このような 3DAP を開発するためには，3 次元 CG や入出力装置に関する高度な知識やプログラミング能力，インタフェース設計能力などがプログラマに要求され，多大な時間と労力を必要とする．

3DAP を開発するプログラマに課される，このような負荷を軽減するために，本研究では 3 次元仮想環境を提供するアプリケーションのユーザインタフェースを容易に構築するための 3 次元ユーザインタフェースツールキットを提案する．ここでツールキットとは，種々の関数群からなるソフトウェアライブラリである．3 次元ユーザインタフェースツールキットによって，3DAP のインタフェース設計や入出力装置の制御を支援することで，簡単かつ短時間で 3DAP を作成できるので，プログラマに課される 3DAP 開発時の負荷を軽減することができる．また，3 次元ユーザインタフェースツールキットで提供されるインタフェース部品を組み合わせることで 3DAP のインタフェース部分を容易に作成できることから，3DAP のプロトタイプの作成に有効と考えられる．

2 次元グラフィカル・ユーザ・インタフェース (2 次元 GUI) の構築のためのツールキットとしては，OSF Motif や Open Look，Tk など様々なものが知られている [10, 11]. 3 次元仮想環境を構築するためのツールキットの研究もなされているが [12, 13, 14, 15]，これらは 3DUI の構築に主眼を置いたものではなく，主に仮

想物体の作成に重点を置いたものである．

Brooke は，アプリケーションプログラムはその中枢となる部分の設計とインタフェース部分の設計を完全に切り分けて考えることが望ましいと述べている [16]．3 次元ユーザインタフェースの構築を主目的としたツールキットを用いることで，アプリケーションのインタフェース部分を独立に設計，実装することが可能となる．また，インタフェース部分を切り分けることで，アプリケーションの保守改善が行ないやすくなる．例えば，アプリケーションプログラムが持つ機能を改善する場合，インタフェース部分は変更せずにアプリケーションプログラムの関連する部分のみを検討すればよいし，逆に，アプリケーションの外観を変更する場合，アプリケーションプログラムの内容を変更せずに，インタフェース部分の該当箇所を変更すればよい．

また，Tsao らの提案する CRYSTAL[17] や Card らの提案する 3D/Rooms[18] などのように，仮想環境内で複数のアプリケーションを動作させることが可能なシステムでは，個々のアプリケーションとのインタラクションを 3DUI ツールキットで提供されるインタフェース部品を通じて行なうことで，アプリケーションの種類によらない一貫したインタフェースをユーザに提供することができる．

本論文では，第 2 章で 3 次元ユーザインタフェースツールキットの概要を述べ，3 次元ユーザインタフェースツールキットとしてどのような要件を満たすべきかを考察する．第 3 章では，本研究で提案する 3 次元ユーザインタフェースツールキットの設計方針や特徴を説明し，その実装について述べ，第 4 章で，実装した 3 次元ユーザインタフェースツールキットを用いたプログラム例を紹介する．第 5 章で，本研究で試作した 3 次元ユーザインタフェースツールキットについて考察し，最後に第 6 章において本研究の成果をまとめ，今後の課題について言及する．なお本論文では以降より，3 次元ユーザインタフェースを 3DUI と略記する．

2.　3DUI ツールキットの概要

本章では，3DUI ツールキットの概要について説明する．前述のように 3DUI ツールキットとは，3 次元仮想環境を利用したアプリケーションのユーザインタフェースを容易に構築するためのツールキットである．以下では，3DUI の構成要素について述べ，3DUI ツールキットが満たすべき要件について考察する．

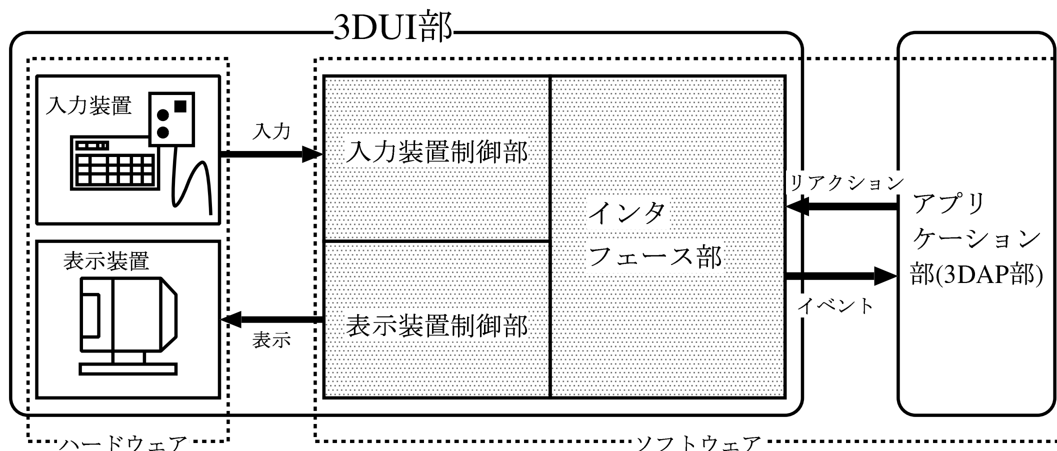


図 1　3DUI と 3DAP の関係

2.1　3DUI の構成

一般に 3DUI を構成する要素には，通常の計算機のインタフェースを構成する要素と同様に，ハードウェア的な要素として入力装置および表示装置が挙げられ，またソフトウェア的な要素としては，入力装置制御部，表示装置制御部およびインタフェース部が挙げられる．各々について以下で簡単に説明する．また，3DAP との関係を図 1に示す.

入力装置 仮想環境とユーザとの間のハードウェア的なインタフェースである．キーボードやマウス，3 次元位置入力装置などがある．3 次元位置入力装置については 3.5節で述べる．

表示装置 仮想環境をユーザに提示するための装置である．CRT，ヘッド・マウント・ディスプレイ (HMD)，プロジェクタなどがある．

入力装置制御部 入力装置をソフトウェア的に制御する．

表示装置制御部 表示装置をソフトウェア的に制御する．

インタフェース部 仮想環境とユーザとの間のソフトウェア的なインタフェースである．3 次元的な形状を持つスイッチやメニューなどのインタフェース部品を提供し，また仮想環境全体の管理を行なう．

図 1の各要素間の関係を理解するために，例として，ユーザがマウスでアプリケーションの終了ボタンを押す場合について考えてみよう．まず，ユーザはマウス (入力装置) のカーソルを終了ボタン (インタフェース部品) の位置に移動させ，マウスボタンをクリックする．このとき，マウスカーソルが終了ボタン上にあれば，入力装置制御部は，マウスボタンが押されたことを終了ボタンに通知する．終了ボタンは，自分自身が押されたことを表すフラグを設定する．このフラグの値から，アプリケーション部は終了ボタンが押されていることが分かるので，終了のための処理を行ない，インタフェース部に対し，このアプリケーションに関係する部位の消去を指示する．インタフェース部はアプリケーション部からの指示に従い表示装置制御部へ描画命令を出し，表示装置制御部が実際に表示装置への描画を行なう．

3DUI ツールキットでは，上述した要素のうち，インタフェース部，入力装置制御部，表示装置制御部の設計，実装を支援する．入力装置制御部，表示装置制御部は，プログラマがシステムのハードウェア構成を意識することなくインタフェース部分を作成できるようにするためのプログラムである．プログラマは，インタ

フェース部で提供される部品に様々な処理を行なう関数を対応づけ，それらの部品を適切に組み合わせることで，アプリケーションのインタフェース部分を作成することができる．

2.2　3DUI ツールキットが満たすべき要件

3 次元仮想環境は，2 次元 GUI が提供する平面的な世界に「奥行き」を付加した世界である．2 次元 GUI の平面的なインタフェース部品をそのまま仮想環境内に配置すると，ユーザが仮想環境内を移動し，斜めや真横からこのインタフェース部品を見た場合に分かり難い．ユーザの移動に合わせて全てのインタフェース部品の向いている方向を更新することも考えられるが，このような処理は計算コストが高くなる上，多人数で 1 つの仮想環境を共有する場合などの実装が困難である．もちろん，文字情報の提示など平面的なインタフェース部品も必要であるが，特に平面的な形状を必要としないインタフェース部品には立体的な形状を持たせることで，ユーザは任意の角度からインタフェース部品を確認，操作することが可能である．また立体的な形状を持つインタフェース部品ならば，複数のユーザが仮想空間を共有する場合でも，どのユーザもインタフェース部品を確認，操作することが可能になる．したがってインタフェース部品に立体的な形状を持たせることは有効であると考えられる．

またインタフェース部品は，アプリケーション部が提供する諸機能と密接に関係する．例えば，ドアに付けられたスイッチボタンが押されるとチャイムの音が出る，あるいは空間中に浮かぶスケーラで背景色の濃さや明るさを変更できるなど，アプリケーション毎にインタフェース部品への様々な対応づけが考えられる．よって，インタフェース部品に様々な処理を対応づけることができる機能が必要となる．

さらに一般的な 2 次元 GUI 環境では，入力装置としてキーボードやマウス，表示装置としてディスプレイを用いているが，3 次元仮想環境を提供する 3DAP では，入力装置や表示装置に様々な機器を使用できる．例えば，2 次元 GUI 環境でのマウスに相当する 3 次元マウスやユーザの頭に装着して使用する HMD などの機器が挙げられる．したがって 3DUI ツールキットには，インタフェース部品だ

けでなく，様々な入力装置と表示装置の利用を支援する機能が必要である．

まとめると，3DUI ツールキットは次の要件を満たす必要がある．

1. 3 次元的な形状を持ち，様々な処理の対応づけが可能なインタフェース部品を提供すること．

2. 入力装置と表示装置の利用を支援する機能を提供すること．

次章以降では，本研究で行なった 3DUI ツールキットの設計および試作，また 3DUI ツールキットを用いた場合のプログラム例について述べる．

3. 汎用 3DUI ツールキットの設計と試作

本章では，前章で述べた要件を満たす 3DUI ツールキットの設計方針と構成および，その実装について説明する．また，本研究では 3DUI ツールキットの設計にオブジェクト指向設計を採用したので，その有効性についても述べる．

3.1 3DUI ツールキットの設計方針

3DUI ツールキットを利用するユーザは，3DAP を作成するプログラマであると考えられる．現在のプログラムは，要求される機能の高度化と多様化に伴ってプログラムサイズも大きくなり，開発期間も限られているので，数人からものによっては数百人の開発者によって協同開発されている．また商品として世に送り出す場合，保守が必要となり，次の商品を開発する際には既存のプログラムを利用した開発が行なわれる．したがって，保守や再利用が行ないやすいプログラムの作成が望まれる．また，様々な 3DAP のインタフェース部分を 3DUI ツールキットを用いて作成するためには，3DUI ツールキットで提供されるインタフェース部品の組み合わせ方や個々のインタフェース部品の機能をプログラマが自由に決定できる必要がある．そこで，3DUI ツールキットの設計方針として，次の 2 点に注目した [19].

1. 可読性の高いプログラムが容易に作成できること．

2. インタフェース部品の機能が拡張できること．

本研究では，この設計方針に基づき，3DUI ツールキットを設計，試作した．以下では，3DUI ツールキットの設計および試作について述べる．

3.2 オブジェクト指向設計の有効性

本研究で提案する 3DUI ツールキットは，インタフェース部，入力装置制御部，表示装置制御部の 3 つの部分から構成される．本研究では，各部の設計に際し，オブジェクト指向設計を採用した．

オブジェクト指向の考え方においては，一般に，オブジェクトは概念や抽象あるいは対象となる問題に対して明確な境界と意味を持つものとして定義される [20]．オブジェクトには，主に 2 つの役割がある．それは，実世界を理解しやすくすることと，コンピュータ上への実装に関して実質的な基盤を与えることである．

オブジェクトという単語は，しばしば文献中で漠然と使用される．あるときはオブジェクトは単一のものを指し，またあるときには類似したもののグループを指している．通常は，厳密に 1 個のものを参照したいとき，インスタンスという用語を使用する．そして，類似したもののグループを参照するためにクラスという用語を使用する．同じクラスに属するオブジェクトは，属性と振舞いに関して同一パターンを持ち，その属性値や他のオブジェクトとの関係の違いによって個別性を出す．また，あるクラスに属するオブジェクトは，共通の属性や振舞いに加えて，共通の意味を持つ．例えば，馬小屋と馬とは，どちらも値段と年齢を持っているにも関わらず異なるクラスに属すると考えられる．しかしもし馬小屋と馬が純粋に金融財産として見なされるならば，それらを同じクラスとして考えることができる．意味の解釈は各アプリケーションの目的に依存していると言える．

また，オブジェクト指向設計には，汎化と継承という概念がある．汎化と継承は，クラス間の違いを維持しながら，一方で類似点を共有するための強力な抽象化手法である．例えば，次のような状況をモデル化したいとする．全ての装置は，属性値として，製造者，重量，価格を持つ．またポンプは吸引力と流量を，タンクは容量と圧力を属性値として持つとする．汎化とはクラスとそれを 1 回以上特殊化したものとの関係である．特殊化のもとになるクラスを親クラス (スーパークラス) と呼び，特殊化された結果の各クラスを子クラス (サブクラス) と呼ぶ．例えば，装置はポンプとタンクの親クラスである．子クラスの集合に共通の属性と操作は，親クラスに付加され，全ての子クラスで共有される．各子クラスは親クラスの特性を継承するという．例えば，ポンプは属性として製造者，重量，価格を装置から継承する．

オブジェクト指向設計を用いた 3DUI ツールキットの設計について以下に述べる．3DUI ツールキットは，インタフェース部，入力装置制御部，表示装置制御部の 3 つの部位からなる．オブジェクト指向の考え方を用いれば，3DUI ツール

キットというクラスにインタフェース部オブジェクト，入力装置制御部オブジェクト，表示装置制御部オブジェクトという 3 つのオブジェクトが属していると考えられる．そして，各オブジェクトに適用できる関数名やその関数の引数 (関数インタフェース) を各オブジェクト間で共通にすることで，アプリケーションプログラムをインタフェース部品や表示装置などに特有の関数の並びとしてではなく，オブジェクトが持つ関数の並びとして捉えることができる．すなわち，図 2 で示す疑似コードのように，プログラム上，インタフェース部品や表示装置などの区別なしに共通の関数インタフェースを用いたプログラミングが可能である．

```
Interface_Parts  P;  // インタフェース部品
Display_Device   D;  // 表示装置

P.Init();            // インタフェース部品の初期化
D.Init();            // 表示装置の初期化
```

図 2 **オブジェクト間で共通の関数インタフェース**

さらに，インタフェース部が提供するインタフェース部品のオブジェクト指向的な設計として，次のような設計が考えられる．まず，全てのインタフェース部品に共通に使用される関数や変数データを定義する親クラスを作成し，各インタフェース部品のクラスをこの親クラスからの継承によって作成する．

各インタフェース部品のクラスは，その種類毎に固有に持つ変数や配列などのデータとそのインタフェース部品に適用できる関数を定義している．各インタフェース部品のクラスで定義される関数は，そのインタフェース部品に専用の関数であると考えることができる．例えば，プッシュボタンには，プッシュボタン専用の初期化関数や描画関数が定義でき，ダイアログボックスには，ダイアログボックス専用の初期化関数や描画関数が定義できる．それぞれの関数の内部では，各インタフェース部品に固有な変数などのデータを扱っている．また，各インタフェース部品のクラスを設計する上で，これらの初期化関数や描画関数に同じ関

数名を持たせることも可能である．このように設計することで，種類の異なるインタフェース部品間で，同種の操作を行なう関数に同じ関数名を適用できるので，関数名が覚えやすくプログラムの可読性も向上する．

