オブジェクト指向によるアニメーションデータベースシステムのデータモデルと視覚化機能に関する研究

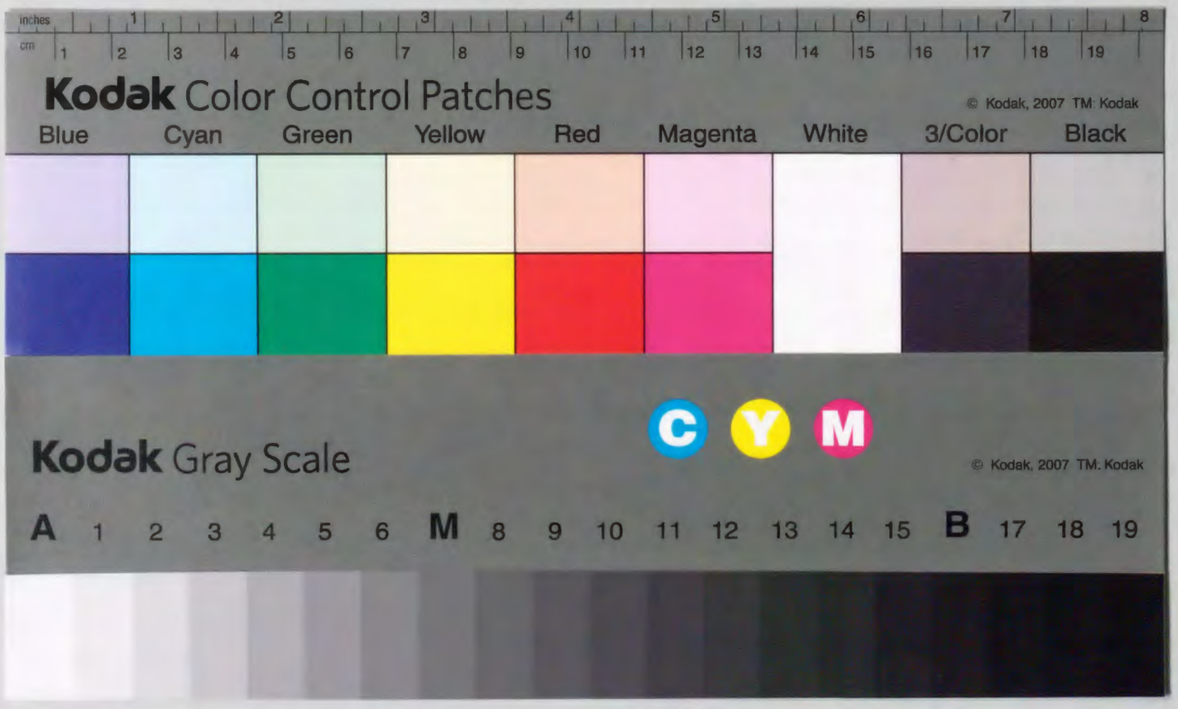
金子, 邦彦
Graduate School of Engineering, Kyushu University

https://doi.org/10.11501/3099879

出版情報：九州大学, 1994, 博士（工学）, 課程博士
バージョン：
権利関係：

オブジェクト指向によるアニメーションデータベース
システムのデータモデルと視覚化機能に関する研究

金子 格



オブジェクト指向によるアニメーションデータベースシステムのデータモデルと視覚化機能に関する研究

金子邦彦

１９９５年１月

目次

第１章

はじめに

コンピュータグラフィックスは，１９６３年にアイヴァン･サザランドが作成したスケッチパッドが発祥とされている．コンピュータグラフィックスを利用した３次元グラフィックスアニメーション[37]の特徴は，３次元の物体の形状と動きを人間の視覚と聴覚に訴えた形で分かりやすく表示できることである．当時は，３次元グラフィックスアニメーションには性能，価格，使いやすさの面で問題があった．しかし，近年はグラフィックスワークステーション上で１０万ポリゴン／秒の性能が実現するなど描画用ハードウエアが進歩した．近い将来には，３次元グラフィックスアニメーションを利用したゲーム機の登場により，３２ビットないし６４ビット処理で１００ＭＩＰＳ，１０万ポリゴン／秒を超える性能を持った描画用コンピュータチップの普及が進むと思われる．

技術の進歩と共に３次元グラフィックスアニメーションはますます広い分野に応用されるようになり，当初利用されていたアート･エンターテイメント分野だけでなく，ＣＡＤ[4]，ＣＡＭ，ＣＡＥ，地図情報システムなどの産業分野，景観シミュレーション，広告，プレゼンテーションなどの商業分野，サイエンティフィックビジュアリゼーション，仮想現実感などの研究･教育分野にも普及した[18]．

以上の背景から，研究領域としての３次元グラフィックスアニメーションの重要性は高い．近年，実世界の表現能力や画像の写実性の向上の観点から，リアルな現象のモデル[15]，３次元形状の表現法[50]，柔軟な物体の形状の表現

法[47]，写実的な描画アルゴリズム[1]などの研究が活発に行なわれている．

以上のように，ハードウエアの性能，価格面の進歩と共にコンピュータグラフィックスの応用分野が広がり，その利用者も増えつつある．ところが，アニメーションを使ったアプリケーションを開発するには，コンピュータグラフィックスの知識が必要である．従って，３次元グラフィックスアニメーションを容易に扱えるようなアニメーション用プラットホームの必要度は高い．現在までに，グラフィックス描画機能を使いやすくまとめたプラットホームが数多く整備されてきた．従来のプラットホームは，アプリケーションから利用可能なライブラリやアニメーション作成用のツールとして機能し，グラフィックス分野に精通していない一般の技術者やプログラマに対してアプリケーション開発，アニメーション作成の支援を行ない，労力の負担の軽減に効果があった．

一方で，描画用ハードウエアの進歩によって，従来では考えられなかったような多量のグラフィックスデータを扱うことが可能となってきた．画像１枚あたりのデータ量を増やすことで，例えば，細部を精密に表現した画像，地平線の彼方までも続くような画像が可能になる．その結果，画像の写実性，迫力が向上するので，今後もデータ量の増加が続くと予想される．さらに，大容量ネットワーク，２次記憶装置の進歩と共に，多量のグラフィックスデータを１つの計算機で管理し複数の利用者に提供することが可能となるようなハードウエア環境の整備が進んでいる．ところが，従来のプラットホームではデータを単なるファイルの集まりとして管理しているため，利用者は多量のグラフィックスデータの検索，共有，再利用をうまく行なうことができない．

多量のグラフィックスデータの検索，共有，再利用の課題に対して，データベースシステムの立場からアニメーション用データ検索システムとしてのアニメーションデータベースシステムの研究[26][39][40]が行なわれてきた．それらの研究では，データベース管理システムのデータ管理メカニズムを利用して，グラフィックスデータの検索，共有，再利用を行なっている．

従来のアニメーションデータベースでは，描画対象の形状とその動きのデータベース化を行なっていた．ところが，従来のアニメーションデータベースでは，リアルな絵の描画を行なうために必要な実体の表面属性，テクスチャ，光源，音源などのデータが扱われていなかった．さらに，グラフィックスデータ，イメージ，グラフィックスアルゴリズムの関連情報も扱われていなかった．さらに，従来のデータベースでは，データベース部分と描画部分とが独立した構造になっており，システムはデータベース部分からグラフィックスデータを検索し描画部に渡すことで描画を行なっていたため，メモリサイズよりも大きなデータの描画が難しい．以上のように，従来のアニメーションデータベースでは，描画対象の形状とその動きのデータベースに限られていたことから，（1）リアルな絵の生成，（2）グラフィックスとイメージに関する同期，（3）巨大データの効率の良い描画，の３つの機能を持っていない．従って，従来のアニメーションデータベースは，アニメーション用プラットホームと比べて機能面で劣っており，実際に利用するには機能的に不十分であった．

また，アプリケーションの種類によって必要な実体の種類，実体の表現法，描画に用いるべきグラフィックスアルゴリズム，動きの表現法に関するモデルが変わる．アプリケーションが必要とするすべてモデルをシステムの構築以前に予測することは不可能である．従って，従来のアニメーションデータベースシステムでは，個々のモデルに対応してデータベーススキーマが定義される．一度作成されたデータは，長い期間に渡って保存されるため，モデルの種類ごとに複数のデータベースが存在する．しかし，異なる種類のデータを必要とする複数のアプリケーションのプラットホームとしてアニメーションデータベースを利用するという観点にはあまり重点がおかれていなかったため，複数のモデルが存在するようなアニメーションデータベースの研究はなかった．

本研究では，以上のアニメーションデータベースの課題に対して，新しいアニメーションデータベースのプロトタイプとしてのＭＯＶＥを提案する．ＭＯＶＥではグラフィックスに関する複数のモデルを扱えることが特徴である．

複数のモデルの登録を可能とするために，ＭＯＶＥはオブジェクト指向データベースシステム上に構築され，３次元コンピュータアニメーションの基

本モデルが定義されている．この基本モデルは，グラフィックスデータとイメージとグラフィックスアルゴリズムの統合，同期，動きのモデル，巨大データの描画の機能を持つように設計され，従来のアニメーションデータベースにおける機能面の不十分さを補っている．

本研究では，ＭＯＶＥの基本モデルの検証を目的として，これをオブジェクト指向データベースシステムＯＮＴＯＳ上に実装した．本論文では，次の手順で．ＭＯＶＥのアニメーション用プラットホームとしての機能を検証する．

２章では，従来のアニメーション用プラットホームとアニメーション用データベースの現状をまとめている．従来のプラットホームは，描画対象であるシーンの記述方式の観点から１）シーン記述言語，２）グラフィックスデータ型，３）アニメーションデータベースの３種類に分類できる．それぞれにおけるグラフィックスデータの保存法，描画法をまとめるとともに，従来のプラットホームがデータをファイルで管理しているためにデータ共有･検索において問題が発生する原因を示す．次に，多量データ共有，検索を可能とするグラフィックスサーバの機能を持ったアニメーションデータベースの研究をまとめ，従来のプラットホームと比べて機能的に劣る部分を指摘し，アニメーションデータベース固有の問題を挙げる．

３章では，ＭＯＶＥのシステム構成と，そのオブジェクト指向データベース上の実現について説明する．最初に，アプリケーションが必要とするモデルを登録可能とすることによって得られる利点を示す．次に，オブジェクト指向を用いた理由を説明する．そして，ＭＯＶＥと従来のアニメーション用プラットホームの機能を比較する．

４章では，ＭＯＶＥの基本モデルについて説明する．３次元グラフィックスアニメーションでは，グラフィックスデータ，イメージ，グラフィックスアルゴリズムの３つを扱う必要がある．ＭＯＶＥの基本モデルはこれらグラフィックスデータ，イメージ，グラフィックスアルゴリズムが統合されている．まず，グラフィックス分野における３つの同期問題を示し，ＭＯＶＥの基本モデルを用いた解決法を説明する．次に，ＭＯＶＥの基本モデルを用いてアプリケーションに固有なモデルを登録可能であることを説明する．

５章では，ＭＯＶＥでの動きのオブジェクト表現を説明する．ＭＯＶＥでは，系という論理単位で動きの格納と描画を行なう．まず，ＭＯＶＥの系オブジェクトに関して，動きのモデルの定義を行なう．次に，ＭＯＶＥにおける３次元の物体とその動きの描画機能を示す．

６章では，リアルなアニメーションを容易に行なうためのグラフィックスアルゴリズムとグラフィックスデータの統合の必要性について説明し，代表的なグラフィックスアルゴリズムであるポリゴンメッシュアルゴリズムと，立体音響アルゴリズムとグラフィックスデータの統合を実装によって検証した．

７章では，基本モデルの有効性をプロトタイプの試作によって検証する．まず，プロトタイプの題材として用いた古典力学のシミュレーションの重要性を説明する．次に，古典力学の剛体系のシミュレーションのデータベースを構築することで，実世界と実体に関する３次元グラフィックスと動きをオブジェクトとして表現し，ＭＯＶＥ上のデータベースに格納可能であることを示す．又，力学シミュレーションの力学方程式を解くための力学ソルバを説明する．

８章では，ＭＯＶＥの動きの描画機能を説明し，アニメーション描画の効率化のためのインデックスの有効性を示す．実際にインデックスを実装して，多量データの描画法の検証を行なった．インデックスを用いなければ扱えないような多量データを必要とする例として，九州大学箱崎地区のウオークスルーアニメーションを取り上げ，その作成と性能評価について述べる．

第２章

アニメーション用プラットホームの課題

2.1 アニメーション用プラットホーム

３次元グラフィックスアニメーション[18]では，実世界の実体に関するデータは計算機内にモデル化され，コンピュータグラフィックスと立体音響で視覚化される．実体の形状は３次元的な空間属性として表現され，実体の動きは時間とともに変化する属性として表現される．これら実体の形状，動きやその他の属性を変えることで異なったアニメーションを得られるという点が，実体に関する情報を持たない場合のアニメーションとの違いである．

従来，３次元コンピュータアニメーションにおける実体の表現法と描画法に関して，リアルな現象のモデル[15]，３次元形状の表現法[50]，自然界の物体の形状の表現法[47]，リアルな描画法や高速な描画法[1]の観点からの研究が行なわれてきた．これらの研究の結果，リアルに変形する３次元物体のモデル化が可能となった．また，リアルな描画法に関しては，レイトレーシング法，ラジオシティ法，ポリゴンシェーディング，テクスチャマップなどの描画アルゴリズムが開発され，それらを利用することでフォトリアリスティックな画像が得られるようになった．高速な描画法に関しては，グラフィックスワークステーション上で１０万ポリゴン／秒の性能が実現するなど描画用ハードウエアが進歩した．近い将来には，３次元グラフィックスアニメーションを利用したゲーム機の登場により，３２ビットないし６４ビット処理で１００ＭＩＰＳ，１０万ポリゴン／秒を超える性能を持った描画用コンピュータ

チップの普及が進むと思われる．

以上のように，３次元グラフィックスアニメーションのためのグラフィックスハードウエアの普及，研究の進歩と共に，応用分野や利用者が増えつつある．ところが，グラフィックスハードウエアを直接操作してアニメーションを得るにはコンピュータグラフィックスに関する専門的な知識が必要である．従って，グラフィックスハードウエアへのインタフェースとして抽象度の高いグラフィックス描画機能をまとめることで，３次元グラフィックスアニメーションを容易に扱えるようなアニメーション用プラットホームが数多く整備されてきた．

アニメーション用プラットホームを利用したアニメーションの作成は，描画対象となる３次元的な仮想世界を表現するシーンを計算機上に記述し，１シーンのデータをプラットホームの描画部に渡すことで行なわれる．従来から，シーン記述法とアニメーション用プラットホームが多数開発されてきた．従来のシーン記述法は大きく分けて次の３種類，（1）独自言語，（2）プログラミング言語の拡張，（3）グラフィックスデータ型，フォーマットに分類できる．

第１の独自言語は，シーン記述に専用の言語を用いる方法であり，MIRA-3DやRayshadeなどが知られている．

第２のプログラミング言語の拡張による方法は，コンピュータグラフィックスの機能を汎用的なプログラミング言語から利用できるようにしたツールキットの実装である．ツールキットの機能は，（1）コンピュータグラフィックス描画ライブラリ，（2）マウスなどのグラフィカルインタフェースによるデータ生成ツールの２つである．プログラミング言語cのライブラリや，Ｃ＋＋[6]のクラス（型）として，Open-GL[43]，IRIS-GL，PEXlib[52]，IRIS Inventor，IRIS Performer，RenderManなどが実用化されている．

第３のグラフィックスデータ型，フォーマットは，グラフィックスの形状と動きのデータの表現法を統一する方法である．特定のプログラミング言語に依存しないようなグラフィックスデータの型やファイルフォーマットが標準化されている．このようなグラフィックスデータ型についてはいくつかの

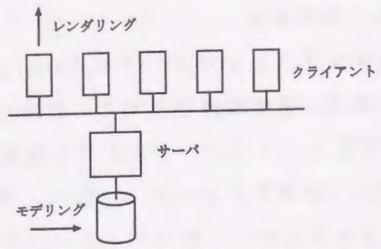


図 2.1: グラフィックスサーバの構造

研究が行なわれ[8][32][35][13][48][51]，ＰＨＩＧＳ，ＧＫＳ，ＨＯＯＰＳなど実用化もされている．また，ファイルのフォーマットとしては，ＣＧＭ[17]，ＡｕｔｏＣＡＤ　ＤＷＦ，ＤＸＦ，Ｗａｖｅｆｒｏｎｔ　ＯＢＪなど多数が実用化されている．この方式では，シーン記述のデータ形式が定まっているので，異なったアプリケーションでのデータ交換が可能である．

グラフィックスハードウエアを直接操作するよりもシーン記述を行なう方が分かりやすく使いやすい．アニメーション用プラットホームは，シーン記述の内容を描画するライブラリやツールとして機能する．プラットホームはアニメーション作成，アプリケーション開発の負担の軽減，複数のグラフィックスハードウエアへのアプリケーションの移植性の向上に有効であった．その結果，プラットホームを利用することで，グラフィックス分野に精通していない一般の技術者を含めた多くの利用者やプログラマの労力の負担の軽減に効果があった．

2.2 アニメーション用プラットホームの課題

描画用ハードウエアの進歩によって，従来では考えられなかったような多量のグラフィックスデータを扱うことが可能となった．データ量を増やすことで，細部を精密に表現した画像，地平線の彼方までも続くような画像が可能になるなど，画像の写実性，迫力が向上することから，今後もデータ量の増加が続くと予想される．さらに，大容量ネットワーク，２次記憶装置の進

歩と共に，図2.1のように多量のグラフィックスデータをサーバで管理し複数のクライアントに提供が可能なハードウエア環境の整備が進んでいる．

グラフィックスサーバの利用によって，アニメーションの作成，データの可視化だけでなく，次のような用途が可能となる．

i) アニメーションの作成支援:
過去に作成したアニメーションのグラフィックスデータを，基本データの集まりとして保存する．後で，必要な部分を検索･複写して，別の作品に利用する．そのことで，コンピュータアニメーション作成の手間を軽減する．

ii) 動画像，音声のプレゼンテーション:
アニメーションは動画像と音声という２種類の要素の組合せとして保存される．これら要素としての動画像と音声には，表示のタイミングを属性として指定可能である．

iii) 空間的実体のデータベース:
景観シミュレーションなどのアプリケーションでは，山，海，平野，建築物，道路などの３次元の実体のデータベースが，ネットワークを介して複数の利用者に提供される．
実体は，グラフィックスデータと，文字列，数値などのデータの対として保存される．こうして保存された文字列，数値などのデータによって，単に計算された動画像や音声だけでなく，テロップなどと組み合わされたようなアニメーションが得られる．

iv) 実体の動きのデータベース:
３次元物体の形状や位置の時間的な変化もデータベース化されているので，再利用できる．ある特定の動作をいろいろな物体にさせたい場合に用いる．

v) シミュレーションとビジュアライゼーション:
工学設計の正しさを確認するために，製品をデータベース上にモデル化し，動作をアニメーション表示する．製品のモデルには，動きのシミュレーションモデルも含めることができる．

製品の持つパラメータを変えて，製品の動きをいろいろと変えることができる．また，動きの法則は，何らかの言語で記述することができる．時間的変化を生じる現象は，アニメーションにすることにより容易に把握することができる

グラフィックスサーバの実現において，（1）同時実行制御，（2）効率の良いデータ検索，（3）異種データの統合などの課題がある．

第１の課題として，複数の利用者がグラフィックスサーバ上のデータを共有することから，複数の利用者が同時にアクセスした場合のアクセス制御（同時実行制御）を行なう必要がある．従来のプラットホームにおいて，シーン記述の内容はファイルとして保存されている．ところが，同時にファイルへの書き込みを行なう場合の同時実行制御はオペレーティングシステムではなくアプリケーション側で行なわなければならない．従って，アプリケーションの開発が面倒である．

第２の課題として，シーン記述の内容が独自言語やプログラミング言語やファイルフォーマットの文法に従って記述されていることにより，利用者がデータの検索，再利用を行なう場合において，シーン記述文法の知識のある利用者でなければシーン記述の内容を理解できないという課題がある．従って，ある条件に合致するシーン記述の検索を計算機に行なわせるための条件の記述は一般の利用者にとっては難しい．個々のファイルは構造を持たない文字列であるので，大きなサイズを持ったシーンへの検索を効率良く行なうことも難しい．

アニメーションのシーンに登場する実体のデータを一般の数値･文字などのデータと比較すると，実体の種類に多様性があるという特徴がある．例えば，地理，地形，都市情報，ＣＡＤ，景観シミュレーションなど，アプリケーションの対象領域が変わると，登場物の種類も，等高線，道路，海，山，平野，建築物，道路などと変わる．そのためにアプリケーションによって異なる実体が必要である．

しかも，実体の表現形式はアプリケーションによって変わることが多い．例えば，６辺がx，y，z軸に平行な直方体の形状の表現形式は，（1）２頂点を

覚える．（2）中心点と，縦，横，高さを覚えるなどの表現が考えらる．等速直線運動の表現形式は，（1）時刻＝０における位置と速度を覚える．（2）ある２時刻における位置を覚えるなどの表現が考えられる．いかなる表現形式が最適かはアプリケーションの種類によって変わる．以上のように，実体の最適な表現形式はアプリケーションによって変わるから，実体の表現を予め定めておくことは不可能である．さらに，アプリケーションの種類，目的によって，描画に用いるべきグラフィックスアルゴリズムも変わることが多い．

以上のように，アプリケーションの種類によって必要な実体の種類，実体の表現法，描画に用いるべきグラフィックスアルゴリズム，動きの表現法に関するモデルが変わる．その結果，従来から，数多くのプラットホームが用意され，利用者の種類，アプリケーションの目的，アプリケーションの機能，アプリケーションの対象領域に応じてプラットホームが選択されてきた．モデルごとにプラットホームを選択することは，従来のアプリケーションでは有効であった．しかし，グラフィックスサーバでは（1）検索の難しさ，（2）描画の難しさの２つの問題が発生する．

アプリケーションデータはファイルとして表現されているが，モデルの種類ごとにファイルのフォーマットが変わる．従って，検索時にはデータの種類を意識する必要があり，検索が難しい．

確かに，従来の多くのアニメーション作成用アプリケーションは，複数のグラフィックスフォーマットを扱えるように実装されているので，描画にグラフィックスフォーマットを用いることが有効であるように思える．しかし，描画用のファイルフォーマットとアプリケーションデータのフォーマットとは一致しないので，アプリケーションごとに，描画用フォーマットへの変換プログラムを開発する必要がある．従って，グラフィックスフォーマットとプログラミングに関する専門的知識を必要とする．他のグラフィックスライブラリなどを用いる方法でも，アプリケーションごとにインタフェース部分を開発する必要があるため，プログラミングの知識が必要となる．

2.3 アニメーションデータベース

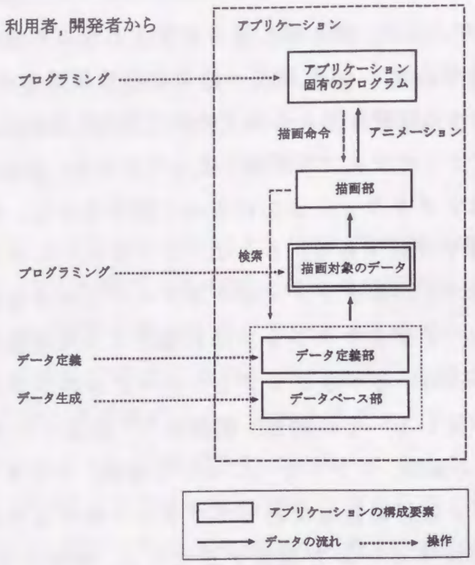


図 2.2: アニメーションデータベースとアプリケーション

従来のプラットホーム上へのグラフィックスサーバの構築について，シーンをファイルとして保持していることによるいくつかの問題があった．そこで，データベースシステムの立場からアニメーション用データの検索システムとしてのアニメーションデータベースシステムANIMENGINE[26]，Animation Platform[39][40]の研究，Kemper，Wallrathのサーベイなどが行なわれてきた．それらの研究では，データベース管理システムのデータ管理メカニズムを利用して，大きなサイズのデータの検索，共有，再利用を行なっている．

従来のアニメーションデータベースでは，リアルな絵の描画を行なうために必要な実体の表面属性，テクスチャ，光源，音源などのデータが扱われていなかった．さらに，グラフィックスデータ，イメージ，グラフィックスアルゴリズムの関連情報も扱われていなかった．以上のように，従来のアニメーションデータベースでは，データベース部分のデータの種類が形状と位置に限られていたことから，（1）リアルな絵の描画，（2）グラフィックスとイ

メージに関する同期，（3）巨大データの効率の良い描画，の3つの機能を持っていない．従って，従来のプラットホームと比べて機能面で劣っており，実際に利用するには機能的に不十分であった．

従来のアニメーションデータベースでは，データベース部分と描画部分とが独立した構造になっており，データベースからグラフィックスデータを検索し描画部に渡すことで描画を行なっていた．

一般に，アプリケーションは描画部，アプリケーション固有のデータ定義，アプリケーション固有のプログラミング，アプリケーションデータの4つの部分から図2.3のように構成される．アプリケーションの開発とアニメーションの作成は，これら4つの部分に対応して，データの生成，データの登録，描画及びアプリケーションのプログラミングによって行なう．

従来，複数のモデルが存在するようなアニメーションデータベースの研究には焦点があてられていなかった．従来のアニメーションデータベースシステムでは，利用者は個々のモデルに対応してデータベーススキーマの定義を行なう．一度作成されたデータは，長い期間に渡って保存されるため，モデルの種類ごとに複数のデータベースが存在する．

従来のアニメーションデータベースシステムでは，複数のモデルを扱うには，スキーマ定義を行なうだけでなく，データベースごとに描画部とのインタフェースが必要となる．従って，複数のモデルを持つデータの扱いには，労力と，コンピュータグラフィックスの知識が要求される．この労力を減らすには，複数のモデルから利用可能な描画部とそのインタフェースが必要である．

第 3 章

アニメーションデータベースMOVE

3.1 システム構成

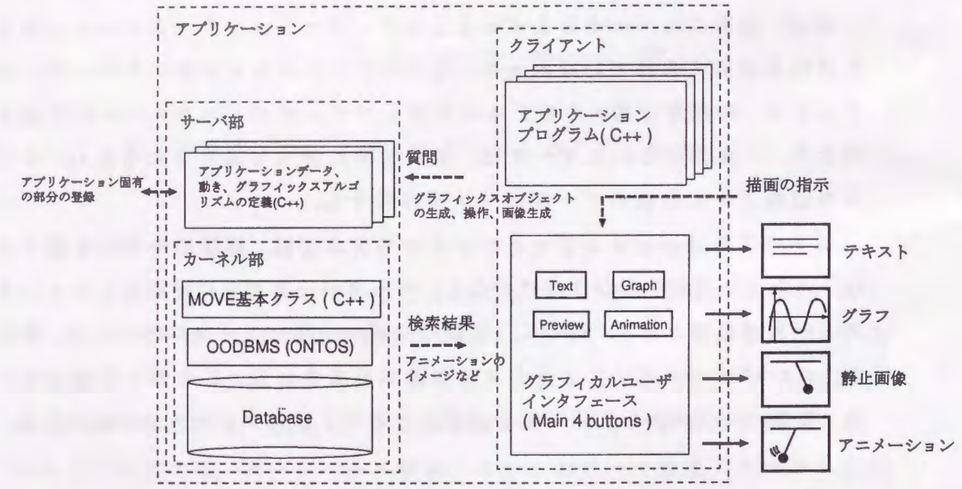


図 3.1: ＭＯＶＥの構造

ＭＯＶＥは，コンピュータグラフィックスのデータとアルゴリズムのデータベースである．ＭＯＶＥは，３次元コンピュータアニメーションを用いるアプリケーションプログラムの開発並びにアニメーション作成と表示を容易にするためのプラットホームとして試作された．ＭＯＶＥを用いることで，アプリケーションは，ＭＯＶＥのデータベースの内容を映像･音声のアニメー

