第2008-02号

グラフィックスアプリケーション開発のための OpenGL の可能性に関する調査研究

関西大学総合情報学部　関西大学先端科学技術推進機構
教授　田中　成典

平成21年9月

研究関係者紹介

たなか しげのり

田中 成典

関西大学総合情報学部 教授, 博士(工学)

主な関連著書・論文:

【３次元に関する関連図書（一部)】
１）デジカメ活用によるデジタル測量入門, 森北出版, 2000.12.
２）DirectX8－３Ｄの基礎とゲームの作り方ー, I/O別冊, 工学社, 2002.1.
３）Java3Dグラフィックス入門, 森北出版, 2002.6.
４）DirectX8 & VC++ －３Ｄの基礎とゲームの作り方ー, 工学社, 2002.8.
５）建設業界のための３次元情報, 山海堂, 2006.9.
６）できる！使える！バーチャルリアリティ～3次元VRの街づくりUC-win/Road入門～, 建通新聞社, 2006.10.
７）基礎からわかる画像処理, 工学社, 2007.11.
【建設CALSの関連図書（一部)】
１）建設CALS/ECに向けた電子国土の動向を探るーCAD/CG/GIS/GPSの統合ー, 山海堂, 2001.5.
２）e-Japan 電子政府の実現に向けて建設業界のためのXML, 工学社, 2002.10.
３）e-Japan 電子政府の実現に向けて建設業界のためのデータモデル, 工学社, 2003.7.
４）e-Japan 電子政府の実現に向けて建設情報の利活用, 工学社, 2004.11.
５）基礎からわかるGIS, 森北出版, 2005.3.
６）Logical Smart for SXF Ver.2.0, 建通新聞社, 2005.11.
【査読論文（一部)】
１）Webリンク構造解析と自然言語処理による組織関係の抽出についての研究, 情報処理学会論文誌, 2006.
２）ステレオビデオカメラを用いた交通量算出システムに関する研究開発, 情報処理学会論文誌, 2006.
３）特徴点追跡による3Dモデルの自動生成に関する研究, 日本知能情報ファジィ学会論文集, 2007.
４）SXFの同一性判別コンポーネントの実装研究, 情報処理学会論文誌2007.
５）カテゴリ分類と時系列情報に基づくブログスパム判定手法の提案, 情報処理学会論文誌, 2008.

・ 研究成果：査読付き論文 ９３件, 著書 ５１件, 国際会議 ６０件
・ 特許等取得件数： １７件（審査請求中を含む）

しばさき りょうすけ

柴崎 亮介

東京大学空間情報科学研究センター センター長・教授, 博士(工学)

主な関連著書・論文:

１）GIS入門, 日本測量協会, 1992.
２）多様な観測データや知識を用いた地物の時空間変化の再構成手法, 地理情報システム学会 GIS-理論と応用, 2003.
３）三次元GISの基礎技術, 写真測量とリモートセンシング, 2004.
４）Simulation-based Estimation of Multipath Mitigation Using 3D-GIS and Spatial Statistics, Proceedings of ION GNSS, 2006.
５）建設分野における地理空間情報基盤の構築に向けた地名辞典に関する研究, 土木情報利用技術論文集, 2007.
６）工事完成図を利用したGISデータの整備を支援するCAD-GIS連携の手引き書の作成, 地理情報システム学会講演論文集, 2007.

・特許等取得件数： ５件(2007年取得のもの)
・2003～2006年度の投稿論文実績 271編

きたがわ えつじ

北川 悦司

阪南大学経営情報学部 准教授, 博士(情報学)

主な関連著書・論文:

１）写真測量技術を用いた 2D デジタル画像からの 3D モデル空間の創出に関する基礎研究，土木情報システム論文集，2000.
２）2D デジタル画像を用いた Web/3D モデルハウスの構築に関する研究，土木情報利用技術論文集，2003.
３）デジタルビデオカメラを用いた３次元モデル自動生成システムの研究開発，土木情報利用技術論文集，2004.
４）３次元衛星電波経路シミュレーションに関する研究開発，土木情報利用技術論文集，2004.
５）特徴点追跡による３D モデルの自動生成に関する研究，日本知能情報ファジィ学会論文集，2007.

・特許等取得件数： ９件（審査請求中を含む）
・研究成果：査読付き論文 ８件，著書 １０件，国際会議 ８件

くぼた さとし

窪田 諭

岩手県立大学ソフトウェア情報学部 講師, 博士(工学)

主な関連著書・論文:

１）コンクリート橋における維持管理業務の To-be モデルの構築に関する研究，土木情報利用技術論文集，2004.
２）道路管理における空間基盤データの利活用システムと運用モデル土木情報利用技術論文集，2006.
３）四次元情報を整備した道路マネジメントシステムの構築に関する研究，土木情報利用技術論文集，2006.
４）空間基盤データの整備と活用における官民協働の実証研究，土木学会論文集，2007.

・研究成果：「地域空間基盤データの共有化手法に関する調査，国土交通省」2003 年 3 月，査読付き論文 １８件，国際会議 ８件

ものべ かんたろう

物部 寛太郎

宮城大学事業構想学部 助教, 博士(情報学)

主な関連著書・論文:

１）SXF の同一性判別コンポーネントの実装研究，情報処理学会論文誌，2007.
２）空間認知を考慮した道案内地図による歩行者交通支援に関する研究，土木情報利用技術シンポジウム論文集，2006.
３）WWW 自動探索による電子地図の属性情報自動抽出システムの研究開発，土木情報利用技術論文集，2004.
４）建設業における３次元情報の活用に関する文献調査，土木情報利用技術論文集，2004.
５）数値地図 2500 を用いた道案内地図作成支援システムの研究開発，土木情報システム論文集，2002.

・特許等取得件数： ２件（審査請求中を含む）
・研究成果：査読付き論文 ５件，著書 １３件，国際会議 ５件

よしだ ひろや

吉田博哉

神戸情報大学院大学情報技術研究科 助教, 博士(情報学)

主な関連著書・論文:

１）ステレオビデオカメラを用いた交通量算出システムに関する研究開発，情報処理学会論文誌，2006.
２）ETC データを利用した高速道路の交通量分析に関する研究開発，土木情報利用技術論文集，2006.
３）混雑空間内における人物流動システムの研究開発，土木情報利用技術論文集，2005.

・特許等取得件数： ５件（審査請求中を含む）
・研究成果：査読付き論文 ４件，著書 ９件，国際会議 ２件

目次

1　はじめに

1.1　研究の背景

我が国の建設業界においては，CAD データ交換標準フォーマットである SXF の開発が行われ，各社の商用製品への実装や国土交通省直轄事業の電子納品に使用される等，２次元の CAD が主流となっている. 一方, 国土交通省 CALS/EC アクションプログラム 2005 では，３次元 CAD の利活用に関する構想が提案されている．しかし，現在，利用されている３次元 CAD エンジンは，AutoCAD や SolidWorks をはじめ，外国の 3 次元 CAD が大半であり，国産は皆無である．安価な３次元 CAD エンジンが存在しないため，建設事業の設計・施工フェーズで３次元 CAD データが利活用されていない. そのため, 国産の安価な３次元 CAD エンジンの早急な開発が切望されている.

1.2　研究の目的

本研究では，３次元 CAD エンジンの開発に欠かせないグラフィックスライブラリの現状を把握し，特に，グラフィックスアプリケーション開発に実績のある OpenGL にフォーカスを当て，その可能性に関する調査研究を実施する.

本研究の意義としては，将来，３次元 CAD エンジンの開発に必要な要求仕様書（概略設計書）と実装仕様書（基本・詳細設計書）が作成された場合，現在の情報処理技術で本当に実現可能かを前もって掌握・理解でき，要求仕様書と実装仕様書の作成工程においての手戻りを少しでも軽減できる．また，わが国の CAD ベンダが３次元 CAD エンジンの開発に着手する場合の参考資料となり得る．したがって，最も核となる点を明らかにすることによって，社会基盤情報の革新に寄与できる.