以上の理由から，3DUI ツールキットの実装にはオブジェクト指向的なプログラミングが可能な言語が有効である．C 言語は現在最も普及している言語であるが，オブジェクト指向プログラミングを行なうのは困難である．C++は C 言語をオブジェクト指向プログラミングが可能なように拡張したもので，多数ある既存の C 言語のライブラリも利用できる [21, 22]．そこで 3DUI ツールキットの実装には C++を用いることにした．また仮想物体の描画には，現在，最も普及している 3 次元グラフィックスライブラリである OpenGL[23] を用いた．

3.3　3DUI ツールキットの構成

本ツールキットの構成を図 3に示すようにインタフェース部，入力装置制御部，出力装置制御部からなる 3 部構成とした．3DUI ツールキットは OpenGL をはじめ，C や C++で利用可能なライブラリを用いて実装した．

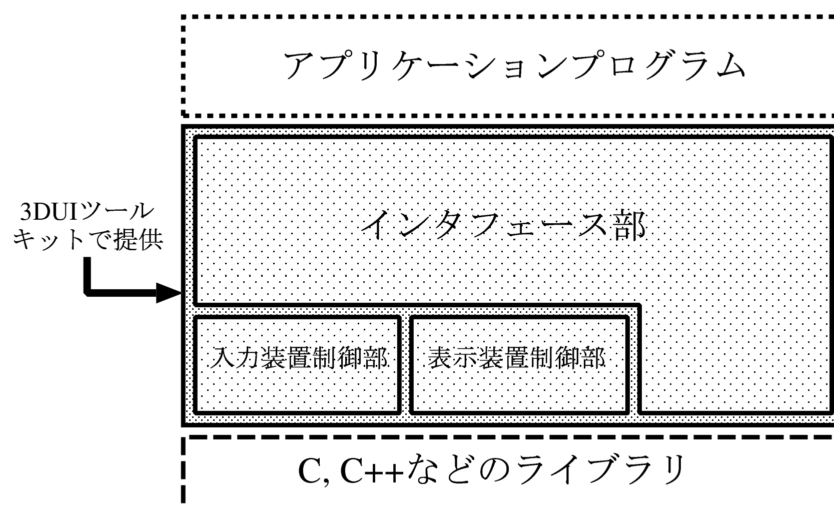


図 3　3DUI ツールキットの構成

入力装置制御部は，入力装置から得られる入力信号をアプリケーションプログラムに受け渡す機能を提供し，表示装置制御部では，表示装置の初期化，画面表示などの機能を提供する．入力装置制御部，表示装置制御部の実装については，3.5.2項，3.6.2項で詳しく説明する．

3DAP では入力装置，表示装置に様々な機器構成が考えられるが，使用する機器毎にプログラムを変更することは非効率的である．そこでパラメータファイルを用意し，入力装置，表示装置などの機器に関係する部分をプログラムから切り分けた．つまり構成機器を変更する場合，プログラマはプログラムを変更せずに，パラメータファイルで該当する箇所のみを変更すればよく，機器構成を意識せずにプログラムの設計，実装を行なうことができる．またパラメータファイルを用いることで，プログラマだけでなくアプリケーションユーザも入力装置や表示装置などの使用機器を容易に変更できる．パラメータファイルでは，種々の入力装置，表示装置の設定を行なう．パラメータファイルの詳細は付録 Bで述べる．

入力装置，表示装置の各制御部は，プログラマが入力装置や表示装置の機器構成を意識することなくアプリケーションプログラムを作成できるようにするためのものであり，特にアプリケーションプログラム内で使用する部位ではない．アプリケーションプログラムは，プログラマがインタフェース部品の振舞いを定義し，それらのインタフェース部品を適切に組み合わせることで作成できる．2 次元 GUI で提供されるスイッチやメニューなどと同等のインタフェース部品には 3 次元的な形状を持たせた．例えばプッシュボタンはワイヤフレームの立方体に球を埋め込んだ形状を持つ．インタフェース部品が 3 次元的な形状を提示することにより，ユーザは任意の方向からインタフェース部品を認識することができる．本研究で実装したインタフェース部品については，3.4節で詳しく説明する．

3.4　インタフェース部

インタフェース部では，2次元GUIで提供されるスイッチやメニューに3次元的な形状を持たせたインタフェース部品を提供する．また仮想環境全体の描画に関する管理も行なう．本節ではインタフェース部品のクラス構成を述べた後，本研究で実装した種々のインタフェース部品について説明する．また，これらのインタフェース部品を仮想環境内に描画する仕組みについても説明する．

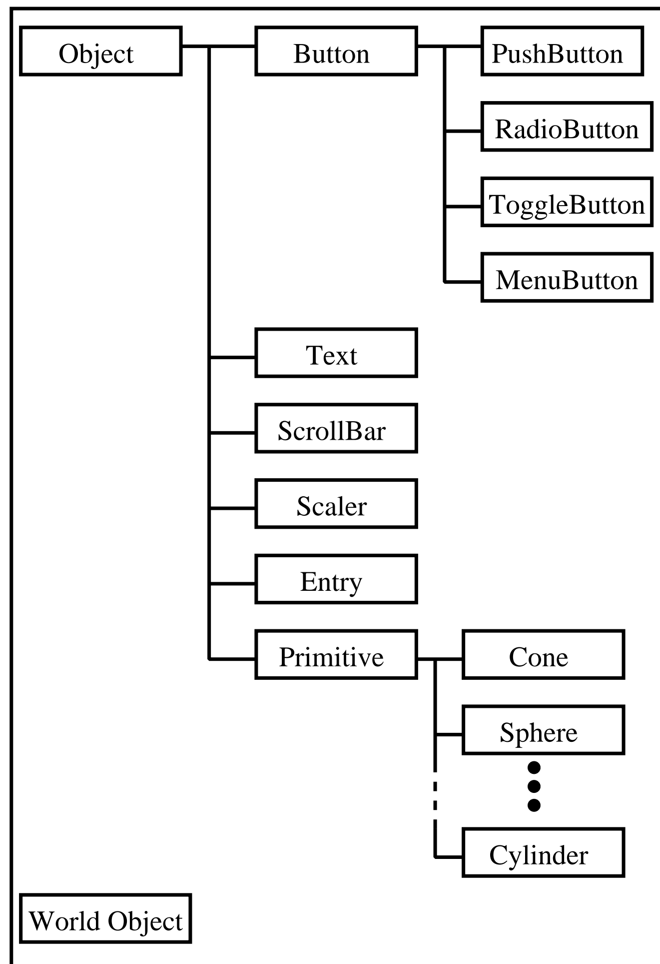
3.4.1　インタフェース部品のクラス構成

オブジェクト指向的に設計した各インタフェース部品のクラスでは，その種類毎に固有のデータや関数群を有する．C++では，クラスや継承などのオブジェクト指向的な概念を用いたプログラムを作成することが可能である．C++が提供するクラスは，構文的にはC言語の構造体に似ているが，そのクラスに属する全てのオブジェクトに適用できる関数の定義や属性データなどを含む．またC++の継承を用いて親クラスから子クラスを派生させると，子クラスでも親クラスのデータや関数が使用できる．さらに子クラスに固有のデータや関数を追加して定義できる．継承を利用することでインタフェース部品の持つ性質を階層的に定義できる．

本研究で提案する3DUIツールキットでは，インタフェース部品の設計に関して3.2節後半で述べた設計手法を採用した．すなわち，全てのインタフェース部品が共通に持つデータや関数を最上位の親クラスに持たせ，インタフェース部品毎に固有のデータや関数は子クラスに持たせた．

以上のように設計することで，各部品に共通して適用可能な関数のインタフェースが統一され，可読性の高いプログラムが作成できる．本ツールキットのクラス構成(クラス木)を図4に示す．図4において，最左のObjectクラスが最上位クラスであり，全てのインタフェース部品に共通のデータや関数を持つ．またクラス木の根と中間ノード(Objectクラス，Buttonクラス，Primitiveクラス)は具体的な形状を持たず，自分以下のクラスで共通に利用されるデータや関数を定義している．各クラスの詳細説明を付録Aに示す．

インタフェース部品



入力・表示装置制御部

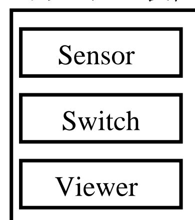


図 4 **本ツールキットのクラス構成**

3.4.2　インタフェース部品の例

実装したインタフェース部品を以下に紹介する．また，その形状を図 5から図11に示す．

プッシュボタン： カーソルで選択され，押下などのアクションを受けると対応する関数を呼び出す．

図 5 **プッシュボタン**

トグルボタン： カーソルで選択され，押下などのアクションを受ける毎にオン，オフが切り替わる．例えば文字列に下線を引くかどうか，計算結果の表示を行なうかどうかなどの二者択一を行なう際に用いられる．

図 6 **トグルボタン**

ラジオボタン： カーソルで選択され，押下などのアクションを受けると複数の選択肢の中から1つだけが排他的に選択される．すなわち，複数のラジオボタンを並べ，ユーザはその中から1つだけを選択することができる．

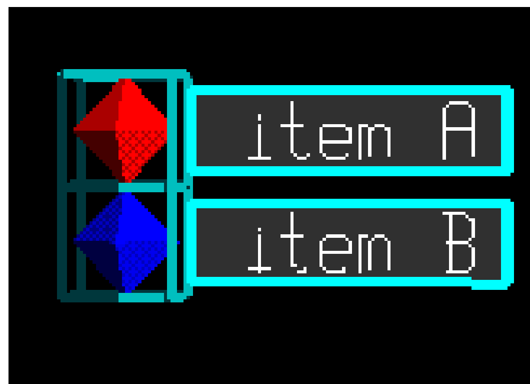


図7 **ラジオボタン**

メニューボタン： カーソルで選択され，押下などのアクションを受けることで，そのメニューボタンの下に関連する項目が掲示される．いわゆるプルダウンメニューである．アクションを与える毎にメニューの掲示，非掲示が切り替わるので，特殊なトグルボタンと考えることもできる．これまでに述べた3種類のボタンとメニューボタンが，項目として利用可能である．項目にメニューボタンを入れることで，階層的なプルダウンメニューを作成できる．図8は，3つのプッシュボタンをメニュー項目に持つ例である．

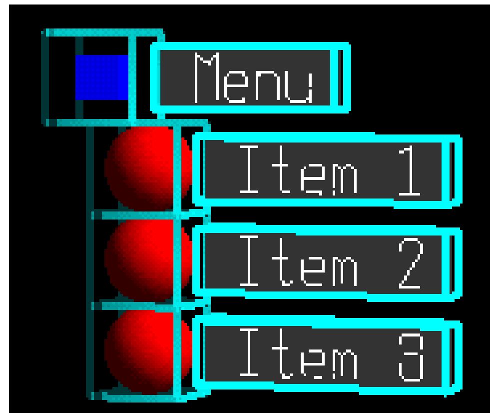


図 8 **メニューボタン**

テキスト： 文字列が表示できる．一枚の板 (背板) に文字列が書かれ，背板の周囲をワイヤフレームで表現された直方体で囲んだ形状である．用途として，ダイアログボックスやボタンなどのインタフェース部品に文字情報を付加するラベルが考えられる．必要であればスクロールバーを持つ．

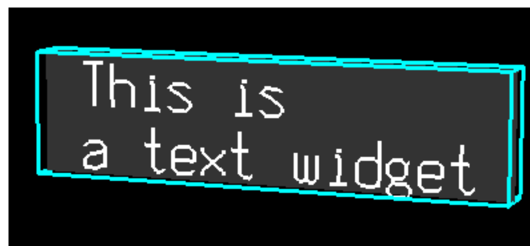


図 9 **ダイアログボックス**

スケーラ： 中央のつまみ (図 10では真中の球) をカーソルで移動させることで，スケーラが持つ内部変数の値を連続的に変更できる．またスケーラの両端のボタンを押すことで，内部変数の値を離散的に変更できる． スケーラはスライダとも呼ばれる．

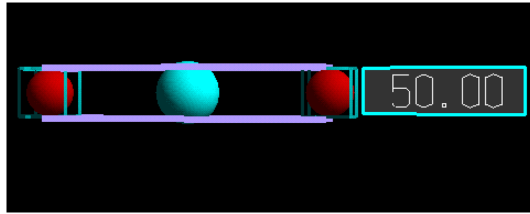


図 10 **スケーラ**

エントリ：1行の文字列を入力または編集することができる．カーソルによって選択することで入力可能状態になり，改行で入力終了となる．

図 11 **エントリ**

各インタフェース部品のクラスでは，大きさや配色，フレームやラベルの着脱など簡単なカスタマイズを行なう関数を定義している．各インタフェース部品のクラスおよびそのクラスで定義されている関数などについては付録Aで述べる．

プッシュボタンなどのインタフェース部品には，プログラマが定義した関数を対応づけて，ボタンの押下などのアクションを受けた際に対応づけられた関数を起動させる必要がある．次項では，3DUI ツールキットが提供する関数を対応づける機構について説明する．

3.4.3　バインディング

プッシュボタンなどのオブジェクトにプログラマが定義した関数を対応づけ，このオブジェクトに対して，押下などの特定のイベントが発生したときに，その対応づけられた関数を実行するという機構を Object クラスは提供している．この機構を**バインディング**と呼ぶ．バインディングを行なう関数 BindFunction は，引数に，関連づける関数へのポインタをとる．図 12は，プッシュボタンに関数をバインドする疑似コードである．

```
void  A();              // 関数 A()


PushButton *pb;         // プッシュボタンを宣言
      …
pb->BindFunction(A);  // プッシュボタンに関数 A() を
                        // バインド
```

図 12　**関数をバインドする疑似コード**

3.4.4　インタフェース部品の新規作成

3.4.2項で述べたインタフェース部品のほか，直方体や球などの基本的な幾何形状を提供する Primitive クラスがある．Primitive クラスで提供する幾何形状を組み合わせることで，新たなインタフェース部品を構築することも可能である．実際，本ツールキットで提供しているインタフェース部品の形状も，Primitive クラスを用いて作成している．Primitive クラスで定義される関数などの詳細は付録**??**で述べる．

幾何形状を適切に組み合わせ，様々な機能を付加することで，新たなインタフェース部品を構築することができる．新たなインタフェース部品を作成する場合，または既存のインタフェース部品に様々な機能を付加する場合に，前述したバインディングや以下で述べる制約という機構を利用することができる．

Primitive クラスで提供される幾何形状に，ユーザが定義した関数をバインドすることで，新たなインタフェース部品が構築できる．その際，ユーザからのアクションに対して，幾何形状に何らかの制約を付加できる必要がある．例えば，スケーラのつまみはドラッグされてもある直線上でのみ移動できるようにしなければならない．また 3DAP によっては，ドラッグできないオブジェクトを作成したい場合も考えられる．仮想オブジェクトに制約を付加し，その挙動を制御する研究も行なわれているが [24, 25]，本研究では，処理を単純化するために，オブジェクトに制約の有無を表すビット列を持たせた．制約 1 つにつき 1 ビットを対応させ，0 か 1 かでそのビットに対応する制約が付加されているかど うかを判別する．付加できる制約の種類として，オブジェクトが直線上でしか移動しない直線制約や，どのようなアクションに対しても移動することのない静止制約などを提供している．これらの制約のオブジェクトへの付加は，Object クラスで定義される関数 AddCostraint によって行なわれる．制約に関する詳細は付録 A で述べる．