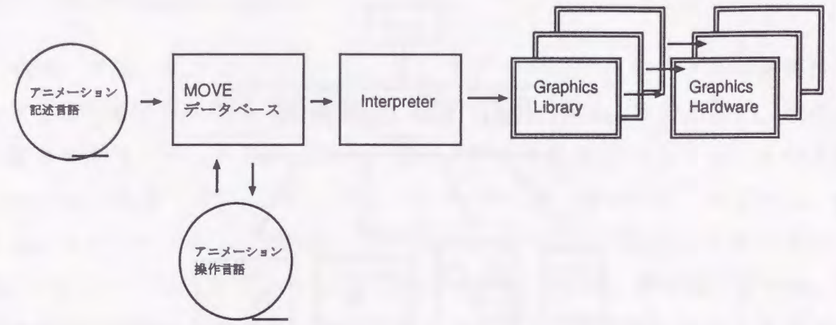


図 3.2: ＭＯＶＥの構造

ションで容易に得ることができる．ＭＯＶＥは，データベース部，データ定義部，描画部の３つが共有可能である．ＭＯＶＥと従来のアニメーションデータベースの相違点は，アプリケーション固有のモデルに対して描画部の共有を行なっている点と，動的な実体の登録が可能となっている点である．

ＭＯＶＥは，データベースと描画部が分離していて，図3.1のように”コア”，”サーバ”，”アプリケーション”の３つのレイヤに分かれている．コアは，ＭＯＶＥの基本部分である．コアには，ＭＯＶＥのデータベースが存在する．アプリケーションプログラムは，ＭＯＶＥのデータベースにすべてのデータを蓄える．そのことによって，複数のアプリケーションプログラムがデータを共有，再利用することが可能である．データベースには，アプリケーションプログラムが利用する３次元グラフィックスアニメーションのデータも存在する．コアには，リアルなアニメーションを描画するためのコンピュータグラフィックスのアルゴリズムが実装されていて，アプリケーションプログラムは，ＭＯＶＥを用いてアニメーションを容易に作成することができる．

描画部もアプリケーションから共有可能なモジュールである．描画部はアプリケーションに対して次のようなインタフェースを持つ．（1）アプリケーションプログラムが描画部に描画を命令する．（2）データベース内のデータから描画に必要なデータが検索され，検索結果は描画部に渡される．

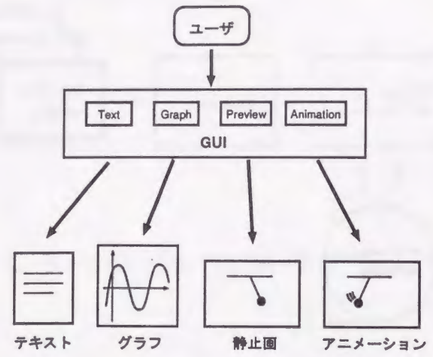


図 3.3: ＭＯＶＥの機能の概念図

（3）描画部は，この検索結果から実装されているコンピュータグラフィックスのアルゴリズムを使用してアニメーションを生成し，アプリケーションプログラムへ渡す．

アニメーションの画像生成には，時間がかかるから，描画部は，画質と速度が異なる複数の描画部が存在し，プレビュー，完成など局面ごとに異なった描画法を選択することを可能としている．

ＭＯＶＥでは，以下の原理により，データ定義部，データベース部の共有を行なっている．ＭＯＶＥは，クラス定義の内容を，データベースのスキーマとして管理している．

ＭＯＶＥには，コンピュータの初心者が簡単にＭＯＶＥのデータを視覚化できるような，単純なグラフィカルユーザインタフェースを持っている．グラフィカルユーザインタフェースには，図に示したように，ＭＯＶＥでは，利用者がデータベース内のデータを検索して，アニメーションなどいろいろな形式で表示するという機能がある．

3.2 オブジェクト指向

従来，画像，音声などのマルチメディアデータのデータベースの重要性をさまざま研究者が指摘してきた[7][57]．ＭＯＶＥも，画像，音声の格納，表示が可能なマルチメディアシステムの一種である．３次元グラフィックスのアニメーションには，（1）イメージ，（2）モデルの２種類のデータがある．第１のイメージデータは，動画像，音声のデータである．第２のモデルデータは，アニメーション化すべき実体に関するデータである．動画像，音声は，登場物のデータからコンピュータグラフィックスと立体音響のアルゴリズムを用いて自動的に生成される．３次元グラフィックスアニメーションの実体は，３次元物体，カメラ，光源，マイクロホン，音源の５種類に分類できる．以上のことから，ＭＯＶＥで扱うアニメーションデータは次の２種類に分類される．それらは（1）動画像･音声のイメージデータ，（2）３次元グラフィックスと動きに関するデータである．

ＭＯＶＥの構築では，これら２種類のデータのそれぞれのデータ形式の設計を行なう必要がある．これらのデータはより単純なデータの集まりとして表現される．例えば，画像は画素の２次元の配列として，音声はサンプルの１次元の配列として，３次元の形状はポリゴン[18]のような基本形状の集まりとして定義することができる．しかし，これらのデータを数値，文字列などの従来の事務データと比較する次のような４つの特徴がある．

- プレゼンテーションのモデル
  アニメーション作成を初心者でも容易に行なうには，計算機のウインドやスピーカーとなどの入出力装置とグラフィックスデータ間の関連の情報の管理が必要である．
- 動きのモデル
  アニメーションの登場物は動的世界を表現するから，アニメーションの登場物の属性は時間の経過に従って変化するような属性である．例えば，動きや変形は空間属性の時間変化として表現できる．
  アニメーションにおけるグラフィックスデータは，時間的に属性が変化す

る．グラフィックスデータの属性の時間変化のモデル化が必要である．そのことで，容易にシミュレーションモデルを書けるようにするための仕組を用意する．さらに，描画を自然に行なえるようにする．
動きの定義では，シミューレーションを用いることで，自然な実体の動きを得ることができる．

- グラフィックスのモデル
  - 空間属性:

    物体の空間的構造は，３次元形状を持った要素の集まりとして表現される．物体の３次元形状は，３次元的な空間属性(x,y,z)をもとに表現される．実際には，３次元形状データの操作の容易さの観点から，３次元形状の空間属性の表現形式は，有限個のパラメータとそれらの形状との関係として表現されることが多い．

  - グラフィックスに固有の属性:

    リアルなアニメーションは光学と音響学を利用することで生成できる．従って，物体表面の反射率，カメラの構成であるレンズの特性，マイクロホンの感度の特性，光源や音源の性質などにおける光線の反射，光線の透過，音波の反射，音波の吸収などの光線と音波に関する現象についてモデル化が必要である．これらの現象のモデルは関数として自然に表現できる．従って，アニメーションの登場物は，これらの現象を表現した関数を属性として持つ必要がある．

- データベース構造
  - データの巨大化:

    マルチメディアデータの大きさは一般には数メガバイトから数十メガバイト程度になることが多い．例えば，ビデオのデータについては，６０分の長さのNTSCの品質を持つ動画像とCDの品質を持つ音声の場合，フレーム枚数が１８００枚，１フレームの解像度が６４０×４８０，１画素が0.5バイト[57]，１秒当たり４４１００サンプル，１サンプルあたり４バイトとすると，動画像は３００ＭＢ，音声

は１０ＭＢである．
一方，グラフィックスデータも，対象世界が広がるにつれて，データの量が多くなる．画像１枚あたりのデータ量を増やすことで，例えば，細部を精密に表現した画像，地平線の彼方までも続くような画像が可能になる．その結果，画像の写実性，迫力が向上するので，今後もデータ量の増加が続くと予想される．

  - データ構造の複雑さ:

    アニメーションの登場物が持っている属性を計算機上に表現する場合，データ構造が複雑になることが多い．例えば３次元形状の表現の場合，空間構造と表面特性に関する属性がないとアニメーション化した結果の見栄えがよくない．３次元形状をポリゴンなどの基本的な描画単位を葉として持つようなツリーとして表現することで，空間構造と表面特性に関する属性を自然に表現することができる．

  - データの管理単位:

    従来のアニメーションシステムは，アニメーションを単位として，ファイルなどの形で管理されている．アニメーションがグラフィックスデータの集合として，グラフィックスデータ単位での検索や，動きのモデルの定義，グラフィックスデータへの付加的属性の追加が難しい．

従来，多くの研究者は，オブジェクト指向データベース上[66][46]を用いることで，レコードやフィールドやタップルよりも，マルチメディアデータを自然に表現でき，効率良く処理できると指摘していた[23]．さらに，ＯＮＴＯＳ，Ｖｅｒｓａｎｔ，ＯｂｊｅｃｔＳｔｏｒｅなど多くのオブジェクト指向データベースが普及し，オブジェクト指向の観点からマルチメディアデータベースに関していろいろな研究が行なわれてきた．アニメーションデータを自然に表現したいという理由から，ＭＯＶＥもオブジェクト指向データベース上に実装した．
オブジェクト指向の概念とは，在来の抽象データ型の概念に加えて，継承

の概念を取り入れたものである．抽象データ型においては，特定の種類の対象物全体が，クラスとして，一括化される．その上で，個々の対象物は，それぞれ固有のクラスに属するインスタンスとして構成され，クラスに応じた属性を持つことになる．継承とは，上位クラスの性質は，下位クラスに引き継がれるというものであり，クラス全体を階層的に構成することができる．

オブジェクト指向の概念は，複雑なシステムのモデル化に優れている．特定の対象モデルは，１つのインスタンスとなる．共通な性質をもつモデルは，全体としてクラスを構成する．あらかじめクラスを定義しておけば，属性値を与えることで，新たなモデルが生成できる．モデルの持つ part-of の構造については，クラス間の関係として表現される．また，is-a 関係の表現は，クラス間の継承機能として実現されている．

オブジェクト指向データベースでは，すべてのオブジェクトに固有のオブジェクト識別子(ＯＩＤ)が定義されていることから，複合オブジェクトの表現にはＯＩＤを用いる．従って，ＭＯＶＥでは，アニメーションは，互いにＯＩＤを介して繋がった複合オブジェクトとして表現されている．例えば，シーンとそのシーンに登場する実体との関係は複合オブジェクトとして表現されている．

ＯＮＴＯＳの検索機能には，ＯＩＤを指定してオブジェクトを検索するという巡航操作と，ＳＱＬによる検索の２種類がある．ＯＮＴＯＳにおける巡航操作では，指定されたＯＩＤを持つオブジェクトがメモリ上にあれば，そのアドレスを返し，メモリ上になければ，ディスクからメモリ上に転送して，アドレスを返す．描画時における検索は，オブジェクトのＯＩＤを出発点として巡行操作を繰り返すことで行なわれる．

後述するように，アニメーション分野では，データベース上に表現されたオブジェクト間の関係データを用いて，あるオブジェクトに関係するオブジェクトの検索が多用される．これらの関係は，ＯＯＤＢＭＳのリファランスを用いて表現されている．ＯＯＤＢＭＳでは，リファランスからオブジェクトを得る検索が高速である（前後との関係がない）．我々は，ＯＯＤＢＭＳに数十のクラスを定義することでＭＯＶＥを実装した．アプリケーションは，

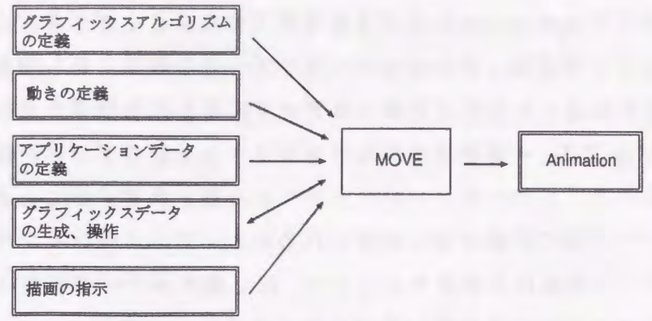


図 3.4: ＭＯＶＥにおけるアニメーション記述とアニメーション操作

さらにいくつかのクラス定義を行なうことで実装される．オブジェクト間の関連はリファランスで表現する．

3.3 ＭＯＶＥの機能

ＭＯＶＥは，（1）ＭＯＶＥの基本クラスをベースとして，アプリケーション固有のデータを定義する機能（アニメーション記述），（2）データベース内のデータを容易に操作，描画できるような機能（アニメーション操作）という２つの機能に特徴がある．この２つの機能を自然に実装できるようにするために，ＭＯＶＥシステムの構造は，個別のアプリケーション，ＭＯＶＥの基本クラスとアプリケーション用クラス，データベースシステムという3層構造として設計された．従って，すべてのアプリケーションプログラムは，コアと利用者定義クラスを共有することで，データとデータ定義の共有が可能である．

ＭＯＶＥでは，データベース上の実体オブジェクトは，形状や位置を表す，基本的なオブジェクトの集まりとして表現されている．この実体オブジェクトの構造は複合オブジェクトとして表現され，データベース中では各要素がリンクで繋がる構造となっている．グラフィックスデータは，論理的に意味のある単位に分割されて保存されている．そのことで，ＭＯＶＥでは，データの共有･再利用が物体，カメラ，光源，マイクロホン，音源という３次元グラ

フィックスアニメーションの実体の単位で行なうことが可能である．ＭＯＶＥのデータの単位は，ファイルベースのデータの単位よりも抽象度が高いので，利用者は大きなサイズを持ったデータに対しても検索を行ないやすい．

ＭＯＶＥでは，一度作成されたアプリケーションのデータが保存されることと，他のアプリケーションがデータベース部を共有することとによって，データベース生成の繰り返しが省かれるから，アニメーション作成が容易になる．データ定義部を共有することで，同じ種類のデータを扱う複数のアプリケーションがデータ定義の内容を共有することが可能となる．その結果，アプリケーションは一度行なったデータ定義の内容を再利用できるので，アプリケーションの作成を容易にする．

ＭＯＶＥでは，アプリケーション固有のモデルの登録，データ定義部と描画部の共有の２つを達成するために，オブジェクト指向データベース管理システム上[46][23]のデータベースシステムとして開発した．

ＭＯＶＥの利用者は，アプリケーションが必要とするモデルの登録を，新たなクラスの実装として行なう．すべての種類の実体オブジェクトに対するＭＯＶＥの基本クラスはEntityである．基本クラスには，描画部と連係した描画用メソッドとしてグラフィクスに関する基本機能が実装されている．従って，新たなアプリケーション用クラスを，ＭＯＶＥの基本クラスのサブクラスとして実装することで，この基本クラスのメソッドがアプリケーションクラスに継承される．以上のように，継承を用いることで描画部の共有を行なっている．ＭＯＶＥでのデータ定義はＣ＋＋[6]で行なわれる．データ定義の内容は，データベースのスキーマとしてＤＢＭＳにより管理される．

ＭＯＶＥ上のアプリケーションのデータはすべてＭＯＶＥの基本クラスの派生クラスのオブジェクトとしてデータベースに格納される．そのことで，実体を単位としたデータの共有，再利用が可能となる．さらに，個々の応用に必要な実体の定義が基本クラスの派生クラスの定義として可能となることから応用の開発が容易になると期待できる．

ＭＯＶＥを従来のアニメーション用プラットホームと比較すると，ＭＯＶＥはデータだけでなくデータ定義も管理し，しかも，グラフィックスデータ

の動的な性質も定義できるので，アプリケーションの量が少なくて済むという特徴がある．従って，描画部がファイルやグラフィックスライブラリを用いて実装されている場合と比較して，データ定義が容易になる．さらに，ＭＯＶＥでは，データ定義はＭＯＶＥのサーバ部に保存され，複数アプリケーションで共有が可能となっている．

3.4 ＭＯＶＥの実装

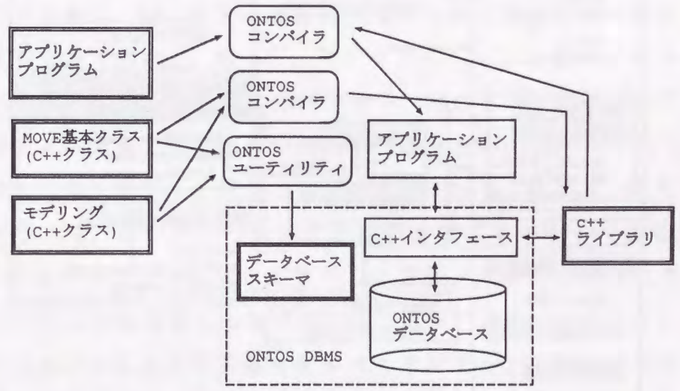


図3.5: ＯＮＴＯＳによるＭＯＶＥの実装

図3.5には，ＭＯＶＥとＯＮＴＯＳの関係を示している．ＭＯＶＥでは，アニメーションの基本モデルはＣ＋＋[6]のクラスとして実装されている．これらのクラスのＣ＋＋プログラムは，ＯＮＴＯＳのユーティリティで処理され，ＯＮＴＯＳのスキーマとして管理される．

Ｃ＋＋は，他のプログラミング言語と比べて，次のような特徴がある．

i) Ｃ＋＋はオブジェクト指向言語である．
ii) Ｃ＋＋はＣの拡張であるので，Ｃの知識を持ったプログラマにとって扱い易い．
iii) Ｃ＋＋はコンパイル言語なので，smalltalk-80などのインタプリタ言語よりも実行効率が良い．

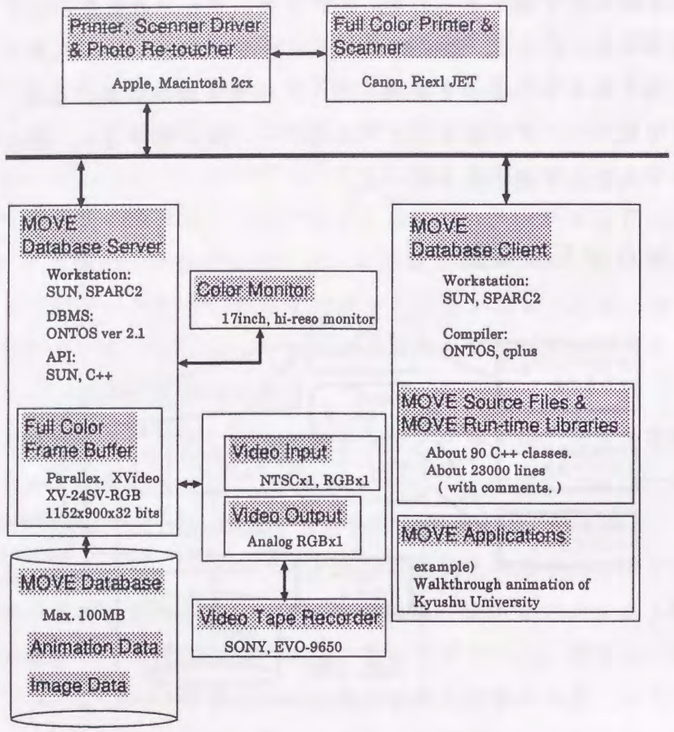


図 3.6: ＭＯＶＥの実装

iv) Ｃ＋＋インタフェースを持ったＯＯＤＢＭＳがすでに実用化されている．

一方，モデリングは，ＭＯＶＥの基本クラスのサブクラスを定義することで行なう．従って，モデリングもＣ＋＋のプログラムとして行われ，ＭＯＶＥはデータ定義をＣ＋＋で行なう．ＭＯＶＥのサーバ部のインタフェースはＣ＋＋である．ＯＮＴＯＳのユーティリティによって処理される．アプリケーションプログラムのコンパイル時には，これらＭＯＶＥのクラスライブラリおよびモデリングされたクラスのライブラリと結合される．

ＭＯＶＥのデータベース部は，オブジェクト指向データベースシステムＯＮＴＯＳ[42]である．従って，オブジェクトの生成，操作は，プログラミング言語

Ｃ＋＋のプログラムから行なう．従って，アプリケーションはＣ＋＋である．

ＭＯＶＥには，データベース上の蓄えられるデータベースオブジェクトと，アプリケーションの終了時に消滅するトランジエントオブジェクトの２種類がある．

3.5 アプリケーション用実体の定義

アプリケーションが必要とする実体の定義はクラス定義によって行なう．このクラス定義は，ＭＯＶＥの基本的クラスである実体クラスのサブクラスを定義することで行なう．従って，アプリケーションの実体クラスはすべてＭＯＶＥの実体クラスの派生クラスであり，ＭＯＶＥの実体クラスは，アプリケーションが必要とする実体の定義を行なうための基本クラスでもある．実体クラスには，描画用メソッドが実装されている．従って，新たな種類の実体のクラスを，ＭＯＶＥの実体クラスのサブクラスとして実装することで，この描画用メソッドが新たに定義した実体クラスに継承される．

アニメーションの描画には実体の運きの情報が必要である．あらかじめアプリケーションが必要とする動きをすべて予測することは不可能なので，動きのモデルはアプリケーションによって変わる．そこで，アプリケーションが必要とする動きを実体の属性として容易に定義できるような機構が必要となる．ＭＯＶＥでの動きの登録は，系の単位で，系の派生クラスを定義することで行なう．動きは系オブジェクトの属性として表現されているので，系とその動きは１つのオブジェクトとして表現される．アプリケーションの実体クラスの定義では，ＭＯＶＥの実体クラスの at，evolve，appearance の３つのメソッドの多重定義を行なうことで，動きの定義を行なう．feature，shapeFeature，motionFeature のメソッドは，ＭＯＶＥの実体クラスから継承される．

第 4 章

アニメーションモデル

4.1 ＭＯＶＥの基本クラス

	クラス名
アニメーション	Animation
カット	Cut, MPEGCut
質	Quality, HDTVQuality, EDTV2Quality, CDQuality
トラック	Track, ImageTrack, AudioTrack
フレーム	ImageFrame, AudioFrame
シーン	Scene
空間インデックス	SpatialIndex
系	System, Dynamic, Static
実体	Entity
ロケーション	Location
カメラ·マイクロホン	Viewer, Camera, Microphone
３次元物体	Object3d
３次元形状	Shape3d
光源·音源	LightSource, SoundSource
描画対象	TAppearance, TDrawable

表 4.1: ＭＯＶＥの基本クラス

ＭＯＶＥでは，グラフィックスデータ，イメージデータ，グラフィックスアルゴリズムの３つを扱う．ＭＯＶＥでは，これらのデータやアルゴリズムを表現するための基本クラスが，表4.1のように定義されている．

4.2 イメージ

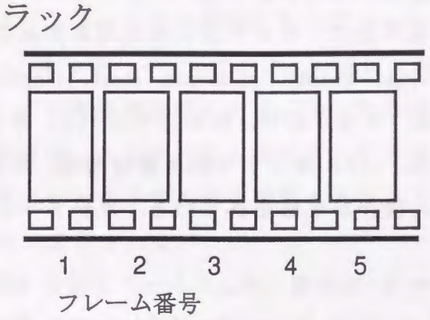


図 4.1: アニメーションにおけるトラックとフレーム関係

4.2.1 フレーム

イメージはコンピュータ合成された動画像と音声である．時間的に変化するイメージデータは，フレームと呼ばれる静止画像または単位時間の長さの音声を，時間順に並べたものとして表現できる．フレームはイメージデータの最小単位であり，計算機が描画処理を行なった結果の計算単位に対応する．

フレームには，type，quality の２種類の属性と，フレームへの書き込みを行なうための操作が定義されている．

```
Frame::write(const int frame_mum, const BLOB& data);
```

4.2.2 カット

カットは，マルチトラックとして定義される．トラックはイメージを表現するフレームの列として表現される．

カットには，ＪＰＥＧの列，ＴＩＦＦの列，ＰＩＣＴの列，ＭＩＦＦの列，ＰＰＭの列，ＭＰＥＧ，ＭＰＥＧ２，ＡＵ，ＳＬ（signed long integer）などのフォーマットがある．ＭＯＶＥでは，カットをこれら標準的なフォーマットで表現することも可能で，イメージの再利用を容易にしている．

カットには，type，quality，slicetime，start，length，dirty の６種類が属性が定義されている．カットには，次のように（1）カットの描画，（2）フレームへの書き込み，（3）カットの表示開始時刻，長さ，スライスタイム，質の変更の３種類の操作が定義されている．カットへの描画では排他制御が行なわれる．

```
Cut::shot(const CutTime, const Quality&, const Float slicetime);
Cut::write(const int frame_mum, const BLOB& data);
Cut::starttime(const ATime);
Cut::length(const CutInterval);
Cut::slicetime(const CutInterval);
Cut::quality(const Quality&);
```

イメージを単にフレームの集合として表現したのでは，利用者が編集，撮影を行なう際に不便である．そこで，イメージデータをある論理的な単位に分解し，表示はこの論理的な単位で行なう．この単位がカットである．我々はカットをひとまとまりの実体が登場するような一連のフレームの列として定義している．

利用者から見ると，あるアニメーションのイメージはカットの集まりとして表現されていて，アニメーションの描画，編集はカットを単位として行なう．利用者があるカットの描画を指示すると，ＭＯＶＥは描画処理をそのカットを構成するひとまとまりの実体を対象として，カットを構成するフレームの枚数だけ繰り返す．

描画の単位の定義には我々の定義以外にもいろいろな方法が考えられる．カットの途中で実体が完全に入れ替わってしまう場合では，描画処理の効率が落ちる．たとえば，描画の単位をアニメーションとすると効率が悪い．ＭＯＶＥではカットを単位として描画を行なう．そのため，カットに必要なデータだけが処理されるので描画より効率的である．

以上から，アニメーションは時間順にカットに分かれ，カットは複数のトラックから構成される．トラックは時間順に並んだフレームである．以上のようにアニメーション，カット，トラック，フレームには階層構造がある（図4.1）．ＭＯＶＥでは，この階層構造はオブジェクト間の親子関係として表現している．この親子関係は，オブジェクト間のリンクで表現している．利用者がアニメーションオブジェクトの名前を指定すると，アニメーションを構成するカットが時間順に表示される．

ＭＯＶＥでは，表示が可能なカットとして，動画像･音声を対象とすることから，フレームクラスは，動画像のトラック，音声のトラックのそれぞれに対応したクラス ImageTrack，AudioTrack の２種類をトラッククラスのサブクラスを持つように設計した．

4.3 グラフィックス

グラフィックスデータは，イメージの合成に必要なモデルのデータである．ＭＯＶＥの基本モデルでは，グラフィックスデータの基本単位は系である．系の詳細な定義は，５章で説明する．グラフィックスデータの動きの登録，変更，描画は系へのクラス定義，属性値の変更，メソッドの実行として行なわれる．

アニメーションとして表示される実世界は，基本的実体オブジェクトの複合オブジェクトとして表現される．アニメーションの作成は，（1）アニメーションに登場する要素としての実体の作成，（2）実体を組み合わせることによる，描画対象となる実世界の構成，（3）実世界のデータから画像を生成という３手順で行なう．この作業を自然に行なうために我々はシーンという概念を導入した．我々は，各カットに登場する実体の集まりをシーンとして

定義している．シーンには，複数の３次元物体，カメラ，マイクロホン，光源，音源から構成される．

アニメーションのカットの描画は，カットに対応するシーンを仮想的なカメラで撮影することとして行なわれる．あるカットを構成するトラックは，各カットで定まった同じシーンから描画されたものであるという条件を仮定すると，同一シーンを構成するトラックの描画は同時に行なうのが適当である．

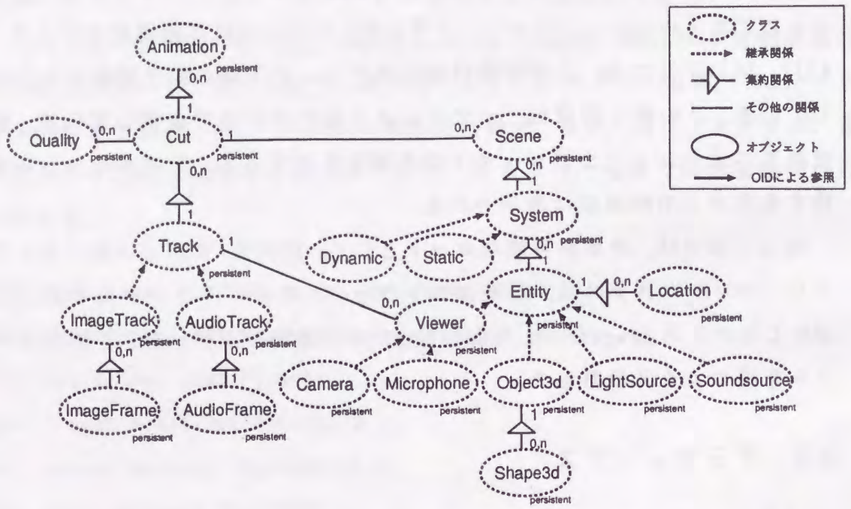


図 4.2: ＭＯＶＥにおけるアニメーションの基本クラス

従って，アニメーションオブジェクトは，シーンとカットの２つのサブオブジェクトから構成される．さらに，カットはフレームから，シーンは実体から構成される複合オブジェクトである．そのため，アニメーションは，フレーム，カット，シーン，実体から構成される複合オブジェクトとして実装される．実体には，カメラ，３次元物体，光源，マイクロホン，音源の５種類がある．実体には位置･姿勢が属性として定義されている．