2　市販3次元CADの実態調査

2.1　はじめに

3 次元 CAD エンジンの開発にあたり, 既存の 3 次元 CAD の仕様および機能を把握する必要がある．そこで，本章では既存の 3 次元 CAD の仕様と機能について調査する．まず，汎用 3 次元 CAD，機械系 3 次元 CAD と土木系 3 次元 CAD の機能と特徴について調査した．次に，既存の 3 次元 CAD の機能比較を行うため，市販の 3 次元 CAD が利用しているモデリングカーネルやモデル表現などについて調査した．

2.2　調査方法

調査方法は, 文献と Web ページを用いた調査を中心とし, 情報が足りない場合は, 各 CAD ベンダに問い合わせを行った．しかし，どのグラフィックスライブラリを使用するかなどの製品の仕様に関わる質問事項については，回答を得ることが困難であった．

2.3　既存の3次元CADの特徴

本節では，3 次元 CAD エンジンを汎用，機械系，土木系の 3 つに分け，各分野に特化した 3 次元 CAD の特徴について整理する．

2.3.1　汎用3次元CAD（AutoCAD）

本項では，汎用 3 次元 CAD の代表的なものとして，オートデスク社の AutoCAD について調査した. AutoCAD は, オートデスク社により開発された汎用 3 次元 CAD であり, 建設・土木・機械・電気などの幅広い分野に利用されており，特に建設業界で圧倒的なシェアを誇っている．AutoCAD は，汎用性と操作性に優れていることや 2 次元図面との親和性が高いことから，既存の 3 次元 CAD 製品の中で最も利用者が多い製品である．

ただし, AutoCAD は, あくまで設計ツールに過ぎず, 基本的なソリッドモデリング機能, サーフェスモデリング機能については揃っているが，高度な描画や分析を行うことは難しい．この問題を補うために，同社は，3 次元に特化した CAD として Autodesk Inventor を開発している．3 次元 CAD としての性能は, 図形描画や機能面ともに AutoCAD より優れているが, 機械設計に特化した製造業向けの CAD エンジンとなっている．なお，AutoCAD には 3 次元機能が付いていない AutoCAD LT，建築設計向けの AutoCAD Architecture，さらに土地開発や道路，環境プロジェクトなど土木設計向けの AutoCAD Civil3D など，各分野に特化した製品が存在する．

2.3.2　機械系3次元CAD（Autodesk Inventor）

本項では, 機械系 3 次元 CAD の代表的なものとして, オートデスク社の Autodesk Inventor について調査した．Autodesk Inventor は，オートデスク社により開発された機械系 3 次元 CAD であり，機械設計を支援するための機能を多く備えていることから，製造業全般で利用されており，特に自動車業界と家電業界で利用されている．

基本的な機能としては，パーツ設計およびアセンブリ設計，設計されたモデルを利用した図面作成機能がある．3 次元でパーツやアセンブリ形状を設計し，その 3 次元モデルを利用して図面ビューを作成することで，作図の時間を削減できる．また，3 次元モデルと 2 次元図面は連携しているため，設計変更は随時反映される．

特徴的な機能としては，3D グリップ機能，オートドロップ機能，大規模アセンブリの管理機能が挙げられる．3D グリップ機能では，配置した図形の基点を選択し，図形の移動，回転や尺度変更ができる．オートドロップ機能では，部品の配置位置をマウス操作で指定するだけで，用意されている大量の部品の中から適切な大きさの部品が自動的に選択され，配置することができる．大規模アセンブリの管理機能では，多数の 3 次元モデルの中から読み込む部品を制限できる．大規模な機械や建築物の設計は，数百個，数万個のパーツからなる．このような場合，全てのパーツを表示するとデータ量が膨大となるため，システムの動作が遅くなり作業の妨げとなる．そのため，モデルを開く時に必要な部分だけを表示することで，作業中のメモリ消費量を抑えることができる．

2.3.3　土木系3次元CAD（UC-Win/Road）

本項では, 土木系 3 次元 CAD の代表的なものとして, フォーラムエイト社の UC-win/Road について調査した．UC-win/Road は，道路事業，都市計画，公共事業などの分野に利用されている．土木用 3 次元 CAD を利用することで，都市設計や街づくりに必要な概念設計，概略設計を 3 次元モデルで構築することが可能である．また，UC-win/Road は，オートデスク社の Autodesk Civil3D やベントレー社の MicroStation の拡張アプリケーション InRoads と相互にスムーズなデータ交換を実現している．

基本的な機能として，土地造成機能や道路断面の自動作成機能，シミュレーション機能を備えている．ここでは，地形入力機能，道路定義機能，道路生成および交通流生成機能について調査した．

地形入力機能では，国土地理院の 50 メートルメッシュ標高データを読み込み，3 次元 VR 空間上に反映できる．また，任意の地形の生成や地形の編集が可能である．道路定義機能は，パラメータやマウス操作で入力された道路，河川，湖や飛行パスの各種線形から 3 次元モデルを自動で生成する．これは，道路の長さや形を変更するための平面線形と道路の高さや断面の変化を変更するための縦断線形の 2 種類の線形の編集を行うことで実現される．道路生成機能は，道路，曲線道路，交差点の形状などを簡単に表現できる．また，交通ルートや信号制御を設定すると，3 次元 VR 交通シミュレーションを実行できる．

2.4　既存の3次元CADの機能比較

本節では，既存の 3 次元 CAD の機能比較として，各 CAD のモデリングカーネル，モデルの表現方法，座標系について調査した．

2.4.1　モデリングカーネル

モデリングカーネルは，ソリッドモデルベースの 3 次元 CAD の核となるソフトウェアライブラリである．モデリングカーネルには，基本形状の生成，フィレットの生成，曲面の生成，属性の付加，集合演算といったソリッドモデリングで必要とされる基本的な機能を備えている．

自社で独自のモデリングカーネルを開発しているところもあるが, 大半の CAD ベンダは，モデリングカーネルの提供元とライセンス契約を結び，費用を支払うことで自社製品に導入している．そして，CAD ベンダは，導入したモデリングカーネルに対して，レンダリング機能やパラメトリック機能など必要な機能を付加し，他社との差別化を図った上で，3 次元 CAD を完成させる．

現在，市販されているモデリングカーネルは，ACIS7，Parasolid，DESIGNBASE などが有名であり，多数の既存 CAD 製品で導入されている．表 2.1 に各 3 次元 CAD が使用しているモデリングカーネルについて纏めた．

表 2.1　モデリングカーネル

独自カーネル	ACIS	Parasolid	DESIGNBASE
CATIA CADDS I-DEAS Pro/ENGINEER AutoCAD AutoCAD architecture Autodesk Inventor OneSpaceDesinger Thinkdesign	AlibreDesign Cealum XXen CimatronE KEYCREATOR CADPAC-CREATOR 3D IronCAD	NX4 Caelum XXen DesignFlow SolidWorks SolidEdge Unigraphics MicroStation TOPSolid VISI-CAD IronCAD	図脳 RAPID3D 図脳 RAPID3DPRO

表 2.1 に示すように，モデリングカーネルのシェアは，Parasolid を利用した CAD 製品がもっとも多く，次いで ACIS，DESIGNBASE の順である．これは，CAD 製品の価格や性能の順とほぼ一致している．3 次元 CAD によっては，複数のモデリングカーネルを利用している製品も存在する．例えば，IronCAD や CaelumXXen は，ACIS と Parasolid の 2 つのモデリングカーネルを利用している．同一のモデリングカーネル利用した CAD 製品では，データ交換が比較的容易であるため，複数のモデリングカーネルを利用することで，製品価格を抑えつつ，データの汎用性を高め，性能の向上を実現している．一方，独自カーネルを利用した CAD 製品は，モデリングカーネルでは実現できないような，複雑な形状の描画や

大規模データの高速な処理に優れているが，製品価格が高いという特徴がみられた．

2.4.2　3次元モデルの表現方法

3 次元モデルの表現方法には，ワイヤーフレームモデル，サーフェスモデル，ソリッドモデルの 3 種類が存在する．既存の CAD 製品の 3 次元モデルの表現方法は，ソリッドモデルが主流である．また，ソリッドモデルを採用している製品は，同時にサーフェスモデル，ワイヤーフレームモデルも利用できることが多い．ソリッドモデルが主流な理由を以下に纏める．