3.4.5 インタフェース部品の描画

仮想空間の構築には，シーングラフ [13] の概念を採用した．シーングラフは仮想空間の構成要素を木で表したものである．文献 [13] では照明や視点，オブジェクトの特性などを表すノードが存在するが，本ツールキットでは，オブジェクト指向的に設計する場合，物体の特性などは物体自身に持たせるべきだと考え，オブジェクトの特性を別ノードにはせずオブジェクト自身に持たせた．このように設計することで，描画時のシーングラフの走査順序を考慮する必要なく，プログラムを作成できる．図 13は仮想空間内にプッシュボタン 2 つからなるメニューボタンと球を置いた場合のシーングラフの例である．またこのシーングラフを構成するための疑似コードを図 14に示す．

シーングラフの根には，World オブジェクトが置かれる．World オブジェクトは仮想空間中の光源の設定や視点の初期設定，シーングラフに対する描画命令の発行など，仮想空間の描画に関連する種々の処理を行なう．描画命令は，シーングラフの根である World オブジェクトからシーングラフ上の各葉に向かって伝搬する．例えば図 13の場合，World オブジェクトは “メニューボタン” と “球” の 2

つの子を持つので，これらの2つのオブジェクトに対し描画命令を出す．メニューボタンは2つのプッシュボタンを子に持つので，各々のプッシュボタンに対し描画命令を出し，自分自身であるメニューボタンを描画する．メニューボタンの子である2つのプッシュボタンおよびWorldオブジェクトの子である球は，子を持たないので自分自身の描画のみを行なう．このように仮想空間内にオブジェクトを描画するためには，そのオブジェクトをシーングラフのノードにする必要がある．シーングラフの構築は，図14に示すように，関数AddChildはWorldクラスおよびObjectクラスで定義されている．

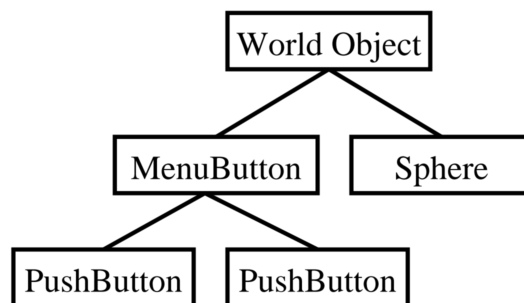


図13 **シーングラフの例**

```
World  *world;
PushButton  *pa, *pb;
MenuButton  *m;
Sphere  *s;

m->AddChild(pa);
m->AddChild(pb);
world->AddChild(m);
world->AddChild(s);
```

図14 **図13のシーングラフを構成する疑似コード**

3.5　入力装置制御部

入力装置制御部では，入力装置の初期化，センサの位置・姿勢に関するデータの抽出，キャリブレーションなどの機能を提供する．3 次元仮想環境とユーザとのインタフェースのための入力装置として様々なものが考えられる．例えば，2 次元 GUI 環境と同様にキーボードやマウスを使用したり，2 次元 GUI におけるマウスに相当する入力装置として，3 次元位置入力装置を利用することができる．

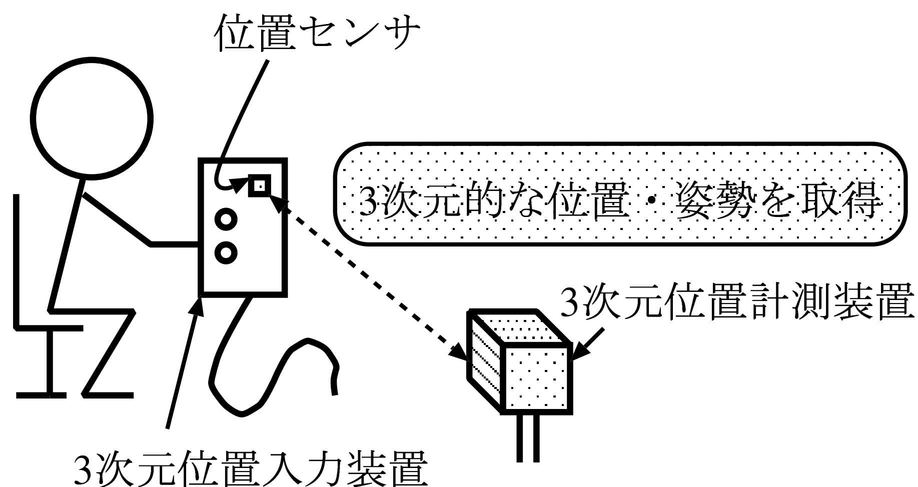


図 15　**入力装置の構成例**

3 次元位置入力装置は，3 次元的なカーソルを実現するためのもので，センサと 1 つ以上のスイッチを持つ．3 次元位置入力装置を用いる場合，ユーザの近傍に 3 次元位置計測装置 (を配置し，このセンサと 3 次元位置入力装置の持つセンサによって，3 次元位置入力装置の 3 次元的な位置と姿勢を得ることができる (図 15)．図 16は 3 次元位置入力装置，図 17は磁気式の 3 次元位置計測装置である．また，3 次元位置入力装置と HMD を用いた実際の作業風景を図 18に示す．

以下では，本ツールキットで支援する 3 次元位置計測装置の種類とその特徴について述べた後，本ツールキットの入力装置制御部について述べる．

図 16　3 次元位置入力装置

図 17　3 次元位置計測装置

図 18　3 次元位置入力装置と HMD を用いた作業風景

3.5.1　3 次元位置計測装置

3 次元位置入力装置は，それ自体が持つセンサと 3 次元位置計測装置から構成される．本ツールキットでは，3 次元位置計測装置として次の 3 種類の使用を支援する．

- Polhemus 社 3SPACE Fastrak
- Polhemus 社 3SPACE IsotrakII
- Shooting Star Technology 社 ADL-1

Fastrak, IsotrakII は，交流磁界を用いた 3 次元の位置および姿勢の計測装置である．また ADL-1 は，可変抵抗を用いた 6 関節機械式計測装置である．

3.5.2　入力装置制御部の実装

前述したように 3DAP の機器構成には，様々な入力装置の利用が考えられる．本ツールキットでは，プログラマが種々の 3 次元位置入力装置を抽象化して扱える Sensor クラスおよび Switch クラスを定義した (図 4参照)．Sensor クラスは，3 次元位置入力装置の位置と姿勢を得るためのクラスであり，Switch クラスは 3 次元位置入力装置に付随しているスイッチの状態 (押されているかどうか) を調べるためのクラスである．

また，利用する入力装置に応じてプログラムを変更するのは，プログラムの汎用性に欠ける．そこで本ツールキットでは，入力装置などのシステム構成機器に依存する部分はパラメータファイルで設定し，ソースコードを変更せず機器の切替が行なえるようにした．

パラメータファイルでは，使用する 3 次元入力装置の指定やその入力装置に必要なパラメータの設定，入力装置からの信号を得るシリアルポートの指定などを行なう．使用する入力装置を切替えるためには，パラメータファイルを装置に応じて変更すればよい．プログラム実行時には，このパラメータファイルを引数として与えて実行する．パラメータファイルの詳細は付録 Bで説明する．

3.6　表示装置制御部

表示装置制御部では，表示装置の初期化，画面表示などの機能を提供する．表示装置も入力装置と同様に様々なものが考えられる．また表示手法にも幾つかの方法が存在する．本節では，種々の表示装置と表示手法について説明した後，本ツールキットの表示装置制御部について述べる．

3.6.1　表示装置と表示手法

本ツールキットが想定する表示装置は，通常の CRT の他，HMD およびプロジェクタであり，各機器で単眼視と両眼立体視をサポートする．表示装置の例を図 19から図 21に示す．また，両眼立体視に必要とされるステレオ画像の表示手法として，以下の表示手法をサポートする．

フレームシーケンシャル方式 右目画像と左目画像を 120Hz で交互に画面全体に描画する時分割方式の 1 手法である．主に液晶シャッター眼鏡を使用する Fish Tank 型のアプリケーションで用いる．Fish Tank 型とは，ディスプレイ画面を窓に見立て，この窓から仮想空間内を観察するものである．つまりユーザが頭の位置を移動させると，それに応じた視野の画面が描画される [26].

フィールドシーケンシャル方式 画面の偶数行に左目画像，奇数行に右目画像を 60Hz で交互に描画する．フレームシーケンシャル方式と同様に時分割方式の 1 手法である．主に HMD を用いたアプリケーションで利用される．

画面分割方式 画面全体を田型に分割し，左上部に左目画像，右上部に右目画像を表示する．画面の下部は使用しないため表示される画像の解像度は低くなる．主に HMD を用いたアプリケーションで利用される．

これらの表示手法をサポートすることで，両眼立体視が可能な機器のほとんどをサポートできると考えられる．

図 19 通常の CRT

図 20 Mediamask(OLYMPUS)

図 21 See-Through Vision(島津製作所)

3.6.2　表示装置制御部の実装

本ツールキットでは，プログラマが種々の表示装置を抽象化して扱える Viewer クラスを定義した (図 4参照)．Viewer クラスは，仮想空間内でのユーザの「目」に相当するもので，前述した表示手法を用いて仮想世界を表示する．

3 次元位置入力装置と同様に，HMD などの表示装置にも位置センサを取り付け，利用者の近傍に設置した 3 次元位置計測装置によって，HMD などの位置，姿勢，すなわちユーザの頭部の位置，姿勢を計測する (図 22)．この計測値から適切な画像を作成し表示装置に描画する．このような構成は，視点追従型のアプリケーションを作成する場合に必要となる．視点追従型のアプリケーションとは，ユーザが観察している場所や方向，すなわちユーザの頭部の位置･姿勢に合わせて表示する画像を変更するアプリケーションである．視点追従型のアプリケーションでは，3.5.2項で述べた Sensor クラスでユーザの頭部の位置・姿勢を計測し，その情報を基に Viewer クラスで表示装置への描画を行なうといった連係が必要となる．

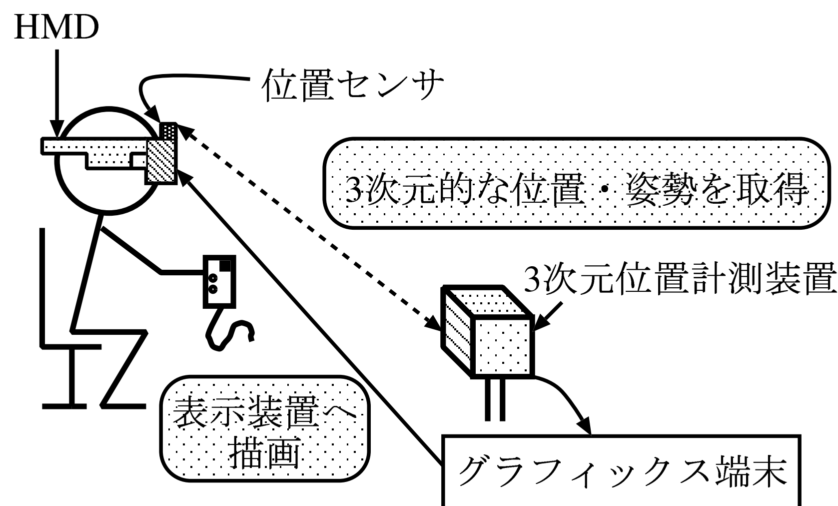


図 22　HMD を用いたシステム構成例

入力装置制御部と同様に表示装置制御部でも，システムの機器構成に依存する部分は，パラメータファイル (付録 B参照) での設定が可能であるため，プログラムを変更せずに機器の切替が行なえる．パラメータファイルでは，表示手法の選択，視体積の設定，ビューポートの位置や大きさの設定などが可能であり，多様な表示装置に対応する．

4.　プログラム例

ここでは簡単なプログラムを挙げ，本ツールキットを用いたプログラミング例について説明する．

4.1　プログラムの概要

作成するアプリケーションプログラムでは，必要となるインタフェース部品の宣言，設定を行ない，シーングラフに登録する．インタフェース部品の設定は，部品の種類毎に異なるが，通常，インタフェース部品を仮想空間内の適切な位置に配置し，そのインタフェース部品の動作を定義した関数をバインドする必要がある．これらの設定およびシーングラフへの登録は，宣言を行なった後ならばいつ行なってもよい．

以下では，実際にプログラム例を挙げる．4.2節では，簡単なアプリケーション例を詳細に渡り説明し，4.3節で，やや複雑なアプリケーション例を概観する．さらに 4.4節では，Sensor クラスや Viewer クラスを用いて，3 次元ポインタを作成するプログラムを説明する．

4.2　例 1 – プッシュボタンとダイアログボックス –

図 23はプッシュボタン 1 つを仮想空間に置き，プッシュボタンを押すとダイアログボックスが表示されるプログラムである．基本的にはインタフェース部品の宣言，設定，シーングラフへの登録を繰り返している．このプログラムの実行結果を図 24に示す．

まずプログラムのメイン関数では，仮想環境の初期化 (4 行目) を行なう．関数 InitFunc は仮想環境全体の管理を行なう World オブジェクト (変数名は world) の作成や描画環境の設定などを行なう．

7 行目から 10 行目でプッシュボタンを生成し種々の設定を行なっている．7 行目で，プッシュボタンをグローバル座標系の適切な位置に配置している．特に指定しない場合は，グローバル座標系 (ワールド座標系) の原点に描画される．

```
1: World  *world;
2: void main(int argc, char **argv){
3:    // 仮想環境の初期化.
4:    InitFunc(argc, argv);
5:
6:    // ボタンの作成.
7:    PushButton *b = new PushButton;
8:    b->Move(-2, 1, 4);
9:    b->String("PushButton");
10:   b->BindFunction(CreateDialog, F_END);
11:   world->AddChild(b);
12:
13:   MainLoop();
14: }
15:
16: // ダイアログボックスを生成する関数
17: void CreateDialog(){
18:
19:   // ダイアログボックスの生成.
20:   Text *t = new Text;
21:   t->String("Dialog Box");
22:   world->AddChild(t);
23: }
```

図 23 **プログラム例 1**

9 行目で，ボタンに付随するラベルに書く文字列を指定している．文字列を指定しなかった場合，ラベルは生成されない．

10 行目では，ユーザの定義した関数 CreateDialog をプッシュボタンにバインドしている．CreateDialog は，ダイアログボックスを生成する関数である．3.4.3 項で述べたように，Object クラスで定義されている関数 BindFunction を用いて，ボタンが押された時に起こるアクションを定義できる．

11 行目で，プッシュボタンを World オブジェクトの子供として指定すること

で，World オブジェクトが管理するシーングラフにプッシュボタンを登録することができる．このようにシーングラフのノードにすることで，仮想環境への描画対象となる．つまり宣言や設定を行なってもシーングラフに登録していなければ描画されない．

20 行目からダイアログボックスの生成，設定を行なっている．22 行目でもプッシュボタンの場合と同様に，シーングラフへの登録を行ない，ダイアログボックスを描画対象にしている．

図 24 **プログラム実行例 1**

4.3　例 2 – *Small World* の作成 –

本節では，前節よりもやや複雑なプログラム例を紹介する．このプログラムは，箱庭的な仮想世界 (Small World) を提供するものである．Small World には，緑の地面に道路があり，道路脇に 12 本の木が植えられ，3 つの神殿が建てられている．さらに，中央にはインタフェース部品であるスケーラとダイアログボックス，2 つのラジオボタンが浮かんでいる．プログラムの実行結果を図 25に示す．

図 25　**昼の風景**

Small World では，スケーラで地下水の水位を自由に変更できる．図 25に示す初期設定では，地下水の水位は 0 メートルなので地表に地下水は溢れ出していない．この水位をスケーラで，5 メートル，16 メートルと変更した際の Small World の様子を図 26, 図 27に示す．また，ラジオボタンで昼夜を変更することも可能である．図 28に夜景を示す．図 28では，中央付近にスポットライトが当てられている．

図 26 **水位上昇** (1)

図 27 **水位上昇** (2)