以上をまとめると，ＭＯＶＥにおけるアニメーションの基本要素は，アニメーション，カット，フレーム，シーン，実体，カメラ，３次元物体，光源，

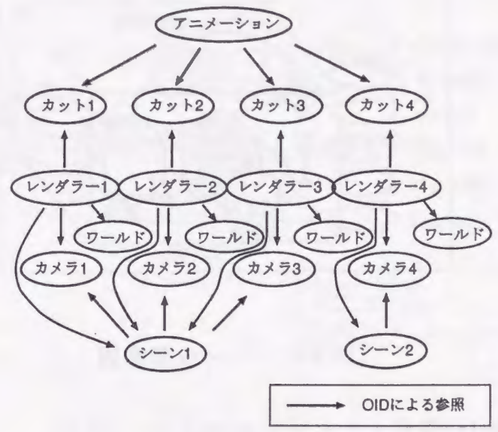


図 4.3: アニメーション，カット，レンダラー，ワールド，シーンの関係

ロケーションである．これら９つの要素は，ＭＯＶＥ上にクラスとして実装されている．図4.2には，これらの９つの要素に対するクラスについて，クラス間の関係を示している．

この図に示したように，オブジェクト間には，１対１，１対多，多対１の関係がある．例えば，カット，シーンという２種のオブジェクトの間には，多対１の関係がある．実体の位置･姿勢の属性は，Location クラスのオブジェクトとして表現され，実体オブジェクトとロケーションオブジェクトの間には，１対１の関係がある．図4.3には，２カット，４フレームからなるアニメーションのオブジェクト表現の例を示している．

ＭＯＶＥでは，アニメーションそのものも名前付けられた１つのオブジェクトとして表現されている．オブジェクトに対する操作は，操作対象のオブジェクト名，操作に対応したメソッド名やメソッドの引数を指定することで行なわれる．アニメーションクラスで定義されている代表的な操作は，（1）アニメーションの描画，（2）アニメーションの表示，（3）カットの挿入，（4）カットの削除の４つである．それぞれ，メソッド名と，メソッドの引数は，Ｃ＋＋スタイルで次のように表される．

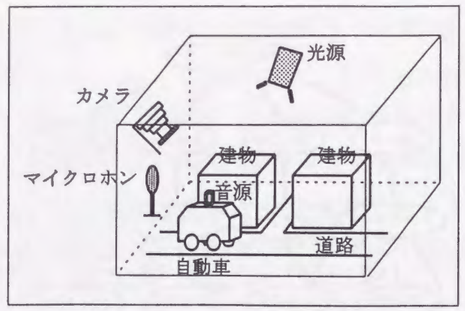


図 4.4: シーンの概念図

```
Animation::display();
Animation::shot();
Animation::add( const Scene& );
Animation::remove( const Scene& );
```

アニメーションの描画は，シーンオブジェクトの内容に従って，そのシーンに対応した動画像または音声のカットを書き換えることですることで行なわれる．アニメーションの描画のためのコンピュータグラフィックスまたは立体音響のアルゴリズムは，カメラまたはマイクロホンオブジェクトのメソッドとして実装される．あるシーンに対しては，そのシーンの中のカメラとマイクロホンの総数だけのカットが対応している．

4.3.1 シーン

ＭＯＶＥのグラフィックスデータの基本単位はシーンである．シーンは，ある動画像または音声として表示される３次元の動的な仮想世界を表現する．ＭＯＶＥでは，シーンはより基本的なデータである実体から構成された複合オブジェクトである．そのため，実体を単位としたデータの共有，再利用が可能になる．

シーンは実体の集まりとして定義される．

シーンオブジェクトは，あるシーンに登場する３次元物体，それらを照ら

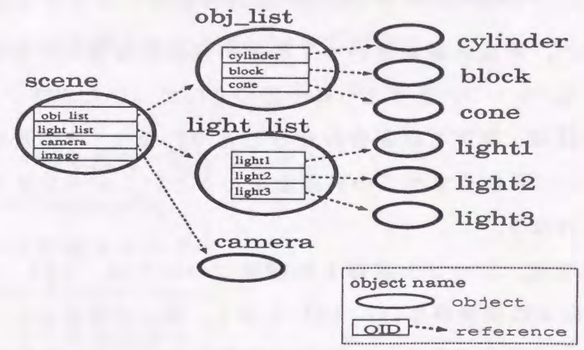


図 4.5: シーンのオブジェクト構造

す光源，それらをいかに視覚化するかの表現に用いる．シーンは，視覚化に必要なすべてのオブジェクトに対して，リンクを持っている．そのことで，シーンを視覚化するには，カメラとシーンを指定するだけである．

シーンに対しては，（1）シーンと実体の関連に関する質問，（2）シーンの視覚化，（3）シーンへの実体の追加，削除，（4）シーンの複写，の４種類の操作が定義されている．シーンの複写は，あるシーンに対してカメラや光源だけを変えたような別のシーンを作成するような場合に用いる．

シーンは，動的な仮想世界を表現していることから，シーンの時間変化の表現が必要である．ＭＯＶＥでは，シーンは系の集まりとして表現されていて，系はそのシーンを構成する互いに影響を及ぼさないような複数の実体の集合であると定義される．系は実世界中の３次元物体を，互いに影響を及ぼし合わないような実体集合に分類した時に得られる実体集合である．従って，シーンの実体の個数と系の個数とは一致しない．例えば，シーン内のすべての実体が互いに影響を及ぼしあっているような場合には，そのシーンの実体の個数は１である．以上の意味で，実体はシーンの動きを記述するための基本的な単位である．言い替えると，実体単位で動きは記述される．

4.3.2 シーンと実体の関連に関する操作

シーンには，作成と描画という２種類の代表的な操作が定義されている．ＭＯＶＥでは，シーン，実体ともに一意の名前がついたオブジェクトであり，シーンの作成は，実体を組み合わせることで行なう．そのため，シーンの描画は，シーンに現れるすべての実体を求めるというシーンクラスの基本操作を利用して行なう．

ＭＯＶＥでは，シーンと実体との関連については，（1）１つのシーン内の実体の個数には制限がない．（2）しかし，同一の実体が２つ以上のシーンに現れることも可能である．この２つの特徴によって，実体オブジェクトは又，複数のシーンに対して登場可能であり，実体はオブジェクトとしてデータベース化されていることから，実体の共有，再利用が可能となっている．

```
animation->scene().install( cone );
animation->scene().install( cyliner );
animation->scene().install( block );
animation->scene().install( lightSource );
animation->scene().install( soundSource );
animation->scene().install( camera );
animation->scene().install( microphone );
```

実体の組み合わせは，シーンオブジェクトへの実体の挿入，削除として行なわれる．シーンへの登場物の挿入，削除は，install()，uninstall() メソッドを用いて行なわれる．install() の引数は実体名である．

4.3.3 実体

ＭＯＶＥの実体クラスは，アプリケーションの実体のための基本クラスである．ＭＯＶＥでは，アニメーションの実体には，（1）実体のコピーの作成，（2）実体のパラメータの変更，（3）実体の動きの検索という３種類の操作がある．実体のパラメータとは，実体に定義された属性のうち時間によって変化しない値として定義される．ＭＯＶＥでは，実体の動きの検索は，時刻を引数として時間変化する属性の値を検索するようなメソッドとして定義される．

i) パラメータの名前の一覧を表示せよ

```
NameList* features();
```

ii) パラメータの値を求めよ

```
FeatureVal feature(char*);
```

iii) パラメータの値を変更せよ

```
void feature(char*, FeatureVal);
```

iv) 時間変化変数の時刻 t における値を検索せよ

```
StateVariables(X)* at( Time ); StateVariables(X)* evolve( StateVariables(
Time );
```

v) 見え方を求めよ

```
void appearance( SP(Entity)*, StateVariables(X) ):
```

以上に示したメソッドの説明では，Ｘはクラスパラメータである．従って，StateVariables(X) はクラスＸで定義されている時間変化する属性の集合を表すオブジェクトのクラス名であり，時間変化する属性の値を求めるメソッド at では，時刻が引数である．StateVariables(X) クラスのオブジェクトは，概念的には，長さ０以上の１次元配列として実装される．もし，ある実体クラスＸのすべてのオブジェクトが動かない場合には，StateVariables(X) の長さは０である．これら実体に関するメソッドは，動きの定義と操作に用いる．詳細は，５章で説明する．

SP(Entity) は，見え方の情報をオブジェクトとして蓄えるためのクラスである．実際の見え方の定義と，SP(Entity) の実際の実装は，６章で説明する．

4.3.4 実体の位置と姿勢

シーンは，実体の集まりとして定義される．実体には，３次元物体，カメラ，光源，マイクロホン，光源の５種類がある．

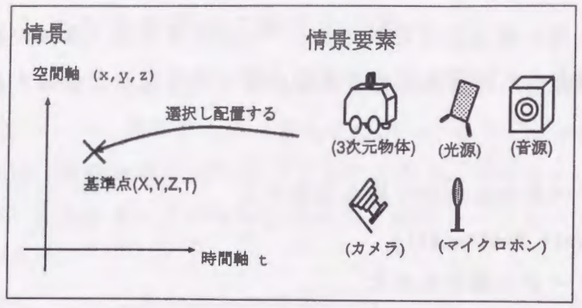


図 4.6: シーンと実体との関係

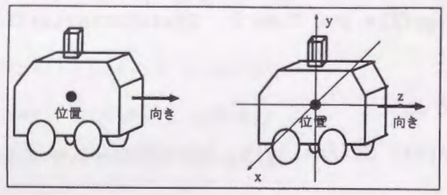


図 4.7: 実体の空間的配置

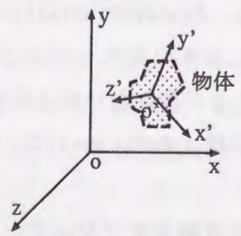


図 4.8: 実体の 3 次元的な位置·姿勢

3 次元物体は，形状だけでなく 3 次元的な位置・姿勢の属性が必要である．これらの実体は，自分自身の 3 次元的な位置・姿勢を属性として持っている．

シーンにおける実体の動きは，その実体のシーンに対する相対的な位置，存在の時間変化として定義できる．例えば，建物，木，自動車などの位置・姿勢は，シーンの座標系との相対的な関係として表現される．ＭＯＶＥの実体クラスには，次に示すような位置と姿勢に関する性質と操作が定義されている．

- 物体の位置·姿勢は，6 つの自由度も持つ
- 位置の自由度: 3 (x, y, z)
- 回転の自由度: 3 (θ, φ, ρ)
- 物体座標系の点 $P(x, y, z)$ を，世界座標系上の点 $P(x', y', z')$ に変換する
- 物体座標系のベクトル $v(x, y, z)$ を，世界座標系上のベクトル $v(x', y', z')$ に変換する
- 世界座標系上の点 $P(x', y', z')$ を，物体座標系の点 $P(x, y, z)$ に変換する
- 世界座標系上のベクトル $v(x', y', z')$ を，物体座標系のベクトル $v(x, y, z)$ に変換する
- 物体座標系に，回転(主軸，回転角)，移動(移動量)を加える

以上の属性と関数は，物体に固有の座標系を考えることで自然に表現できる．従来のアニメーションシステムでも，実体の動きは，実体に固有の座標系を考え，この座標系を，シーンの世界の原点に対して，どこに位置し，どれくらい傾けるかという量の時間変化で表現していた．

ＭＯＶＥでは，この固有の座標系がオブジェクト化され，動きの法則を座標系オブジェクトのメソッドとして定義可能としている．この固有の座標系と，その時間変化とを１つのオブジェクトとして表現したＭＯＶＥのクラスがロケーションクラスである．ロケーションは，実体の位置·姿勢の時間変化を１つのオブジェクトとして表現する．そのメソッドは，実体の動きと物体固有の座標系のパラメータの関連を属性として持っている．ロケーションクラスには，指定された時刻での位置·姿勢を知るためのメソッドが定義されている．

ロケーションオブジェクトは，実体オブジェクトと１対１に対応する関係にある．ロケーションは，図4.8に示したように，実体とシーン間の空間的な配置関係を表現することによって，実体の位置，姿勢の時間変化を表現する．

ロケーションは，アニメーション要素と世界の２つの座標系間における，回転，拡大，縮小，平行という４種類の座標変換の組み合わせとして定義される．これは，変換前の座標値を$(x,\ y,\ z)$，変換後の座標値を$(x',\ y',\ z')$として，次のように表現することができる．

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ P_x & P_y & P_z & 0 \end{bmatrix}$$

a_{ij}とP_iは空間属性をもった変数である．ロケーションは，配列a_{ij}を属性として持つ．振り子の場合では，これらの属性の値は次のようになる．

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l\cos\theta & -l\cos\theta & 0 & 1 \end{bmatrix}$$

ロケーションには，指定された時刻での位置･姿勢を知るためのメソッドが次のように定義されている．

```
Location::location( Matrix43* mat, EntityTime at );
```

4.3.5 実体のインスタンシング

一般にオブジェクトの生成にはオブジェクトを生成するための引数や，クラス名が必要である．表示すべきグラフィックスオブジェクトの生成には指定すべきパラメータの数が多い．オブジェクトの生成に必要な引数の個数を減らすための工夫として，仮想コンストラクタとインスタンシングがある．仮想コンストラクタやインスタンシングは基本的にはオブジェクトのコピーの作成として処理される．グラフィックスでは位置の属性を除いては同一の属性値を持った２つの実体を必要とすることが多い．この場合では仮想コンストラクタはインスタンシングと比べてメモリの量が多い．

ＭＯＶＥでは，実体クラスに仮想コンストラクタとインスタンシングのためのメソッドの両方が定義されている．インスタンシングは実体オブジェクトにinstaning()メソッドを実行することで行なわれる．このメソッドには引数は必要ない．実体オブジェクトのインスタンシングでは，ロケーションオブジェクト以外には浅いコピーが行なわれ，ロケーションオブジェクトには深いコピーが行なわれる．

4.3.6 ３次元物体

３次元物体は，ロケーションオブジェクトと，形状オブジェクトの２つを構成要素として持つ複合オブジェクトとしてデータベース化されている．例えば，３次元物体の一種である円柱は，図6.6に示したようなオブジェクト構造をしている．

形状オブジェクトを，３次元物体オブジェクトとは区別した理由は次の通りである．それは，コンピュータグラフィックスでは複数の３次元物体が，同じ形である場合が多い．例えば，あるシーンに同じ形をした木が１００本立っているような場合である．このようなとき，これらの木は同じ形状オブジェクトを参照する複数の３次元物体オブジェクトとして表現される．つまり，木の形状を表す１つの形状オブジェクトを１００個の３次元物体オブジェクトが参照する．３次元物体が形状オブジェクトからなる複合オブジェクトであることにより，形状を表すデータの再利用が可能となる．

4.4 イメージとグラフィックスの統合

カットはイメージだけでなく，シーンへのリンクを持つ．

従来のマルチメディアデータベースの研究の多くは，文書，動画像，音声などのデータに焦点をあてている．３次元グラフィックスアニメーションのモデルには，３次元物体，カメラ，光源，マイクロホン，音源のという５種のデータが必要である．従来のアニメーションデータベースシステムは，このうち３次元物体のみをあつかっているため，我々の目的には不十分であった．

ＭＯＶＥの基本モデルは，グラフィックスデータとそれ以外のイメージ，

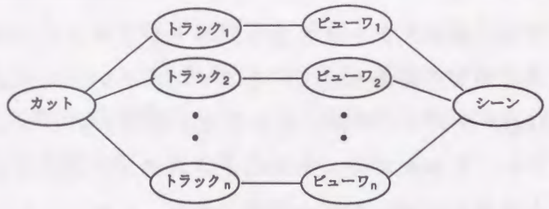


図 4.9: シーン，カット，ビューワーオブジェクトの関係

グラフィックスアルゴリズム，カメラ，光源，マイクロホン，音源の４種の実体との統合モデルでもある．ＭＯＶＥの基本クラスは，アニメーションを構成するイメージとモデルという２種類のデータを統合するので，アニメーション自体と，描画の対象となる実世界とがオブジェクト化され，オブジェクト間のリンクでイメージとモデルの関連が表現されている．

アニメーションに関する基本的な要素をオブジェクトとして表現する方法についても，いくつかの研究が行なわれてきた[8][13][32][33][48][51]．我々の研究は，多様な種類の実体や動的な実体ののデータベースを作成している点で，従来の研究とは異なる．

クラス Animation は，アニメーションをオブジェクトとして表現するＭＯＶＥの基本クラスである．アニメーションには，作成と表示という２種類の操作がある．アニメーションの表示は，動画像と音声という２種類のイメージデータを表示することとして行なわれる．

4.4.1 プレゼンテーションにおける同期

３次元グラフィックスアニメーションにおけるマルチメディアプレゼンテーションでは，（1）プレゼンテーション時における早送り，巻き戻し，スキップの機能，（2）アニメーションのイメージが複数のウインドであらかじめ指定されたタイミングで表示されるというオーサリング，（3）あるシーンを構成するトラックが同期して表示されるという lip-sync，（4）オーサリングを考慮した画像計算の４種類の課題がある．

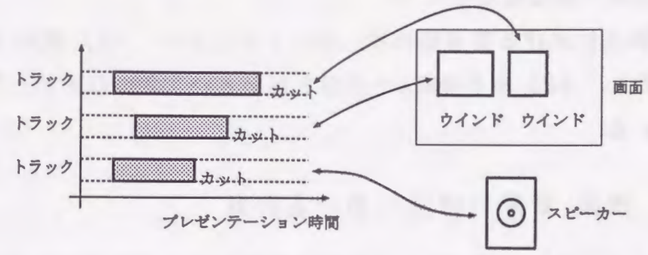


図 4.10: プレゼンテーションのモデル

4.4.2 映像･音声の表示における同期

シーンを構成する系の時間的性質に関する規則は，（1）系がある定まった時刻で生成，消滅する規則，（2）系の属性の時間的変化を定める規則の２種類である．

実体は，始まりの時刻と長さの属性をもった区間クラスのサブクラスであり，実体が属するすべてのシーンにおける見え始めの時刻と見える長さが区間の属性として記述される．

区間の属性 startTime が系の生成時刻を，区間の属性 duration が系の存在する時間的長さを表現するという方式で，１番めの生成，消滅に関する規則を表現する．この２種の規則の記述を独立に行なうには，系の属性の時間的変化をシーン全体の時刻を用いて記述するより，系に固有な時刻を用いて記述する方が適している．

ＭＯＶＥが出力する映像･音声のアニメーションは，データベース内のデータから計算で求めれる．その表示の時に，一般の映像･音声のデータベースと同様に，２つのアニメーションを並べて同時に再生したり，複数の動画像と音声の同期再生が必要である．

ＭＯＶＥでは，ある情景をある形式で表示したいというスコアが用意されている（図 4.3.1）．

計算機画面のウインド，スピーカーを表すオブジェクトに，トラックを対応させる．カットは，映像･音声の品質の情報を持ち，この情報は映像･音声

の生成処理の際に使用される.

同期再生における基本操作は，カットクラスの，（1）次の1枚のフレームを描画と，（2）ある時刻tへの巻き戻しという2つのメソッドとして実装されている.

4.4.3 映像·音声の描画における同期

描画のタイミングは，カットの開始時刻によって決まる.

同一のシーンを複数のカメラで描画する場合や，あるシーンに対して立体音響の処理を行なう場合には，複数の描画処理を同期させる必要がある.

第5章

動きのモデル

実体の動きの表現法はアプリケーションによって変わる．アプリケーションが必要とする動きの表現法は予測できない．そこでMOVEでは，アプリケーションごとに動きの登録を行なうことを可能にしている．動きの登録に関して，（1）動きの基本モデル，（2）動きの登録，（3）動きの操作，（4）動きの描画の3つの課題がある.

5.1 動きの基本モデル

MOVEの動きの格納形式は，形状特徴，運動特徴，運動法則及び見え方に関する規則の4つを実体オブジェクトの属性とする方式で行なう．この方式では，動きの対象として3次元物体の位置，色，幾何形状，光源の特性，カメラの特性などのグラフィックス属性が指定可能である．この方式では，3次元物体の位置，色，幾何形状，光源の位置，特性，撮影用のカメラの位置，特性などの定義は，実体がどのように見えるかという規則を見え方の規則として格納する．この方法では，実体の動きは系オブジェクトのメソッドで表現される．3次元物体の変形もメソッドとして表現できる．本章では，実体の動きのモデル化で我々がとった方法である，運動と形状のパラメータ表現に関して説明する.

系は，系の運動の自由度から，静的な系と動的な系の2種類に分類できる．静的な系は，運動の自由度が0であるような系であり，動的な系は，運動の自由度が1以上であるような系である．従って，シーンSは，動的な系の集

合M= $\{M_i\}$ と，静的な系の集合D= $\{D_i\}$ という2つの集合の組 S =(M, D) として定義できる．

動的な系には，運動の自由度だけの状態変数があり，状態変数の値は時間変化する．例えば，3次元空間中を自由に動くような質点 M の自由度は3であるから，M からなる系の状態変数の個数は3個である．状態変数の表現形式はアプリケーションの種類によって変わる．同様に，1個の剛体から成る系が3次元空間で自由に動くが回転はしないという場合では，系の状態変数は，3次元の位置を表す（x，y，z）という3個の変数として表現できる．例えば，M の状態変数として，（1）M の位置P(x，y，z)や，（2）M の位置P(θ，ϕ，ρ)などが考えられる．

系の動的な性質は，状態変数に対する時間の関数として表現できる．例えば，M の動的な性質は（x（t），y（t），z（t））として表現することができる．このように，系の動的な性質は，個々の系ごとに定義された，状態変数の時間に対する関数として表現できる．

キーフレームアニメーションでは，S が時間の陽の関数 $S(t)$ として表される．シミュレーションでは，S は時間変数を含む何らかの方程式 $f(S,t)=0$ の解である．この方程式を S について解くことができるとすると，S は時刻 t の関数として表せる．系の動的性質については，一般に，運動特徴[45][36]といわれる動きのパターンそのものを決定するようなパラメータが存在する．例えば，系がある決まった運動法則によって運動し，その運動が微分方程式で記述できる場合，運動特徴は，微分方程式の境界値である．いずれの場合にも，S は有限個のパラメータである運動特徴 F_m と時刻 t の関数として，$S=L_m(t,\ F_m)$ のように表される．L_m は，状態変数 S の時間変化を決定する運動法則であり，系の種類ごとに利用者が指定する．運動特徴 F_m は，アニメーション化すべき系の運動を利用者が変更するためのパラメータである．F_m の取り方は，L_m によって決まる．例えば，等速直線運動では，時刻0における位置と速度は M の運動特徴である．

動的な系 M_i の見え方は，状態変数と形状特徴という2種類のパラメータの関数として表現できる．見え方 A は，状態変数 S と形状特徴 F_s の関数として

$A=L_a(S,\ F_s)$ のように表すことができる．L_a は，系の見え方を決定する規則であり，系の種類ごとに利用者が指定する．形状特徴 F_s は，利用者が M_i の見え方 A を変更するためのパラメータである．例えば，M を球として描画する場合，M の直径は形状特徴である．以上から，F_s が与えられ，S が時間の関数として分かれば A が分かる．

以上をまとめると，動的な系は，形状特徴 F_s，運動特徴 F_m という2種類のパラメータと，運動法則 L_m，見え方の法則 L_a を属性として持つような系オブジェクトとして表現される．動的な系の状態変数 S と，見え方 A は，次のように書くことができる．

$$S=L_m(t,F_m)A=L_a(S,F_s)$$

ここで，

F_m ： 運動法則
F_s ： 形状特徴
L_m ： 運動法則 $L_m(t,\ F_m)$
L_a ： 見え方の規則 $L_a(S,\ F_s)$
S ： 状態変数
A ： 見え方

であり，F_m と F_s は利用者によって値を変更することが可能なパラメータである．

静的な系 D は，運動の自由度が0なので，状態変数 S を持たない．従って，静的な系 D は，L_m，F_m も持たないの．以上のことから，静的な系 D の見え方 A は

$$A=L_a(F_s)$$

のように表すことができる．

5.2 動きの登録

系の動きは，系の状態変数の時間に対する変化 $S(t)$ として定義されている．運動法則 L_m の記述法に関しては，キーフレーム法，スケルトン法などの

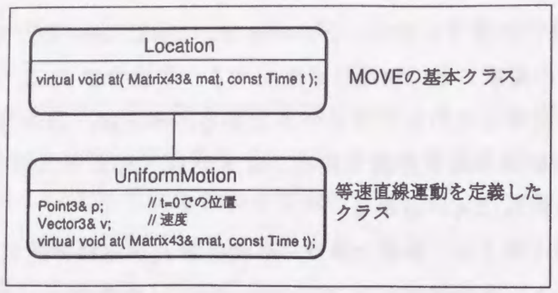


図 5.1: 等速直線運動する系のクラス階層

```
class UniformMotion : public System {
    virtual StateVariabels* state( StateVariables* s, Time t );
    virtual int DOF()  const;
    Float px, py, pz, vx, vy, vz;
}

StateVariabels* UniformMotion :: state( StateVaribales* s, Time t )
{
    TPoint3d p = TPoint3d( this->px, this->py, this->pz ) +
                 TVector3d( this->vx, this->vy, this->vz ) * t;
    s.val[1, p.x];
    s.val[2, p.x];
    s.val[3, p.x];
    return s;
}

int UniformMotion :: DOF()
{
    return 3;
}
```

図 5.2: 等速直線運動する系のクラス定義

代表的なアニメーションの動きの表現法の適用が考えられる．これらの手法


```
class AcceratedMotion : public System {
    virtual StateVariabels* state( StateVariables* s, Time t );
    virtual int DOF()  const;
    Float px, py, pz, vx, vy, vz, ax, ay, az;
}

StateVariabels* AcceratedMotion :: state( StateVaribales* s, Time t )
{
    TPoint3d p = TPoint3d( this->px, this->py, this->pz ) +
                 TVector3d( this->vx, this->vy, this->vz ) * t;
                 TVector3d( this->ax, this->ay, this->az ) * 1/2 * t * t;
    s.val[1, p.x];
    s.val[2, p.x];
    s.val[3, p.x];
    return s;
}

int AcceratedMotion :: DOF()
{
    return 3;
}
```

図 5.3: 等加速度運動のクラス定義

では，動きを（状態変数の値、時刻t）の組が集まった表形式で表現し，動きの変更は表の各セルの値の変更として行なう．しかし，この方式では，リアルな動きを得るにはこの表の値が必ずリアルな動きを表現するような値であることを保証するようなプログラムが必要であり面倒である．動きの変更では，表の値をすべて書き換えるような操作が必要になるので面倒である．

一方，運動法則 L_m の記述法として，シミュレーションモデルを用いる方法も考えられる．シミュレーションモデルでは，動きの変更はシミュレーションのパラメータの変更として行なう．さらに，シミュレーションに基づいて動きを得るので，シミュレーションモデルが正しければリアルな動きを得る

ことができる.