1 点目として，ソリッドモデルは，ワイヤーフレームモデルやサーフェスモデルに比べて，実際に材料を加工するような感覚で設計を行えることが挙げられる．境界のみを定義するサーフェスモデルでは，物体の定義として曖昧さが残ることに対して，ソリッドモデルでは完全に定義することが可能である．これによって立体を削るといった作業や体積計算，重心計算を行うことができる．

2 点目として，コンピュータの性能の向上が挙げられる．ソリッドモデルでは，面情報のみをもつサーフェスモデルに比べ，図形を描画する際の計算量が多い．そのため，コンピュータの性能が乏しかった数年前までは，サーフェスモデルが主流であった．しかし，近年のコンピュータの性能の向上に伴って，ソリッドモデルへの移行が進められ，現在では低価格の CAD 製品も含め，大部分の CAD 製品がソリッドモデルで 3 次元モデルを表現している．

3 点目として，多くの CAD がモデリングカーネルを利用していることである．モデリングカーネルはソリッドモデルベースであるため，モデリングカーネルを利用している CAD 製品は，必然的にソリッドモデルとなる．

分野別に見ると，汎用・機械系 3 次元 CAD は，ほぼすべてソリッドモデルを含んだ表現方法を採っているのに対して，建築・土木系 3 次元 CAD は，汎用・機械系 3 次元 CAD に比べると，必ずしもソリッドモデルである必要性はないと考えられる．例えば，AutoCAD Civil3D，inRoad などはサーフェスモデルを採用している．その理由として，機械の設計は一品一様設計であり，現実空間上に配置する部品あるいは製品の設計である．一方，建築・土木系における設計は，現実空間の設計であるため，本来は実体があるが，実体がない物体としての描画が求められることがある．例えば，地表面の表現が該当する．地表面は，地球という物体の表面である．これをソリッドモデルで表現する場合は，領域が有限なオブジェクトとして捉える必要がある．また，実体を持たない境界面や等高線，厚みを持たない道路の区画線や横断歩道などの路面標識は，サーフェスモデルでなければ表現できないと考えられる．このため，建設分野に特化した汎用 3 次元 CAD エンジンの開発を行うにあたって，3 次元モデルの表現方法はサーフェスモデルが適していると考えられる．

2.4.3　座標系

座標系には，直交座標系，円柱座標系，球座標系がある．直交座標系には，右手系と左手系がある．右手系には，数値座標系の数学系と機械座標系の機械系が，左手系には測地座標系の測地系と視点座標系の視点系が存在する．この他に，空間に図形を描画する手段として，位置情報（座標値）を空間全体で定義したグローバル座標系と図形ごとに定義したローカル座標系が存在する．

本調査では，既存 CAD 製品が右手系と左手系のどちらを採用しているかを問題とした．右手系, 左手系以外の違いについては, CAD を利用するユーザが目視できる違いではなく，あくまで CAD 内部のデータの持ち方の違いであるため，重要視しなかった。調査結果を表 2.2 に示す．

表 2.2　座標系

製品名	座標系
Alibre Design	右手系直交座標
ArchitrendZ	右手系直交座標
AutoCAD	右手系直交座標
AutoCAD Architecture	右手系直交座標（数学系）
AutoCAD Civil3D	右手系直交座標（数学系）
Autodesk Inventor	右手系直交座標
Caelum XXen	右手系直交座標
KEYCREATOR	右手系直交座標
MicroStation	右手系直交座標（数学系）
OneSpace Designer	右手系直交座標
Pro/ENGINEER	右手系直交座標

表 2.2 に示すように，3 次元 CAD には直交座標を利用している製品が多く，左手系のみを採用している製品は調査を行った範囲では存在しなかった．その理由としては，グラフィックスの標準インターフェイスである．OpenGL や Java3D が右手系直交座標を採用していること，JIS B8437，ISO 9787 において，産業用マニピュレーティングロボットにおける座標系として右手系直交座標を選定していることが挙げられる．そのため，右手系直交座標が業界標準となっている．

2.5　おわりに

本章では，市販の 3 次元 CAD の実態を把握するため，3 次元 CAD の既存製品の調査を行った．ここでは，各分野に特化した 3 次元 CAD の特徴を整理し，市販の 3 次元 CAD の機能について比較した．本章におけるこれらの調査は，汎用 3 次元 CAD エンジンを開発する上で必要となる機能を把握するとともに，建設分野に特化した汎用 3 次元 CAD に必要な機能や特徴の把握も実現した．

3　OpenGLのOSや他システムとの相性調査

3.1　はじめに

本章では，OpenGL の可能性について把握するため，各 OS，各開発言語，市販のグラフィックボードとの相性について調査する．まず，OpenGL を利用可能な OS，各 OS における OpenGL の補助ライブラリについて調査した．次に，各開発言語について調査し，開発環境や OpenGL との相性について調査した．最後に，3 次元 CAD 開発に適したグラフィックボードについて調査した．

3.2　OpenGLと各OSとの相性

3.2.1　対象OS

本節では，OpenGL との相性を OS 毎に比較し，3 次元 CAD エンジンを開発する上で，最適な GUI を持つ OS について調査した．調査対象の OS としては，Windows，Linux，Mac とした．

(1) Windows

Windows は，マイクロソフト社の提供する OS 群の総称であり，GUI を搭載し，インテルアーキテクチャ（IA）と呼ばれるプロセッサを搭載したコンピュータ上で動作する．互換性を重視した設計であり，過去の Windows で動作したアプリケーションが現在の WindowsVista でも正常に動作する例も多い．

(2) Linux

Linux は，一般に UNIX 系コンピュータ OS 群の総称であり，その基幹である Linux カーネルのことを指す．現在では PC に限らず，携帯電話のような組み込み型 OS やスーパーコンピュータにまで幅広く応用されている．

(3) Mac

Mac は，アップル社の提供する Unix 系 OS であり，デスクアクセサリと呼ばれるアプリケーションを開発時から搭載し，多機能なサブソフトを少ないメモリ使用量で疑似マルチタスクすることが可能である．また，Mac は，パソコンの性能が低い時代から，主にマルチメディア業界で重宝されている．

3.2.2　動作比較

OpenGL の OS 毎の動作環境を表 3.1 に示す．

表 3.1　OS 毎の動作環境

Windows	Linux	Mac
○	○	○

表 3.1 に示すように，OpenGL は，すべての OS において動作する．そのため，OpenGL は，OS を選ぶことなく開発を行うことができるグラフィックスライブラリである．

3.2.3　補助ライブラリ

3 次元 CAD エンジンを開発する際，ウィンドウ制御やイベント処理を実装する必要がある．しかし，OpenGL は，移植性の高いシステム構築を実現するため，ウィンドウ制御やイベント処理などの機能を保持しません．そのため，OpenGL が保持しない機能を補完するライブラリが提供されている．本項では，各 OS 用の補助ライブラリについて纏める．

(1) GLUT

GLUT とは，GL Utility Toolkit ライブラリの略称である．ウィンドウ制御やイベント処理を行うライブラリで，3 次元モデルの描画関数も保持する．さらに，GLUT は Windows，Mac や Linux など各 OS のウィンドウシステムに対応しているため，GLUT を使用することで環境に依存しないシステムを開発することができる．

(2) WGL

WGL は，マイクロソフト社が提供しているライブラリで，Windows アプリケーション上で OpenGL によるウィンドウ制御などを行うライブラリである．WGL は，任意のウィンドウを作成する際に使用され，デバイスコンテキスト関連の処理などを行うライブラリである．また，デバイスコンテキストには，線，図形やテキストなどを描画するための Windows API が組み込まれているため，デバイスコンテキストを使うことで，デバイスに依存しない描画が可能となる．

(3) CGL，AGL，NSOpenGL

CGL，AGL と NSOpenGL は，いずれも Mac においてウィンドウ制御などを行うライブラリである．WGL と同様，Mac において，GLUT では実現できない機能を補完するライブラリである．