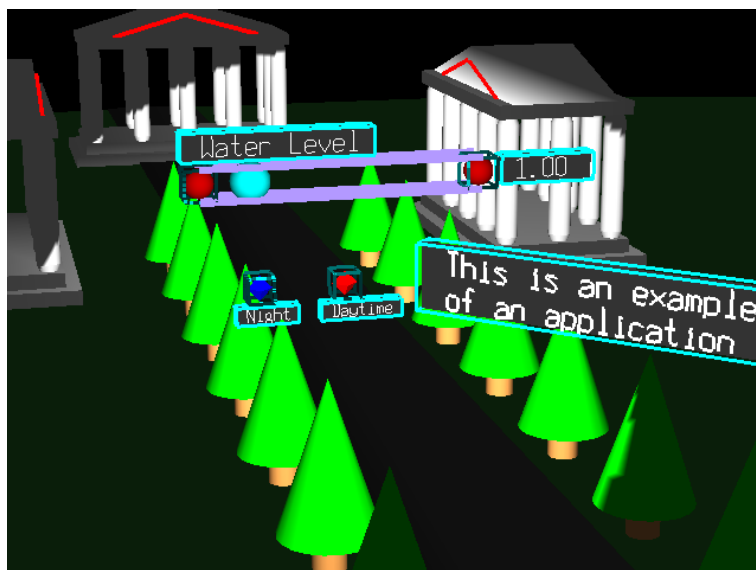


図 28 **夜の風景**

以下では，Small World のプログラムの各部について説明する．プログラムは付録 Cに付載する．このプログラムでは，まず外観となる地面や神殿を作成し，スケーラやダイアログボックスなどのインタフェース部品を作成する．

```
 3: void MakeGround();       // 地面を描画.
 4: void Make12Trees();      // 12 本の木を描画.
 5: void Make3Shrines();     // 3 つの神殿を描画.
```

この部分は，背景となる地面や神殿などの CG を作成する関数を示している．背景は全て Primitive クラスで定義される基本形状で作成した．ここでは簡単のためこれらの関数の内部には触れない．

```
 8: void WaterRise(Object *water, Object *scale){
 9:   float tmp;
10:   tmp = ((Scale*)scale)->GetValue();
11:   water->Size(1, tmp, 1);
12: }
```

この関数はユーザが定義した関数で，スケーラで指定した値に背景の水位を設定する．10 行目でスケーラの値を取得し，11 行目で水塊の大きさを変更している．

```
14: // シーンを夜にする関数
15: void ChangeNight(Object *spotlight){
16:   glClearColor(0,0,0,0);
17:   world->light->Off();
18:   ((Light*)spotlight)->On();
19:   world->Draw();
20: }
21:
22: // シーンを昼にする関数
23: void ChangeDaytime(Object *spotlight){
24:   glClearColor(0.3, 0.5, 1.0, 1.0);
25:   world->light->On();
26:   ((Light*)spotlight)->Off();
27:   world->Draw();
28: }
```

これらの関数もユーザが定義した関数で，空間中に浮かぶ 2 つのラジオボタンにバインドされる．引数に Light クラスのスポットライトを取り，各々のラジオボタンを押すことで昼夜の変更が行なえる．

```
35:   // スポットライトを作成
36:   Light *spot = new Light();
37:   spot->SetPosition(0, 3, -3, 1);
38:   spot->SetSpotDirection(0, -0.2, -1);
39:   spot->SetSpotCutoff(22.5);
40:   spot->Off();
```

スポットライトを作成し，3 次元位置とライトの照射方向，および照射範囲の設定を行なっている．初期設定ではスポットライトを照射しない (40 行目)．

```
49:   // 水位を上昇させるスケーラを作成
50:   Scale *water_scale = new Scale;
51:   water_scale->SetLength(10);
52:   water_scale->Move(-10, 10, 15);
53:   water_scale->String("Water Level");
54:   water_scale->SetRange(1, 20);
55:   water_scale->SetValue(1);
56:   water_scale->
57:      BindFunction(WaterRise, ground, water_scale);
58:   world->AddChild(water_scale);
```

スケーラを作成し，3 次元位置やスケーラに付加する文字情報，値の取り得る範囲などを指定している．また 56 行目では，前述のユーザ定義関数 WaterRise をスケーラに対応付けている．スケーラ以外のインタフェース部品についても同様の手順で宣言，設定されている．

```
84:   world->InitCameraPos(-25, 25, 50);
```

カメラの初期位置を設定している．表示装置がディスプレイの場合のみ実行される．表示装置に HMD を使用する場合，ユーザの頭の位置がカメラの位置になるため，この命令は意味を持たない (ただし，プログラムの実行には支障を来たさない).

このプログラムのソースコードを付録 Cに付載する．メイン部分は，付録 Cに示すように，約 60 行程度である．またインタフェース部品にバインドされる関数や，シーンの背景となる道路や木，神殿などを作成する関数全てを含めても，このプログラムは 220 行程度で作成できる．

4.4　例 3 – 3 次元カーソル –

本節では，Sensor クラス，Viewer クラスを用いて作成した 3 次元カーソルのプログラム例を説明する．3 次元カーソルとは，2 次元 GUI で用いられる 2 次元カーソルの 3 次元版で立体的な形状を持つ．3 次元カーソルは，2 次元 GUI でのマウスに相当する 3 次元位置入力装置 (図 15参照) で操作される．本ツールキットでは，3 次元カーソルを Cursor3D クラスで提供している．3 次元カーソルの形状を図 29に示す.

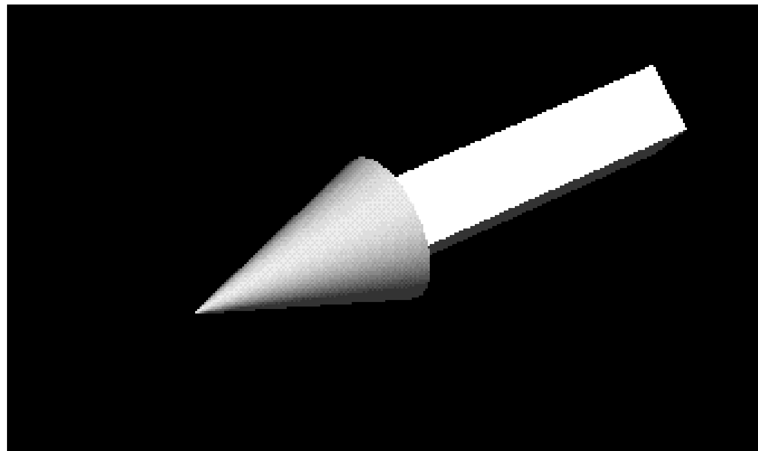


図 29　3 次元カーソルの形状

ここでは，本ツールキットのソースコード内から特に Sensor クラス，Viewer クラスに関連する箇所を 3 次元カーソルの作成を中心に概説する．なお，本節で例示するプログラムは，本ツールキットの構造の細部に触れるものであり，3DUI ツールキットのユーザであるプログラマが作成するプログラムに直接関係するものではないが，Sensor クラスおよび Viewer クラスの挙動を理解する上で重要と考えられる．

```
Cursor3D::Cursor3D(CursorLR lr, Viewer *v,
                   char *sn, int sp, char *sd, char *scf,
                   char *swn, char *swd){
    sensor = new Sensor(sn, sp, sd, scf);
```

上記のコードは，3 次元カーソルのコンストラクタの冒頭部分である．Cursor3D クラスでは，Sensor クラスのインスタンス “sensor” を持つ．sensor は，4×4 行列で表された，3 次元位置入力装置の位置，姿勢を 3 次元位置計測装置から取得するために用いられる．

```
void Cursor3D::ProcessCursor()
{
    cursor_matrix = sensor->PosRot() * offset;
    hitting = world->ProcessHits(cursor_matrix);
```

Cursor3D クラスのメンバ関数 ProcessCursor では，3 次元カーソルが現在指しているオブジェクトを特定するための関数である．また 3 次元カーソルを適切な位置に描画するための設定も行なう．最初の行で，sensor から 3 次元入力装置の現在の位置，姿勢を取得する．そして次の行で，その位置にオブジェクトが存在するかどうかを調べる．実際には，シーングラフを根から順に探索することで 3 次元カーソルの位置にオブジェクトが存在するかを判定する．

```
World::World(int mode){
  if(WORLD_MODE == MODE_3D){
     CursorR = new Cursor3D(CURSOR_RIGHT, viewer,
                            "Fastrak", 4, "/dev/ttyd2",
                            "3space.dat",
                            "Stamp", "/dev/ttyd1");
```

上記は World クラスのコンストラクタの冒頭部分で，3 次元カーソルを作成している箇所である．ここでは，3 次元位置計測装置に Polhemus 社の Fastrak を用いている．実際には，3 次元カーソルを作成するコンストラクタの引数には，パラメータファイルで指定された値を変数として代入する．

```
  viewer = new ylViewer(WorldDraw);
  if(argv[1] != NULL)
          viewer->LoadConfig(argv[1]);
```

上記は Viewer クラスのインスタンス “viewer” を作成している部分で，作成時の引数には，仮想環境を再描画する関数を指定する．また Small World プロ

グラム起動時にパラメータファイルを引数に指定するが，Viewer クラスの関数 LoadConfig でこのパラメータファイルを読み込んでいる．

```
void World::IdleFunc3D(){
     CursorR->ProcessCursor();
     viewer->Draw();
}
```

World クラスの関数 IdleFunc3D は，入力装置によるピッキングやドラッグなどのイベントがない場合に実行される関数である．ここでは，3 次元カーソルがオブジェクトの存在する位置にあるかどうかを判定する作業と，“viewer” によって仮想環境を表示装置に描画する作業を行なっている．このとき呼ばれる関数は，“viewer” の生成時に指定した，仮想環境を再描画する関数 WorldDraw である．

5. 考察

2.2節で 3DUI ツールキットが満たすべき要件として挙げた事柄は，(1)3 次元的な形状を持ち，様々な処理の対応づけが可能なインタフェース部品を提供すること，(2) 入力装置と表示装置の利用を支援する機能を提供すること，の 2 点であった．本章では，幾つかのプログラム例の作成を通じて明らかになった本ツールキットの特性について，2 点の 3DUI 開発の趣旨に照らし合わせて考察する．

まず (1) について，本ツールキットが提供するインタフェース部品について考える．4.2節で挙げたプログラム例では，ダイアログボックスを生成する関数をプッシュボタンに対応づけている．プログラマが定義した関数をインタフェース部品に対応づける関数 BindFunction は，Object クラスで定義されている関数である．Object クラスは，図 4で示すように，全てのインタフェース部品の基礎となるクラスであり，Object クラスで定義された関数は，全てのインタフェース部品に共通に適用することが可能である．また 3.4.3項で述べたように，Primitive クラスで定義された幾何形状に，ユーザが定義した関数をバインドすることで，新たなインタフェース部品を構築できる．

次に (2) について，本ツールキットではパラメータファイルを導入し，通常の 2 次元 GUI 環境で使用するキーボード，マウス，ディスプレイ以外の特殊な機器に依存する部分をプログラムから切り分けた．つまり使用する入力装置や表示装置を変更する場合，プログラムは変更せずに，パラメータファイルの必要な箇所のみを修正すればよい．これによりプログラマは，3DAP の機器構成を意識することなく，プログラムを作成することができる．

以上のように本ツールキットは，最初に設定した要件を満たしていると考えられる．本ツールキットは現段階では，限られたインタフェース部品しか定義されておらず，また十分な形状生成能力を備えてはいないが，短時間で 3DAP を構築し，その概観を把握したい場合には，十分有効であると考える．したがって，3DAP のプロトタイプ作成などの用途が考えられる．

6. むすび

本論文では，3 次元仮想環境を提供するアプリケーションの開発を支援する 3 次元ユーザインタフェースツールキットの設計および実装について述べた．3 次元ユーザインタフェースツールキットは，3 次元的な形状を持つボタンやメニューなどの仮想的なインタフェース部品を提供し，また実際のハードウェアである入力装置や表示装置を制御する機能を提供する．本研究では，インタフェース部品だけでなく各装置の制御部をオブジェクト指向的にモデル化し実装した．こうすることで種々の装置を仮想化し，プログラム上ではボタンなどのインタフェース部品と同等に扱うことが可能となる．

また 3 次元仮想環境を提供するアプリケーションでは，入力装置や表示装置に様々な機器を使用できるので，使用する機器構成に応じてプログラムを変更せずに使用できるようにした．つまり，プログラマが機器構成を気にする必要なくアプリケーションのインタフェース部分を構築できるように，入力装置や表示装置を抽象化した．

本研究では，2 次元 GUI で提供されている部品群をモデルにして，3 次元のユーザインタフェース部品を構築した．しかし 3 次元仮想環境は 2 次元 GUI 環境より操作性の自由度が高いので，3 次元仮想環境内での使用に特化した部品や機能も考慮する必要がある．例えば，ユーザは広大な仮想環境内での作業によって自分の現在の位置や向いている方向を見失いやすいので，仮想環境中でのユーザの移動を制御するナビゲーション機能などが考えられる．

また，今回実装したインタフェース部品は，その配色の変更や機能の追加などは容易に行なえるが，形状を変更することはできないので，インタフェース部品の形状を容易に変更できる機構が必要である．またインタフェース部品の形状の妥当性を検討する必要もある．

本ツールキットが提供するインタフェース部品や直方体，球などの基本的な幾何形状を用いることで，様々なアプリケーションのインタフェース部分を作成することが可能である．インタフェース部品や幾何形状などのオブジェクトを仮想空間内に配置する手法として，現段階では簡単のため，インタフェースプログラム中で直接，配置すべき箇所の 3 次元座標値を指定する手法を採用している．し

かしこの場合，オブジェクトの大きさや配置する 3 次元座標位置を綿密に算出しておく必要がある．また，通常，インタフェース部分の作成時には，オブジェクトの配置箇所や大きさ，配色，形状などを試行錯誤しながら変更することが多い．この変更毎にプログラムを再コンパイルする必要がある．そこでオブジェクトの配置や大きさなどを仮想空間内でインタラクティブに変更する，あるいは設定することができれば，インタフェース部品の再配置が容易になり，アプリケーション開発に必要な時間を短縮することが可能であると考えられる．実際，仮想空間内でインタラクティブに仮想オブジェクトをモデリングするアプリケーションも考案されているので [9]，そのようなアプリケーションとの統合も検討すべきである．

謝辞

本研究の全過程を通して，直接懇切なる御指導，御鞭撻を賜ったソフトウェア基礎講座 横矢 直和教授に衷心より感謝の意を表します．

本研究の遂行にあたり，終始有益な御助言と御鞭撻を頂いた像情報処理学講座 千原 國宏教授，並びにソフトウェア基礎講座 竹村 治雄助教授に厚く御礼申し上げます．

さらに，本研究を通して，有益な御助言を頂いたソフトウェア基礎講座 岩佐 英彦助手，並びにソフトウェア基礎講座 山澤 一誠助手に厚く感謝します．

そして物心両面において常に温かい御助言を頂いたソフトウェア基礎講座の大隈 隆史氏，並びに清川 清氏に深く感謝します．

最後に，本研究の全過程を通して終始有益な御助言を頂いたソフトウェア基礎講座の諸氏，ソフトウェア基礎講座事務補佐員 福永 博美女史，ならびに元ソフトウェア基礎講座事務補佐員 村上 和代女史に深く感謝します．

参考文献

[1] J.N.Latta and D.J.Oberg. A Conceptual Virtual Reality Model. *IEEE Computer Graphics and Applications*, Vol.14, No.1, pp. 23–29, January 1994.

[2] G.G.Robertson, S.K.Card, and J.D.Mackinlay. Information Visualization Using 3D Interactive Animation. *Communications of the ACM*, Vol.36, No.4, pp.57–71, April 1993.