ＭＯＶＥでは，動きの定義に関して．表形式とシミュレーションモデルの両方を扱えるようにしている．動きの定義は，動的な系のクラスの定義として行なう．動的な系のクラス定義は，（1）運動特徴，運動法則，状態変数の個数の定義，（2）形状特徴，見え方の規則の定義に対応する２つのクラスを定義することで行なう．

第１の運動特徴，運動法則，状態変数の個数の定義は，Systemクラスのサブクラスの定義として行なう．例えば，３個の状態変数(x,y,z)，６個の運動特徴$p_x, p_y, p_z, v_x, v_y, v_z$を持ち，運動法則が$(x,y,z) = (p_x,p_y,p_z) + t*(v_x,v_y,v_z)$で表されるような系の定義は，図5.2のように行なう．UniformMotionクラスでは，運動特徴が時刻$t=0$における位置$p(0)$と速さ$v(o)$を表すと解釈すると，運動法則は等速直線運動$vt+p$ををを表す．運動法則は，5.2のように，UniformMotion::state()に記述する．state()の引数は時刻であり，戻り値は状態変数の値である．

第２の形状特徴，見え方の規則，等速直線運動などの定義はクラス定義として行なう．

まず，ＭＯＶＥの(x，y，z)という属性を持つようなクラスLocationのサブクラスとして，等速直線運動という具体的な運動を表すクラスUniformMotionを定義する．

ＭＯＶＥでは，系の状態変数の時間変化に関する定義は，Systemクラスのサブクラスを定義することで行なう．規則，運動特徴の種類のパターンごとにクラスを定義することで行なう．具体的には，運動特徴を属性として持つようなクラスをＭＯＶＥの登場物のサブクラスとして定義する．このサブクラスでは，系の状態変数$S = L_m(t, F_m)$を動きの定義は，Systemクラスのstate()を多重定義して，新しいstate()の定義として，L_mを定義する．DOF()の定義には状態変数の個数を記述する．この手順で，等加速度運動なども定義可能である．定義例を，図（5.2）に示している．

物体形状の変形，物体の色の変化など，見え方の変化に関する処理は，System，Group，Entity，Cut，Track，TAppearance，TDrawableクラスには，

描画時において見え方を決定するためのメソッドが定義されている．利用者が系の種類を増やすことを可能にする．動きの定義は，これら基本クラスのサブクラスの定義として行なう．サブクラスの定義では，基本クラスのメソッドを継承するので，定義が容易である．

ＭＯＶＥでは，以上のように系のクラスを定義することで，運動のパターンがクラスとしてデータベース化される．

5.3 動きの操作

動きの基本モデルに関しては，動きの変更を簡単に行なえることが重要である．ＭＯＶＥの系は，形状特徴と運動特徴とをパラメータとして持つことで，運動や形状のパラメータ化を行なっている．運動と形状の変更はパラメータ値の変更として行なう．形状特徴，運動特徴の値をいろいろと変えることで，動きをいろいろと変えることができるようになる．

例えば，シミュレーションモデルにおける動きの変更は，シミュレーション用パラメータの変更として行なう．ＭＯＶＥでは，シミュレーションモデルは系オブジェクトの属性として定義され，シミュレーションのパラメータは運動特徴そのものである．

5.4 動きの描画

ＭＯＶＥ上での画像の描画は，概念的には，実世界を実体の１種であるカメラが撮影するという形式で行なう．カメラは，描画のためのコンピュータグラフィックスのアルゴリズムが実装されている．トラックの描画は，シーンオブジェクトに対して，カメラ名を引数とした描画質問で行なう．

シーンの状態は時間とともに変化するから，シーンの描画では，シーンの動きを知る必要がある．シーンの描画はトラックを単位として行なう．トラックはフレームの集まりとして定義されているから，あるトラックの描画はそのトラックを構成するフレームの枚数だけ描画を繰り返すことで行なう．以上の処理は，疑似的にＣ＋＋で次のように記述できる．

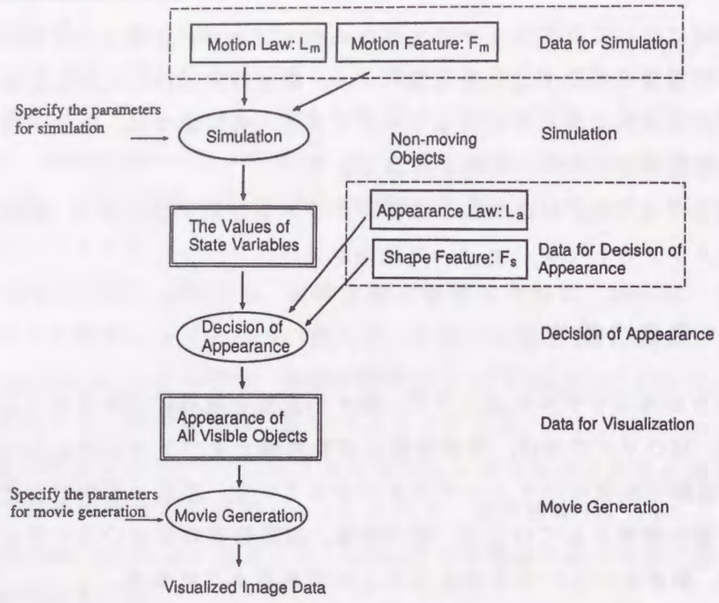


図 5.4: 動きの描画

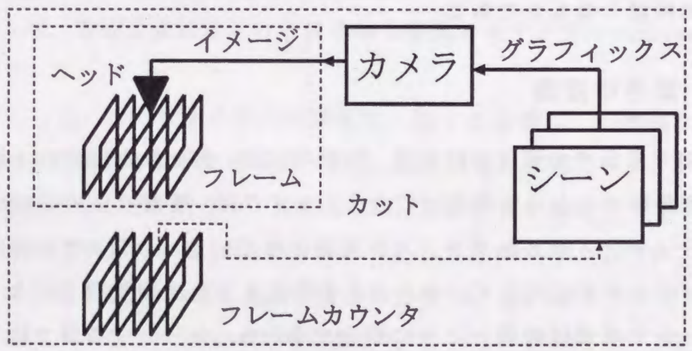


図 5.5: アニメーション描画における処理の流れ

```
Appearace(X) appear;
```



```
appear->assign( track( 0 ) );
for ( i = 0; i < track->duration(); i++ ) {
  appear->assign( track->scene( time(i) );
  appear->draw();
}
```

Xはグラフィックスアルゴリズムを表す名札である．各フレームの描画では，フレームに対応する時刻のシーンの状態を描画することで行なう．

TDrawableクラスのメソッドでは，タイムスタンプ*now*を設定し（1行目），matオブジェクトに計算結果を書き込んでいる（2行め）．

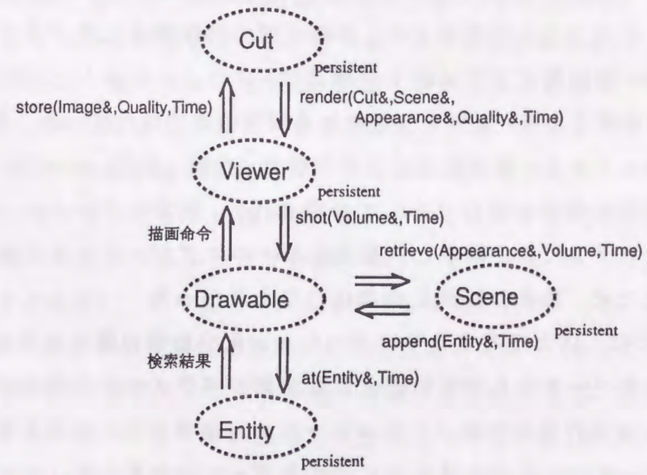


図 5.6: アニメーション描画におけるオブジェクト間のメッセージの流れ

3次元物体，光源，音源，カメラ，マイクロホンの5種の実体が組み合わさって1つのシーンを構成し，データベースとして蓄積されている．本節では，シーンの描画を容易にするためにＭＯＶＥ上に実装されたオブジェクト指向による3次元コンピュータグラフィックスのモデルを説明する．

系の見え方は，運動特徴，運動法則，形状特徴，見え方に関する規則の4

種類の属性から決定される．これら４つの属性はデータベースに格納される．描画時にはこれらのデータベースから検索される．データベース内には系の動きに関する４つの格納されているのに対して，描画時にはある時刻における系の状態変数の値が必要である．状態変数の値の検索は，動的な系に対しては，時刻を引数として行なう．一方，静的な系に対しては，状態変数そのものが存在しない．このよう動的な系と，静的な系とでは描画の処理が変わる．動的な系と，静的な系とが混在したシーンの視覚化を効率良く行なうために，ＭＯＶＥでは，系の状態値の保持は系とは別のクラスTDrawableクラスのオブジェクトで行なっている．さらに，シーンの検索結果自体をTAppearanceクラスの複合オブジェクトとして保持している．

シーンの見え方に関する情報であるTAppearanceクラスの描画用オブジェクトは，描画のために検索され，描画処理の間保持されるグラフィックデータである．描画用オブジェクトの構造は，アニメーションとして見えている（あるいは聞こえている）シーンのある瞬間の構造に対応する．従って，描画用オブジェクトも，論理的には３次元物体，光源，音源，カメラ，マイクロホンの５種類の実体の組合せとして表現される．従来のアニメーションデータベースシステムでは、動きと状態値を同一のオブジェクト上に表現しようとしていたため、動きの格納を自然に行なえなかった

図5.4には，以上の実体のアニメーション化の処理の流れを示している．まず，データベースから運動法則と運動法則のパラメータが検索され，シミュレーションが行なわれる．シミュレーション結果から，状態変数が求まる．次に，データベースから見え方に関するデータが検索され，シミュレーション結果とから，アニメーションが生成されるという流れである．

描画用オブジェクトの検索では，シーン基本クラスに対して，カメラ･マイクロホン名と，時刻を引数として描画対象の集合を複合オブジェクトとして返すような動きの計算を行なう．この検索では，描画すべき系の各々に対して，時刻を引数とした検索が行なわれる．この検索は，実体オブジェクトに時刻tをパラメータとして渡すことで行なう．系は，時刻tにおける状態変数の値と，見え方を求め答として返す．この検索結果は，描画用オブジェクト

上に保持される．描画用オブジェクトでは，状態変数の値と見え方を保持する領域の他に，検索の引数として与えられた時刻，状態変数の時間変化の規則を表すメソッドvoid at(Time t)へのポインタから構成されている．動きの計算は，個々の系ごとに動きの計算を行ないつつ単位時間ごとに描画を繰り返す．こうして，動きの描画の処理は，運動法則に時刻を引数として動きの計算を行ない状態変数を決定，さらに系単位で，見え方を決定することで行なわれる．以上のように，動的な系に対しては描画に必要なデータは描画時に計算される．これらの属性は図5.4では破線で示している．これらの属性は系オブジェクトの属性としてカプセル化されていて，次に示したメソッドで操作される．

動的な系Mに対して，運動法則L_mと運動特徴F_mが計算され，シミュレーションによりMの状態変数Sの値が求まる．次に，形状特徴F_sと見え方の規則L_aが検索され，見え方Aが求まる．同様に静的な系Dに対しては，見え方の規則L_aと，形状特徴F_sが検索されAが決定される．最後に，シーン内の実体の見え方Aの情報が集められアニメーションが生成される．

シーンに対するトラックの描画は，次に示すような３段階の描画処理を，そのトラックのフレームの枚数だけ繰り返すことで行なう．（1）各フレームに対応する時刻に，そのフレームに登場する実体の決定．（2）その時刻における，個々の実体の状態値と描画に必要な情報の計算．（3）こうして求めた情報を描画部に渡すことによる画像の描画．以上の処理は，次に示すように，ＭＯＶＥのオブジェクトが連係して行なっている．図5.6には，アニメーションの描画におけるオブジェクト間のメッセージの流れを示している．

描画用オブジェクトの描画は，個々の実体ごとに動きの計算を行ないつつ行なう．このとき，描画中のフレームのタイムスタンプと，各実体のメソッドとして定義された運動法則から決定される状態変数の時間変化との同期が必要である．そこで，描画用オブジェクト内には，描画対象となるすべての実体についてのハンドラが存在し，この同期を行なう．複数の描画用オブジェクトが同一の実体へのハンドラを持つことがあるから，描画情報はハンドラごとに保持する．ハンドラの生成は，実体クラスのメソッドで行なう．

描画処理は，アプリケーションプログラムが，カットオブジェクトに描画命令を送ることで始まる．描画要求を送られたカットオブジェクトは，次の手順で描画を行なう．

i) 描画用オブジェクトの生成：
描画には，描画アルゴリズムと描画対象が必要である．ＭＯＶＥでは，この２つはそれぞれ，レンダラーを表すViewerクラス，見え方の情報を表すTAppearanceクラスのサブクラスののオブジェクト化されている．描画開始時に，これら２つのオブジェクトがメモリ上に生成される．

ii) フレーム描画の繰り返し：
次に，カットオブジェクトは，レンダラーオブジェクトに対して，renderメッセージをフレームの枚数ｎ回だけ送ることで，画像を要求する．このメッセージの引数には，シーン時刻 ｔが含まれる．シーン時刻 ｔ はフレーム番号 ｉ の関数として，

ｔ＝ｉ＊ｆｐｓ　（ｉ＝０・・・ｎ－１）

と定義される．画像を要求されたレンダラーオブジェクトは，ワールドオブジェクトに対して描画対象に関する情報を要求する．そのワールドオブジェクトは，次の（３），（４），（５）の処理を行なう．

iii) 実体の検索：
まず，描画の対象となる実体オブジェクトのＯＩＤの集合を得る．そのために，レンダラーオブジェクトは，ｒｅｔｒｉｅｖｅメッセージをシーンオブジェクトへ送る．このメッセージ引数は，検索結果を保存すべきワールドオブジェクトのＯＩＤと，シーン時刻と，どれくらいの範囲の実体が必要かという範囲である．

iv) 実体の状態値の検索：
次に，描画対象となる実体のすべてについて，指定された時刻での状態値を検索する．シーン時刻tを指定して，オブジェクトの特徴量を検索することが必要である．そのために実体に定義されたメソッドがat()である．この検索は，ワールドオブジェクトが，（３）の結果得られた登場物

オブジェクトのすべてに対して，ａｔメッセージを送ることで行なう．引数は，結果を保存すべきワールドオブジェクトのＯＩＤとシーン時刻である．

v) 描画に必要な情報の検索：最後の段階が描画である．まず，（４）で得られたオブジェクトの状態値から，形状，構造などの描画に必要な情報が検索され，ワールドオブジェクトに保存される．このワールドオブジェクトがレンダラーオブジェクトへ返されて，描画が始まる．

以上のように，描画時には，（３），（４），（５）で示した３種類の検索がデータベースに対して行なわれる．

映像･音声の生成手法については，アニメーション化のアルゴリズムには，ボリュームレンダリング，レイトレーシング，ラジオシティ，シェーディッドポリゴン，ワイヤーフレームなどのいくつかの種類がある．したがって，描画処理の引数として，これらのアルゴリズムの名称を指定可能としている．TAppearanceクラスには，レンダリングアルゴリズムに対応したサブクラスが定義されている．例えば，シェーディッドポリゴンに対応したTAppearanceがSP(TAppearance)である．TAppearanceクラスには，描画単位となるオブジェクトを生成するためのメソッドが定義されている．以上のクラス構成の利点の１つは，カメラの属性として，レンダラーアルゴリズムの種類を保持することを可能にしていることである．

第 6 章

グラフィックスモデル

6.1 ＭＯＶＥへのグラフィックスモデルの登録

機能	基本クラス
３次元形状の表現法	Shapde3d
物体の表面属性	Shapde3d
カメラの属性	Shapde3d
光源の属性	LightSource
描画アルゴリズム	Camera
カメラの属性	Camera
立体音響アルゴリズム	SoundSource
音源の属性	SoundSource
形状データフォーマット	Shapd3d

表 6.1: グラフィックスモデルの機能

ＭＯＶＥの基本クラスのほとんどは抽象クラスなので，ＭＯＶＥの基本クラスだけではアニメーションの描画を行なうことができない．描画を行なうには，ＭＯＶＥ上へグラフィックスモデルを登録する必要がある．グラフィックスモデルには，表6.1の９種類の機能がある．

描画アルゴリズムには，ワイヤーフレームレンダリング，サーフェスレンダリング，ソリッドレンダリング，ボリュームレンダリングなどのアルゴリ

ズム[1][18]がある．ＭＯＶＥの基本クラスは、特定のグラフィックスモデルに依存していない．そのことによって，複数のグラフィックスモデルを同時に実装できるようになっている．原理的には，描画対象がいくつかの物体の集まりとして表現できる場合には，ＭＯＶＥにはあらゆる種類の描画アルゴリズムを実装可能である．ＭＯＶＥの特徴として，ＭＯＶＥの基本クラス上に従来のアニメーションデータベースでは困難であった写実的な描画のためのグラフィックスモデルをCamera, SurfaceShader, LightSource, Microphone, SoundSource のサブクラスとして定義できるようになっている．グラフィックスモデルの定義は，ＭＯＶＥの基本クラスのサブクラスの定義として行なう．

ＭＯＶＥ上へグラフィックスモデルの定義を行なうには，最初に，使用する描画アルゴリズムを選択する必要がある．工業分野では，３次元的な形状を正確に描画するだけでなく，画像生成の速さも要求される．一般に，描画アルゴリズムについては，描画処理の高速性と，描画の結果得られる画像のリアリティの間にトレードオフの関係がある．

サーフェスレンダリングは，高速に描画するためのグラフィックスハードウエアとの相性が良い．しかも，サーフェスレンダリングは，十分に高品質なイメージを生成するという利点もある．以上の理由により，我々はサーフェスレンダリングを選んだ．

実体	属性
３次元物体	位置·姿勢，形状，質感，テクスチャ
光源	位置·姿勢，色，強度の方向特性
音源	位置·姿勢，音源のサンプル値·距離
カメラ	位置·姿勢，視野角，レンズの特性
マイクロホン	位置·姿勢，マイクロホンの特性

表 6.2: 実体の属性一覧

表6.2には，サーフェスレンダリングに必要なデータの一覧を示している．サーフェスレンダリングに基づくグラフィックスモデルの実装は，表6.2のデ

ータを属性として持つようなクラスの定義によって行なう.

6.2 3 次元形状

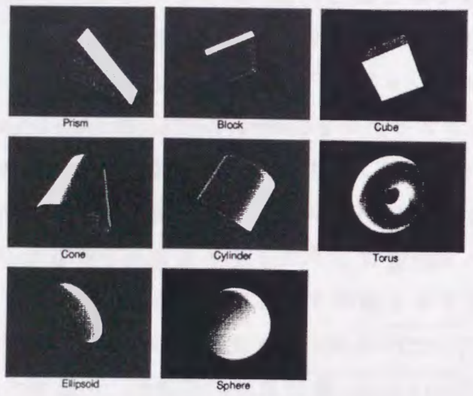


図 6.1: 3 次元物体の基本形状

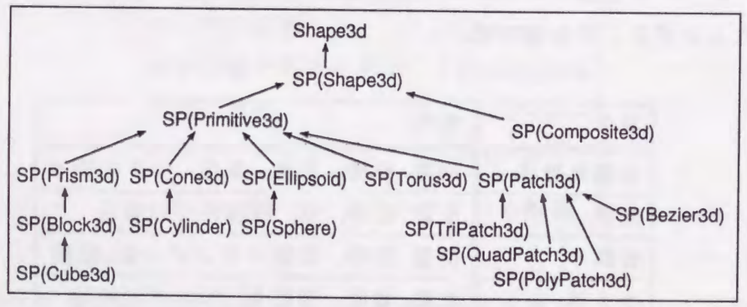


図 6.2: 3 次元物体のクラス階層

3 次元形状の表現は，コンピュータグラフィックスだけでなく，3 次元ＣＡＤ，工学設計，解析，シミュレーション実験などの分野で広く行なわれている．3 次元形状の表現法には，ワイヤーフレームモデル，サーフェスモデ

種類	クラス名	属性
ベジェパッチ	SP(Bezier3d)	縦，横，高さ
直方体	SP(Block3d)	縦，横，高さ
立方体	SP(Cube3d)	一辺の長さ
円錐	SP(Cone3d)	上半径，下半径，高さ
円筒	SP(Cylinder3d)	半径，高さ
円盤	SP(Disc3d)	半径
楕円体	SP(Ellipsoid3d)	3 軸方向の半径
多角形	SP(Polygon3d)	各頂点の座標
多角形パッチ	SP(PolyPatch3d)	
角柱	SP(Prism3d)	各頂点の座標，高さ
4 角形パッチ	SP(QuadPatch3d)	
正多面体	SP(RegularPolyhedron3d)	正多面体
球	SP(Sphere3d)	半径
ドーナツ形	SP(Torus3d)	2 半径
3 角形パッチ	SP(TriPatch3d)	

表 6.3: 基本形状とその属性

ル，ＣＳＧ，Ｂ－ｒｅｐなどのソリッドモデル，雲，炎などの不定型物のモデルなどの種類がある．

ワイヤーフレームモデルでは，形状は線の集まりとして表現される．面に関する情報を持たないので，物体は線の集まりとして表示される．従って，アニメーションの写実性が低い．サーフェスモデルはサーフェスシェーディングアルゴリズムで高速に描画でき，画像のリアリティも高い．ＣＳＧでは，形状は球，立方体，直方体などのプリミティブと呼ばれる基本的な立体の組合せとして表現される．Ｂ－ｒｅｐでは，形状は，点，線，面の集まりとして表現される．これらソリッドモデルは，サーフェスモデルと比べて構造が複雑なので，実装が難しい．以上の理由から，我々はサーフェスモデルを用いた．

ＭＯＶＥにおける形状表現は（1）ＭＯＶＥの基本面の利用，（2）新しい

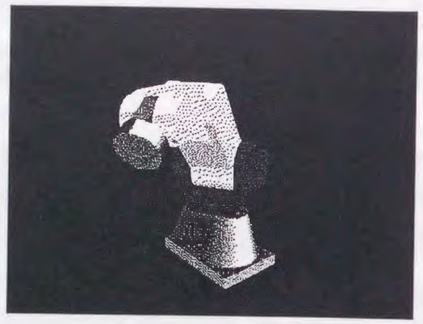


図 6.3: ロボットアームの静止画像

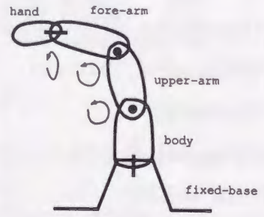


図 6.4: ロボットアームの力学的構造

基本面の定義，の２つの方法がある．

サーフェスモデルでは，形状は基本面と呼ばれる面の集まりとして表現される．基本面は，サーフェスモデルにおける形状の最小単位である．

立体の形状は，基本面と呼ばれる基本的な単位図形の組合せとして表現する．基本面にはいろいろな種類がある．基本面には，直方体，立方体，円柱，円筒，ドーナツ形，楕円体，球，パッチなどの種類がある．第１の方法では，これら基本面を組み合わせることで利用者が望む形の形状を組みあげる．基本面の１種であるパッチを用いることで，曲面の形状はパッチと呼ばれる多角形の集まりとして精度良く近似できる．

ＭＯＶＥでは，基本面はSurfacePrimitive3dクラスのサブクラスのオブジ

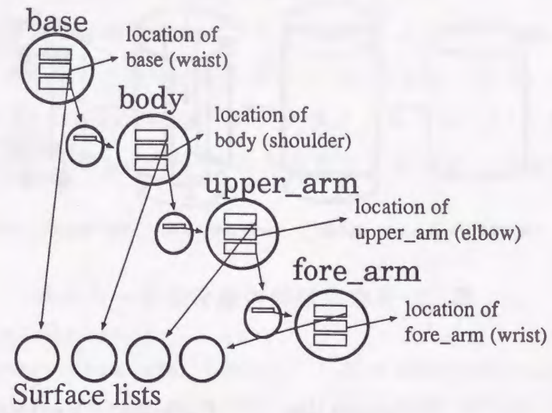


図 6.5: ロボットアームの複合面オブジェクト表現

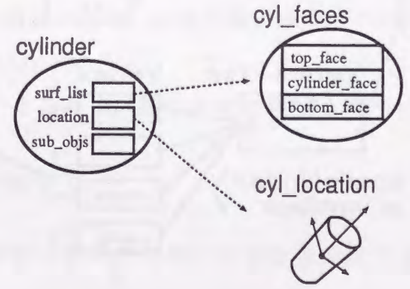


図 6.6: 円筒面のオブジェクト表現

ェクトである．これらの基本面については，表6.3に示すような属性を持っている．利用者が基本面を生成するには，表6.3に示した属性を指定することで行なう．基本立体の操作は，表１に示す各クラスの属性を操作することで行なう．

ＭＯＶＥの３次元形状オブジェクトはCompositeクラスのオブジェクトとして表現される．Compositeクラスのオブジェクトは基本面オブジェクトの

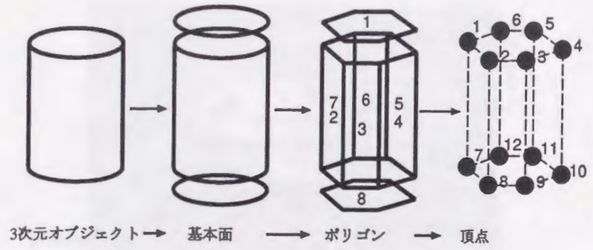


図 6.7: 3 次元形状の幾何要素への分割

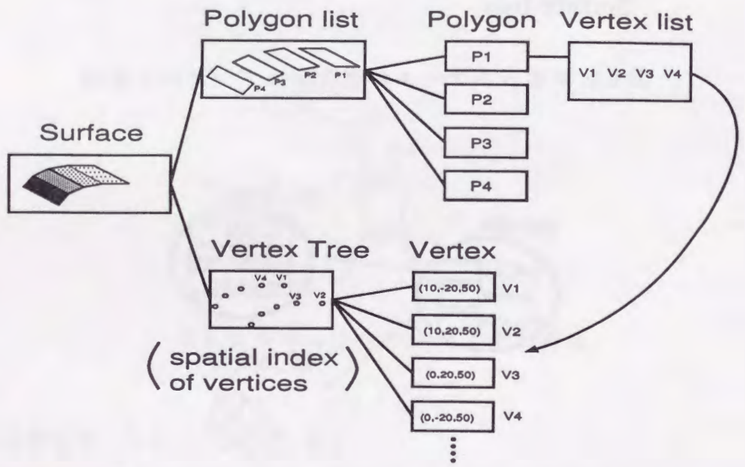


図 6.8: 3 次元形状のオブジェクト表現

集まりである．一般に，3 次元物体の形状は基本面の集まりとして表現される．利用者は，最適な基本面をＭＯＶＥの中から選択する．例えば，図6.7に示すような円筒を，ポリゴンメッシュと円盤という 2 種類の基本面で表現する場合，円筒は，円柱面 1 枚と円盤 2 枚という 3 つの幾何形状の集まりとして表現され，円柱面はｎ個のパッチと 2ｎ個の頂点が組合わさった，ｎ角柱として近似される．このように，ポリゴンメッシュ面の形状はポリゴンの集

まりで近似され，ポリゴンは周囲の点で表現される．

第 2 の方法は新しい基本面のクラスを定義し，そうして定義した新しいプリミティブを利用することで形状を表現する方法である．新しいプリミティブの定義は，SP(PrimitiveSurface) のサブクラスとして新しいクラス SamplePrim を定義し，次のように patchOut() を多重定義することで行なう．

```
void SamplePrim::patchOut( int res )
{
  ...
   this->patchBand( handle,
   f1, zradprim1, ( zradprim1 * dz2 ) / ( f1 * this->radius_x * dx2 ),
   f2, zradprim2, ( zradprim2 * dz2 ) / ( f2 * this->radius_x * dx2 ),
   x_arr, y_arr, u_arr, v1, v2, res, MV_SPHERICAL );
}
```

patchBand() は，利用者による基本面の定義を短くし，容易にするためのＭＯＶＥのライブラリ関数である．多重定義の容易さの点では，これら基本形状について，SP(Shape3d) というインタフェースクラスが存在し，SP(Shape3d) クラスを多重定義することで，基本形状の種類を増やすことが可能である．

6.3 光線の伝搬モデル

Kajiya[21] らは光線の伝搬が次に示すような方程式でよく近似できることを示した．

$$i(x,x') = \int r(x,x',x'')l(x',x'')dx''$$

ここで，$i(x,\ x')$ は x' から入射して点 x へ至るような光線の強さを表す．関数 $r(x,\ x',\ x'')$ は，物体の表面での反射を表し，関数 $l(x',\ x'')$ は入射する光の強度分布を表す．以上の方程式で記述される光線伝搬をローカルリフレクションモデルという．

ＭＯＶＥでは，$r(x,\ x',\ x'')$ を SurfaceShader オブジェクトの属性として，$l(x',\ x'')$ を光源オブジェクトの属性として実装した．