3.3　OpenGLと各開発言語との相性

3.3.1　対象言語

本項では, 3 次元 CAD エンジンを開発する上で, 重要となる開発言語について調査した．対象言語としては，C，C++，Visual Studio 系言語，Delphi，Java とした．

(1) C，C++

C，C++では，Borland C++ Builder について調査した．C++は，オブジェクト指向プログ

ラミングのために開発された言語である．従来の C 言語のプログラミングに C++のクラスを利用して，手続き型としてプログラミングすることができる．

(2) Visual C++

Visual C++は，C++でマイクロソフト社の提供する.NET Freamwork や MFC を利用して，簡便に Windows アプリケーション開発を行うための統合開発環境に組み込まれた言語である．

(3) Visual C#

Visual C#は，Visual C++の拡張型で，より簡易に Windows アプリケーション開発を行うことができる．Visual Basic とクラスライブラリを共有するが，C++や VC++の標準関数はサポートされていない．

(4) Visual Basic

Visual Basic は，マイクロソフト社の提供する統合開発環境により提供される手続き型言語 Basic による Widnows アプリケーションを安易に開発するための言語である．VisualBasic では，MFC や.NET Freamworks をサポートしている．

(5) Java

Java は，環境に依存しない言語を目的に開発され，仮想マシンを構築してその上でアプリケーションを稼働することができる．また，Java には，独自の 3D 描画ライブラリである Java3D がある．

(6) Delphi

Delphi は，Borland 社の提供する Windows アプリケーション開発のビジュアル開発環境である．Delphi は，主要なデータベースへの多様なアクセス形態を標準で有し，WindowsAPI へのフルアクセスも可能である．有償，無償を問わず，世界的に多数の補助ライブラリが公開されている．

3.3.2　言語比較

(1) OpenGLとの相性

上記の 6 つの開発言語において OpenGL との相性を評価する．OpenGL は，一般的に C 言語ライブラリの使用が前提であり，C/C++上で制御されることを想定して作られている．その為，その他の言語から OpenGL を利用する際には，別途クラスライブラリや DLL を用意する必要がある．たとえば，Java で OpenGL を使用するには，JOGL と呼ばれるライブラリが必要となる．VB で OpenGL を使用するには，専用の DLL が必要となり，これらはインターネット上で公開されている．Delphi で OpenGL を使用するには，OpenGL 制御用ユニットファイルを用意する必要がある．また，OpenGL の DLL は.NET に対応していない為，VC# で OpenGL を使用するには，OpenTK などのラッパークラスが必要となる．以上の理由から，

OpenGL ともっとも相性のよい開発言語は，C/C++であるといえる．

3.4　OpenGLと市販のグラフィックボードとの相性

3.4.1　3次元CADの適したグラフィックボード

グラフィックボードは，描画処理を行うグラフィックチップと表示する描画情報を保持するメモリ領域によって構成されたマイクロプロセッサである．グラフィックボードがCPUに代わって3次元グラフィックス処理を行う機能を3Dアクセラレータと呼ぶこともあり，現在販売されている全てのグラフィックボードには3Dアクセラレータ機能が搭載されている．3Dアクセラレータ機能によって処理される項目のうち，特にピクセルシェーダやバーテックスシェーダが 3 次元描画速度に大きく影響し，ピクセルシェーダが高いほど 3 次元モデルの描画が滑らかになる．また，3次元モデル描画では映像を処理しながら表示面を作成する必要があるため，多くのメモリ容量を必要とする．

3.4.2　対象グラフィックボード

対象とするグラフィックボードは，コンシューマ向けとプロフェッショナル向けの用途別に，主要なグラフィックボードメーカの製品の特徴を比較する．

(1) GeForce（NVIDIA社）

GeForce は NVIDIA 社製のコンシューマ向けグラフィックボード製品系列である．寒色系の発色に強く，全体的にシャープな描画となり，ゲームアプリケーション用途の DirectX に最適化されている．

(2) Quadro（NVIDIA社）

Quadro は NVIDIA 社製のプロフェッショナル向けグラフィックボード製品系列である．CAD などのワークステーション用途に設計され，OpenGL に最適化されている．CAD ベンダ側では，Quadro での動作保証はしても GeForce での動作保証はしない場合が多い．

(3) Redeon（ATI社）

Redeon は ATI 社製のコンシューマ向けグラフィックボード製品系列である．暖色系の発色に強く，全体的にぼやけた感じの描画となり，動画再生（MPEG-2，DivX，WMP）に最適化されている．

(4) FireGL（ATI社）

FireGL は ATI 社製のプロフェッショナル向けグラフィックボード製品系列である．FireGL は Redeon をベースに OpenGL 向けに特化しており，消費電力を低減した派生品が存在する．

3.4.3　機能比較

前項で対象とした 4 種のグラフィックボードについて評価する．評価項目はグラフィッ

クボードの開発用途，適応ライブラリ，値段，描画傾向，消費電力／発熱量の 5 項目とした．値段に関しては，各製品シリーズ内で上下の変動が大きいが，その中でも特に安価の傾向の強い製品に◎，全体的に安価な傾向の製品に○，比較的に割高の製品に△として評価した．比較結果を表 3.2 に示す．

表 3.2　グラフィックボードの機能比較

	開発用途	適応 ライブラリ	値段	描画傾向	消費電力／ 発熱量
GeForce	ゲーム	DirectX	◎	寒色系が鮮やか シャープ	比較的少ない
Quadro	CAD	OpenGL	△	寒色系が鮮やか シャープ	比較的少ない
Radeon	動画	DirectX	○	暖色系が鮮やか ややぼかし	比較的多い
FireGL	CAD	OpenGL	△	暖色系が鮮やか ややぼかし	比較的多い

表 3.2 に示すように，OpenGL に適応するものは，Quadro と FireGL の２つである．そして，描画傾向と消費電力/発熱量の比較結果では，Quadro の方が優れていると考えられる．そのため，3 次元 CAD を開発する上で最も適している市販のグラフィックボードは Quadro であるといえる．

3.5　おわりに

本章では，汎用 3 次元 CAD エンジンを開発する上で使用する OpenGL を利用した際に最も相性が良い OS, 開発言語, そしてグラフィックボードを選択するため, OpenGL と各 OS, 開発言語，市販のグラフィックボードとの相性調査を行った．調査結果から OS は Windows を利用し，開発言語には，十分な統合開発環境が提供されている Visual C++が最もよいのではないかと考えられる．グラフィックボードには，NVIDIA 社の Quadro が最もよいと考えられる．

4　3次元CADエンジンのプロトタイプの開発に向けた機能調査

4.1　はじめに

本章では，3 次元 CAD エンジンの開発に向けて実装すべき機能について纏める．まず，汎用 3 次元 CAD の機能について調査する．次に，その機能の中から本研究で開発する 3 次元 CAD エンジンのプロトタイプで実装する機能について整理する．

4.2　汎用3次元CADの機能調査

4.2.1　調査内容

本節では，汎用 3 次元 CAD の機能についての調査を行った．調査した 3 次元 CAD は，既存の 3 次元 CAD 製品の中でも特に利用者数が多い，オートデスク社の AutoCAD，ベントレー社の MicroStation，ダンソー・システムズ社の CATIA と SolidWorks の 4 つとした．調査内容としては，各汎用 3 次元 CAD の機能の中でモデルの描画や編集といった OpenGL に関連があると考えられる機能とした．調査方法としては，文献と Web ページの調査を中心に行い，情報が足りない場合は，直接，ベンダに問い合わせを行った．