[3] R.Gossweiler, R.J.Laferriere, M.L.Keller, et al. An Intruductory Tutorial for Developing Multiuser Virtual Environments. *Presence*, Vol.3, No.4, pp.255–264, 1994.

[4] D.Kurmann and M.Engeli. Modeling Virtual Space in Architecture. *Proc. Conf. on Virtual Reality Software and Technology(VRST '94)*, pp.77–82, 1996.

[5] L.Treinish and D.Silver. Visualizing the Visible Human. *IEEE Computer Graphics and Applications*, Vol.16, No.1, January 1996.

[6] J.Lansdown and S.Schofield. Expressive Rendering: A Review of Nonphotorealistic Techniques. *IEEE Computer Graphics and Applications*, Vol.15, No.3, May 1995.

[7] G.G.Grinstein and D.A.Southard. Rapid Modeling and Design in Virtual Environments. *Presence*, Vol.5, No.1, pp.146–158, 1996.

[8] K.Coninx, F.V.Reeth, and E.Flerackers. A Hybrid 2D/3D User Interface fir Immersive Object Modeling. *Proc. Computer Graphics International 1997*, pp.47–55, 1997.

[9] K.Kiyokawa, H.Takemura, Y.Katayama, H.Iwasa and N.Yokoya. VLEGO: A simple two-handed modeling environment based on toy blocks. *Proc. Conf. on Virtual Reality Software and Technology(VRST '96)*, pp.27–34, 1996.

[10] E.F.Johnson and K.Reichard. *Motif1.2 Power Programing.* Management Information Source,Inc., 1993.

[11] J.K.Ousterhout. *Tcl and Tk Toolkit.* Addison-Wesley, 1994.

[12] R.C.Zelezik, K.P.Herndon, D.C.Robbins, et al. An Interactive 3D Toolkit for Constructing 3D Widgets. *Computer Graphics(SIGGRAGH '93)*, pp.81–84, 1993.

[13] P.S.Stauss and R.Carey. An Object-Oriented 3D Graphics Toolkit. *Proc. ACM Conf. on User Interface Software and Technology(UIST'92)*, pp.341–349, 1992.

[14] G.Leach, G.A.Qaimari, M.Grieve, et al. *Human-Computer Interaction: INTERACT'97*, chapter Elements of a Three-dimensional Graphical User Interface. Champman & Hall, 1997.

[15] *WorldToolKit Reference Manual version2.0.* Asahi Electronics Co.,Ltd.

[16] John Brooke. *Interacting with Virtual Environments*, chapter Designing Flexible and Adaptable Interfaces. John Wiley & Sons Ltd, 1994.

[17] J.Tsao and C.J.Lumsdes. CRYSTAL: Building Multicontext Virtual Environments. *Presence*, Vol.6, No.1, pp.57–72, 1997.

[18] S.Card, G.G.Robertson, and J.D.Mackinlay. The Information Visualizer, An Information Workspace. *Proc. ACM Conf. on Human Factors in Computing Systems(CHI '91)*, pp.181–188, 1991.

[19] 世利至彦，清川清，竹村治雄，横矢直和. 汎用３次元ユーザインタフェースツールキットの設計. 電子情報通信学会 1997 年総合大会, 1997. A-16-16.

[20] J.Rumbaugh. *Object-Oriented Modeling and Design.* Prentice Hall,Inc., 1991.

[21] S.Oualline. *Practical C++ Programming.* O'Reilly & Associates, 1995.

[22] M.A.Ellis and B.Stroustrup. *The Annotated C++ Reference Manual.* AT & T Bell Telephone Laboratories,Inc., 1992.

[23] J.Neider, T.Davis, and M.Woo. *OpenGL Programming Guide.* Addison-Wesley, 1993.

[24] C.Elliott, G.Schechter, R.Yeung, et al. TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications. *Computer Graphics(SIGGRAPH '94)*, pp.421–434, 1994.

[25] N.Hiraki, K.Kiyokawa, H.Takemura, and Y.Yokoya. Imposing Geometric Constraints on Virtual Objects within an Immersive Modeler. *Proc. International Conference on Artificial Reality and Tele-existence '97(ICAT '97)*, pp.178–183, 1997.

[26] C.Ware, K.Arthur, and K.S.Booth. Fish tank virtual reality. *Proc. ACM Conf. on Human Factors in Computing Systems(INTERCHI '93)*, pp.37–42, 1993.

付録

A.　3次元ユーザインタフェースツールキット仕様書

本仕様書は本研究において実装した3次元インタフェースツールキットの仕様を述べることを目的とする．本仕様書では，実装したツールキットの構成の概要について述べた後，本ツールキットで提供する2種類の表示モードについて述べ，各クラスについて詳細を述べる．

A.1　ツールキットの概要

本ツールキットは１）インタフェース部，２）入力装置制御部，３）出力装置制御部の３つの部分から構成される．インタフェース部に属する大部分のクラス群は継承により階層的に定義されるクラス構成を持つが，入力装置制御部，出力装置制御部に属するクラスは特に継承関係を持たない．

本ツールキットの設計思想を理解するために重要なクラスとして，Object クラス，World クラス，Primitive クラスが存在する．クラス階層の根として定義されているのが Object クラスである．すなわち，仮想空間内に描画される物体はすべてインタフェース部に属し，Object クラスを継承したクラスのインスタンスとして宣言される（ただし，World オブジェクトが描画する床や座標軸は除く）． World クラスはインタフェース部に属するが Object クラスとの継承関係はなく，主に仮想環境全体の管理を行うクラスである．以降，Object クラスとその継承クラスのインスタンスを単にオブジェクトと呼ぶ．また World クラスのインスタンスを World オブジェクトと呼ぶ．各3次元インタフェース部品の形状は Primitive クラスのインスタンスで構成されている．Primitive クラスは，基本的な幾何形状を提供するクラスで，これも Object クラスの継承クラスである．実装言語には C++を採用し，3次元コンピュータグラフィックスライブラリとして OpenGL を用いた．

A.2　2D モードと 3D モード

本ツールキットは，通常のディスプレイ画面やプロジェクタをサポートする 2D モードと，HMD をサポートする 3D モードを提供している．2D モードでは 1 つのウィンドウまたは画面全体に仮想環境を表示するもので，視点の移動や仮想物体の操作にキーボードやマウスを使用する．これに対して 3D モードでは HMD などを用いて没入型仮想環境を提示するためのもので，視点の移動や仮想物体の操作には 3 次元位置センサなど人工現実感技術で用いられる特殊なデバイスを使用する．なお，モードの切替はパラメータファイルで行なう．

各モードでの処理の主な違いは仮想物体のピック処理の実装方法である．2D モードでは，OpenGL が提供するセレクション機構を用いてピック処理を実装した．3D モードでは，Cursor クラス（後述）が 3 次元カーソルの位置を各オブジェクトに知らせ，各オブジェクトがカーソルとの当たり判定を行う．

A.3　Button クラス

Button クラスは Object クラスの継承クラスである．ただし，Button クラス自体は，形状は持たない抽象クラスである．Button クラスでは，Button クラスの子クラスである PushButton クラスや，ToggleButton クラスなどで共通に使用される変数や関数を定義している．

各種ボタンクラスのインスタンスは，ラベルを付加することで，その機能や状態を文字情報として提示することができる．また，Button クラスでは，ピック時に視覚的フィードバックを与えるためのリアクションを定義している．リアクションとは，ピック時またはドラッグ時に実行される，オブジェクトの種類毎に固有の処理のことである．実際には，リアクションはオブジェクトの種類毎に関数 Reaction で定義される．関数 Reaction 内部では，ユーザによってバインドされた関数も呼び出される．この関数 Reaction をユーザが再定義することはできないが，ユーザは関数 BindFunction を用いることで様々な処理をオブジェクトにバインドすることが可能である．PushButton クラスで定義されている関数 Reaction を図 1 に示す．図 30では，プッシュボタンの色を青に変更し，ユーザによってバインドされた関数 (もしあれば) を実行する．

```
void  PushButton::Reaction(){
         ball->Color(BLUE);
         CallBindFunction(); // バインドされた関数の実行
}
```

図 30 **プッシュボタンでの関数** Reaction

Button クラスに特有の変数データは，ボタンに付加できるラベルのみであり，また Button クラスで定義されている関数は次の通りである．

Draw ボタンの形状を描画する．

String ボタンに付加するラベルの文字列を指定する．

```
// ボタンに付けることのできるラベルの位置
#define  ABOVE  0
#define  UNDER  1
#define  RIGHT  2
#define  LEFT   3

class Button : public Object
{
   public:
       // ボタンに付加するラベル
       Label    *label;
       // コンストラクタ・デストラクタ
       Button();
       ~Button();
```

```
        // ボタンの形状描画
        virtual void     Draw();
        // ボタンに付加するラベルを設定
        void             String(char *s, int anchor);
        void             String(char *s);
};
```

A.4 Camera クラス

ユーザの「目」に対応するカメラの設定を行うクラス．2D モードでのみ使用される．3D モードでは，Camera クラスに代わり Viewer クラスが Sensor クラスと連係して，カメラの設定を行なう．2D モードでは，一つのウィンドウ内に仮想環境を構築するが，Camera クラスは，このウィンドウの設定に必要な変数データを持つ．以下に，Camera クラスの持つ変数データを挙げる．

X_POS_OF_WINDOW 画面座標系でのウィンドウの位置 (x 座標).

Y_POS_OF_WINDOW 画面座標系でのウィンドウの位置 (y 座標).

width ウィンドウの幅.

height ウィンドウの高さ.

fovy カメラの視野角.

nearDistance 視点から最も手前にあるクリップ面までの距離.

farDistance 視点から最も奥にあるクリップ面までの距離.

title 表示ウィンドウのタイトル.

Camera クラスで定義される関数は主に，カメラの移動に関連する関数などである．以下に，Camera クラスで定義される関数を述べる．

FrontMove カメラを前進させる.

BackMove カメラを後退させる.

RightMove カメラを右に平行移動させる.

LeftMove カメラを左に平行移動させる.

TurnMove カメラをパンする.

ReshapeWindow ウィンドウサイズを変更した際に，ウィンドウの再描画を行なう.

InitCameraPos カメラの初期位置を設定する.

getFovy 現在のカメラの視野角を返す.

getNear 視点から最も手前にあるクリップ面までの距離を返す.

getFar 視点から最も奥にあるクリップ面までの距離を返す.

```
class Camera {
 public:
    Camera(int, int);
    Camera();
    ~Camera(){};
    void FrontMove();   // カメラを前に移動
    void BackMove();    // カメラを後ろに移動
    void RightMove();   // カメラを右に移動
    void LeftMove();    // カメラを左に移動
    void HorizontalRotation(float);  // カメラを水平方向に回転
                                     // (正…右回転，負…左回転)
    void VerticalRotation(float);    // カメラを鉛直方向に回転
                                     // (正…上回転，負…下回転)
```

```
    // カメラの位置を指定
    void InitCameraPos(float, float, float);
};
```

A.5 Constraint クラス

オブジェクトの振る舞いに制約を与えるための特殊クラス．Object は継承していない．制約を表す unsigned int 型の変数 constraints を持つ．本ツールキットが提供している制約 1 種類につき constraints の 1 ビットが割り当てられ，ビットの ON/OFF によって対応する制約を受けるかどうかが決定される．Constraint クラスはこの他に，直線制約で用いる直線の端点データなど，制約付けに必要な情報を保持する．

オブジェクトに対する制約付けは，Object クラスの関数 AddConstraint によって行われる．付加する制約の指定は，制約の種類を表す定数の論理和を AddConstraint の引数に与えることで行う．本ツールキットで提供する制約とその制約を表す定数を以下に示す．

SELF_MOVE シーングラフ上での自分の親はドラッグされないが，自分以下の子はドラッグされる．この制約を用いない場合 (デフォルト設定) でドラッグの対象となるのは，選択したオブジェクトを含むオブジェクト木全体である．オブジェクト木は，シーングラフの部分木で，World オブジェクトの直接の子を根とする木を指す．

STOP 静止 (ドラッグされない).

VIEW_TRACE 視点追従．視点変更に関わらず常に画面の同じ場所にオブジェクトを描画する．

LINEAR_MOVE 直線制約．オブジェクトをドラッグした際，ある直線上でしか移動しないようにする．

A.6 Entry クラス

1 行の文字列を入力または編集することのできるインタフェース部品（エントリ）を生成するクラス．例えば，あるドキュメントが始めてディスクに保存される場合，ユーザはそのファイル名を指定する必要がある．このような場合にファイル名を入力するのに用いることができる．文字列を入力するには，エントリ上でマウスボタンをクリックし，エントリをフォーカスされた状態にする．エントリがフォーカスされると，キーボードからの入力はエントリに渡され，文字列として表示される．エントリの形状は，長方形の板 (背板) の上に文字列を並べ，その板の周りをワイヤーフレームの直方体で囲んだものである．フォーカスされると，ワイヤーフレームの色が赤くなる．

入力された文字列は，エントリが持つ変数 string に格納される．Entry クラスで定義される関数には次のものがある．

GetString エントリに入力された文字列へのポインタを返す．内部的には，入力された文字列分の領域を新たに確保し，文字列の複製を作成する．返される値はこの新しく確保した領域の先頭アドレスである．

SetFrameColor ワイヤーフレームの色を指定する．

SetBackColor 背板の色を指定する．

SetStringColor 文字列の色を指定する．

```
class Entry : public Object
{
    public:
        TextLine    *string;     // 表示される (入力された) 文字列

        Entry();
        ~Entry();
        void     Draw();          // 描画を実行する
```

```
        void     Reaction();      // ピック時のリアクションを指定
        void     NotReaction();   // リアクションからの復帰処理
        // ラベルの指定
        void     String(char*);
        // フレームの色を変更する.
        void     SetFrameColor(float, float, float);
        void     SetFrameColor(Vector3D);
        void     SetFrameColor(float*);
        // 背板の色を変更する.
        void     SetBackColor(float, float, float);
        void     SetBackColor(Vector3D);
        void     SetBackColor(float*);
        // 文字列の色を変更する.
        void     SetStringColor(float, float, float);
        void     SetStringColor(Vector3D);
        void     SetStringColor(float*);
};
```

A.7 Light クラス

仮想環境内の光源の設定を行うクラス．World オブジェクトの作成時にデフォルトで 1 つの光源が作成される．最大 8 個まで光源の設定が行える．Light クラスで定義されるデータを以下に示す.

ambient 環境光の色.

diffuse 拡散光の色.

specular 鏡面光の色.

position 光源の位置.

direction スポットライトの照射方向.

exponent スポットライトの輝度分布.

cutoff スポットライトの広がり角度.

Light クラスで定義されている関数を以下に示す.

SetAmbient 環境光の色を指定する.

SetDiffuse 拡散光の色を指定する.

SetSpecular 鏡面光の色を指定する.

SetPosition 光源の位置を指定する.

SetDirection スポットライトの照射方向を指定する.

SetExponent スポットライトの輝度分布を指定する.

SetCutoff スポットライトの広がり角度を指定する.

On ライトを点ける.