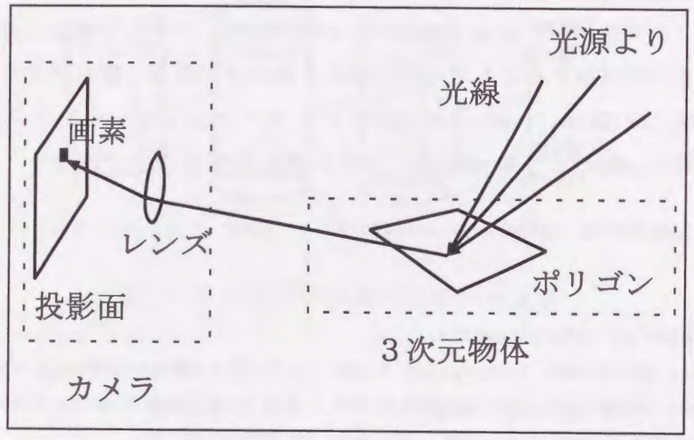


図 6.9: 光線の伝搬モデル

6.3.1 表面属性

画像生成には，位置と形状に関する情報以外に，色，テクスチャ，光線の反射率などに関する情報が必要である．これらの属性は，SurfaceShaderオブジェクトの属性として表現される．物体表面の材質はグラフィックスのリアリティを向上に効果的である．ＭＯＶＥには，Constant，Matte，Metal，Plastic，TextureMap, Wood，Marble, Bumpy, Planet, の 9 種類の表面属性のクラスがSurfaceShaderのサブクラスとして実装される．表面属性クラスは，表6.4に示したようなシェーディング関数$r(x,\ x',\ x'')$を属性として持っている．なお，パラメータk_a，k_s，α，k_d，nの値は利用者が設定可能である．

6.3.2 光源

ローカルリフレクションモデルにおいて，光線の伝搬の結果を得るには積分方程式を解く必要がある．しかし，積分方程式を精度良く解くのは処理時間がかかる．そこで，光源に関する関数$l(x',\ x'')$がアンビエントa，強さI，向きd，色cを用いて次のように表現できる場合に，$l(x',\ x'')$の種類を限定す

材質	シェーディング関数 $r(x,\ x',\ x'') =$	オブジェクトの属性
constant	K_a	K_a
matte	$K_a + K_d\theta$	$K_a,\ K_d$
metal	$K_a + K_s \cos^n \alpha$	$K_a,\ K_s$
plastic	$K_a + K_d\theta + K_s \cos^n \alpha$	$K_a,\ K_d,\ K_s$
texture map	$f(x')$	$f(x')$

k_a: ambient

k_s: specular

α : shinyness

k_d: diffuse

n: metalness

表 6.4: 表面属性クラス

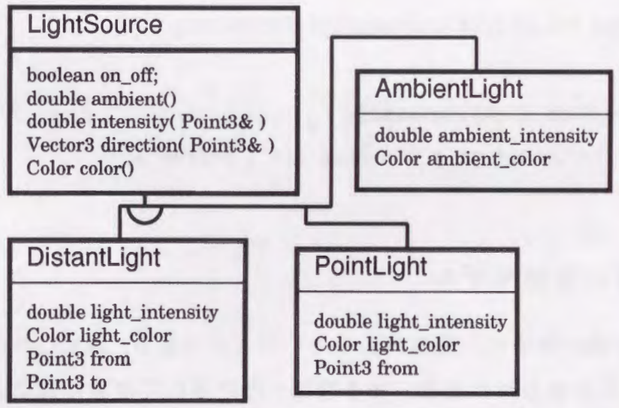


図 6.10: 光源クラス

ることで，積分を行なわないで済むようにした．

$$l(x',\ x'') = a(x' - x'' \neq d) = a + I \times c(x' - x'' = d)$$

この４つのパラメータだけですべての種類の光源を表現することはできない．しかし，コンピュータグラフィックスで代表的な環境光源，遠隔光源，点光源の３種類を表現することができる．

ＭＯＶＥの光源は，環境光源，遠隔光源，点光源の３種類である．6.10にはこれらのクラス設計を示している．

- 環境光源

  環境光源はあらゆる方向から均一の強さで光線が到達する場合である．
- 遠隔光源

  遠隔光源からの光線はすべて平行である．太陽光は遠隔光源である．
- 点光源

  点光源は，ある１点から光線が発射している場合である．

$$I = \frac{I_0}{4\pi \| x' - x'' \|^2}$$

点光源はＣ＋＋で次のように記述できる．

```
double PointLight::intensity( TPoint3& p )
{
    return light.intensity /
          4 * pi * ( p-from ) * ( p-from );
};
```

6.4 音の伝搬モデル

音の伝搬の特徴は，遅延があるということである．図6.11のように，音源から時刻tに発せられた音は、ある時間τだけ遅れて聴者に聞こえる．τは聴者と音源間の距離dによって下式で表される。

$$\tau = \frac{d(t)}{V_s} \qquad \text{(但し、}V_s\text{は音速)}$$

音の大きさは距離の2乗に反比例して小さくなる。距離がd_0の時に聞こえる音の大きさをA_0とすれば、聴者での音の大きさ$A(d)$は、次の式で表される．

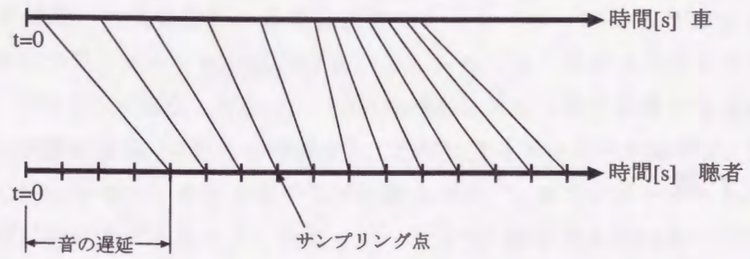


図 6.11: 音の遅延

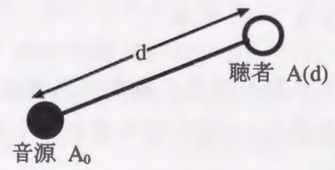


図 6.12: 音の減衰

$$A(d) = A_0 \cdot \left(\frac{d_0}{d}\right)^2$$

第７章

アプリケーションの実装

7.1 シミュレーションモデル

一般に，時間とともに動く実体の動きの表現法には，（1）利用者が動きを明示的に指定する方法と，（2）利用者がシミュレーションモデルを記述する方法の２つがある．

利用者が動きを明示的に指定する方法には，キーフレーム，スケルトンなどがある．明示的に指定する方法では，単に，動きは状態変数を時刻tの関数$P_i(t)$として，時間を縦軸に，状態変数を横軸にもつ配列として表現可能である．しかし，動きを時間tの関数として指定できない場合が多い．例えば，３体問題では，３個の質点がもつ合計９個の状態変数を時間の関数として利用者が明示的に指定せねばならないため，動きを明示的に指定する方法で表現できない．

利用者がシミュレーションモデルを記述する方法では，利用者のモデルに従って計算機が動きを自動的に求める．シミュレーションモデルでは，動きを時間tの明示的な関数で記述する必要がないため，動きを明示的に指定する方法での記述能力の問題はない．しかも，シミュレーションによって動きを決める方法には，自然な動きが得られるという特徴がある．

シミュレーションモデルには，さまざまな方法が考えられるが，３次元グラフィックスアニメーションにおける代表的な手法に，力学[12]がある．力学アニメーションは，物体の物理的特性(質量など)と物体が存在する世界の

特性を指定して現実世界の物理法則をシミュレーションにより動きを求める．３次元アニメーションにおけるシミュレーション手法としての力学は，（1）手法として確立しており，（2）物理的に正しい動きが得られるという２つの特徴がある．これらの特徴から，力学によるアニメーションは，建築，機械CAD，ロボティクスなどの工学分野を始め，人体アニメーション，仮想現実感におけるダイレクトマニピュレーション，高校や大学における力学の教育など，さまざまな分野に応用されている[3][20][31]．

本章では，力学に関する基本事項として，動きの表現の対象とする系の例を用いて，拘束，一般座標，ラグランジェの運動方程式という３つの事項について説明する．そして，古典力学の力学系を，１つの動的系オブジェクトとして表現することで，系の運動法則の記述がラグランジェの運動法則を立式する同時に，古典力学の法則をＭＯＶＥの系オブジェクトのメソッドとして表現することを行なう．我々は，系の運動法則の記述において利用可能なクラスをＭＯＶＥのクラスとして実装することを行なった．

7.2 運動法則としての古典力学

古典力学の対象は，互いに力を及ぼし合うようなn個の質点M_iの集合$\{M_i\}$ $(1 \leq i \leq n)$からなる質点系である．すべての質点には内力と外力が作用している．内力f_{ij}は，質点M_jから質点M_iへの力として定義され，外力F_iは，系の外から質点M_iへの力をとして定義される．すべての質点M_iには，質量m_iと，３次元の座標$P_i(x_i,\ y_i,\ z_i)$とう２つの属性がある．現実世界のあらゆる物体の力学的な側面は，物体を質点の集合として表現することで良く近似できる．例えば，剛体は互いの距離$|M_i - M_j|$が一定であるような質点の集合として表現できる．

質点系を支配する運動法則には，作用反作用の法則と力学的エネルギー保存の法則がある．それぞれ，次の方程式で表される．

$$f_{ij} = f_{ji}$$

$$m_i \cdot \ddot{P}_i = \sum_j f_{ij} + F_i$$

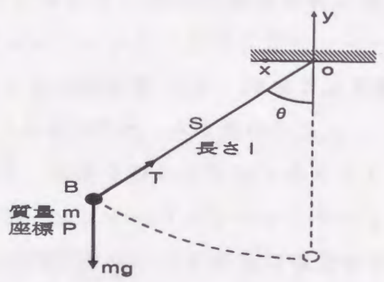


図 7.1: 単振り子

質点系 $\{M_i\}$ に対する動きのシミュレーションとは，すべての質点に対して，座標値 $\{P_i\}$ $(1 \leq i \leq n)$ の時刻 t に対する関数を，以上の運動法則の解として求めることであると定義できる．

7.3 動きの拘束

力学アニメーションでは，系の運動法則と力学的パラメータの指定が必要である．ところが，望んだ動きが得られるような力学的パラメータの決定には試行錯誤が必要である．この課題に対する代表的なアプローチとして，動きの拘束がある．動きの拘束の方を用いることで，利用者が指定すべき力学パラメータの数を減らすことができる．

宇宙空間における惑星の運動は，最も単純な古典力学系である．それぞれの惑星は，他の惑星から引力を受けながら運動する．惑星に十分な外力を与えれば，惑星を自由に運動させることができる．しかし，実世界の力学系の多くは，たとえ外力を加えた場合でも，自由に運動することはない．それは，接触面，接触点，リンクのような動きの拘束が存在するからである．

例えば，図7.1のような単振り子では，玉ＢがひもＳを通して天井の一点ｏにつながって，玉は前後，左右に動くが，ひもを離れて動くことはないという動きの拘束が存在する．この系は，ひもの重さ，空気抵抗は無視し，紐は伸び縮みしないとすると，P をＢの位置とおくことで次の運動方程式が得られる．

$$m \cdot \ddot{P} = (0, -mg, 0) + (-T\sin\theta, T\cos\theta, 0)$$

動きの拘束 C は次のように表現できる．

$$z = 0$$

玉Ｂはある一平面上（この平面をxy平面にとっても以後の議論の一般性を失うことはない）を平面運動する場合，ひもが垂線となす角を θ とおくと，θ はこの系の状態変数である．玉Ｂの位置は θ を用いて次のように表すことができる．

$$P = (l\sin\theta, -l\cos\theta, 0)$$

一般に，P_i の動きの拘束Ｃは，内力 f_{ij}，外力 F_i，質量 m_i，座標 P_i，時刻 t の間の関係に関する方程式である．

もし，質点系において，拘束という概念を考えずに動きを表現するならば，質点系 $\{M_i\}$ に作用しているすべての力：内力 f_{ij}，外力 F_i を時間の関数として指定する必要がある．このやり方は非現実的である．

7.4 動きの拘束の表現法

系の拘束Ｃから系を構成するすべての質点の座標値 $\{P_i(t)\}$ を計算機が求めるにはさまざまな手法がある．力学シミュレーションでは，物体の運動に関する運動の法則が系の種類によって変わるため，系の種類ごとに運動法則を記述する必要がある．従って，運動法則の記述には，力学に関する専門的な知識が必要である．利用者が，簡単に力学シミュレーションを行なうには，系の記述が容易に行なえるような拘束の解法が必要である．この観点から，拘束力よる方法と，ラグランジェの運動方程式による方法の２つが適している．

ある系を構成する複数の質点 M_i が何らかの拘束に従って運動するとき，この M_i 間には，拘束を維持させるような力が働いている．この力を拘束力という．拘束力による方法では，利用者は，系の運動法則の記述では，拘束を表す

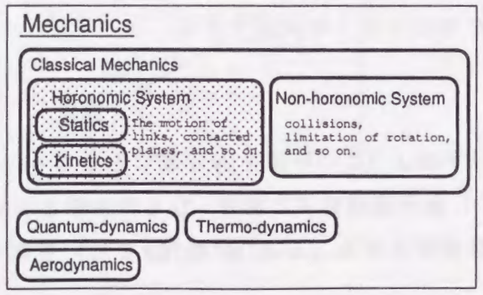


図 7.2: 力学におけるホロノミック系の位置付け

方程式のみを記述する．運動を求めるには他にニュートンの運動方程式が必要であるが，ニュートンの運動法則は系の種類で変わらないため，利用者は改めて記述する必要がない．

計算機は，次の手順で動きを求める．まず利用者が記述した拘束の方程式から，系を構成する全質点間の拘束力が求まる．次に，この拘束力をニュートンの運動方程式に代入して，内力 f_{ij} と外力 F_i の項を消去する．そうして，系の状態変数 S_i が求まることから，全質点の位置，Pが求まる．

例えば，単振り子の拘束 Cs を表す拘束の方程式は内力Tを用いて次のように記述できる．

$$|T| = mg\cos\theta$$

$$T = (-mg\sin\theta, mg\cos^2\theta - mg, 0)$$

この式から，玉Bへの外力の総和Fが求まり，運動Pも求まる．

$$F = (-mg\sin\theta\cos\theta, -mgsin^2\theta, 0)$$

$$P = (l\sin\theta, -l\cos\theta, 0)$$

ただし，$\frac{d^2\theta}{dt^2} = -mg\sin\theta$. $\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta$.

以上の例で，TからFやPを求める部分は計算機に行なわせる．以上の物体間の拘束から拘束力を求める方式について，すでに，我々は，オブジェクト指向の観点から考察を行なった[63]．

力学形のうち，ホロノミック系では，ラグランジェの運動方程式を用いる方が一般的である．ホロノミック系は，系がm個の自由度を持ち，系が持つ自由度mの個数だけの変数の組 $q_k(1 \leq k \leq m)$ の関数として，すべての座標値 P_i が表現できるような系として定義されている．ホロノミック系は，大学教養課程レベルの力学を網羅していて，ほとんどの場合の動きの表現が可能である7.2．例えば，ホロノミック系は図7.3に示した力学要素の表現能力がある．

ラグランジェの運動方程式では，利用者は系の状態変数 q に対してラグランジアンLを求める必要がある．ラグランジアンLは，運動エネルギーTと位置エネルギーVによって次のように定義されている．

$$L(q, \dot{q}, t) \equiv T(q, \dot{q}, t) - V(q, t)$$

ラグランジアンLを次の式に代入することにより，運動方程式が求まる．こうして得られた運動方程式を特にラグランジェの運動方程式という．

$$\frac{d}{dt}\left[\frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}_i}\right] - \frac{\partial L(q, \dot{q}, t)}{\partial q_i} = 0$$

振り子の場合，状態変数は θ の１つであり，運動エネルギーT，位置エネルギーV，ラクランジアンL，ラグランジェの運動方程式は次の通りである．

$$T = \frac{1}{2}ml^2\dot{\theta}^2.$$

$$V = mgl(1 - \cos\theta).$$

$$L = \frac{1}{2}ml^2\dot{\theta}^2 - mgl(1 - \cos\theta).$$

$$\frac{\partial L}{\partial \dot{\theta}} = ml^2\dot{\theta}, \frac{\partial L}{\partial \theta} = -mgl\sin\theta.$$

$$ml^2\ddot{\theta} = -mgl\sin\theta.$$

以上のように，ラグランジェの運動方程式では，利用者は状態変数 q_i を選び，状態変数に対するラグランジェの運動方程式を立てる．計算機は，ラグ

ランジェの運動方程式から，状態変数の時間変化を求める．ラグランジェの運動方程式は，一般に，時間を独立変数とし，一般座標を時間の未知関数とする二階の(連立)微分方程式である．

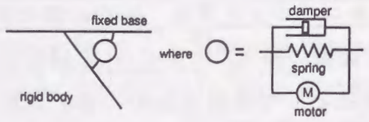


図 7.3: 力学要素

ラグランジェの運動方程式を，拘束力による方法と比べると，（1）拘束力を求める必要がないので，計算の処理が短いが，（2）ラグランジェの運動方程式では，記述可能な系はホロノミック系に限られているという相違点がある．しかし，力学シミュレーションの多くは，ホロノミック系として処理できるか，ホロノミック系として近似したシミュレーション結果に対して付加的な処理を行なうことでできるため，ラグランジェの運動方程式の方が使われることが多い．

7.5 複振り子のラグランジェ運動方程式

ＭＯＶＥの応用の一つとして力学シミュレーションを行なった．ＭＯＶＥでは，シミュレーションの対象をＭＯＶＥの系オブジェクトとして表現する必要がある．ＭＯＶＥで表現可能な系は，有限個の状態変数を持つ系（集中定数系）に限られる．今回は，高校・大学教養課程レベルの力学の問題のうち構造を持った力学系として，複振り子の実装を行なった．

図7.4に示すように，固定点oに長さl_1の糸を結び，その下端に質量m_1のおもりを吊す．これからもう1つの長さl_2の糸で質量m_2のおもりを吊す．この系を1つの鉛直面内で運動させる．l_1，l_2の糸が鉛直線となす角をそれぞれθ，ϕとする．ただし，糸のたるみや伸縮はないものとする．この系の状態変数はθ，ϕである．θ，ϕに関するラグランジェの運動方程式は次の通り．

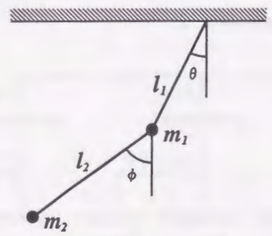


図 7.4: 複振り子の運動特徴

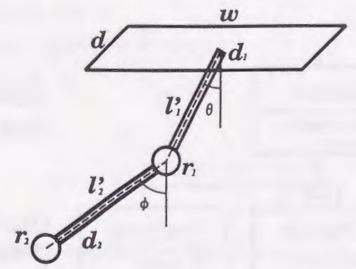


図 7.5: 複振り子の形状特徴

図 7.6: 複振り子の静止画像

$$\begin{bmatrix} (m_1+m_2)l_1^2 & m_2 l_1 l_2 \cos(\phi-\theta) \\ m_2 l_1 l_2 \cos(\phi-\theta) & m_2 l_2^2 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{bmatrix}$$

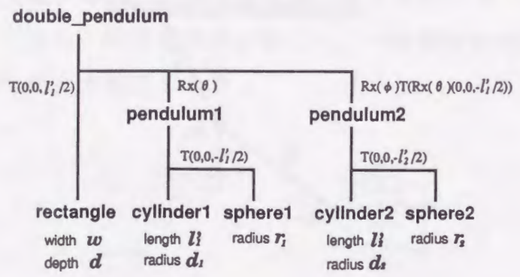


図 7.7: ＭＯＶＥにおける複振り子のオブジェクト構造

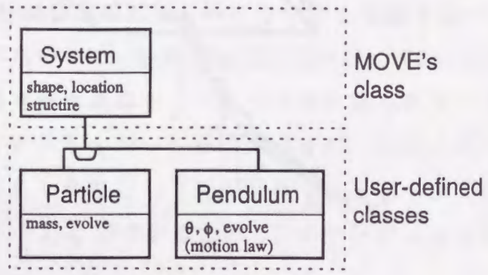


図 7.8: 複振り子に関するクラス定義

$$= \begin{bmatrix} m_2 l_1 l_2 \sin(\phi-\theta)\dot{\phi}^2 - (m_1+m_2) l_1 g \sin\theta \\ -m_2 l_1 l_2 \sin(\phi-\theta)\dot{\theta}^2 - m_2 l_2 g \sin\phi \end{bmatrix}$$

５章で示したように，系オブジェクトでは，運動特徴と形状特徴量を属性値として，動きをメソッドとして保持していた．この方程式からも分かるように，状態変数は，θ，ϕ，運動特徴は，m_1，m_2，l_1，l_2，$\theta(0)$，$\dot{\theta}(0)$，$\phi(0)$，$\dot{\phi}(0)$ である．運動特徴のうち，m_1，m_2，l_1，l_2 が複振り子の構造を表すパラメータであり，$\theta(0)$，$\dot{\theta}(0)$，$\phi(0)$，$\dot{\phi}(0)$ は，ラグランジェの方程式に対する初期値である．形状特徴の選び方は複振り子の視覚的な見せ方によって変わる．例えば，複振り子の形状を，１枚の板，２つの円筒，２つの球という３種類，５個の基本的な３次元物体の組合せとして，図7.5のように表現する場合，r_1，r_2，l_1'，l_2'，d_1，D_2，d，w（図7.7）が形状特徴である．最後に，系の動きは

• 一般座標の数

```
f = 2;
```

• ラグランジェの方程式

```
Matrix A(2,2);
Vector B(2);
A(0,0) = (m1+m2)*l1*l1;
A(0,1) = A(1,0) = m2*l1*l2*cos(q(1)-q(0));
A(1,1) = m2*l2*l2;
B(0) = m2*l1*l2*sin(q(1)-q(0))*dq(1)*dq(1)
        - (m1+m2)*l1*g*sin(q(0));
B(1) = -m2*l1*l2*sin(q(1)-q(0))*dq(0)*dq(0)
        - m2*l2*g*sin(q(1));
ddq = A.lu(B);
```

• 運動特徴

```
l1 = 0.3, l2 = 0.4; // the length of rod [m]
m1 = 0.4, m2 = 0.6; // the mass of ball[kg]
```

• 状態変数の初期値

```
q(0) = 0.5236;    // θ [rad]  (30度)
q(1) = 1.0472;    // φ [rad]  (60度)
dq(0) = 0.0;      // θの角速度 [rad/s]
dq(1) = 0.0;      // φの角速度 [rad/s]
```

図 7.9: 力学ソルバへの入力例（複振り子）

ラグランジェの運動方程式が表現していることから，ラグランジェの運動方程式そのものを系のメソッドとして表現する．

7.6 古典力学のシミュレーションの実装

今回ＭＯＶＥに実装した力学アニメーションシステムにおけるシミュレー

```
5.0 0.591676 0.581358 -5.01031 0.880143 -2.50945 -15.355
5.2 0.4262 -2.98694 -23.1317 0.28289 -2.45594 11.2879
5.4 -0.32913 -2.78071 23.0547 -0.155087 -3.02611 -14.2496
 :      :       :       :        :       :       :
 :      :       :       :        :       :       :
9.0 -0.83466 -2.39986 32.9925  -0.653296 2.26049 -10.2263
```

図 7.10: 力学ソルバの出力例（複振り子）

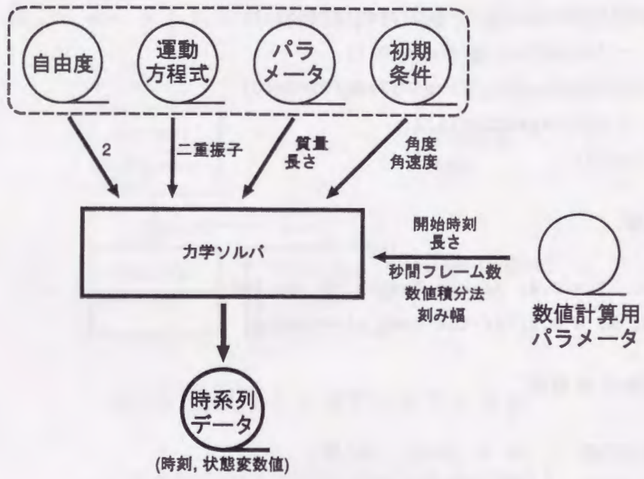


図 7.11: ラグランジェの運動方程式のソルバ

ションは，ラグランジェの運動方程式を解くことで行なう．ここでは，系のメソッドとして表現されたラグランジェの運動方程式を解き，系の状態変数の時間変化を求める役割を持った力学ソルバをオブジェクトとして実装した．

ラグランジェの運動方程式は連立微分方程式で記述可能である．そこで，力学ソルバには，図7.6に示した，（1）系の状態変数の個数，（2）系のラグランジェの運動方程式，（3）運動方程式である微分方程式に与えられる初期値，（4）系の運動特徴，形状特徴の４種類を入力として与える．すると，

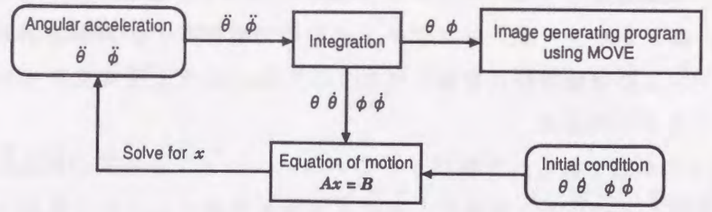


図 7.12: ＭＯＶＥの数値積分オブジェクトの処理

```
State st(f);
Integrator *integ;
Euler *eu = new Euler( h, dt );
ModifiedEuler *me = new ModifiedEuler( h, dt );
RungeKutta *rk = new RungeKutta( h, dt );
integ = rk;
st.Calculation( integ, start, length );
```

図 7.13: 力学ソルバを利用するＣ＋＋プログラム例

一般座標の値の時間変化（図 7.10）が結果として返ってくる．この解は，タイムコードをキーとした，タイムコードの時刻における状態変数の値を列にもつ表の形式である．

一般に，ラグランジェの運動方程式は，時間を独立変数とし，状態変数を時間の未知関数とする二階の連立微分方程式である．ラグランジェの運動方程式は微分方程式なので，解析的に解くことは難しい．そこで，ＭＯＶＥでは，オイラー法，修正オイラー法，ルンゲクッタ法などの数値積分法で解いている．数値積分では，計算すべきデータの開始時刻[秒]，計算すべきデータの長さ[秒]，計算すべきデータのきざみ幅[秒]，数値積分法（オイラー法，修正オイラー法，ルンゲクッタ法のいずれかの指定），数値積分のきざみ幅[秒]を指定が必要である．数値積分では，指定されたきざみ幅で数値積分を繰り返す（図7.12）．

すでにＭＯＶＥには，Ｃ＋＋から簡単に利用できるような形式で，オイラー法，修正オイラー法，ルンゲクッタ法の力学ソルバが実装されている．ラグランジェの運動方程式を解くには，図7.6に示したようなＣ＋＋プログラムを書くことで行なう．

以上の手順で得られた複振り子のアニメーションを巻末の図Ａに示している．複振り子の場合と同様に，系のクラスを定義することで複振り子以外の力学系のシミュレーションも可能である．付録１には，さまざまな力学系の状態変数，運動法則，運動特徴を示している．巻末の図Ｂには，倒立振子のアニメーションを示している．

一般に，ＭＯＶＥにおけるアニメーションの作成は，次の６段階で行なう．

i) 実体クラス，系クラスの定義ＭＯＶＥデータベースへの登録
ii) 動きのモデルの定義
iii) （必要なら）カメラ，光源などの定義
iv) オブジェクトの生成
v) スコアの指定
vi) アニメーションの描画

7.7 グラフィカルユーザインタフェース

シミュレーションとビジュアライゼーションを利用したアニメーションの作成は試行錯誤が行なわれることが多い．そのための使いやすいインタフェースが重要である．特に，初心者が，容易にアニメーションの作成，修正を行なえるようなインタフェースが重要である．例題として，力学系のシミュレーションとビジュアライゼーションのインタフェースの開発を行なった．

7.8 グラフィカルユーザインタフェースの機能

マウスやウインドを利用するグラフィカルユーザインタフェースは初心者にとくに有効である．ＭＯＶＥには，系オブジェクト操作用のグラフィカルユーザインタフェースが準備されている（図7.14，巻末の図Ｃ）．このインタ