4.2.2　調査結果

汎用 3 次元 CAD の機能の調査結果を表 4.1 に示す．

表 4.1　汎用 3 次元 CAD の機能

大機能	小機能	汎用CAD			
		AutoCAD	MicroStation	CATIA	SolidWorks
曲線作成	線分の作成	○	○	○	○
	折線の作成	○	○	○	○
	円の作成	○	○	○	○
	円弧の作成	○	○	○	○
	楕円の作成	○	○	○	○
	楕円弧の作成	○	○	○	○
	ベジェ曲線の作成	○	○	○	○
	NURBS（B-スプライン）曲線の作成	○	○	○	○
面作成（プリミティブ）	直方体の作成	○	○	○	○
	角柱の作成	○	○	○	○
	円錐の作成	○	○	○	○
	円柱の作成	○	○	○	○
	球の作成	○	○	○	○
	トーラス体の作成	○	○		
	ウェッジの作成	○	○		
面作成（2D-3D）	押出しによる面の作成	○	○	○	○
	回転による面の作成	○	○	○	○
	スイープによる面の作成	○	○	○	○
	ルールドによる面の作成	○	○	○	
	バウンダリーによる面の作成	○	○	○	
	ロフト（スキニング）による面の作成	○	○	○	○
	フィレットによる面の作成	○	○	○	○
	メッシュによる面の作成	○	○	○	
注釈作成	文字の作成	○	○	○	○
	直線寸法の作成	○	○	○	○
	角度寸法の作成	○	○	○	○
	半径寸法の作成	○	○	○	○
	直径寸法の作成	○	○	○	○
	引き出し線の作成	○	○	○	○
	バルーンの作成		○	○	○
	ハッチングの作成				○
モデル編集	平行移動（複写）	○	○	○	○
	回転移動（複写）	○	○	○	○
	ミラー移動（複写）	○	○	○	○
	スケール（拡大・縮小）	○	○	○	○
	稜線面取り	○	○	○	○
	稜線フィレット（丸め）	○	○	○	○
	シェル化（薄肉化）	○	○	○	○
	オフセット（厚み付け）	○	○	○	○
	モデルの色の変更	○	○	○	○
	モデルの線種の変更	○	○	○	○
	モデルの線幅の変更	○	○	○	○
曲面編集	切断（分割）	○	○	○	○
	結合	○	○	○	○
	延長	○	○	○	○
	トリミング	○	○	○	
	アントリミング	○			
	反転	○	○	○	○
拘束	幾何拘束		○	○	○
	寸法拘束		○	○	○
	2次元平面の拘束を評価		○		
	2次元平面をパラメトリックに登録		○		
集合演算	ブーリアン演算（和）	○	○	○	○
	ブーリアン演算（差）	○	○	○	○
	ブーリアン演算（積）	○	○	○	○
	平面による切断（分割）		○		
	曲面による切断（分割）		○		

4.3　OpenGLを用いて実装可能な機能調査

4.3.1　調査内容

本項では，前節で調査した内容を基から 3 次元 CAD エンジンのプロトタイプの機能について選定する．選定基準としては，表 4.1 の機能の中から特に，OpenGL と関連が深く，3 次元 CAD エンジンを開発する上で必要性が高いものから順に選定した．

4.3.2　調査結果

機能一覧を表 4.2 に示す．

表 4.2　機能一覧

大機能	小機能	大機能	小機能
点作成	点マーカの作成	曲面作成	ベジェ曲面の作成
曲線作成	線分の作成		NURBS曲面の作成
	折線の作成	注釈作成	文字の作成
	円の作成		直線寸法の作成
	円弧の作成		角度寸法の作成
	楕円の作成		半径寸法の作成
	楕円弧の作成		直径寸法の作成
	ベジェ曲線の作成		引き出し線の作成
	NURBS（B-スプライン）の作		バルーンの作成
	クロソイド曲線の作成	モデル編集	色の変更
面作成（プリミティブ）	直方体の作成		線種の変更
	円錐の作成		線幅の変更
	円柱の作成		
	球の作成		
	トーラス体の作成		
	ウェッジの作成		

4.4　おわりに

本章では，3 次元 CAD エンジンのプロトタイプを開発する上で，汎用 3 次元 CAD の機能について調査を行った. ここでは, 利用者数の多い 3 次元 CAD である AutoCAD, MicroStatio, CATIA，SolidWorks の４つの CAD エンジンについて調査を行った．そして，この調査結果を基に，本研究で開発する 3 次元 CAD エンジンのプロトタイプの機能について整理した．次章以降では，整理した機能について OpenGL での実装方法について纏める．

5　OpenGLによる3次元モデルの表現方法

5.1　はじめに

本章では，OpenGL を利用して 3 次元モデルを描画する際の表現方法について調査，実装する．まず，実装すべき幾何要素について整理する．次に，3 次元モデルのデータフォーマットである ISO10303（STEP）でのモデルとの対応について調査する．そして，OpenGL で用意されている幾何要素について調査し，それを基に，幾何要素毎の表現方法について纏める．

5.2　実装すべき幾何要素

本研究では，点要素，曲線要素，面要素の３つの幾何要素に分けて OpenGL によるモデルの表現方法について調査した．本研究で調査するモデルを表 5.1 に示す．

表 5.1　本研究で調査するモデル

要素	モデル	要素	モデル
点要素	点マーカ	面要素	直方体
曲線要素	線分		円錐
	折線		円柱
	円		球
	円弧		トーラス
	楕円		ウェッジ
	楕円弧		ベジェ曲面
	ベジェ曲線		NURBS 曲面
	NURBS 曲線		
	クロソイド		

5.2.1　STEPにおけるモデル要素とエンティティ

本研究では，ISO10303（以下，STEP）に基づいたデータ交換を目的としている．そのため，STEP で定義されているモデルの幾何情報を表すエンティティを基に各モデルを定義する．まず，本項では，STEP におけるモデル要素とモデルの幾何情報について纏める．

(1) STEPのモデル要素

STEP のモデル要素には，幾何情報と位相情報を保持する．幾何情報とは曲線，曲面など個々の要素がどのような形状をしているかを表わしたもので，位相情報とは，各要素のつながりの情報のことである．STEP のモデル要素を図 5.1 に示す．

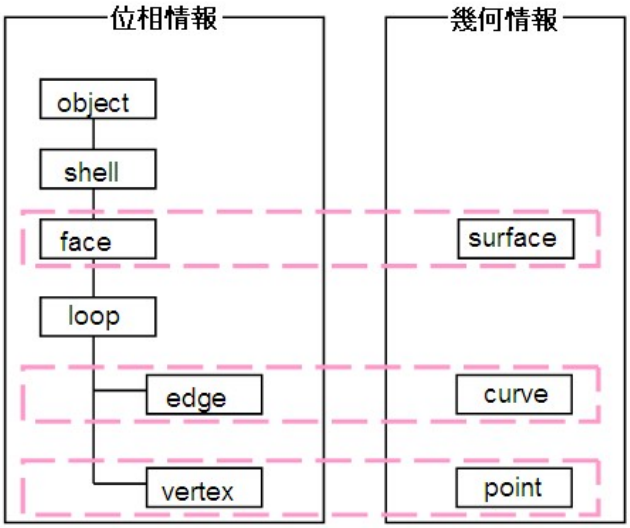


図 5.1　STEP のモデル要素

STEP のモデル要素では，図 5.1 に示すように，頂点に関する位相情報である vertex と点に関する幾何情報を表す point が対応する．また，稜線に関する位相情報である edge と曲線に関する幾何情報を表す curve が，さらに，面に関する位相情報である face と面に関する幾何情報を表す surface がそれぞれ対応する．

(2) エンティティとモデルの関係

STEP で定義されているエンティティと本研究で実装するモデルの対応関係を表 5.2 に示す.