Off ライトを消す.

```
class Light {
    public:
     Light();
     ~Light();
     void On();       // ライトを点ける
     void Off();      // ライトを消す
     // 環境光の色を指定する(RGB と α を指定. 以下同じ)
     void SetAmbient(float, float, float, float);
     // 拡散光の色を指定する
     void SetDiffuse(float, float, float, float);
     // 鏡面光の色を指定する
     void SetSpecular(float, float, float, float);
```

```
    // 光源の位置を指定する
    void SetPosition(float, float, float);
    // スポットライトの照射方向を指定する
    void SetSpotDirection(float, float, float);
    // スポットライトの輝度分布を指定する
    void SetSpotExponent(int);
    // スポットライトの広がり角度を指定する
    void SetSpotCutoff(int);
};
```

A.8 MenuButton クラス

プルダウンメニューを作成するクラス．クリックする毎にメニュー項目の掲示,非掲示が切り替わるので，特殊なトグルボタンと考えることもできる．メニューボタンをクリックすると，そのメニューボタンの下に関連する項目が掲示される．Button クラスの子クラスで提供される 4 種類のボタン全てが，項目として利用可能である．特に，項目にメニューボタンを入れることで，階層的なプルダウンメニューを作成できる．

このクラスに特有の関数として，次のものがある．

AddItem 引数で渡されたボタンをメニュー項目に追加する．

```
#define  NEWLINE   -1     // 各項目の間隔
#define  INDENT    0.5    // メニュー階層のインデント

class MenuButton : public Button
{
    public:
        Cube            *button;
        WireCube        *frame;
```

```
	List<Object*>	*ItemList;	// メニュー項目を格納するリスト

	MenuButton();
	~MenuButton();
	// 色を指定する
	void	Color(Vector3D);
	void	Color(float*);
	void	Color(float, float, float);
	// 描画を実行する
	virtual void	Draw();
	// ピック時のリアクションを指定
	void	Reaction();
	// リアクションからの復帰処理
	void	NotReaction();
	// 選択・非選択を表すフラグの値を設定する
	void	SetValue(int);
	// 現在のフラグの値を設定する
	int	ReturnValue();
	// メニュー項目のリストに新たな項目を追加する.
	Void	AddItem(Button*);
};
```

A.9 Object クラス

仮想空間に描画されるすべてのオブジェクトの基底となるクラス．Object クラスで定義されるデータを以下に挙げる．

parent シーングラフ上での自分の親へのポインタ.

ChildList シーングラフ上での自分の子供のリスト.

id 各インスタンスに固有な識別子（ID）.

Matrix 親の座標系での変換行列（平行移動，回転）.

ScaleMatrix 拡大，縮小に用いる行列.

constraints オブジェクトの振る舞いの制約.

BindFuncList, BindDragFuncList オブジェクトにバインドされる関数を管理するためのリスト．リストで関数を保持することで複数の関数を 1 つのオブジェクトにバインドできる．このリストには，ピック時に呼ばれる関数のリストとドラッグ時に呼ばれる関数のリストがある．バインドした関数は，ピックの場合ピック開始時に，ドラッグの場合ドラッグ終了時に呼ばれる.

各オブジェクトは各々に固有の ID を持っている．この ID は World オブジェクトが持つ ID テーブルで管理される．ID の持つ情報は ID 番号とオブジェクトの種類である．なお，ID テーブルは，2D モードのピック処理に用いられる.

「制約」はオブジェクトの振舞いを制限するためのものである．例えば，オブジェクトをある直線上でしかドラッグできないようにする（直線制約），オブジェクトをドラッグできないようにする（静止制約）などの制約が考えられる．これらの制約は Constraint クラスで提供される.

次に Object クラスで定義されている関数を以下に示す.

Draw オブジェクトを描画する．Object クラスのインスタンスは自分自身の形状を持っていない．したがって Object クラスでは，単にシーングラフ上での自分の子供に対し，描画命令を出すだけである．Object クラスの継承クラスで形状を持つ場合，描画関数を定義している.

Reaction オブジェクトがピックされたときのリアクションを定義．例えば，プッシュボタンがピックされた際に，プッシュボタンの色を変更する視覚的フィードバックなどが設定できる．また，ピック時に起動される関数が指定されている場合，その関数の呼び出しもここで行なわれる．

NotReaction ピック終了時，すなわち，マウスボタンが離されたときのリアクションを定義．

DragReaction ドラッグ時のリアクションを定義．

NotDragReaction ドラッグ時のリアクションからの復帰処理を行う．

AddChild 引数に指定したオブジェクトをシーングラフ上での自分の子供にする．すなわち，Object クラスが保持するリスト ChildList に，引数で渡されたオブジェクトを格納する．

Move 平行移動する．

Rotate 回転する．

Size 大きさを変更する．

GetPos 現在のワールド座標位置を返す．

BindFunction, BindDragFunction 関数をバインドする．オブジェクトをピックまたはドラッグする際に， 呼び出される関数をオブジェクトにバインドできる．バインドする関数は，その引数と共にリストで管理される．また，複数の関数を 1 つのオブジェクトにバインドすることも可能である．

AddConstraint オブジェクトに制約を付加する．オブジェクトに直線制約を付加する際には，その直線をあらわす端点データも引数で与える必要がある．

上記の他，Object クラスに対してコンストラクタ，デストラクタを用意している．コンストラクタは，クラスのインスタンスを初期化する特別な関数であり，デストラクタはコンストラクタと対をなすもので，オブジェクトが必要なくなった時に，そのオブジェクトが保持するリソースを解放し，クリーンアップする特別な関数である．コンストラクタ，デストラクタは共に C++で提供される機能である．

関数 Copy を使用することで，オブジェクトの複製が作成できる．

Object *a = new Object; Object *b = new Object; a = b-¿Copy();

この場合，"a" と "b" はそれぞれ別々のオブジェクト (識別子が異なる) であり，変数データも各々別々の領域がとられる．

```
class Object {
    public:
     Object         *parent;          // 親へのポインタ
     List<Object*>  *ChildList;       // 子供のリスト
     Id             *id;              // 各インスタンスに固有な ID
     Matrix       Matrix;           // 親に対する変換のための行列
     Matrix       SizeMatrix;       // スケーリングのための行列
     Matrix       RotateMatrix;     // 回転行列
     Constraint     *constraints;     // 制約
     List<BF*> *BindFuncList; // バインドする関数とその引数のリスト
     List<BF*> *BindDragFuncList;  // 上のドラッグ版

     int operator ==(const Object*);
     int operator !=(const Object*);
     int operator <=(const Object*);
     int operator >=(const Object*);
```

```
Object();
~Object();

virtual void Draw();                    // オブジェクトを描画.
virtual void AddChild(Object*);         // 子供をリストに追加する.

Object* Copy();                         // オブジェクトのコピー.
virtual void Reaction();          // ピック時のリアクションを指定.
virtual void NotReaction(){};     // リアクションからの復帰処理.
virtual void DragReaction(){};    // ドラッグ時のリアクションを指定
virtual void NotDragReaction();   // ドラッグ終了時のリアクションを
                                  // 指定
// ドラッグの際，シーングラフ上の親も一緒に動かす.
void  ParentMove(float, float, float);
void  ParentMove(Matrix);
// 親のリアクションを実行する.
void  ParentReact();
// 大きさを設定
void  Size(const Vector3D);
void  Size(const float, const float, const float);
void  Size(const float);
// オブジェクトの回転
void  Rotate(const float rad, const float,
             const float, const float);
void  Rotate(const float, const Vector3D);
// オブジェクトの平行移動
void  Move(const float, const float, const float);
void  Move(const Vector3D);
```

```
    void  Move(Matrix);
    // 新たに関数をバインドする.
    void BindFunction(void (*func)(...), ...);
    // バインドする関数を追加する.
    void AppendFunction(void (*func)(...), ...);
    // BindFunction のドラッグ版.
    void BindDragFunction(void (*func)(...), ...);
    // Appendfunction のドラッグ版.
    void AppendDragFunction(void (*func)(...), ...);
    // バインドされた関数を実行する.
    void CallBindFunction();
    // CallBindFucntion のドラッグ版.
    void CallBindDragFunction();
    // 制約を付加する.
    void AddConstraint(const unsigned int constraint);
    // 直線, 回転制約の際に用いる端点データを指定する
    void AddConstraint(const unsigned int constraint,
                       Object *Left, Object *Right);
    // 自分の位置をベクトル形式で得る.
    // PM() の結果から現在のワールド座標を返す.
    Vector3D& GetPos();
    // 自分の位置を行列形式で得る.
    Matrix GetMatrix();
    // 行列を指定する.
    void SetMatrix(Matrix);
    // キーバインドの設定
    virtual void KeyboardFunc(unsigned char, int, int);
};
```

A.10 Primitive クラス

Primitive クラスは Object クラスの継承クラスで，様々な幾何形状を提供する．Primitive クラスは，その継承クラスである Sphere クラスや Cube クラスで共通に使用される変数や関数を定義する抽象クラスである．Primitive クラスでは，基本形状の色を変更する関数が定義されている．

Color 色の変更を行なう．色の指定は RGB 値を直接指定する方法と，色を表す文字列を指定する方法がある．RGB 値とは，スクリーン上の各ピクセルに放射される赤，緑，青の光の割合を一つにまとめたもので，例えば，黒に相当する RGB 値は (0, 0, 0)，紫の RGB 値は (1, 0, 1) となる．後者の場合，”WHITE” や”BLACK”などの文字列を指定する．

Primitive クラスの継承クラスとして定義されている基本形状のクラスを以下に示す．各形状の生成に必要な数値をコンストラクタの引数に与える．

Cube クラス 直方体や立方体を生成する．生成時 (コンストラクタの引数) に縦，横，高さを指定する．

WireCube クラス ワイヤーフレームの直方体や立方体を作成する．生成時に縦，横，高さを指定する．

Sphere クラス 球や楕円球を生成する．生成時に半径を指定する．楕円球は，球を作成した後，Object クラスの Size 関数を適用することで生成できる．

Cone クラス 円錐を生成する．生成時に底面の半径と高さを指定する．

Circle クラス 円を生成する．生成時に半径を指定する．

Cinder クラス 円柱を生成する．生成時に上面と下面の半径および高さを指定する．

Line クラス 線分を生成する．2 点以上の頂点データを指定する必要がある．また，3 点以上の頂点データを与えることで，それらの頂点を順に結んだ折れ線が生成できる．頂点データは，ワールド座標系での x，y，z 値の 3 値からなる．頂点データをコンストラクタの引数に渡して直線または折れ線を生成するが，コンストラクタの引数の最後には終端文字 (L_END) を付加する必要がある．

Polygon クラス 任意の多角形を生成する．直線の場合と同様に，コンストラクタの引数に頂点データを指定する．このとき，引数の最後には終端文字 (P_END) を付加する必要がある．3 点以上の頂点データを指定する必要があり，3 点未満では何も描画しない．また，指定した多角形はそれ自体を交差することができず，凸状であることが必要である．頂点データが，これらの条件を満たさない場合，その結果は予測不能となる．

```
// 文字列で指定可能なプリミティブの色
enum ColorType {
    WHITE, BLACK, RED, BLUE, GREEN, YELLOW, GRAY, BROWN, PINK,
    SKYBLUE, YELLOWGREEN, NAVY, OCHER
};

class Primitive : public Object {
   public:
     Primitive();
     ~Primitive();

     // 色の指定
     void Color(Vector3D);
     void Color(float*);                          // ベクトルで指定
```

```
    void Color(GLfloat, GLfloat, GLfloat);  // RGBで指定
    void Color(ColorType);                  // 文字列で指定
};

class Cube : public Primitive {
   public:
    Cube();
    Cube(float, float, float);
    ~Cube(){};

    Cube* Copy();   // コピー
    void Draw();    // 描画
};

class WireCube : public Primitive {
   public:
    WireCube();
    WireCube(float, float, float);
    ~WireCube(){};
    WireCube* Copy();  //コピー
    void Draw();       // 描画
};

class Cinder : public Primitive {
   public:
    Cinder();
    Cinder(float base, float top, float h);
    ~Cinder(){};
    Cinder* Copy();  // コピー
```

```
    void Draw();       // 描画
};

class Sphere : public Primitive {
    public:     Sphere();
     Sphere(float r);
     ~Sphere(){};
     Sphere* Copy();   // コピー
     void Draw();      // 描画
};

#define  MAX_VERTEX   256
#define  L_END   -9999  // 可変引数リスト (各頂点データ)
                        // の終りを表す

class Line : public Primitive {

    public:
     Line();
     Line(int, ...);
     ~Line();
     void Draw();         // コピー
     Line* Copy();        // 描画
     void Width(float);   // 直線の太さを指定する
};

class Cone : public Primitive {
    public:
     Cone();
```

```
     Cone(float r, float h);
     ~Cone(){};
     Cone* Copy(); // コピー
     void Draw();  // 描画
};

class Circle : public Primitive {
    public:
     Circle();
     Circle(float);
     ~Circle(){};
     Circle* Copy(); //コピー
     void Draw();    // 描画
};

#define  P_END   999  // 可変引数リスト (各頂点データ) の
                      // 終りを表す

class Polygon : public Primitive {
    public:     Polygon();
     Polygon(int, ...);
     ~Polygon();
     Polygon* Copy();  // コピー
     void Draw();      // 描画
};

class Octahedron : public Primitive {
    public:     Octahedron();
     ~Octahedron();
```

```
    Octahedron* Copy(); // コピー
    void Draw();        // 描画
};
```

A.11 PushButton クラス

最も一般的なボタン (プッシュボタン) を生成するクラス．プッシュボタンはワイヤーフレームの立方体の中に球を埋め込んだ形状を持つ．通常，プッシュボタンが押された時に呼ばれる関数を指定して使用する．押下などのアクションを受けることで対応する関数を実行する一般的なボタン．PushButton の形状はワイヤフレームの立方体に球を埋め込んだものなので WireCube と Sphere をメンバに持つ．

```
class PushButton : public Button
{
   public:      Sphere  *ball;
    WireCube  *frame;
    PushButton();
    ~PushButton(){};
    // ボタン (ここでは球) の色を設定する．
    void Color(Vector3D);
    void Color(float*);
    void Color(float, float, float);
    void Reaction();          // ピック時のリアクションを指定
    void NotReaction();       // リアクションからの復帰処理
    // コピーを作る．
    PushButton* Copy();
};
```

A.12　RadioButton クラス

複数の選択肢の中から 1 つだけを排他的に選択する場合に用いられるボタン（ラジオボタン）を生成するクラス．ラジオボタンには，ワイヤーフレームの立方体に正 8 面体を埋め込んだ形状を持たせた．このクラスでは，自分自身が選択されている状態にあるかを示す変数 value を持つ．選択されていれば 1，そうでなければ 0 となる．また，このクラスでは以下の 2 つの関数が定義される．

GetValue　value の値を返す．

SetValue　value の値をセットする．

関数 SetValue は，主にラジオボタン内部の処理やラジオボタンの初期化に用いられる．プログラムを作成する際は，関数 ReturnValue で特定のラジオボタンが選択されているかどうかを検出し，その結果に応じて行なう処理を条件分岐させて使用することが考えられる．

シーングラフ上で自分と共通の親を持つラジオボタンを同一セットのラジオボタンと考える．例えば，シーングラフが図 31のような場合，R1, R2, R3 が 1 セット，さらに R4，R5 が 1 セットとなる．ここで，O1, O2 は，幾つかのラジオボタンのセットを作成するための「器」的な役割を果たしている．異なるセット間に特に関連はなく，各セット内でラジオボタンの排他的な選択が行なわれる．