フェースを用いることで，アニメーションオブジェクトの検索とアニメーション化を行なうこともできる．

機能は，次の５種である．

i) アニメーションの選択：
データベース内のアニメーションオブジェクトの一覧表が表示される．利用者は一つの系を選ぶ．選んだ系に対しては，グラフ，プレビュー，アニメーション，説明表示の４種類の基本操作が可能である．例えば，『単振子』オブジェクトの選択後，"Graph" ボタンを選ぶと，単振子の運動のグラフが表示される．

ii) シミュレーション
シミュレーションの実行は，アニメーションオブジェクトを選択した時点で自動的に行なわれる．シミュレーションでは，形状特徴と運動特徴の設定が可能である．

iii) 説明表示
説明表示では，選択した系の現象を説明するようなテキストの表示を行なう．テキストの内容は，説明文，図，一般座標のとり方，パラメータ，運動方程式である．

iv) グラフ：
グラフのボタンを押すと，シミュレーション結果のグラフが表示される．グラフでは，選択した系の時間に対する一状態変数の時間変化をグラフで表示する．グラフでは，表示する時間の範囲，表示形式，グラフの精度が指定可能である．グラフの表示形式には，デカルト座標，円柱座標，極座標の３種類が選択可能である．

v) プレビュー：
プレビューは，ある時刻での系の状態を１枚静止画像として表示する機能である．プレビューは，見たい時刻の指定が可能である．

vi) アニメーション：
アニメーションオブジェクトを選んだあとで，次に，"アニメーション" のボタンを押すと，アニメーションが計算され，アニメーション表示ウ

インド上に表示される．

vii) アニメーションの再生，逆再生，一時停止：

アニメーション表示ウインドは，ビデオテープレコーダと同様のインタフェースと持っている．アニメーションの表示中は，アニメーションの再生，逆再生，一時停止，初期状態に戻すリセットの4つのボタンが使用可能である．再生と逆再生ボタンを選んだ時は，表示と同時にタイムゲージも動く．ストップボタンを押すことでアニメーションの表示を一時停止することができ，さらにタイムゲージを動かすことで任意の時刻にからの再生が可能である．

viii) 運動特徴，形状特徴，状態変数表示

アニメーションの表示中は，表示に連動して，一般化座標(速度，加速度)など運動特徴，形状特徴，状態変数の値も表示される．

ix) 運動特徴と形状特徴の変更

"Change"ボタンを押すとウィンドウが開き，そこで初期値，質量や長さなどの運動特徴と形状特徴の変更を行なうことができる．そうして変更を行なった後で再び基本操作を行なうと，シミュレーションが再び行なわれる．この機能によって，運動特徴と形状特徴の変更とアニメーションの表示という繰り返しを容易に行なうことができる．

x) シミュレーションとアニメーションのパラメータの変更

パラメータには，アニメーションの１秒間のフレーム数，数値計算の積分器の種類，数値計算の精度，アニメーションの開始時刻，アニメーションの長さがある．これらのパラメータも変更可能である．積分器には，オイラー法，修正オイラー法，ルンゲクッタ法の３種類が実装されている．

力学シミュレーションでは，系の動きを予測することが困難なため，望みの動きを得るまでには試行錯誤が必要になることが多い．試行錯誤は，パラメータを変更して繰り返しシミュレーションし，アニメーションを表示させることで行なう．８，９，１０番目の機能は，そのための機能である．"Change"ボタンを押すとウィンドウが開き，そこでパラメータを変更することができる．

7.9 Tcl/Tkによるインタフェースの開発

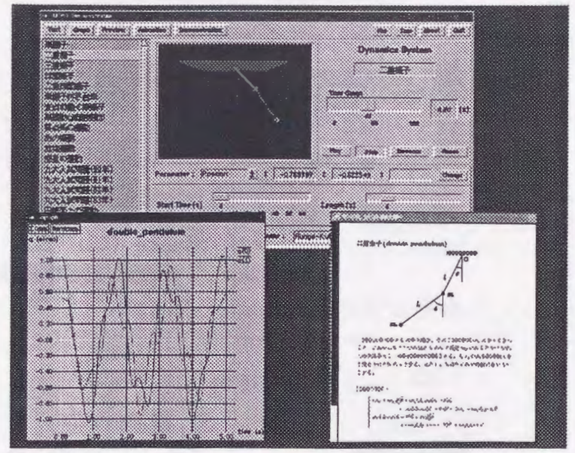


図 7.14: 力学アニメーション用インタフェースの動作画面

GUIは，オブジェクト指向のGUIツールキットであるTcl/Tkを用いて開発した．Tcl/Tkは，コマンド言語Tclと，グラフィカルユーザインタフェース構築のための部品集であるTkとから構成される．Tcl/Tkを用いると，高度なウィンドウベースのアプリケーションの作成が容易である．

開発の手順は，Tcl/Tkがライブラリとして持っている，フレーム，ボタン，メニュー，メニューボタン，スケール，ラベル，リストボックス，スクロールバー，エントリ，ラジオボタン，キャンバスというという１１種類の部品を組み合わせることで行なった．Tcl/Tkの代表的な部品であるボタン，メニュー，メニューボタン，スケール，ラベル，リストボックス，スクロールバーには次のような機能がある．

- ボタン

  Tcl/Tkのボタンオブジェクトには，-commandオプションを用いてコマンドを指定することができる．指定されたコマンドは，利用者がそのボタンへマウスカーソルを移動させ，マウスの１ボタンを押すことで呼び出

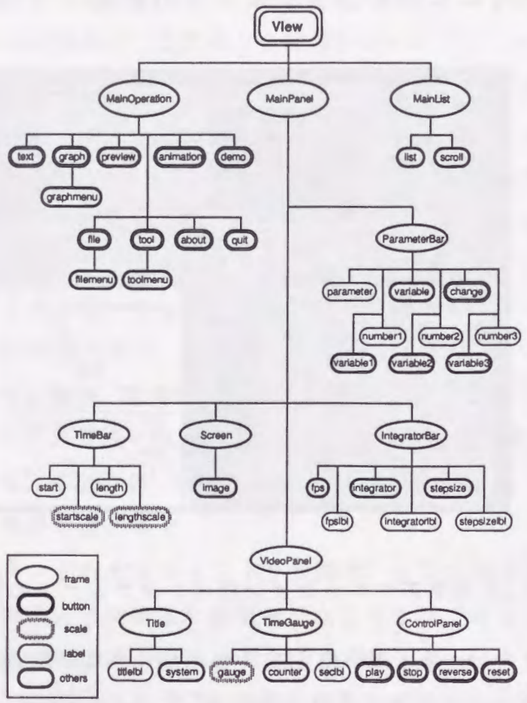


図 7.15: Tcl/Tk 部品の階層構造

すことができる.

- メニュー

  メニューは，文字列のリストを表示するための部品である.

- メニューボタン

  メニューに表示された文字列そのものをボタンとして利用するための部品がメニューボタンである.

- スケール

  長方形とスライダーとを組み合わせたものがスケールである．長方形は，スライダーの稼働範囲を決定する．スライダーの位置はマウスをドラッグすることで動かすことができる．長方形内におけるスライダーの相対

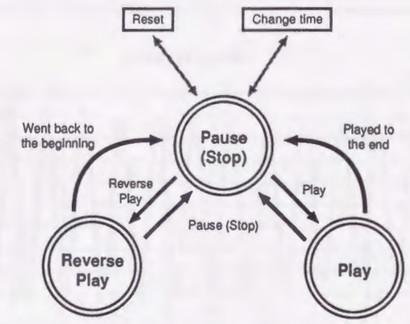


図 7.16: VideoPlayer の状態遷移

位置によって，スケールの値が決定する.

- ラベル

  文字列を表示するためのボタンがラベルである.

- リストボックス

  リストボックスは，メニューと同様，文字列のリストを表示するための部品である．リストボックスは，要素としての文字列を挿入，削除することができる.

- スクロールバー

  スクロールバーは，メインとなるウインド，２つの矢印を両端に持った１つの長方形，スライダーの３つから構成される．スライダー位置を動かすことで，ウインド内の表示が上下方向に動く．スライダーの位置はマウスで動かすことができる.

今回の実装では，オブジェクトの検索にメニューとリストボックスを，メソッドの引数の決定にスケールを，メソッドの選択と実行にボタンを利用した.

7.10 ユーザインタフェース実装の考察

図 7.15には，我々が開発したGUI の部品の階層構造を示している．階層は木構造になっていて，木の葉がTcl/Tk の部品に対応している．Tcl/Tk がオブ

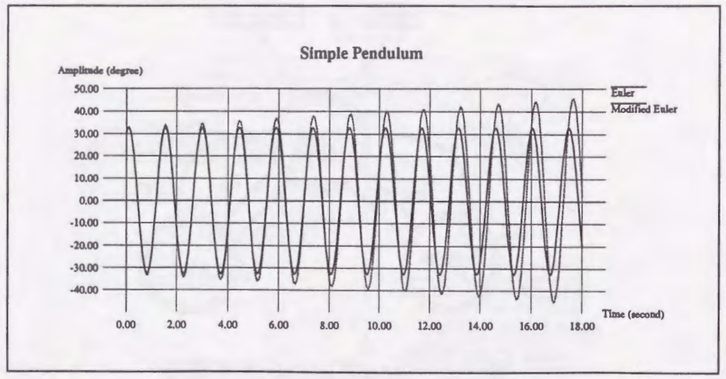


図 7.17: オイラー法と修正オイラー法の比較

ジェクト指向であったことから，アニメーションを表示する機能の状態遷移（図7.16）を自然に実装することができた．

積分器には，オイラー法，修正オイラー法，ルンゲクッタ法を実装したが，オイラー法は誤差が大きい．今回のユーザインタフェースのグラフ表示の機能によって，これら３種類の誤差の大きさを容易に比較することができた7.17ので，単にアニメーションを表示するだけでなく，場合によって他の表示手段を用いることが有効であることが分かった．

写実的な３次元グラフィックスの表現には，３次元物体の形状や位置，物体表面の光線反射，仮想カメラの位置，光源の位置，光源からの光線の拡散の特性の表現が必要である．これらの定義をいかに行なうかという課題に対して，

第 8 章

描画の高速化

8.1 ウオークスルー

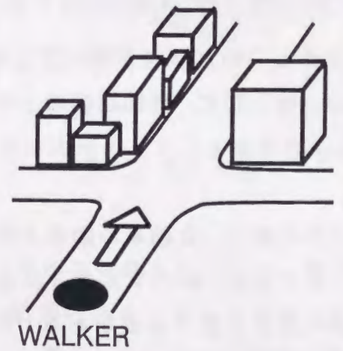


図 8.1: ウオークスルー

２次元の地図情報システムは，カーナビゲーション，環境評価などに利用されている．一方，３次元のデータを必要とするような，景観シミュレーション，仮想現実感などの応用の必要性が高まっている．３次元グラフィックスアニメーションで地図情報の表示を行なうと，建物の高さが直観的に把握できるなどの利点がある．

地図情報システムへの３次元グラフィックスの適用の代表的な応用例として，ウオークスルーがある．ウオークスルーは，仮想的な視点が，建築物，道

路など仮想的な３次元的世界の中を動き回り．景観をシミュレーションするものである（図8.1）．ウオークスルーを表示することで，あたかもそのシーンの中を歩いているかのような感覚を得ることができる．ウオークスルーは、都市の景観を分かりやすく表示することができることから、都市開発のための有効なプレゼンテーションの手段として普及している．このことから，３次元コンピュータアニメーションを利用したウオークスルーは、重要な研究分野となった．巻末の図Ｃには、九州大学の箱崎キャンパス全景のコンピュータグラフィックスの絵を示している．

8.2 ＭＯＶＥ上のウオークスルー用データベース

8.2.1 ウオークスルー用データのＭＯＶＥへの定義

地図情報のウオークスルーの必要性が高いことから，我々は，ＭＯＶＥ上のアプリケーションの一例として，地図情報のウオークスルーを実装した．我々は，ウオークスルーの対象として，九州大学の箱崎キャンパスとその周辺を選んだ．

建築物の形状，カメラの動き，自動車の動きと形状などのデータからウオークスルーを行なう．ところが，ＭＯＶＥの３次元物体のクラスが持たないような固有の属性を持つ実体を扱う必要がある．九州大学の箱崎キャンパスにおいて，こういった固有の属性を持つ実体は，建築物，道路，自動車，木の４種類である．例えば，建物固有の属性には，名称，番号，所属，材質，階数がある．以上のように，アプリケーション固有のデータの対象として実世界のモデル化を行なう．

形状の記述には，ＭＯＶＥのShape3dクラスをそのまま利用することができる．建築物は，名前，材質，階数などの，ＭＯＶＥの３次元物体のクラスが持たないような属性を含むので，ＭＯＶＥのクラスをそのまま使う場合と，固有の属性がデータベースに格納できない．ＭＯＶＥでは，実体の種類ごとにクラスを定義することで，固有の属性を持つ実体のデータベース化を行なう．そこで，固有の属性を定義したクラスをObject3dのサブクラスとして定義する．

ウオークスルーにおいて，カメラや自動車などは運動する．動きについては，５章で示した方法で実装した．

以上のモデリングの結果得られるクラス階層を，付録３に示している．

8.2.2 ウオークスルー用データの作成

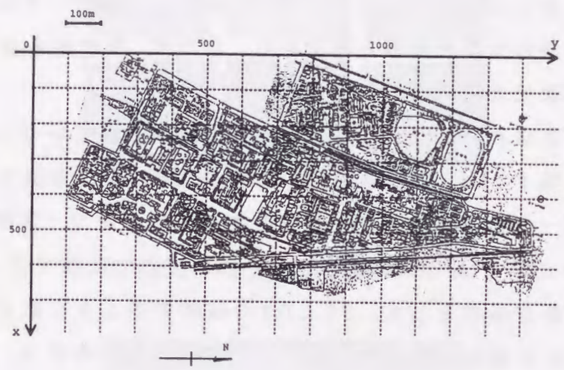


図 8.2: 施設図面

ウオークスルーには、描画の対象となる物体に関する色と形状のデータが必要である．ところが、迫力のあるウオークスルーの作成には，多量のデータが必要である．従って，データの作成の手間が大きい．

ウオークスルーにおいて必要な建物，道路などの３次元形状データが必要である．近年、２次元の地図をベースとした地図情報システムの普及により、都市の建物、道路、等高線などの形状の２次元データは容易かつ安価に入手可能となった、そのことから，２次元の地図情報システムの２次元のデータを，３次元に変換することが一般に行なわれている．我々も，３次元データ生成ツールを開発し，３次元への変換を行なった．我々は，最も普及した２次元の地図情報データベースの１つであるＺＭＡＰのデータを利用した．３次元への変換は次の手順で行なう．（1）ＺＭＡＰの２次元物体のデータをファイル形式で取り出し，（2）３次元データに変換し，（3）３次元のデータをＭＯＶＥに格納という手順をとった．

図8.2に示した２次元形状データはＺＭＡＰの１ユニット分を示している．データ量は，約100Kバイトである．人間が生活する現実の空間において，地表面をxy平面，鉛直方向をz軸に沿って取る．すると，ＺＭＡＰにおいて建物などの形状は，z方向に沿ってxy平面に投射した多角形として表現されている．３次元化コンバータは，ＺＭＡＰの２次元的な建物データを対象として３次元化を行なう．３次元コンバータが出力するデータは，３次元物体を構成する多角形の頂点座標と色である．そこで，投影面を底面に持つような多角柱を３次元化の結果として得た．

ただし，ＺＭＡＰには高さや色のデータは含まれていないので，正確な高さや色を表現することはできない．そこで，何らかの方法で高さや色や形状に関するデータを確率的なデータとして与えることで，３次元化を行なう．３次元化の処理では，物体毎に高さや色を適当に乱数で変える．高さを7.6 m，色は灰色を基本として，0.6±0.1の範囲で明るさを変え，適当な色を付けた．見かけの現実性を向上させるために，乱数の分布は，実際の航空写真から決定した．

8.2.3 アニメーション作成

対象は，九州大学の箱崎キャンパスを中心とした範囲１．６ｋｍ×２．０ｋｍである．データベースは，５１８３の建築物（ポリゴン数でおよそ１００００）で構成される．１建築物あたり，およそ２０ポリゴンで構成される．データベースサイズは，ＯＮＴＯＳ３．０を用いて，およそ１４．７Mバイトであった．

以上で完成した，九州大学のシーンのデータベースをもとに，ウオークスルーを作成した（図8.3，巻末の図Ｅ，巻末の図Ｆ）．手順は，まずＭＯＶＥデータベースに九州大学とその周辺のシーンを作成する．次に，このシーン内にカメラと光源のオブジェクトを加え，カメラの位置と向きを時間的に変化させることで，ウオークスルーを得た．カメラと自動車の動きは，あらかじめ指定された経路に沿って運動するものとして表現した．

同一の視点を複数の場所から同時に表示した方が，１つの場合よりも周囲

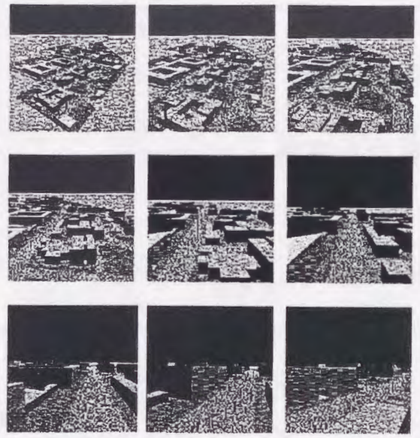


図 8.3: 九州大学ウオークスルー

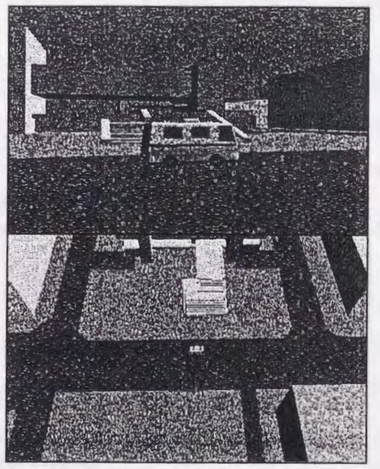


図 8.4: 同期再生中の２つのアニメーション

の状況の把握が容易である．九州大学箱崎キャンパスを表現するシーンオブジェクトについて，カメラオブジェクトを２つ生成し，２つのトラックから

構成されるシーンを得ることができた．そうして作ったマルチトラックのシーンは，表示の際にも同期して再生される（図8.4）．

8.2.4 描画における課題

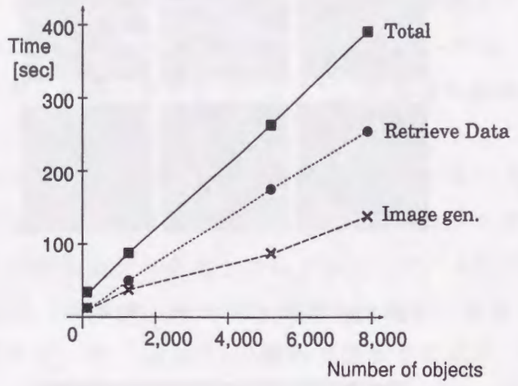


図 8.5: データベースサイズと処理時間（インデックス無し）

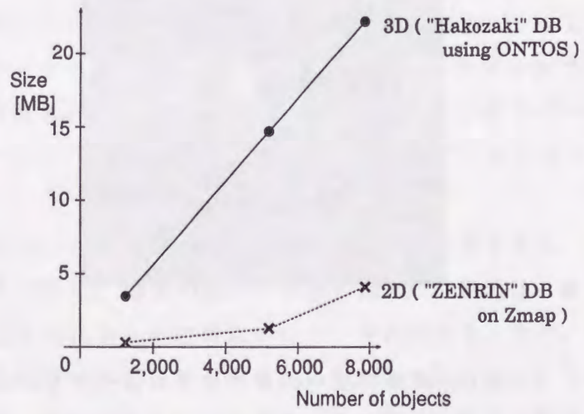


図 8.6: ２次元と３次元のデータベースサイズの比較

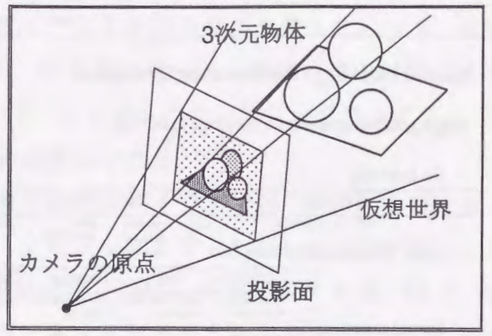


図 8.7: 可視範囲

性能測定を，ＳＵＮＳｐａｒｃＳｔａｔｉｏｎ１０／４０ＺＸ，メモリ４８Ｍバイト，Ｓｏｌａｒｉｓ２．３上で行なった．図8.5には，６４０×４８０の大きさのフルカラーイメージの描画にかかった時間を示している．この図から分かるように，描画時間はデータに比例する．

描画では，グラフィックスデータからイメージを合成する画像生成時間と，ディスクからメモリ上へデータを転送する検索時間がかかる．

画像生成は現在ソフトウエアで行なっているためかなりの時間がかかっている．しかし，ＳＵＵ ＳｐａｒｃＳａｔｉｏｎ１０ＺＸのグラフィックスハードウエアは３１，０００ポリゴン／秒の性能を持つことから，ＭＯＶＥをグラフィックスハードウエアの性能を最大限に発揮するように改良すれば，画像生成時間は１０倍から２０倍程度高速化できるので問題はないと考える．

測定結果では，画像生成時間よりも検索時間の方が時間時間がかかっている．図8.6に示したように，同じ対象の地図情報が，２次元と３次元では，３次元の方がデータ量が多い．従って，３次元データを扱うアプリケーションでは，２次元の場合よりも検索時間の問題が重要である．

8.2.5 描画への範囲検索の利用

ウオークスルーのグラフィックスデータはディクス上に存在する．従来のグラフィックスパッケージの多くは，描画時にグラフィックスデータをすべ

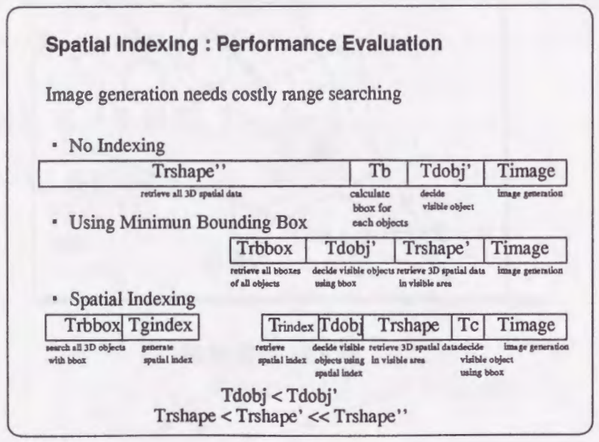


図 8.8: 各種の検索処理と処理時間の関係

てメモリ上に読み込むような方式になっている．ところが，視点から見える範囲は図8.7のようにカメラを頂点とする４角錐領域である．従って，本当は見えないために不要なデータもディスクから読み込んでいる．そのため，データベースサイズにほぼ比例して処理時間が増えている．これが，検索に時間がかかっている原因である．

描画に不要なグラフィックスデータをディスクから読み込まないで済むには，４角錐領域の範囲と交差するような３次元物体の検索を行なうことが有効である．これは一種の範囲検索であるが，この範囲検索の特徴として，（1）対象が広がりを持った３次元物体である，（2）検索領域が４角錐領域である，という２つの特徴がある．範囲検索に関して，３次元の空間データデータベースや計算幾何学の立場から，さまざまな研究が行なわれてきた[14][9][5]．しかし，従来の範囲検索の研究の多くは点データ(x,y,z)の集合への矩形領域の範囲検索を行なっており，広がりを持った３次元物体に対する４角錐領域の範囲検索はあまり研究がなされていなかった．

描画における範囲検索の特徴として，画像生成時に実体の基本面に関する隠れ面消去が行なわれることから，範囲検索の答えとしては指定された範囲

に交差する立体だけでなく余分な立体を含んでいても，画像生成上問題がない．この特徴から，広がりを持った３次元物体に対する４角錐領域の範囲検索の方法には，（1）ミニマムバウンディングボックス，（2）空間インデックスの２つの方法が考えられる．

ミニマムバウンディングボックスでは，広がりを持った３次元物体は，その実体を含むような最小の直方体（ミニマムバウンディングボックス）で近似される．ミニマムバウンディングボックスとは，xy, yz, zx 平面のいずれかに平行な６つの面で構成される直方体である．データベースには，その実体の属性としてミニマムバウンディングボックスの右上と左下の頂点の座標を格納する．検索時には，個々の実体ごとに，ミニマムバウンディングボックスの属性を読み込んで実体の可視性の判定を行ない，その結果候補となった実体のみの形状データを読む．

空間インデックスでは，ミニマムバウンディングボックスでは，物体の個数をN，表示される物体の個数をnとして，検索時間はO(N)となる．空間インデックスは，あらかじめ与えられた実体集合に対してある程度の時間を書けて前処理を行ない，理想的にはディスクからの転送時間をO(logN) + O(n)までの向上を目指す方法である．空間インデックスでは，データベースサイズに対する処理時刻のオーダーがミニマムバウンディングボックスより小さいため，多量のデータを扱う場合に有効である．

以上をまとめると，画像処理において，（1）範囲検索を行なわない，（2）ミニマムバウンディングボックス，（3）空間インデックスの３つの場合の処理時間を比較すると，図8.8のようになる．

8.3 空間インデックスの実装

地図情報のウオークスルーでは，図8.9のような５角形領域の範囲検索として行なう．これは，２次元の範囲検索である．２次元の範囲検索の空間インデックスは，メッシュ, Ｒ＊－Ｔｒｅｅなどいろいろな種類がある．最適なインデックスは，データの空間分布，視点の位置によって変わる．

メッシュでは，２次元の空間を等間隔のメッシュで分割する．ＭＯＶＥで

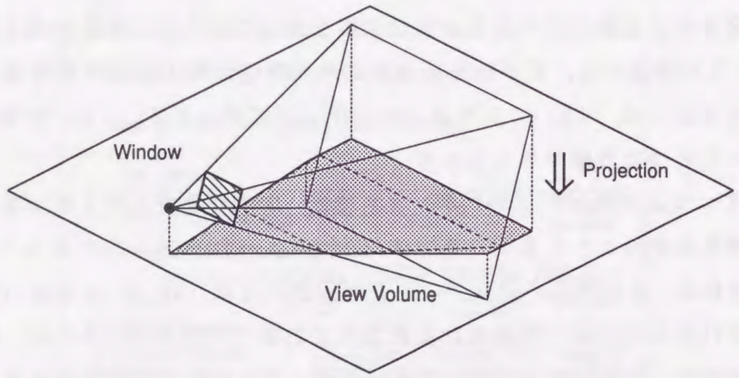


図 8.9: 描画時における範囲検索

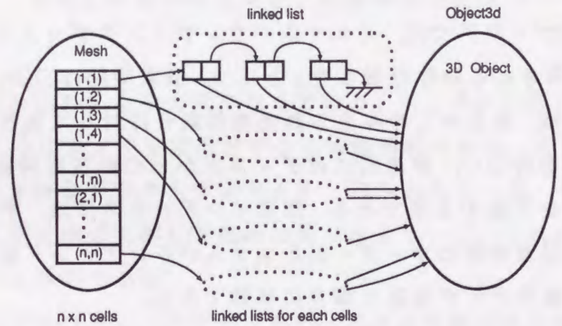


図 8.10: メッシュにおけるデータの格納構造

は，メッシュは図8.10のような構造になっており，メッシュの各区画はCellクラスのオブジェクトである．Cellには，メッシュと交差する３次元物体のＯＩＤの集合が格納されている．範囲検索は，（1）与えられた範囲と交差するメッシュを求め，（2）次に３次元物体のＯＩＤの集合を求める，という手順で行なう（図8.11）．