表 5.2　STEP で定義されているモデルの幾何情報

要素	エンティティ名	概要	モデル
点要素	cartesian_point	座標点を表すエンティティ	点マーカ
曲線要素	line	直線を表すエンティティ	線分
	polyline	折線を表すエンティティ	折線
	circle	円を表すエンティティ	円，円弧
	ellipse	楕円を表すエンティティ	楕円，楕円弧
	bezier_curve	ベジェ曲線を表すエンティティ	ベジェ曲線
	b_spline_curve_with_knots	ノットを伴う有理 B スプライン曲線のエンティティ	NURBS 曲線
面要素	plane	平面を表すエンティティ	直方体，ウェッジ
	sphere_surface	球面を表すエンティティ	球
	conical_surface	円錐面を表すエンティティ	円錐
	cylindrical_surface	円柱面を表すエンティティ	円柱
	toroidal_surface	スイープ面を表すエンティティ	トーラス
	bezier_surface	ベジェ曲面を表すエンティティ	ベジェ曲面
	b_spline_surface_with_knots	ノットを伴う B スプライン曲面を表すエンティティ	NURBS 曲面

5.3　OpenGLで実装可能な幾何要素

OpenGL では，3 次元空間上に様々なモデルを描画することができる．本節では，OpenGL のライブラリを使用して描画可能な 2 次元モデルと 3 次元モデルについて纏める．

5.3.1　2次元モデル

本項では，OpenGL が描画できる 2 次元モデルについて解説する．OpenGL では，点や線といったプリミティブといわれる基本的なモデルが用意されている．OpenGL で描画できる 2 次元モデルを図 5.2 に示す．

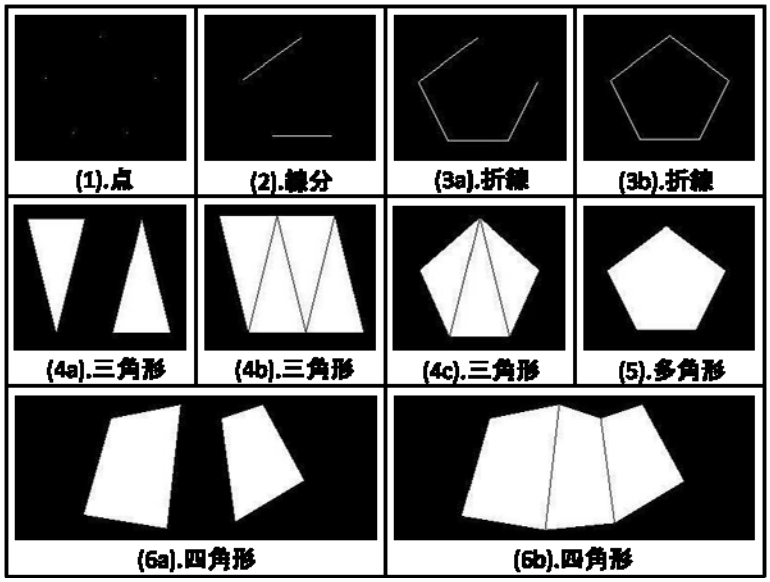


図 5.2　OpenGL で描画できる 2 次元モデル

(1) 点

図 5.2 の(1)は，OpenGL による点の描画例である．指定した各頂点座標に点を描画する．

(2) 線分

図 5.2 の(2)は，OpenGL による線分の描画例である．線分は，2 つの頂点座標間を結び直線である．

(3) 折線

図 5.2 の(3a)は，OpenGL による折線の描画例である．図 5.2 の(3a)の折線は，各頂点座標を指定した順に結ぶ直線である．

図 5.2 の(3b)は．OpenGL による始点と終点を連結した折線の描画例です．図 5.2 の(3b)の折線は，各頂点座標を指定した順に結び，最後に始点と終点が連結される．

(4) 三角形

OpenGL では，三角形（図 5.2 の(4a)），一辺を共有した連続する三角形（図 5.2 の(4b)）と一辺を共有し扇状に連続する三角形（図 5.2 の(4c)）の 3 種類の三角形が描画できる．

三角形（図 5.2 の(4a)）は，各頂点を指定した順に 3 点を組にし，三角形を描画する．一辺を共有した連続する三角形（図 5.2 の(4b)）は，各頂点を指定した順に 3 点を組にし，一辺を共有した帯状の三角形を描画する．一辺を共有し扇状に連続する三角形（図 5.2 の(4c)）は，各頂点を指定した順に 3 点を組にして，一辺を共有した扇状の三角形を描画する．

(5) 多角形

図 5.2 の(5)は，OpenGL による多角形の描画例である．多角形は，指定した各頂点座標を頂点とする多角形を描画する．

(6) 四辺形

OpenGL では，四辺形（図 5.2 の(6a)）と一辺を共有する連続した四辺形（図 5.2 の(6b)）の 2 種類の四辺形が描画できる．

四辺形（図 5.2 の(6a)）は，各頂点座標を指定した順に 4 点ずつ組にした四辺形を描画する．一辺を共有する連続した四辺形（図 5.2 の(6b)）は，各頂点を指定した頂点順に 4 点ずつ組にした四辺形を，一辺を共有しながら帯状に四辺形を描画する．

5.3.2　3次元モデル

OpenGL には，OpenGL ライブラリと OpenGL ユーティリティライブラリという 2 つのライブラリから構成されているが，2 つのライブラリには，3 次元モデルを描画するためのプリミティブが用意されていない．そのため，OpenGL において 3 次元モデルを描画する場合は OpenGL Utility Toolkit （GLUT）などの補助ライブラリを使用する必要がある．GLUT で描画できる 3 次元モデルを図 5.3 に示す．

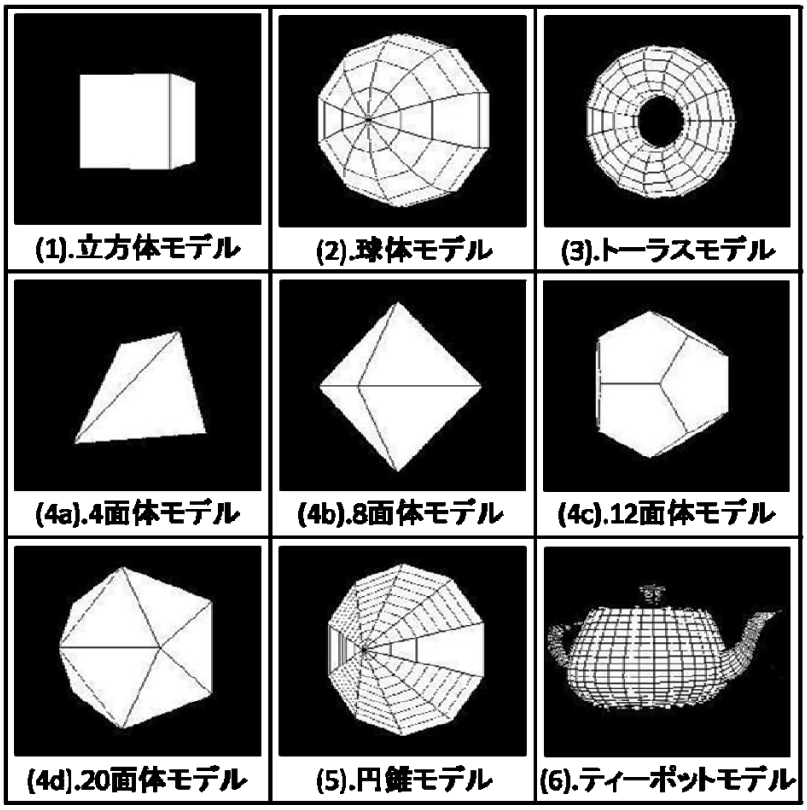


図 5.3　GLUT で描画できる 3 次元モデル

(1) 立方体モデル

立法体モデル（図 5.3 の(1)）は，モデルの一辺の長さを指定することで描画できる．

(2) 球体モデル

球体モデル（図 5.3 の(2)）は，モデルの半径，緯度方向の分割数と経度方向の分割数を指定することで描画できる．そして，緯度と経度方向の分割数が多いと綺麗な曲面を描画できるが，速度が遅くなる．

(3) トーラスモデル

トーラスモデル（図 5.3 の(3)）は，モデルの内半径，外半径と分割数を指定することで描画できる．

(4) 多面体モデル

GLUT では，4 面体，8 面体，12 面体と 20 面体の 4 種類の多面体モデルが描画できる．また，多面体モデル（図 5.3 の(4a)から(4d)）は，いずれもモデルの大きさがデフォルト値で定められているため，他のモデルのように値を指定することはできない．