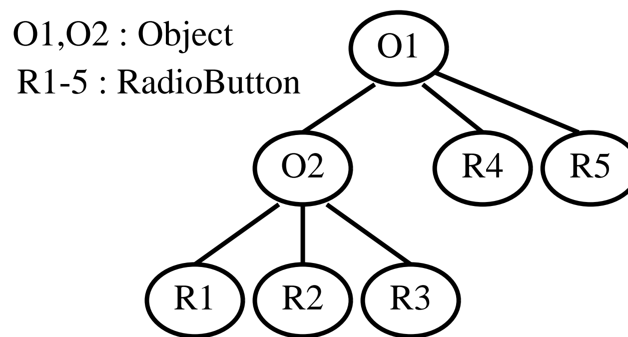


図 31　**ラジオボタンを含むシーングラフ**

```
class RadioButton : public Button
{
    public:
     Octahedron  *button;
     WireCube  *frame;
     RadioButton();
     ~RadioButton(){};
     void SetValue(int);      // 選択・非選択を表すフラグの値を設定する
     int ReturnValue();       // 現在のフラグの値を設定する
     void Color(Vector3D);    // 色を指定する
     void Color(float*);
     void Color(float, float, float);
     RadioButton* Copy();     // コピーを作る
     void Reaction();         // ピック時のリアクションを指定
     void NotReaction();      // リアクションからの復帰処理
};
```

A.13 Scaler クラス

スケーラを生成するクラス．スケーラは連続量を持つ変数に対応し，中央のつまみをドラッグすることでその変数値を連続的に変更できる．また，スケーラの両端のボタンを押すことで変数値を離散的に変更することもできる．スケーラは変数 value を持ち，つまみ，および両端のボタンで value の値を更新する．スケーラの提供する関数は次の通り．

SetValue value の値を設定する．

GetValue value の値を返す．

SetRange value の範囲を指定する．

String タイトルを付ける．

SetLength スケーラの長さを指定する．

```
class Scale : public Object
{
    public:
     PushButton *HeadLeft, *HeadRight;    // スケーラの端のボタン
     Sphere *ball;                        // スライダ
     Object *Lstop, *Rstop;               // スライダのストッパー
     Label *title;                        // スケーラのタイトル
     Scale();
     ~Scale();
     void Draw();                        // 描画を実行する
     void SetLength(float);              // スケーラの長さを指定する
     void  SetValue(const float);        // 値を設定する
     float GetValue();                   // 現在の値を返す
     void  SetRange(const int max, const int min);  // 値の範囲を指定

     void  SetDecimal(const int);      // 小数点のシフト数を設定
     void  String(char*);              // ラベルを付加する
};
```

A.14 Sensor クラス

3 次元位置計測装置から位置・姿勢を取得するクラス．センサの位置と姿勢を取得する．位置と姿勢は 4×4 行列か 3 次元ベクトルの組の形で表現される．現在サポートしている 3 次元位置計測装置は次の 3 種類である．

- Polhemus 社 3SPACE Fastrak
- Polhemus 社 3SPACE IsotrakII
- Shooting Star Technology 社 ADL-I

このクラスを使用する際，ユーザがパラメータファイルで以下の 2 つの行列を指定する必要がある (図 32参照).

- 測定の基準となる座標系における位置計測装置のソースの位置姿勢を示す行列
- センサを基準とする座標系 (レシーバ座標系) での，測定したい点の位置と姿勢を示す行列

以上の行列を設定しておくことで，測定の基準となる座標系における測定したい点の位置と姿勢を示す行列を求めることができる.

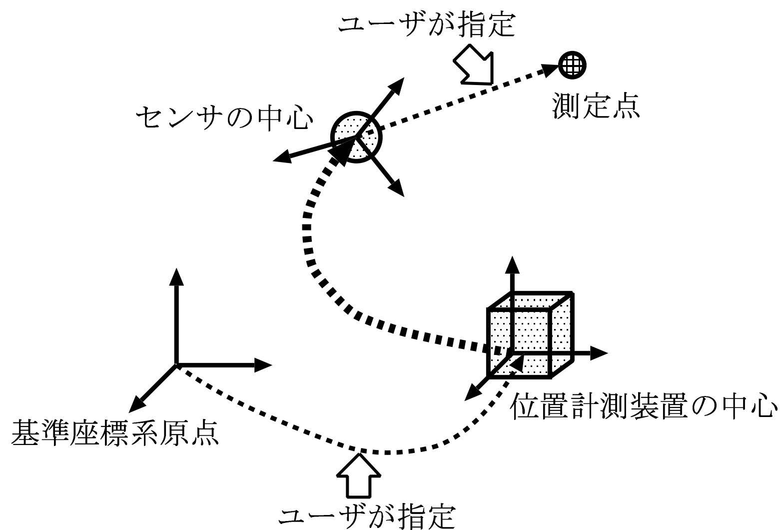


図 32 **測定点と Sensor クラスでの座標変換の関係**

Sensor クラスでは，上述のようにユーザがパラメータファイルで指定する 2 つの行列をデータとして持つ．また，Sensor クラスで定義される関数を以下に示す.

PosRot センサの位置姿勢を 4×4 行列で返す.

Pos センサの位置を 3 次元ベクトルで返す.

Rot センサの姿勢を 3 次元ベクトルで返す.

```
class Sensor
{
   public:
     // レシーバ座標系での測定点の位置姿勢
     // (測定点座標値からレシーバ座標値への変換行列)
     Matrix measure_point;

     // このセンサが移動する空間の基準座標系でのソースの位置姿勢
     // (ソース座標値からこのセンサの移動する空間の基準座標値への
     //  変換行列)
     Matrix origin;

     // sensor_name     : "Adl", "IsotrakII", "Fastrak", "None"
     // d_port          : ignored for "None"
     //                 : 1 for "/dev/ttyd1"
     //                 : 2 for "/dev/ttyd2"
     // p_num           : ignored     for "Adl", "None"
     //                 : 1, 2        for "IsotrakII"
     //                 : 1, 2, 3, 4  for "Fastrak"
     Sensor(char *sensor_name,   int sensor_number,
                 char *device,   char *calib);
     Sensor(char *sensor_name,   int d_port,
                 int p_num,      char *calib);
     ~Sensor ();
```

```
    // These calls are available for ALL instance.
    Matrix RawPosRot();
    Matrix PosRot();
    Vector3D Pos();          // return 0 for "None"
    Vector3D Rot();          // return 0 for "None"
};
```

A.15 Switch クラス

3 次元位置入力装置に付随するスイッチの状態 (押されているかどうか) を調べるためのクラス．Switch クラスでは，各スイッチを 1 ビットに対応させたビット列を 3 次元位置入力装置から受け取り，このビット列を調べることで現在押されているスイッチを知ることができる．Switch クラスで定義される関数には次のものがある．

Read 各スイッチに対応したビット列を受け取る．

```
#ifndef SWITCH_RIGHT
#define SWITCH_RIGHT_1ST       (1<<12)
#define SWITCH_RIGHT_2ND       (1<<11)
#define SWITCH_RIGHT_3RD       (1<<10)
#define SWITCH_RIGHT_4TH       (1<<9)
#define SWITCH_RIGHT_5TH       (1<<8)

#define SWITCH_RIGHT ((1<<12) | (1<<11) | (1<<10) | (1<<9) | (1<<8))

#define SWITCH_LEFT_1ST        (1<<4)
#define SWITCH_LEFT_2ND        (1<<3)
#define SWITCH_LEFT_3RD        (1<<2)
#define SWITCH_LEFT_4TH        (1<<1)
```

```
#define SWITCH_LEFT_5TH        (1<<0)

#define SWITCH_LEFT ((1<<4) | (1<<3) | (1<<2) | (1<<1) | (1<<0))

class Switch
{
   public:
      // sw_name     : "Stamp", "TP1040", "None"
      // device      : device name, e.g. "/dev/ttyd1" for "Stamp"
      //             : host name,   e.g. "ytpl00"     for "TP1040"
      Switch(char* sw_name, char* device);
       ~Switch();

      int Read();       // スイッチの現在値を返す
};
```

A.16 Text クラス

文字列を表示するためのクラス．一枚の板 (背板) に文字列が書かれ，背板の周囲をワイヤーフレームで囲んだ形状を持つ．必要であればスクロールバーを持つことが可能である．用途として，ダイアログボックスや各種ボタンの付加情報を与えるラベルなどが考えられる．Text クラスが持つデータを以下に示す．

StringList 文字列を格納するためのリスト．

line_number 表示する文字列の行数．

string_color 文字列の色．

back_color 背板の色．

frame_color フレームの色.

height 背板の高さ.

width 背板の幅.

また，Text クラスには，以下の関数が定義されている.

String 表示する文字列を設定する.

SetWidth 一行あたりの文字数を設定する.

SetHeight 何行表示するかを指定する．文字列の行数が，指定した値を越えるとスクロールバーが付随する.

GetWidth 一行あたりの文字数を返す.

GetNumberOfLine 全行数を返す.

SetStringColor 文字の色を変更する.

SetBackColor 背板の色を変更する.

SetFrameColor フレームの色を変更する.

```
class Text : public Object
{
    public:
      Text();
      ~Text();
      void Draw();                    // テキストの描画
      // 文字列を設定する
      void String(char*);
      // 数字を文字列として設定する
      void String(int);
      // 浮動小数点数を文字列として設定する
```

```
    void String(float);
    // テキストをクリア
    void Clear();
    // 一行あたりの文字数を返す.
    int ReturnWidth();
    // 現在の全行数を返す.
    int  GetNumberOfLine();
    // フレームの色を指定する
    void SetFrameColor(float, float, float);
    void SetFrameColor(Vector3D);
    void SetFrameColor(float*);
    // 背板 (テキストパネルの色を指定する)
    void SetBackColor(float, float, float);
    void SetBackColor(Vector3D);
    void SetBackColor(float*);
    // 文字の色を指定する
    void SetStringColor(float, float, float);
    void SetStringColor(Vector3D);
    void SetStringColor(float*)
    // 一行あたりの文字数を設定 (テキストの横幅)
    void SetWidth(int);
    // 何行表示するかを設定する.
    // この数を越える場合, スクロールバーがつく.
    void SetHeight(int);
    // テキストの何行目から表示するかを指定する.
    void SetHeader(int);
};
```

A.17 ToggleButton クラス

クリックする毎に ON/OFF が切り替わるボタン（トグルボタン）を生成するクラス．例えば，文字列に下線を引くかどうか，計算結果の表示を行なうかどうかなどの二者択一を行なう際に用いられる．ラジオボタンと同様にトグルボタンも変数 value を持ち，クリックする毎に ON/OFF が切り替わる．トグルボタンでもラジオボタンと同様の関数が用意されている．

GetValue value の値を返す．

SetValue value の値をセットする．

関数 SetValue は，主にトグルボタン内部の処理やトグルボタンの初期化に用いられる．プログラムを作成する際は，関数 ReturnValue で特定のトグルボタンが選択されているかどうかを検出し，その結果に応じて行なう処理を定義して使用することが考えられる．

```
class ToggleButton : public Button
{
    public:     Cube  *button;
     WireCube  *frame;
     ToggleButton();
     ~ToggleButton(){};
     void Reaction();           // ピック時のリアクションを指定
     void NotReaction();        // リアクションからの復帰処理
     void Color(Vector3D);      // 色を指定する
     void Color(float*);
     void Color(float, float, float);
     ToggleButton* Copy();      // コピーを作る
     void SetValue(int);    // 選択・非選択を表すフラグの値を設定する
     int ReturnValue();     // 現在のフラグの値を設定する
};
```

A.18 Viewer クラス

ユーザが指定した表示装置に仮想空間を描画するためのクラス．表示装置として通常のディスプレイの他に，HMD，プロジェクタが使用可能である．またViewer クラスでは，表示形式として，単眼視画像，フレームシーケンシャルステレオ画像，フィールドシーケンシャルステレオ画像，画面分割式ステレオ画像をサポートする．

Viewer クラスのコンストラクタには引数として仮想空間を描画する関数（World クラスの Draw 関数等）を指定する．また，仮想空間の描画に必要となるデータ（ビューポートの大きさや位置，表示形式等）はパラメータファイルで設定する．

Viewer クラスで定義されている関数は，主に仮想空間を描画する関数，パラメータファイルから描画に必要なデータを取得し Viewer クラスが持つ変数に格納するための関数およびこの変数の現在の値を調べるための関数である．

SetCamSensor 表示装置に付加されたセンサの位置姿勢を計測する 3 次元位置計測装置を指定する．

SetCameraToRecv 両眼中心位置座標系でのセンサの位置・姿勢を設定する．

GetCameraToRecv 両眼中心位置座標系でのセンサの現在の位置・姿勢を返す.

SetProjectionParam ビューボリュームの設定に必要なパラメータおよび表示装置のキャリブレーションに必要なパラメータを設定する．

GetProjectionParam 上記の関数で設定した値を返す.

SetScreenParam ビューポートの位置と大きさの指定およびステレオタイプの指定を行なう．

GetScreenParam 現在のビューポートの位置と大きさ，ステレオタイプを返す.

SetIOD 両眼間隔を指定する.

GetIOD 現在の両眼間隔を返す.

```
class Viewer
{
 public:
  // func : 描画関数
  Viewer();
  Viewer(void (*world_draw)());
  Viewer(void (*world_draw)(),
         void (*viewer_draw)(),
         void (*camera_draw)());
  Viewer(Object *world_obj);
  Viewer(Object *world_obj,
         Object *viewer_obj,
         Object *camera_obj);
  Viewer(void (*world_draw)(),
         void (*viewer_draw)(),
         void (*camera_draw)(),
         Object *world_obj,
         Object *viewer_obj,
         Object *camera_obj);
  ~Viewer();
  // 描画命令
  virtual void Draw();
  // パラメータ群ゲット&セット
  void LoadConfig(char *filename);
  // type = 0 : FishTank(Mono)
  //      = 1 : Separate
```

```
//       = 2 : Field Sequencial
//       = 3 : FishTank(Stereo)
void SetViewerType(int type);
int  GetViewerType();
// ワールド座標系でのビューワ (コクピット) の位置姿勢
// (ビューワ座標系からワールド座標系への変換行列)
void SetWorldToViewer(Matrix wv);
Matrix GetWorldToViewer();
// ワールド座標系でのカメラの位置姿勢
// (カメラ座標系からワールド座標系への変換行列)
// マウスカーソルとの連携で必要になる可能性あり
// 右目 [0] と左目 [1] 両方返す
void GetWorldToCamera(Matrix wc[2]);
// 両眼の間を返す
Matrix GetWorldToCamera();
void SetCamSensor(Sensor *s);
// 両眼中心位置座標系でのレシーバの位置姿勢をセット
void SetCameraToRecv(Matrix cr);
Matrix GetCameraToRecv();
// Viewer のパラメータ
// param[0] = 画面横 (FishTank) or 測定水平画角 (HMD)
// param[1] = 画面縦 (FishTank) or 縦/横 (HMD)
// param[2] = near;
// param[3] = far;
// calib_param[0] = 画面内側の端から無限遠点注視点までのピクセル数
// calib_param[1] = 無限遠点注視点から可視範囲内側の端までの
//                  ピクセル数
// calib_param[2] = 可視範囲外側の端から無限遠点注視点までの
//                  ピクセル数
```

```
  void SetProjectionParam(float param[4], int calib_param[3]);
  void GetProjectionParam(float param[4], int calib_param[3]);
  // Screen のパラメータ
  // param[0] = ビューポート左端(右)
  // param[1] = ビューポート下端(右)
  // param[2] = ビューポート左端(左)
  // param[3] = ビューポート下端(左)
  // param[4] = ビューポート幅
  // param[5] = ビューポート高さ
  // param[6] = ステレオタイプ
  //                      0 : off
  //                      1 : separate
  //                      2 : field sequencial
  //                      3 : crystal eys
  void SetScreenParam(int param[7]);
  void GetScreenParam(int param[7]);
  void SetIOD(float iod);
  float GetIOD();
  void FullScreenOff(int width, int height);
  void FullScreenOn();
  void SetDrawFunc(void (*f1)(),
                   void (*f2)(),
                   void (*f3)());
  void SetDrawObj(Object *o1,Object *o2,Object *o3);
  friend istream &operator>>(istream &in,  Viewer &v);
  friend ostream &operator<<(ostream &out, Viewer &v);
};
istream &operator>>(istream &in,  Viewer &v);
ostream &operator<<(ostream &out, Viewer &v);
```

A.19　World クラス

照光，ID テーブル，フォントセットなどの仮想空間の描画に必要な各種初期設定，およびマウスやキーボードからのイベント管理を行うクラス．仮想空間内のオブジェクトは全て，シーングラフ上で World オブジェクトの子供として扱われる．World オブジェクトは，シーングラフの根になる唯一のオブジェクトである．仮想空間を描画する際は，World オブジェクトがシーングラフ上の子供に対して描画命令を発行する．したがって，プログラム中でオブジェクトを宣言しても，そのオブジェクトを World オブジェクトの子供にしていない場合，仮想空間内に描画されない．

World クラスが持つデータを以下に述べる．特に断らない限り，2D，3D の両モードで共通に使用される．

camera 視点やビューボリュームなどを設定する Camera クラスのインスタンスへのポインタ．2D モードでのみ必要．

light 光源などを設定する Light クラスのインスタンスへのポインタ．

mouse_button, mouse_state 現在押されているマウスのボタンとその状態．2D モードでのみ必要．

PickObj 現在ピックされているオブジェクト．

offset ピックされているオブジェクトの中心とピックした点の間のオフセット．

floor 床描画のためのフラグ．

axes ワールド座標軸描画のためのフラグ．

mode 現在のモード．

Viewtrace_Obj_List 視点追従制約を持つオブジェクトのリスト．視点追従制約を持つオブジェクトはこのリストに格納される．

また，コンストラクタ，デストラクタ以外に以下の関数を定義している．

Draw ワールド全体の描画をする．World オブジェクトを根とするシーングラフを根から順に各ノードを巡回し，各ノードが持つ Draw 関数を呼び出すことで，仮想空間内のオブジェクトの描画を行なう．

KeyboardFunc キーボードからの入力イベントを処理する．

MouseFunc ピックされたオブジェクトを検出し，そのオブジェクトがピックに対するリアクションを定義している場合，それを実行する．

DrawFloor 床の描画設定をする (描画するか否か).