Ｒ＊－ｔｒｅｅでは，広がりを持った空間データを木構造で格納する．Ｒ＊－ｔｒｅｅの各ノードには，そのノードを根とするサブツリーに含まれる

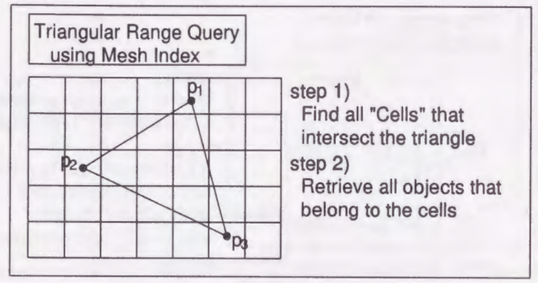


図 8.11: メッシュを用いた範囲検索

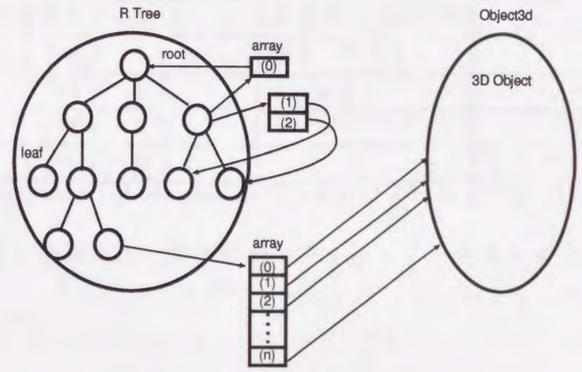


図 8.12: Ｒ＊－ｔｒｅｅにおけるデータの格納構造

空間データのすべてを含むようなミニマムバウンディングボックスが格納されている．Ｒ＊－ｔｒｅｅの空間分割は，各ノードのミニマムバウンディングボックスの重なりが最小となるように行なう．

ＭＯＶＥでは，Ｒ＊－ｔｒｅｅの構造は図8.12のようになっていて，Ｒ＊－ｔｒｅｅの各リーフには，３次元物体のＯＩＤが格納されている．Ｒ＊－ｔｒｅｅへの範囲検索は，図8.13に示したようにＲ＊－ｔｒｅｅの各ノードと与えられた範囲との交差を再帰的に検査することで行なう．

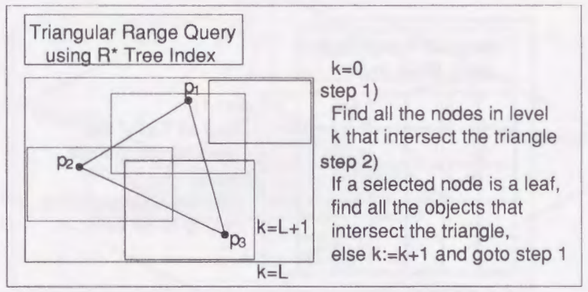


図 8.13: Ｒ＊－ｔｒｅｅを用いた範囲検索

	インデックス無し	メッシュ	Ｒ＊－Ｔｒｅｅ
検索	１５７．０秒	８７．２秒	７１．１秒
画像生成	８６．２	４９．６秒	４９．７秒
生成時間		３１．５秒	１５５．９秒
インデックスサイズ		０．３９Ｍバイト	１．４３Ｍバイト

d=1,600m, α=56, height=225m, 640×480 full color image

表 8.1: メッシュとＲ＊－Ｔｒｅｅの，処理時間，生成時間，インデックスサイズの比較

表８．１には，メッシュとＲ＊－Ｔｒｅｅの，処理時間，生成時間，インデックスサイズの比較を行なっている．空間インデックスを用いることで，描画の高速化ができたことが分かる．メッシュの方が，サイズも小さく生成時間も速いが，検索時間が遅いことが分かる．

図には，ウオークスルー生成における各フレームごとの描画時間を示している．画像生成の時間は各フレームでおよそ同じであるが，検索時間には激しい変動があり，特に１枚目に時間がかかることが分かる．これは，ＭＯＶＥではオブジェクト指向データベースＯＮＴＯＳ上に実装したおり，ＯＮＴＯＳでは１度ディスクからメモリ上に転送したオブジェクトはメモリ上に置かれる．２枚目以降で検索時間が速くなっていることは，ＯＮＴＯＳがウオ

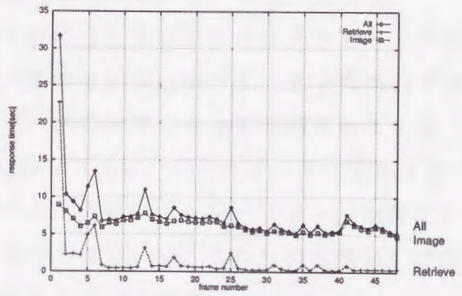


図 8.14: アニメーションのフレーム数と処理時間

ークスルーの間グラフィックスデータをメモリ上にキャッシュしたことの効果を示している．

第９章

おわりに

２章では，アニメーション分野の巨大データの検索と共有の必要性と，３次元グラフィックスアニメーションの動作基盤としてのグラフィックスサーバの重要性を説明した．従来の３次元グラフィックス用のパッケージではデータをファイルで管理しているため，グラフィックスサーバとして利用することが困難である．一方，従来のアニメーションデータベースは，グラフィックスサーバへの適用が可能であるが，実際に利用するには描画の機能面で不十分であった．３次元グラフィックスの分野では，アプリケーションごとに必要となるデータが異なるため，アプリケーションごとにデータベースの定義を行なう必要がある．ところが，従来のアニメーションデータベースは，データベース部分と描画部分とが独立した構造になっており，データベースからグラフィックスデータを検索し描画部に渡すことで描画を行なっているため，新しい種類のアプリケーションデータを扱う際に，データベースの定義だけでなく，描画部に関する実装が必要であった．

３章では，３次元アニメーションアニメーションの新しいプラットホームとしてのＭＯＶＥを提案した．ＭＯＶＥは，従来のアニメーションデータベースが可能としていたデータの共有だけでなく，データ定義の共有，描画部の共有を可能とする．そのことで，３次元グラフィックスアニメーションのアプリケーション開発，アニメーション作成の支援を行なう．この観点から，従来のアニメーションデータベースとＭＯＶＥを比較すると，ＭＯＶＥは，（１）機能面で拡張されていて，（２）アプリケーションごとに必要なデ

ータの定義が容易に可能である，という２つである．

４章では，ＭＯＶＥの３次元グラフィックスアニメーション用基本モデルの説明を行なった．基本モデルはオブジェクト指向のクラスとして，オブジェクト指向データベースＯＮＴＯＳ上に実装されている．ＭＯＶＥの基本クラスは，利用者がＭＯＶＥ上に，グラフィックスアルゴリズム、データ構造、立体音響アルゴリズム、力学シミュレーション、空間インデックスなどの定義ができるように設計されている．これらの定義は，ＭＯＶＥの基本クラスのサブクラスを定義することで行なう．

４章では，基本モデルの構造とデータ操作の説明も行なった．ＭＯＶＥ上でのアニメーション作成と描画は，主に（１）シーン描画，（２）シーンへの実体の追加、退場、（３）シーンにおける実体の位置、姿勢の変更，（４）シーンの複製の生成の４つのデータ操作を用いて行なう．これらのデータ操作は，ＭＯＶＥの基本クラスのメソッドとして実装されているので，利用者にとって使いやすい．

ＭＯＶＥの基本モデルは，“イメージ”、”グラフィックス”、”画像合成アルゴリズム”の３種類のデータをオブジェクトとして，これら３種類のデータの間の関連をオブジェクト間のリンクとして表現している．そのことで，動画像、音声という２種のマルチメディアのイメージデータと，グラフィックスとの統合を行なっている．この基本モデルには，動画像と音声の再生時の同期が自動的に行なえるなどの利点がある．

本論文の５章で示した動きの定義の方法は，まず，シーンに登場する実体集合を状態変数を持った系という単位に分ける．系の状態変数と見え方は時間と共に変化するが，その時間変化は形状特徴，運動特徴という２種類のパラメータと，運動法則，見え方の規則で表現する方法である．系は，３次元物体，光源，カメラ，音源，マイクロホンの５種の実体の集合として定義されているので，３次元物体と光源の動きの同期など，異なった種類の実体の動きの同期を扱うことが可能である．動きの操作は，形状特徴，運動特徴というパラメータの変更として行なうことができる．利用者は，動きの定義を，ＭＯＶＥのSystemクラスのサブクラスの定義として行なうことができる．動き

の定義の実例として等速直線運動などを示した．

動きのある系を格納する場合の問題に，動きの描画がある．シーンには，多数の系が存在するから，描画が難しい．ＭＯＶＥでは，系ごとに状態値を時刻を引数として検索するようなメソッドを定義することで，利用者にとって描画を容易に行なえるようにした．

６章では，ＭＯＶＥ上に，（１）ポリゴンメッシュ法による自由局面の表現，（２）局所反射モデルによる色付け（３）ポリゴンの隠面消去とシェーディングアルゴリズムという画像生成アルゴリズム，の３つの実装を行なった．この実装によって，ＭＯＶＥに代表的な画像生成アルゴリズムの実装が可能なことを実証した．

７章では，ＭＯＶＥの実際のアプリケーションとして，古典力学におけるホロノミック系のシミュレーションを実装した．ホロノミック系の運動法則は，ラグランジェの運動方程式で記述できる．そこで，ラグランジェの運動方程式とそのソルバ（力学ソルバ）をＭＯＶＥ上に実装した．

ＭＯＶＥへの力学系の実装では，力学系の種類ごとにクラスを定義した．一方，力学系の形状モデルと視覚化のための機能はＭＯＶＥに準備されたものをそのまま利用することが可能であった．こうしてＭＯＶＥ上に表現した力学系の動きは運動特徴によってパラメータ化されている．従って動きの操作は，系の属性として定義された運動特徴の値の操作として行なうことができる．この運動特徴の値の変更を容易に行なうためのグラフィカルユーザインタフェースも実装した．

８章では，ウオークスルーのような多量のデータを必要とする場合について，ＭＯＶＥの適用性を検討した．８章では，多量のデータの描画を高速にするためには，４角錐領域の範囲検索を高速に行なうための空間インデックスが有効であることを示した．そして，地図情報に適した２種類の空間インデックス，（１）メッシュ，（２）Ｒ＊－ｔｒｅｅを実装した．

本研究では，アニメーションデータを，オブジェクト指向データベース上のオブジェクトとして格納するという，ＭＯＶＥの基本的なアイデアを示した．オブジェクト指向を用いたことによって，３次元グラフィックスアニメ

ーションのアプリケーションが必要とする多様な種類の実体を扱うことが可能であった．アニメーションデータは，階層構造を持った複合オブジェクトとして表現され，要素オブジェクトの共有，再利用が可能である．

ＭＯＶＥの基本モデルは，従来のアニメーションデータベースのデータモデルと比較して，イメージとグラフィックスの関連情報の表現，写実性のある描画，空間インデックス定義のためのメカニズム，動きの定義や描画を行なうためのメカニズムなどの機能が増えている．従って，ＭＯＶＥは，従来のアニメーションデータベースを機能面で拡張したシステムであるといえる．

今回の研究の結果，ＭＯＶＥの基本モデル上に，グラフィックスデータ構造、形状モデル，画像合成アルゴリズム，立体音響アルゴリズム、力学シミュレーション、動きの定義，アプリケーションの定義，空間インデックスの機能の定義を行なうことが可能であることを示した．今までに普及した形状モデル、画像合成アルゴリズムの種類の数は有限である．従って，ＭＯＶＥ上に，形状モデル、画像アルゴリズムをすべて実装ことが可能である．そのことで，複数のアプリケーションがＭＯＶＥの描画部を共有することが可能になると予想される．

以上のように，ＭＯＶＥはコンピュータグラフィックスやプログラミングの知識のない一般の技術者にも容易に利用できる．画像，音声，グラフィックスのマルチメディアデータを扱う．アプリケーション開発の支援に有効である．

ＭＯＶＥが扱うことができるデータの範囲はグラフィックスモデルによるアニメーションに限定されている．有限要素法，多変量解析など，グラフィックスモデルとして表現できない応用も多い．

われわれの力学ソルバは，ラグランジェの運動方程式しか解けない．ラグランジェの運動方程式を用いてホロノミック系を表現できる．しかし，ホロノミック系以外の空間衝突判定や運動の拘束などの扱いが今後の課題として残っている．

謝辞

本論文の最後にあたって，日頃から熱心に御指導頂いている九州大学工学部情報工学科の牧之内教授，吉田助教授，黒木助手に感謝致します．ＺＭＡＰ使用に関してご援助下さった（株）ゼンリンに感謝致します．また，九州大学箱崎キャンパスの建築物データ入力に協力いただいた九州大学牧之内研究室の吉川克彦氏に感謝します．力学アニメーションの作成に協力いただいた近藤祐介氏に感謝します．最後に，本研究に関して，熱心な助言を頂いた九州大学工学部情報工学科牧之内・吉田研究室の方々に感謝致します．

参考文献

[1] Alan Watt, and Mark Watt: Advanced Animation and Rendering Techniques, Addison-Wesley (1992).

[2] ANSI (American National Standard Institute), "Americal National Standard for Information Processing Systems–Programmer's Hierarchical Interactive Graphics System
(PHIGS) Function Description, Active File Format, Clear-Text Encoding of Archive File", ANSI, X3.144-1988, ANSI, 1988.

[3] Arnaldi B, Dumont G, Hégron G, Magnenat-Thalmann N, Thalmann D, "Animation control with dynamics.In: *State-of-the-art in computer animation*", Springer, pp.113-124, 1989.

[4] Barnhill, Robert E. , and Richard F. Reisenfeld, "Computer Aided Geometric Design", Academic Press, New York (1987).

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", Proc. 1990 ACM SIGMOD Int. Conf. Management of Data (eds. H. Garcia-Molina and H. V. Jagadish), Atlantic City, pp. 322-331, 1990.

[6] Bjarne Stroustrup, "the C++ Programming Language 2nd edition", Addison-Wesley, 1991.

[7] David P. Anderson and George Homsy, "A Continous Media I/O Server and Its Synchronization Mechanism", IEEE Computer, pp. 51-57, October, 1991.

[8] Eric Grant, Phil Ambum, and Turner Whitted, "*Exploiting Classes in Modeling and Display Software*", IEEE CG & A, November, pp.13-20, 1986.

[9] M. W. Freeston, "The BANG File: A New Kind of Grid File", Proc. 1987 ACM SIGMOD Int. Conf. Management of Data (eds. U. Dayal and I. Traiger), San Francisco, pp. 260-269, 1987.

[10] Geoff Wyvill and Craig McNaughton, "*Optical Models.* In:*CG International '90*", Springer-Verlag, 1990:

[11] Graphical Kernel System for Three Dimensions(GKS-3D), ISO/DIS 8805. International Standards Organization, 1987.

[12] Goldstein, Herbert, "Classical Mechanics 2nd edition", Addison-Wesley, 1983.

[13] Mark Green and Hanqiu Sun, "*A Language and System for Procedural Modeling*

and Motion", IEEE CG & A, November, pp.52-64,1988.

[14] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching", Proc, 1984 ACM SIGMOD Int. Conf. Management of Data, pp. 47-57, 1984.

[15] Hahn, J. K. : Realistic Animation of Rigid Bodies, Computer Graphics, 22(4), pp.299-308.

[16] Hans Rijpkema and Michael Girard, "*Computer Animation of Knowledge-Based Human Grasping*", ACM Computer Graphics, vol.25, no.4, pp.339-348, July, 1991.

[17] ISO, Information processing systems - Computer graphics - Metafile for the storage and transfer of picture description information, ISO 8632 Parts 1 to 4, ISO Centrral Secretariat, Geneva (1987).

[18] James Foley, Andries van Dam, Steven Feiner, and John Hughes, "Computer Graphics Principles and Practice 2nd edition,", Addison-Wesley addison-wesley publishing company, 1990.

[19] James K. Hahn, "*Realistic Animation of Rigid Bodies*", ACM Computer Graphics, vol.22, no.4, pp.299-308, August, 1988.

[20] Jane Wilhelms, "Using Dynamic Analysis for Realistic Animation of Articulated Bodies", IEEE CG & A, June, pp.12-27, 1987.

[21] Kajiya, James T., "*The Rendering Equation*", ACM Computer Graphics 20(4), pp. 143-149, August, 1986.

[22] Kim,W., Chou,H., and Banerjee,J., "Operations and Implementation of Complex Objects", Proc.1987 IEEE Data Engineering Conf., Los Alamitos, CA., 626-633, 1987.

[23] W. Kim and F. H. Lochovsky, "Object-Oriented Concepts, Databases, and Applications", ACM Press, 1988.
Proc. 16th International Conference on VLDB, ,August 1990.

[24] Kunihiko Kaneko, Susumu Kuroki, and Akifumi Makinouchi, "Design of 3D CG Data Model of "MOVE" Animation Database System", Proc. the 2nd Far-East Workshop on Future Database Systems (eds. Q. Chen, Y. Kambayashi and R. Sacks-Davis) , Kyoko, pp. 364-372, 1992.

[25] Kunihiko Kaneko, Susumu Kuroki, and Akifumi Makinouchi, "Towards Dynamics Animation on Object-Oriented Animation Database System "MOVE"", Proc. of the 3rd International Symposium on Database Systems for Advanced Applications(eds. S. Moon and H. Ikeda), Taejon, pp. 3-10 1993.

[26] Tsukasa Noma and Tosiyasu L. Kunii, "ANIMENGINE: An Engineering Animation System", IEEE CG & A, October, pp.24-33, 1985.

[27] Toshiyasu L.Kunii and Linig Sun, "Dynamic Analysis-Based Human Animation.In:*CG International '90*", Springer-Verlag, pp.3-15, 1990.

[28] Susumu Kuroki, Katushiko Kikkawa, Kunihiko Kaneko, and Akifumi Makinouchi, "Walkthrough using Animation Database System MOVE", Proceedings of the

Fourth International Conference on Database and Expert Systems Applications (eds. V. Marik, J. Lazansky and R. R. Wagner), Prague, pp. 760-765, 1994.

[29] Susumu Kuroki, Kunihiko Kaneko, Katushiko Kikkawa, and Akifumi Makinouchi, "Performance Improvement of the Animation Database System MOVE", 1995.

[30] N. Kuwano, T. Kandda, Y.Mohri, "Applications of Object-oriented Databases to Publishing Systems", The proceedings of The Second International Symposium on Database Systems for Advanced Applications, April, 421-429, 1991.

[31] Michael McKenna and David Zelter, "Dynamic Simulation of Autonomous Legged Locomotion", ACM Computer Graphics, vol.24, no.4, pp.29-38, August, 1990.

[32] Mangnenat-Thalmann N, Thalmann D, "*The Use of High-Level 3D Graphical Types in the MIRA Animation System*", IEEE CG & A, pp.9-16, 1983.

[33] Magnenat-Thalmann N, Thalmann D, "CINEMIRA: a 3D computer animation language based on actor and camera data types", Technical Report, University of Montreal, 1984.

[34] Nadia Magnenat-Thalmann and Daniel Thalmann, "An Indexed Bibiography on Computer Animation", IEEE CG & A, July, pp.76-86, 1985.

[35] N. Thalmann, D. Thalmann, and M. Fortin, "MIRANIM: An Extensible Director-Oriented System for the Animation of Realistic Images", IEEE Computer Graphics and Applications, Vol.5, No.3, pp.61-73, 1985.

[36] D Thalmann, "*Motion Control: From Keyframe to Task-Level Animation* In:*State-of-the-art in computer animation*", Springer, pp.3-18, 1989.

[37] N.Magnenat Thalmann, D.Thalmann, "*Computer Animation. Theory and Practice. Second Revised Edition*", Spring-Verlag, 1990.

[38] Mohammed Mahieddine and Jean Claude Lafon, "*An Object-Oriented Approach for Modeling Animated Entities* In:*Computer Animation'90* ", Springer-Verlag, pp.177-187, 1990.

[39] Myeong W. Lee and Tosiyasu L.Kunii, "*Design Methodology for Computer Animation Database System*", Proc. DASFAA, pp.73-79,1989.

[40] Myeong W. Lee and Tosiyasu L.Kunii, "*Animation Platform: A Data Management System for Modeling Moving Objects* In:*Computer Animation'91*", Springer-Verlag, pp.169-186, 1991.

[41] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure", ACM Trans, Database Systems, Vol. 9, No. 1, pp. 38-71, 1984.

[42] Ontologic Inc. , "Ontos Object Database version 3. 0 Developer's Guide", Ontologic Inc. , Burlingtion Mass, 1994.

[43] OpenGL Architecture Review Board, "OpenGL Reference Manual: The Official Reference Document for OpenGL, Release 1", Addison-Wesley, Reading, Massachusetts, 1992.

[44] B.-U. Pagel, J.-W. Six, H. Toben, and P. Widmayer, "Towards an Analysis of Range Query Performance in Spatial Data Structures", Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, Washington D. C. pp. 214-221, 1993.

[45] Parke FI, "Parameterized Models for Facial Animation", IEEE Computer Graphics and Applications, vol. 2, no. 9, pp. 61-68, 1982.

[46] Peter Coad, Edward Yourdon, "*Object-Oriented Analysis 2nd edition*", Prentice Hall, 1991.

[47] Przemyslaw W. Prusinkiewicz, and Aristid Lindenmayer: The Algorithmic Beauty of Plants, Springer-Verlag (1990).

[48] Reynolds CW, "Computer Animation with Scripts and Actors", Proc. SIGGRAPH '82, Computer Graphics 16(3), pp.289-296, 1982.

[49] Pixar Corporation, "*The RenderMan Interface Version 3.0*", Pixar Corporation, San Rafael CA, May, 1988.

[50] Requicha A. A. G: Represention for Rigid Solids: Theory, Methods, and Systems, ACM Computing Surveys, pp.437-464 (1980).

[51] Robert C. Zelenik et al. : An Object-Oriented Frameork for the Integration of Interactive Animation Techniques, ACM Computer Graphics, vol. 25, no. 4, pp. 105-112 (1991).

[52] R.J.Rost, "*PEX Introduction and Overview, PEX Version 3.20*", MIT X Consortium, 1988.

[53] H. Samet, "The Design and Analysis of Spatial Data Structures"

[54] Scheifler, R.W.,J.Gettys, and R.Newman,"X Window System," 1988.

[55] Scott D. Roth, "Ray Casting for Modeling Solids" Computer Graphics and Image Processing 18, 108-144(1982)

[56] B. Seeger and H.-P. Kriegel, "The Bubby-Tree, An Efficient and Robust Access Method for Spatial Data Base System", 16th Int. Conf. on Very Large Data Bases (eds. D. McLeod, R. Sacks-Davis and H. Schek), Brisbane, pp. 590-601.

[57] , Simon Gibbs, "Data Modeling od Time-Based Media", ACM SIGMOD, pp. 91-102, 1994.

[58] P. Strauss and R. Carey, "An Object-Oriented 3D Graphics Toolkit", Proc. SIGGRAPH'92, pp. 341-349, 1992.

[59] S. Teller and C. H. Sequin, "Visibility Preprocessing for Interactive Walkthroughs", Proc. SIGGRAPH'91, pp. 61-69, 1991.

[60] D. Rotem, "Spatial Join Indices", Proc. Data Engineering'91, pp. 500-509, 1991.

[61] O. Guenther, H. Noltemeier, "Spatial Database Indices fot Large Extended Objects", Proc. Data Engineering'91, pp.520-526, 1991.

[62] 金子邦彦，黒木進，牧之内顕文，"アニメーションデータベースシステムＭＯＶＥのアニメーションモデリング"，電子情報通信学会データ工学研究会, 1993.

[63] 黒木進, 金子邦彦, 牧之内顕文 : "アニメーション用データベースにおける時間的知識の表現", 情報処理学会第43回全国大会

[64] 吉川克彦，金子邦彦，黒木進，牧之内顕文，"３次元地図情報データベースの構築とその３次元動画像道案内への応用", 電子情報通信学会機能図形情報システムシンポジウム, pp.67-72, 1993.

[65] 村上公一, 広田克彦, 太田雅明, 柿本正憲, 佐藤弘幸, "3次元アニメーション用モデリング言語", 日経CG,1986年10月号.

[66] 横山孝典, 佐藤秀人, "制約に基づくオブジェクト指向知識表現システム", 情報処理学会論文誌, vol.31, no.1, pp.68-75.

付録１　力学系の運動特徴

名前	変数	個数	変数に関する微分方程式	パラメータ
質点	x, y, z	3	$\ddot{x} = \frac{F_x}{m},\quad \ddot{y} = \frac{F_y}{m},\quad \ddot{z} = \frac{F_z}{m}$	m, F
単振子	θ	1	$\ddot{\theta} = -\frac{g}{l}\sin\theta$	l, g
単振動(摩擦あり)	x	1	$\ddot{x} = \frac{1}{m}(-kx - c\dot{x})$	m, k, c
Atwood の機械	x	1	$\ddot{x} = \frac{m_1 - m_2}{m_1 + m_2}g$	m_1, m_2, g
梯子	θ	1	$\ddot{\theta} = \frac{3}{2a}g\sin\theta$	a, g
放物体(空気抵抗あり)	x, y	2	$\begin{cases} \ddot{x} = -\frac{c\dot{x}}{m}\sqrt{\dot{x}^2 + \dot{y}^2} \\ \ddot{y} = -g - \frac{c\dot{y}}{m}\sqrt{\dot{x}^2 + \dot{y}^2} \end{cases}$	m, c, g
二重振子	θ, ϕ	2	$\begin{cases} (m_1 + m_2)l_1^2\ddot{\theta} + m_2 l_1 l_2 \cos(\varphi - \theta)\ddot{\varphi} \\ \quad = m_2 l_1 l_2 \sin(\varphi - \theta)\dot{\varphi}^2 - (m_1 + m_2)l_1 g \sin\theta \\ m_2 l_1 l_2 \cos(\varphi - \theta)\ddot{\theta} + m_2 l_2^2 \ddot{\varphi} \\ \quad = -m_2 l_1 l_2 \sin(\varphi - \theta)\dot{\theta}^2 - m_2 l_2 g \sin\varphi \end{cases}$	m_1, m_2, l_1, l_2
単振子つき台車	z, θ	2	$\begin{cases} (M + m)\ddot{z} - ml\cos\theta \cdot \ddot{\theta} = F - \mu\dot{z} - ml\dot{\theta}^2 \sin\theta \\ \cos\theta \cdot \ddot{z} - l\ddot{\theta} = g\sin\theta \end{cases}$	M, m, l, F, μ
惑星の運動	r, θ	2	$\begin{cases} \ddot{r} = r\dot{\theta}^2 - \frac{GM}{r^2} \\ \ddot{\theta} = -2\frac{\dot{r}\dot{\theta}}{r} \end{cases}$	M, m, G
球面振子	θ, ϕ	2	$\begin{cases} \ddot{\theta} = \dot{\phi}^2 \sin\theta\cos\theta - \frac{g}{l}\sin\theta \\ \ddot{\phi} = -2\dot{\theta}\dot{\phi}\frac{\cos\theta}{\sin\theta} \end{cases}$	m, l
円錐面上の運動	r, θ	2	$\begin{cases} \ddot{r} = r\sin^2\alpha \cdot \dot{\theta}^2 - g\cos\alpha \\ \ddot{\theta} = -2\frac{\dot{r}\dot{\theta}}{r} \end{cases}$	α
質点系の振動(ばね)	x_1, x_2	2	$\begin{cases} \ddot{x_1} = \frac{1}{m_1}(-c_1 x_1 + k(x_2 - x_1)) \\ \ddot{x_2} = \frac{1}{m_2}(-c_2 x_2 - k(x_2 - x_1)) \end{cases}$	m_1, m_2, k, c_1, c_2
質点系の振動(糸)	x_1, x_2	2	$\begin{cases} \ddot{x_1} = \frac{1}{m_1}(-\frac{S}{a}(2x_2 - x_1)) \\ \ddot{x_2} = \frac{1}{m_2}(-\frac{S}{a}(2x_1 - x_2)) \end{cases}$	m_1, m_2, S, a
支点の動きうる単振子	ξ, θ	2	$\begin{cases} \ddot{\xi} + l\cos\theta \cdot \ddot{\theta} = l\sin\theta \cdot \dot{\theta}^2 - \frac{k}{m}\xi \\ \cos\theta \cdot \ddot{\xi} + l\ddot{\theta} = -g\sin\theta \end{cases}$	m, l, k, g

付録２　ＭＯＶＥのクラス一覧

補助

- データベースオブジェクト

  DInstance, TypeId

- 系

  NameList, TStateVariable

- ベクトル・空間座標

  TVector4d, TVector3d, TUnitVector3d, TZeroVector3d, TAffinePoint3d, TPoint3d

- 変換

  TTransform, TPerspective, TProjection, TGeometrical, TScaling, TIdentity, TZeroTransform, TOrthgonal, TTranslation, TRotation, TAboutXAxis, TAboutYAxis, TAboutZAxis