(5) 円錐モデル

円錐モデル（図 5.3 の(5)）は，モデルの半径，高さと分割数を指定することで描画できる．

(6) ティーポットモデル

ティーポットモデル（図 5.3 の(6)）は，ティーポットの大きさを指定することで描画できる．

5.4　点要素の表現方法

本節では，点要素として，点マーカの OpenGL での表現方法について解説する．

5.4.1　点マーカ

点マーカは，任意の座標を示す点データである．点マーカの概要を図 5.4 に示す．

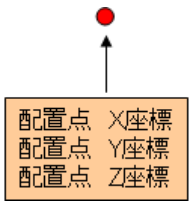


図 5.4　点マーカの概要

本研究では，配置点の X 座標，配置点の Y 座標，配置点の Z 座標，マーカコード，尺度を 6 つの値を基に点マーカを描画する．マーカコードの値により点マーカの形状が異なり，尺度の値により点マーカの大きさが決定する．点マーカのパラメータを表 5.3 に示す．

表 5.3　点マーカのパラメータ

パラメータ	型	説明
m_dStartX	double	配置点の X 座標
m_dStartY	double	配置点の Y 座標
m_dStartZ	double	配置点の Z 座標
m_iMarkerCode	int	マーカコード
m_dScale	double	尺度
m_PointMarkerTopology	PointMarkerTopology	点マーカの位相情報

マーカコードに対応した点マーカの形状を図 5.5 に示す.

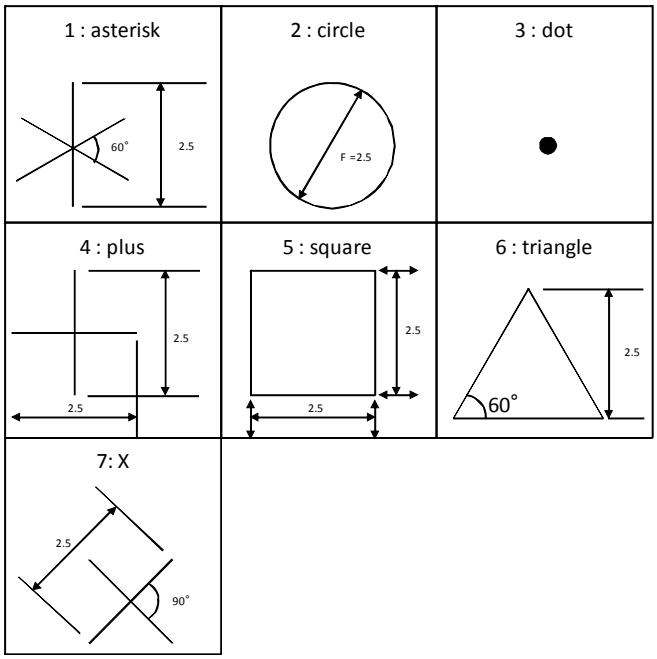


図 5.5　マーカコードに対応した点マーカの形状

(1) 実装方法

OpenGL には，点を描画するための関数が用意されているため，glVertex3d 関数を使用して点マーカを描画する．glVertex3d 関数の定義を次に示す.

```
void glVertex3d(GLdouble x, GLdouble y, GLdouble z)
```

glVertex3d 関数の引数の説明を表 5.4 に示す.

表 5.4　glVertex3d 関数の引数

引数	型	説明
X	GLdouble	描画する点の X 座標
Y	GLdouble	描画する点の Y 座標
Z	GLdouble	描画する点の Z 座標

マーカコードの値により点マーカの形状が異なるため，マーカコード毎の点マーカの描画方法について次で解説する．

- マーカコード１

マーカコード１の点マーカでは，glVertex3d 関数を使用して 3 本の直線を描画することで実現する．glVertex3d 関数を用いて線分を描画するには，glBegin 関数と glEnd 関数を利用する．マーカコード１の点マーカの実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINES);
        // 描画の実行
        glVertex3d(0, 0 - 1.25,0);
         glVertex3d(0, 0 + 1.25,0);
         glVertex3d(-sqrt(3.0) * 0.5 * 1.25, 0.5 * 1.25, 0);
         glVertex3d( sqrt(3.0) * 0.5 * 1.25,-0.5 * 1.25, 0);
         glVertex3d(-sqrt(3.0) * 0.5 * 1.25,-0.5 * 1.25, 0);
         glVertex3d( sqrt(3.0) * 0.5 * 1.25, 0.5 * 1.25, 0);
    // 描画の終了
    glEnd();
```

glBegin 関数は，頂点データの始まりを宣言する関数であり，glEnd は，頂点データの終了を宣言する関数である．そして，直線を描画する際は，glBegin 関数の引数に GL_LINE を指定する．

- マーカコード２

マーカコード２の点マーカは，円形であるため，円を描画することで実現する．しかし，OpenGL には，円を描画する関数が用意されていないため，円を折線に近似して glVertex3d 関数を利用して各頂点を結びことで円を描画する．マーカコード２の点マーカの実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINE_LOOP);
        // 円を描画するための繰り返し
        for(int i=0;i<alX->Count;i++){
            // 頂点の指定
            glVertex3d((double)alX[i],(double)alY[i],(double)alZ[i]);
        }
```

```
    // 描画の終了
    glEnd();
```

各頂点を結ぶには，glBegin 関数の引数に GL_LINE_LOOP を指定する．また，各頂点の算出には，三角関数を利用する．三角関数による各頂点の算出方法を図 5.6 に示す．

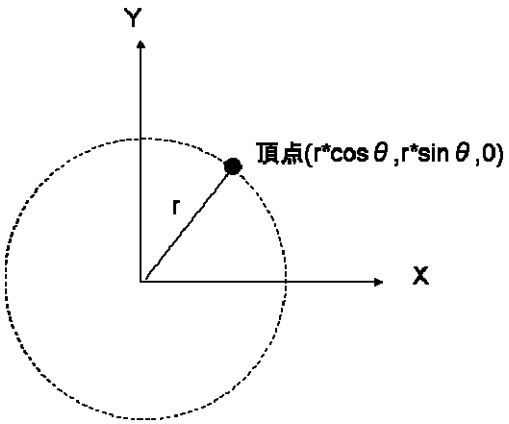


図 5.6　三角関数による座標点の表現

なお，角度 θ の値は，円の分割数と円周率から算出する．角度 θ の算出式を次式に示す．

$$\theta = \frac{2\pi \times r}{iStep} \times i$$

なお，iStep は円の分割数，r は円の半径，i は頂点番号を表す．iStep に代入する値が小さければ，円は多角形のような図形に見え，代入する値が大きければ，綺麗な円に見える．本研究では，iStep の値は 1080 にしている．

- マーカコード３

マーカコード３の点マーカは，glVertex3d 関数を利用して点（ドット）を描画することで実現する．マーカコード３の点マーカの実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_POINTS);
        // 描画の実行
        glVertex3d(0, 0, 0);
    // 描画の終了
    glEnd();
```

glBegin 関数の引数に GL_POINTS を指定する．

● マーカコード４

マーカコード４の点マーカは，glVertex3d 関数を利用して 2 本の直線を描画することで実現する．マーカコード４の点マーカの実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINES);
        // 描画の実行
        glVertex3d( -1.25, 0, 0);
        glVertex3d(  1.25, 0, 0);
        glVertex3d(0, -1.25, 0);
        glVertex3d(0,  1.25, 0);
    // 描画の終了
    glEnd()
```

● マーカコード５

マーカコード５の点マーカは，glVertex3d 関数を利用して四角形を描画することで実現する．マーカコード５の点マーカの実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINE_LOOP);
        // 描画の実行
        glVertex3d(-1.25, -1.25, 0);
        glVertex3d( 1.25, -1.25, 0);
        glVertex3d( 1.25,  1.25, 0);
        glVertex3d(-1.25,  1.25, 0);
    // 描画の終了
    glEnd();
```

● マーカコード６

マーカコード６の点マーカは，glVertex3d 関数を利用して三角形を描画することで実現する．マーカコード６の点マーカの実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINE_LOOP);
        // 描画の実行
        glVertex3d(0-(2.5/sqrt(3.0)), -1.25, 0);
        glVertex3d(0, +1.25, 0);
        glVertex3d(0+(2.5/sqrt(3.0)), -1.25, 0);
    // 描画の終了
    glEnd()
```