DrawAxes ワールド座標軸の描画設定をする (描画するか否か).

FullScreen フルスクリーンで表示する (2D モードのみ)

```
class World
{
  public:
    Camera *camera;      // 視点やビューボリュームなどを設定に使用
    Light *light;        // 照光処理に使用
    World(int mode);
    ~World();
    void Draw();                // ワールド全体の描画
    void AddChild(Object*);     // 子供をリストに追加する
    void DrawFloor();           // 床を描画する
    Void DrawAxes();            // ワールド座標軸を描画する
    void FullScreen();          // フルスクリーンモードで描画する
    // カメラの位置を指定する（2D モード）
    void InitCameraPos(float x, float y, float z);
```

```
    void  Size(const ylVector3D);
    void  Size(const float x, const float y, const float z);
    void  Size(const float x);
    void  Rotate(const float rad, const float x,
                 const float y, const float z);
    void  Rotate(const float rad, const ylVector3D);
    void  Move(const float x,
               const float y, const float z);
    void  Move(const ylVector3D);
};
```

B. パラメータファイル

Viewer クラス，Sensor クラスを使用する上で必要となる値を指定するためのパラメータファイルを以下に付載する．なお，“♯” で始まる行はコメント行である．また，数値の単位は特に指定しない限り **cm** である．

```
#
# パラメータファイル
#

# 表示装置に付加したセンサの，ワールド座標系での位置姿勢.
# 位置，姿勢の順に 3 次元ベクトル形式で指定する.
# Sensor クラスで使用する.
wrl_to_vwr
(0,  0,  0), (0,  0,  0)

# 表示装置に付加したセンサから見た画面の位置 (FishTank 型 VR 用)
# Sensor クラスで使用する.
vwr_to_scr
(0,  0,  -60), (0,  0,  0)

# これより以下のパラメータは Viewer クラスで使用する.

# 画面横 (FishTank) または 水平画角測定測定値 (HMD)
# 画面縦 (FishTank) または アスペクト比 (HMD)
# ビューボリュームの手前の面までの距離
# ビューボリュームの奥の面までの距離
view_param
47      0.55    2       300
```

```
# 表示装置のキャリブレーションに必要なデータ
#       ・画面内側の端から無限遠点注視点までのピクセル数
#       ・無限遠点注視点から可視範囲内側の端までのピクセル数
#       ・可視範囲外側の端から無限遠点注視点までのピクセル数
view_calib_param
371     295     242

# ビューポートおよびステレオタイプの設定
#       ・ビューポート左端
#       ・ビューポート下端
#       ・ビューポート幅
#       ・ビューポート高さ
#       ・ステレオタイプ
#           0 : off
#           1 : separate
#           2 : field sequencial
#           3 : crystal eys
scr_param
(0,  0)  (0,  0)   720   486   1

# 瞳孔間距離
cam_IOD
6.8

# カメラ用センサパラメータ
# 使用する 3 次元位置計測装置の種類，計算機側のシリアルポート番号，
# 計測装置側のポート番号，Polhemus 社製センサ用キャリブレーション
# ファイルを指定.
```

```
cam_sensor
Fastrak        2        1        ./3space.dat

# 両眼中心座標系でのセンサの位置姿勢
cam_to_recv
(3.54253, 7.85466, -1.44959), (123.948, 75.9895, -147.172)
```

C.　プログラム例

```
 1: World *world;
 2:
 3: // 3つの神殿を生成
 4: void Make3Shrines(){
 5:      /* 床の生成 */
 6:      Cube *base1 = new Cube(22, 1, 12);
 7:      Cube *base2 = new Cube(20, 1, 10);
 8:      base2->Move(0, 1, 0);
 9:
10:      Object *pillar1 = new Object;
11:      Object *pillar2;  Object *pillar3;  Object *pillar4;
12:      Object *pillar5;  Object *pillar6;  Object *pillar7;
13:      Object *pillar8;  Object *pillar9;  Object *pillar10;
14:      Object *pillar11; Object *pillar12;
15:
16:      /* 一本の柱を生成 */
17:      Cylinder *ppartA = new Cylinder(0.8, 0.8, 8);
18:      Cylinder *ppartB = new Cylinder(0.6, 0.8, 0.5);
19:      Cylinder *ppartC = new Cylinder(0.8, 0.6, 0.5);
20:      ppartA->Rotate(90, 1, 0, 0); ppartB->Rotate(90, 1, 0, 0);
21:      ppartC->Rotate(90, 1, 0, 0);
22:      ppartB->Move(0, 0.5, 0);     ppartC->Move(0, -8, 0);
23:      pillar1->AddChild(ppartA);   pillar1->AddChild(ppartB);
24:      pillar1->AddChild(ppartC);
25:      pillar1->Move(0, -1, 0);
26:
27:      /* 柱をコピー */
28:      pillar2 = pillar1->Copy();   pillar3 = pillar1->Copy();
29:      pillar4 = pillar1->Copy();   pillar5 = pillar1->Copy();
30:      pillar6 = pillar1->Copy();   pillar7 = pillar1->Copy();
31:      pillar8 = pillar1->Copy();   pillar9 = pillar1->Copy();
32:      pillar10 = pillar1->Copy();  pillar11 = pillar1->Copy();
33:      pillar12 = pillar1->Copy();
34:
35:      pillar1->Move(-9, 11, 4);    pillar2->Move(-4.5, 11, 4);
36:      pillar3->Move(0, 11, 4);     pillar4->Move(4.5, 11, 4);
37:      pillar5->Move(9, 11, 4);     pillar6->Move(9, 11, 0);
38:      pillar7->Move(9, 11, -4);    pillar8->Move(4.5, 11, -4);
39:      pillar9->Move(0, 11, -4);    pillar10->Move(-4.5, 11, -4);
40:      pillar11->Move(-9, 11, -4);  pillar12->Move(-9, 11, 0);
41:
42:      Object *ceiling = new Object;
43:      Cube *ceilboard = new Cube(20, 1, 10);
44:      ceilboard->Move(0, 11.5, 0);
45:      Polygon *roof1 = new Polygon(-10, 12, 4, 10, 12, 4,
46:                           0, 15, 4, P_END);
47:      Polygon *roof2 = new Polygon(-10, 12, -4, 10, 12, -4,
48:                           0, 15, -4, P_END);
```

```
49:      Polygon *roof3 = new Polygon(  0, 15, 4, 10, 12, 4,
50:                          10, 12, -4, 0, 15, -4, P_END);
51:      Polygon *roof4 = new Polygon(  0, 15, 4, 0, 15, -4,
52:                          -10, 12, -4, -10, 12, 4, P_END);
53:      Line *frame = new Line( 7, 12, 5, 0, 14, 5,
54:                           -7, 12, 5, L_END);
55:      frame->Move(0, 0, -0.8);
56:      frame->Color(1, 0, 0);  frame->Width(4);
57:      ceiling->AddChild(ceilboard); ceiling->AddChild(roof1);
58:      ceiling->AddChild(roof2);     ceiling->AddChild(roof3);
59:      ceiling->AddChild(roof4);     ceiling->AddChild(frame);
60:      ceiling->Move(0, -0.5, 0);
61:
62:      /* 神殿の生成 */
63:      Object *shrine = new Object;
64:      shrine->AddChild(base1);    shrine->AddChild(base2);
65:      shrine->AddChild(pillar1);  shrine->AddChild(pillar2);
66:      shrine->AddChild(pillar3);  shrine->AddChild(pillar4);
67:      shrine->AddChild(pillar5);  shrine->AddChild(pillar6);
68:      shrine->AddChild(pillar7);  shrine->AddChild(pillar8);
69:      shrine->AddChild(pillar9);  shrine->AddChild(pillar10);
70:      shrine->AddChild(pillar11); shrine->AddChild(pillar12);
71:      shrine->AddChild(ceiling);
72:
73:      Object *shrine1, *shrine2, *shrine3;
74:      shrine1 = shrine->Copy();     shrine2 = shrine->Copy();
75:      shrine1->Move(-20, -3, -15);  shrine2->Move(0, -3, -30);
76:      shrine1->Rotate(90, 0, 1, 0); world->AddChild(shrine2);
77:      world->AddChild(shrine1);
78:
79:      shrine3 = shrine->Copy();
80:      shrine3->Move(20, -3, -15);
81:      shrine3->Rotate(-90, 0, 1, 0);
82:      world->AddChild(shrine3);
83: }
84:
85: // 12 本の木を生成
86: void Make12Trees(){
87:      Object *tree1, *tree2, *tree3, *tree4, *tree5, *tree6,
88:       *tree7, *tree8, *tree9, *tree10, *tree11, *tree12;
89:
90:      tree1 = tree->Copy();      tree2 = tree->Copy();
91:      tree1->Move(7, 0, 0);      tree2->Move(-7, 0, 0);
92:      world->AddChild(tree1);    world->AddChild(tree2);
93:
94:      tree3 = tree->Copy();      tree4 = tree->Copy();
95:      tree3->Move(7, 0, 10);     tree4->Move(-7, 0, 10);
96:      world->AddChild(tree3);    world->AddChild(tree4);
97:
```

```
 98:      tree5 = tree->Copy();       tree6 = tree->Copy();
 99:      tree5->Move(7, 0, 20);      tree6->Move(-7, 0, 20);
100:      world->AddChild(tree5);     world->AddChild(tree6);
101:
102:      tree7 = tree->Copy();       tree8 = tree->Copy();
103:      tree7->Move(7, 0, -10);     tree8->Move(-7, 0, -10);
104:      world->AddChild(tree7);     world->AddChild(tree8);
105:
106:      tree9 = tree->Copy();       tree10 = tree->Copy();
107:      tree9->Move(7, 0, -20);     tree10->Move(-7, 0, -20);
108:      world->AddChild(tree9);     world->AddChild(tree10);
109:
110:      tree11 = tree->Copy();      tree12 = tree->Copy();
111:      tree11->Move(7, 0, 30);     tree12->Move(-7, 0, 30);
112:      world->AddChild(tree11);    world->AddChild(tree12);
113: }
114:
115: // 地面を生成
116: void MakeGround(){
117:      Cube *ground = new Cube(200, 1, 200);
118:      ground->Color(.2, .6, .2);
119:      ground->Move(0, -4, 0);
120:      ground->AddConstraint(STOP);
121:      world->AddChild(ground);
122:
123:      Cube *ground_water = new Cube(200, 1, 200);
124:      ground_water->Move(0, -5, 0);
125:      ground_water->Color(BLUE);
126:      ground_water->OBJVAL1 = 1;
127:      world->AddChild(ground_water);
128:
129:      Cube *roadNS = new Cube(10, 1, 200);
130:      roadNS->Color(.2, .2, .2);
131:      roadNS->Move(0, -3.8, 0);
132:      roadNS->AddConstraint(STOP);
133:      world->AddChild(roadNS);
134: }
135:
136: // 水位を上昇させる関数
137: void WaterRise(Object *ground, Object *scale){
138:   float tmp;
139:   tmp = ((Scale*)scale)->GetValue();
140:   ground->Size(1, tmp, 1);
141: }
142:
143: // シーンを夜にする関数
144: void ChangeNight(Object *spotlight){
145:   glClearColor(0,0,0,0);
146:   world->light->Off();
147:   ((Light*)spotlight)->On();
148:   world->Draw();
149: }
```

```
150:
151: // シーンを昼にする関数
152: void ChangeDaytime(Object *spotlight){
153:   glClearColor(0.3, 0.5, 1.0, 1.0);
154:   world->light->On();
155:   ((Light*)spotlight)->Off();
156:   world->Draw();
157: }
158:
159: void  main(int argc, char **argv)
160: {
161:   InitFunc(argc, argv);
162:
163:   // スポットライトを作成
164:   Light *spot = new Light();
165:   spot->SetPosition(0, 3, -3, 1);
166:   spot->SetSpotDirection(0, -0.2, -1);
167:   spot->SetSpotCutoff(22.5);
168:   spot->Off();
169:
170:   // シーン内のインタフェース部品以外
171:   // のオブジェクトを作成
172:   Object *ground;
173:   ground = MakeGround();
174:   Make12Trees();
175:   Make3Shrines();
176:
177:   // 水位を上昇させるスケーラを作成
178:   Scale *water_scale = new Scale;
179:   water_scale->SetLength(10);
180:   water_scale->Move(-10, 10, 15);
181:   water_scale->String("Water Level");
182:   water_scale->SetRange(1, 20);
183:   water_scale->SetValue(1);
184:   water_scale->
185:      BindFunction(WaterRise, ground, water_scale);
186:   world->AddChild(water_scale);
187:
188:   // シーンを夜にするボタンを作成
189:   PushButton *night_button = new PushButton;
190:   night_button->Move(-8, 6, 20);
191:   night_button->
192:       BindFunction(ChangeNight, (Object*)spot, F_END);
193:   night_button->String("Night", UNDER);
194:   world->AddChild(night_button);
195:
196:   // シーンを昼にするボタンを作成
197:   PushButton *daytime_button = new PushButton;
198:   daytime_button->Move(-4, 6, 20);
199:   daytime_button->
200:       BindFunction(ChangeDaytime, (Object*)spot, F_END);
```

```
201:   daytime_button->String("Daytime", UNDER);
202:   world->AddChild(daytime_button);
203:
204:   // ダイアログボックスを作成
205:   DialogBox *d = new DialogBox;
206:   d->String("This is an example of an application");
207:   d->Rotate(-45, 0, 1, 0);
208:   d->Move(5, 10, 30);
209:   world->AddChild(d);
210:
211:   // カメラの位置の初期設定
212:   world->InitCameraPos(-25, 25, 50);
213:
214:   MainLoop();
215: }
```