- データ構造

  DArray2d, DArray, DAggregate, DSLink, DSLinkNode, DStack, TBSTree, TBStreeIterat
  TBSTreeNode, DIterator, DSLinkIterator, DStackIterator, TAggregate, TArray, TInstance, TIterator, TSLink, TSLinkIterator, TSLinkNode

- その他

  TMatrix4d

グラフィックスアルゴリズム

- 実体

  SP(Entity)

- ３次元形状

  SP(Articulated), SP(Composite), SP(NFFShape3d), SP(MGFPatch),

- 基本形状

  SP(PrimitiveSurface), SP(Prism), SP(Block), SP(Cube), SP(Cone), SP(Cylinder),

SP(Disc), SP(Torus), SP(Ellipsoid), SP(RegularPolyhedron), SP(Sphere), SP(Polygon3d), SP(TriPatch), SP(QuadPatch), SP(PolyPatch), SP(Patch), SP(Vertex), SP(Bezier),

- 表面属性

  SP(SurfaceShader), SP(SurfaceAttribute), SP(Texture), RGBColor

- カメラ

  SP(Camera), SP(ScanlineZbuffer)

- 光源

  SP(LightSource), SP(DistantLight), SP(PointLight), SP(AmbientLight)

空間インデックス

SpatialIndex, Cell, CellArray2d, Mesh, RstarTree,

アプリケーション

- 実体

  Pendulum, Particle, Building, Road, Tree, Car

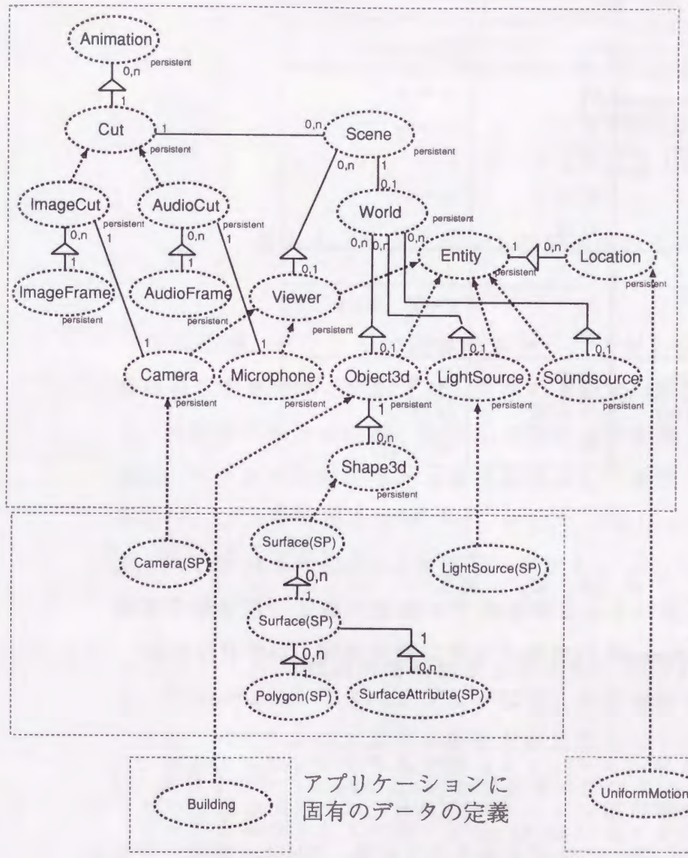
- 運動

  UniformMotion, RotateMotion, AccelatedMotion

- その他

  DynamicsSolver

付録 3　クラス階層



図Ａ３．２ＭＯＶＥのクラス階層階層

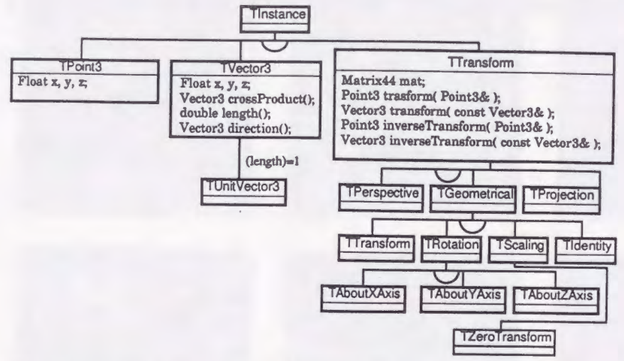
付録 4　ベクトル演算・空間座標

class	operations	mathematical expression
Vector3 (x, y, z, 0)	和	$v_1 + v_2$
	差	$v_1 - v_2$
	スカラー倍	$k \cdot v$, $v \cdot k$, v/k
	内積	$v_1 \cdot v_2$
	外積	$v_1 \times v_2$
	長さ	$\mid v \mid$
Point3 (x, y, z, 1)	和	$p + v$, $v + p$
	差	$p - v$
Transform (4x4matrix)	変換	$M \cdot a$
	逆変換	$M^{-1} \cdot a$
	積	$M_1 \cdot M_2$

v, v_1, v_2　：ベクトル　k　：実数
p　：点　a　：アファインベクトル
M, M_1, M_2　：変換

表Ａ４．１ベクトルと空間座標の演算

３次元グラフィックスアニメーションに関するアルゴリズムの実装では，空間座標やベクトルが頻繁に用いられる．MOVEでは，空間座標とベクトルのためのクラスTVector3d, TPoint3dが実装されている．TVector3dは，３次元のベクトル(x,y,z)をTPoint3dは３次元の点(x,y,z)を表現する．３次元ベクトルと３次元空間座標の属性はともに(x, y, z)で同じである．しかし，３次元ベクトルと３次元空間座標に対する演算は全く異なっている（表Ａ４．１）．例えば，ベクトルには長さが定義されているが，点には長さは定義されていない．従って，ＭＯＶＥでは，３次元ベクトルと３次元空間座標は異なったクラスとして実装した．



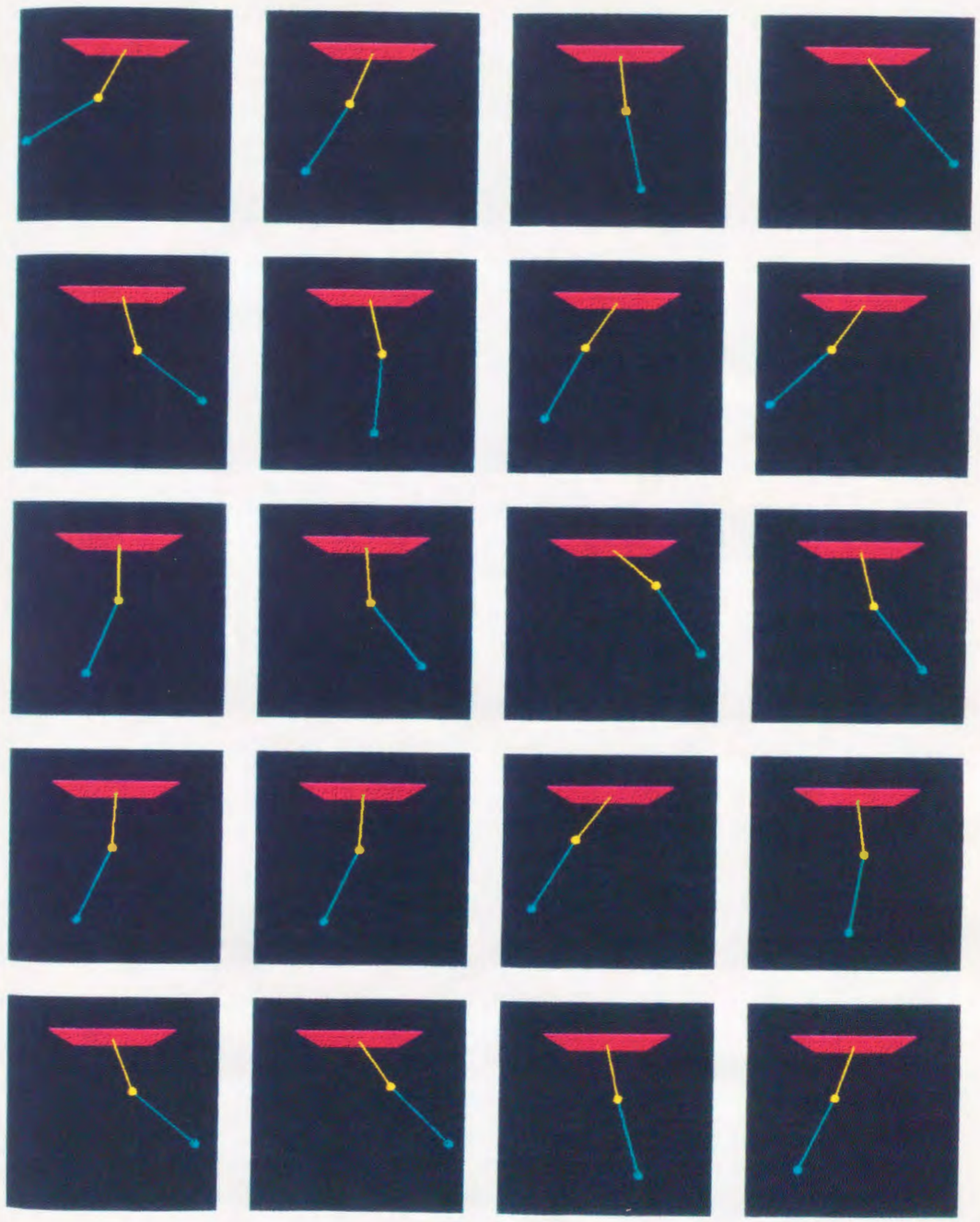
図Ａ４．１ベクトルと空間座標のためのクラス設計

空間座標，ベクトルの演算の中で，グラフィックスで最も用いられる演算は幾何変換である．例えば，実体の運動は，ある時刻ｔでの位置・姿勢L_tと，時刻０における位置・姿勢L_0の間の幾何変換として表現できる．幾何変換は，ベクトルや空間座標を同次座標として表現することで，自然に実装できる[18]．３次元の同次座標では，点(x,y,z)を$(x \times W, y \times W, z \times W, W)$(Ｗは実数，但しＷは０でない)と表現し，ベクトル$(x,y,z)$を$(x,y,z,0)$と表現する．同次座標を用いた幾何変換の手法は確立している．

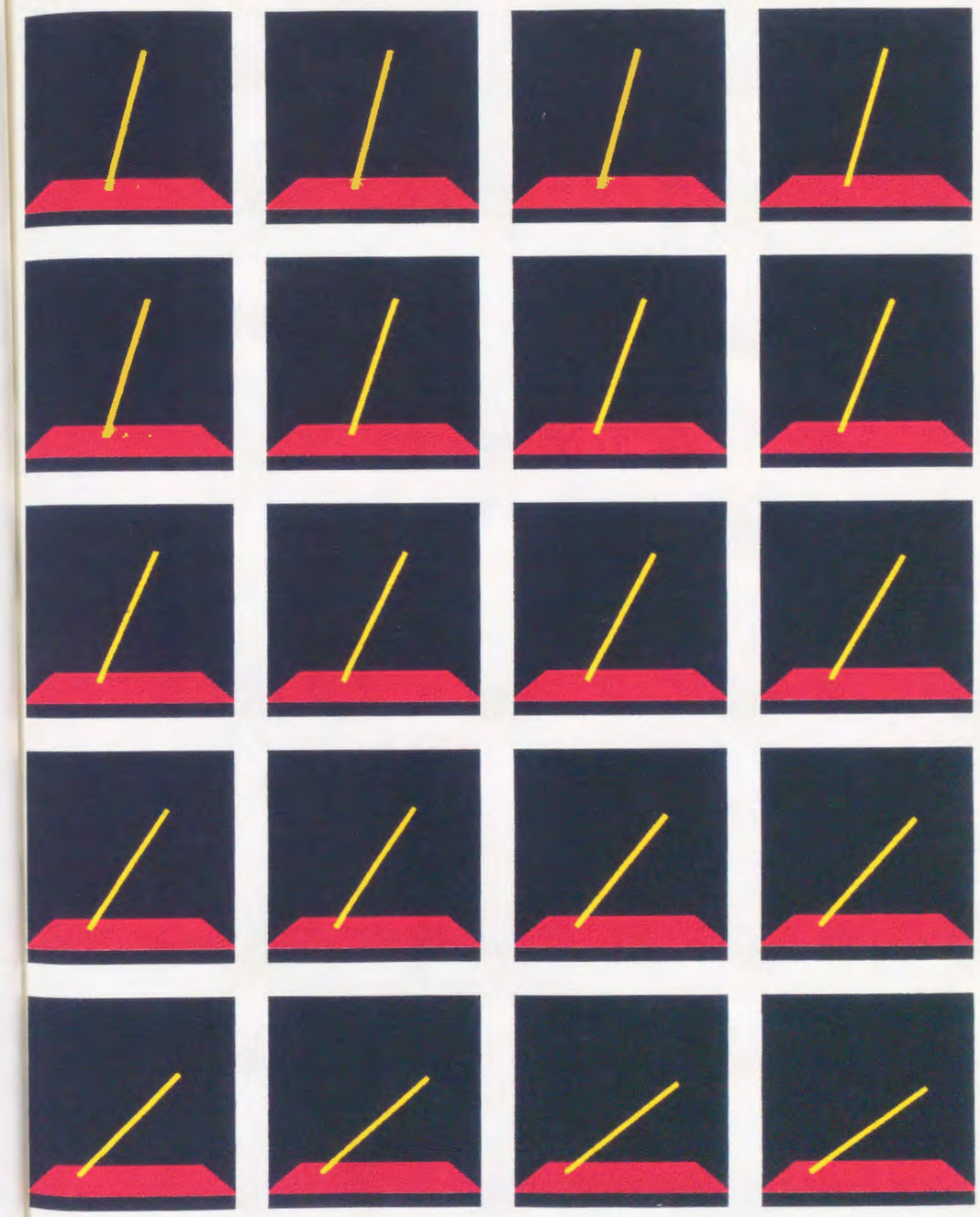
ＭＯＶＥでは，同次座標に関する処理はTTransformクラスに表現されている．TTransformクラスのメソッドには，点の幾何変換，点の投影，ベクトルの回転，ベクトルのスカラー倍などが実装されている．TTransformクラスには，変換を行なうtransform()と逆変換を行なうinverseTransform()の２つのメソッドを実装されている．TTransformは４×４の行列を属性として持つ．従って，線形変換の指定には１６個のパラメータが必要である．１６個のパラメータすべてを指定するのは面倒なので，Transformクラスのサブクラスを実装した．サブクラスの利点は，（1）指定すべきパラメータの個数を減らすことができること，（2）逆変換のパラメータを求めるためのメソッドがより高速なものに多重定義されていることの２つである．

以上のように，空間座標やベクトルに対する演算は，ＭＯＶＥのTVector3d,

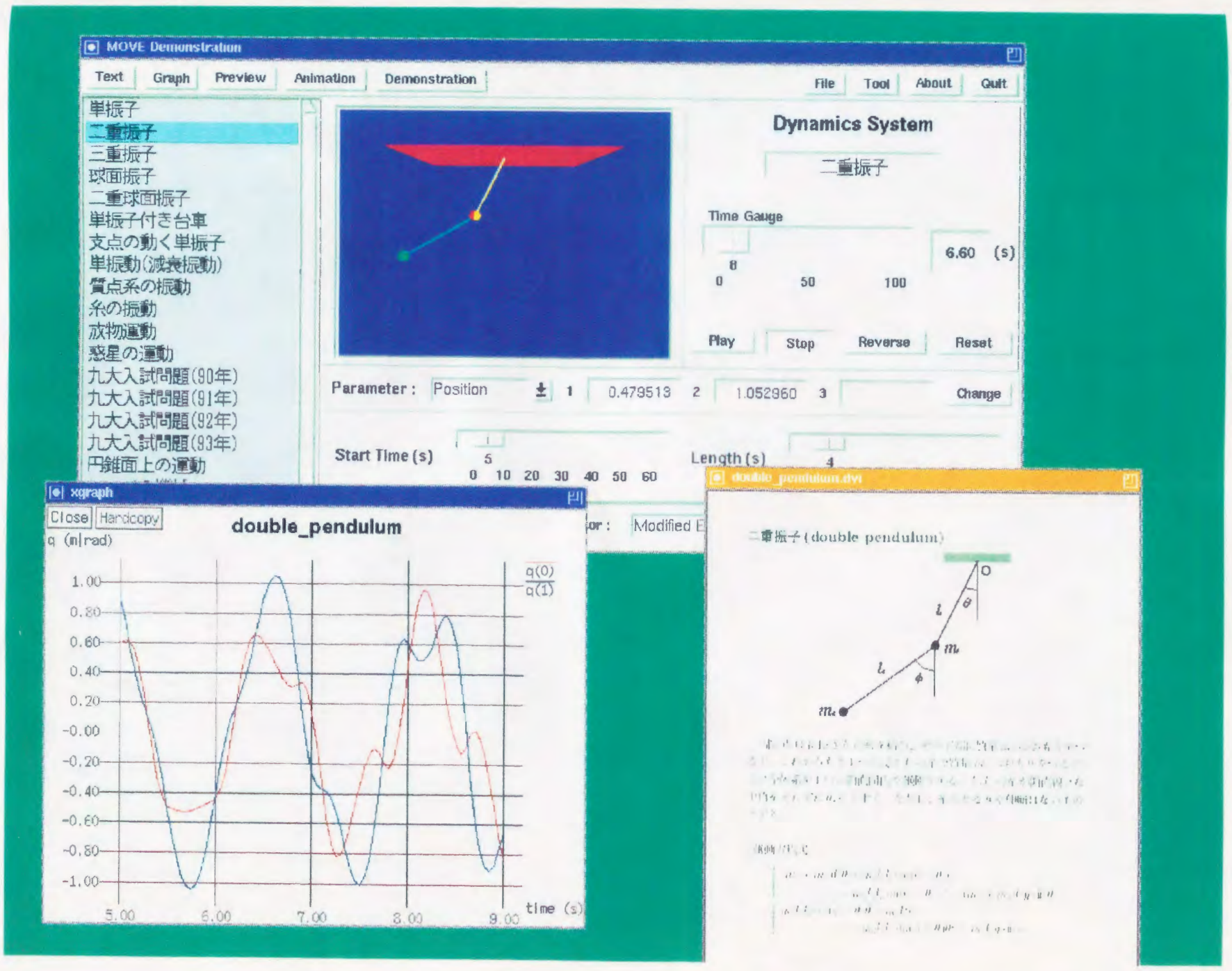
TPoint3d, TTransform クラスとそのサブクラスのメソッドとして実装されている（図Ａ４．１）．



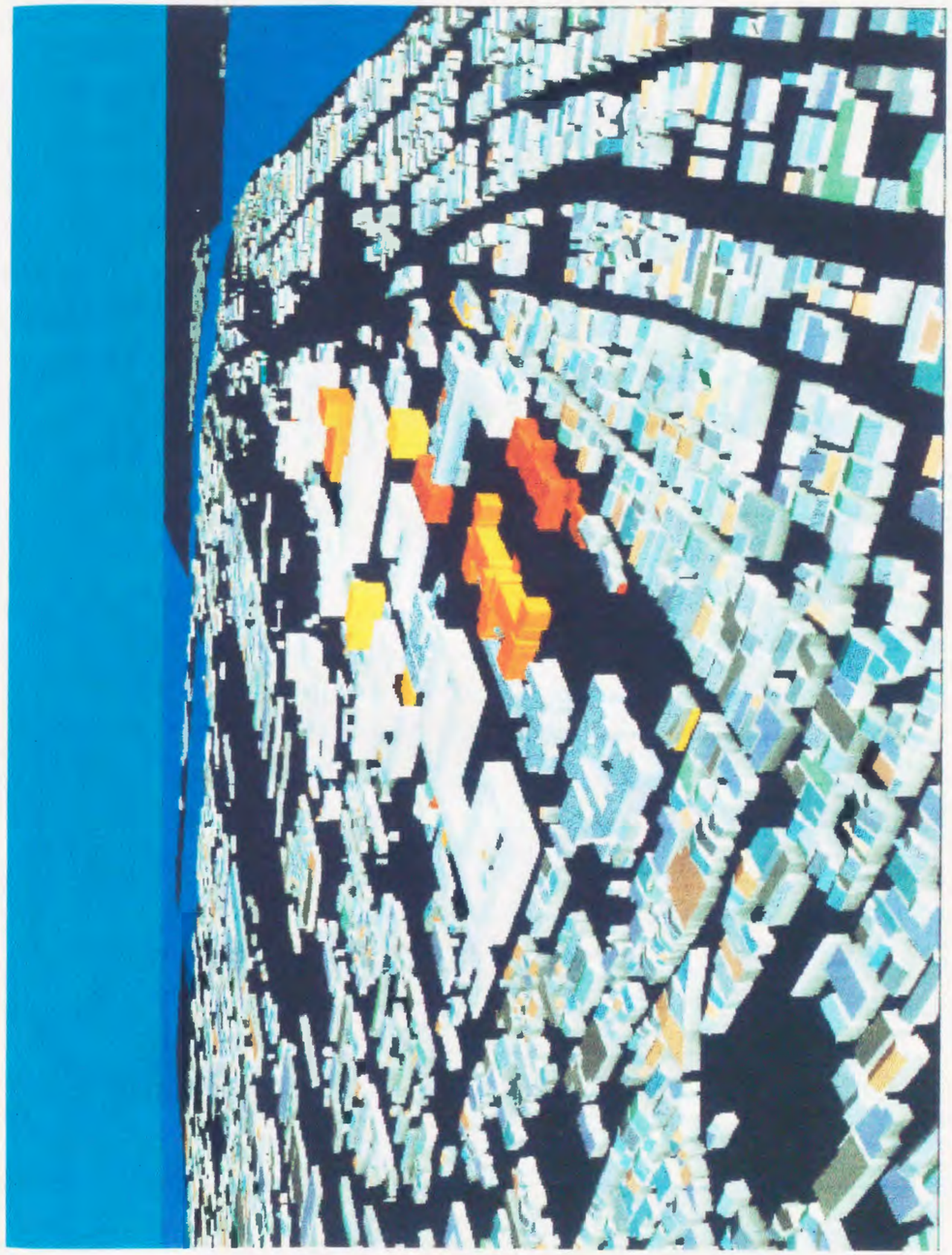
図A，力学アニメーション（複振り子）



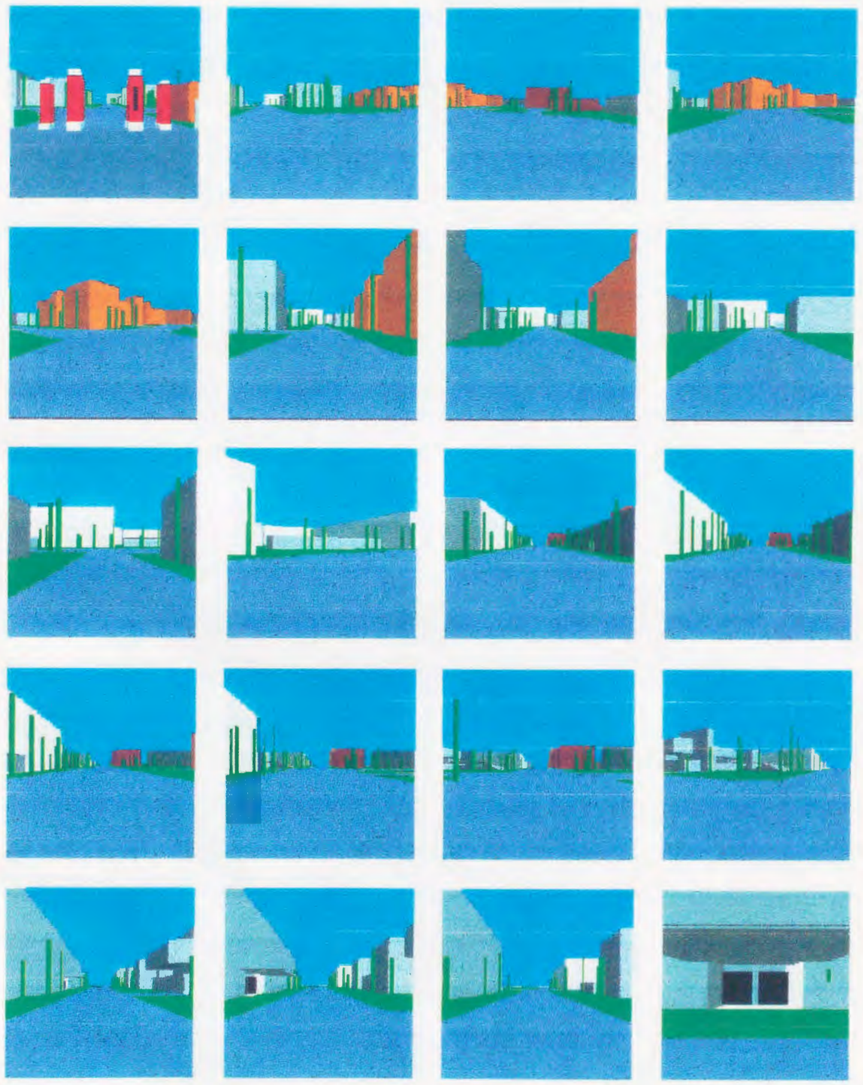
図B，力学アニメーション（倒立振子）



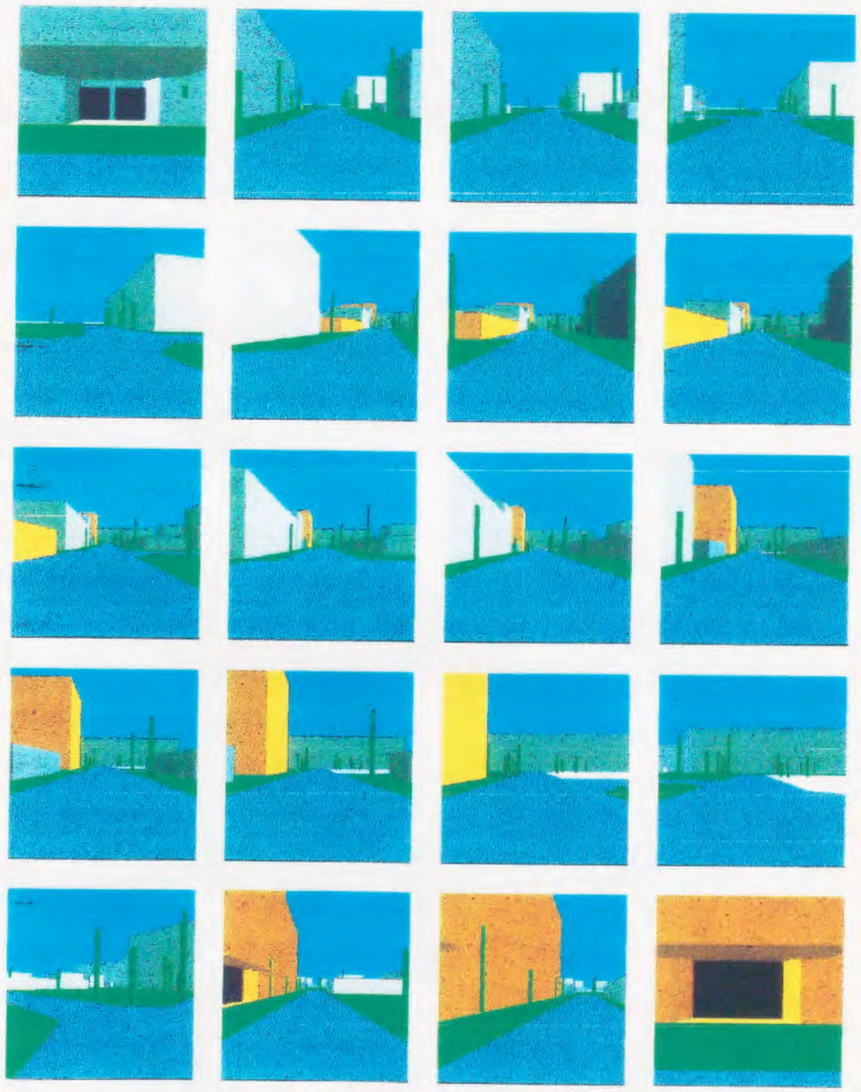
図C，グラフィカルユーザインタフェースの動作画面



図D，九州大学箱崎キャンパスとその周辺のコンピュータグラフィックス



図E，ウオークスルー （九州大学正門から情報工学科まで）



図F，ウオークスルー （情報工学科から大型計算機センターまで）