● マーカコード７

マーカコード７の点マーカは，glVertex3d 関数で 2 本の直線を描画することで実現する．マーカコード７の点マーカの実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINES);
        // 描画の実行
        glVertex3d(0 - ((2.5 * sqrt(2.0)) / 4.0),  ((2.5 * sqrt(2.0)) / 4.0), 0);
        glVertex3d(0 + ((2.5 * sqrt(2.0)) / 4.0), -((2.5 * sqrt(2.0)) / 4.0), 0);
        glVertex3d(0 + ((2.5 * sqrt(2.0)) / 4.0),  ((2.5 * sqrt(2.0)) / 4.0), 0);
        glVertex3d(0 - ((2.5 * sqrt(2.0)) / 4.0), -((2.5 * sqrt(2.0)) / 4.0), 0);
    // 描画の終了
    glEnd();
```

(2) 描画例

本研究で実装した点マーカを図 5.7 に示す.

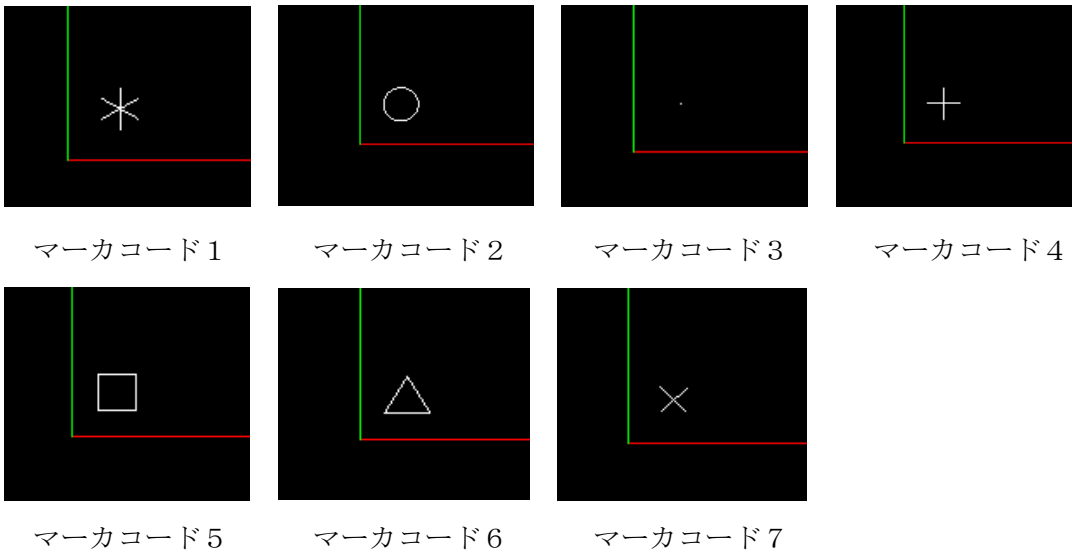


図 5.7　本研究で実装した点マーカ

5.5　曲線要素の表現方法

本節では，曲線要素として，線，折線，円，円弧，楕円，楕円弧，ベジェ曲線，NURBS 曲線，クロソイドについて OpenGL での表現方法について解説する.

5.5.1　線分

線分とは，2 つの異なる位置に存在する点を最短距離で結んだ直線である．線分の概要を図 5.8 に示す.

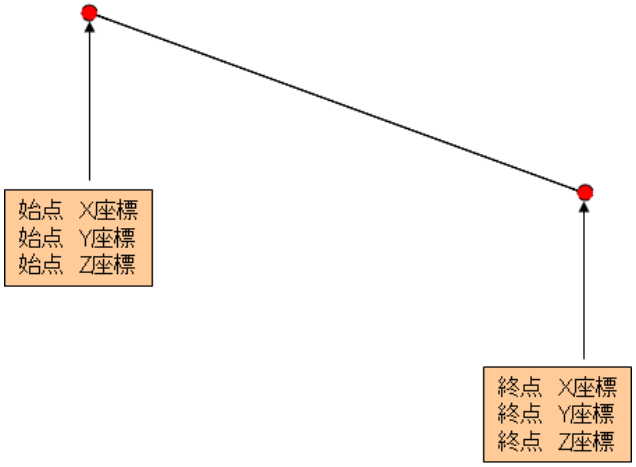


図 5.8　線分の概要

本研究では，始点 X 座標，始点 Y 座標，始点 Z 座標，終点 X 座標，終点 Y 座標と終点 Z 座標の６つの値を基に線分を描画する．線分のパラメータを表 5.5 に示す．

表 5.5　線分のパラメータ

パラメータ	型	説明
m_dStartX	double	始点の X 座標
m_dStartY	double	始点の Y 座標
m_dStartZ	double	始点の Z 座標
m_dEndX	double	終点の X 座標
m_dEndY	double	終点の Y 座標
m_dEndZ	double	終点の Z 座標
m_LineTopology	LineTopology	線分の位相情報

(1) 実装方法

線分の実装方法は，glBegin 関数と glEnd 関数の間に glVertex3d 関数を用いて線分の頂点を指定する．各頂点を結び直線を描画するため，glBegin 関数の引数に GL_LINE_STRIP を指定する．線分の実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINE_STRIP);
        // 描画の実行
        glVertex3d(0.0, 0.0, 0.0);
        glVertex3d(10.0, 25.0, 0);
    // 描画の終了
    glEnd();
```

(2) 描画例

本研究で実装した線分を図 5.9 に示す.

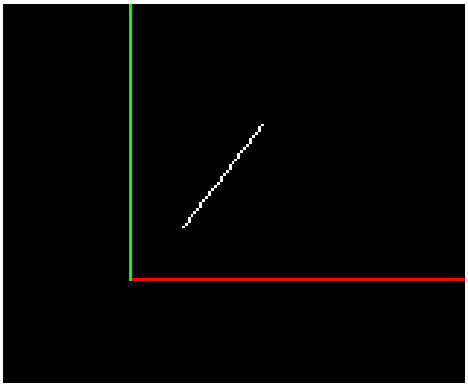


図 5.9　本研究で実装した線分

5.5.2　折線

折線とは，複数の頂点を順に結んだ直線の集合である．折線の概要を図 5.10 に示す.

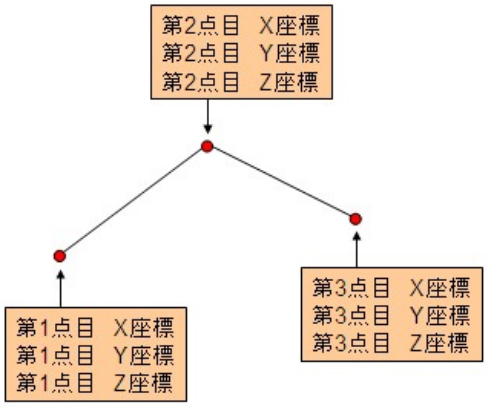


図 5.10　折線の概要

本研究では，頂点数とそれに応じた X 座標，Y 座標，Z 座標の値を基に折線を描画する．折線のパラメータを表 5.6 に示す.

表 5.6　折線のパラメータ

パラメータ	型	説明
m_iNumber	int	頂点数
m_alX	ArrayList	X 座標の配列
m_alY	ArrayList	Y 座標の配列
m_alZ	ArrayList	Z 座標の配列
m_PolylineTopology	PolylineTopology	線分の位相情報

(1) 実装方法

折線の実装方法は，線分同様に glBegin 関数と glEnd 関数の間に頂点数分の glVertex3d 関数を指定する．折線の実装例を次に示す．

```
// 描画の開始
glBegin(GL_LINE_STRIP);
    // 折線を描画するための繰り返し
    for(int i=0;i<m_iNumber;i++){
        // 描画の実行
        glVertex3d((double)alX[i],(double)alY[i],(double)alZ[i]);
    }
// 描画の終了
glEnd();
```

(2) 描画例

本研究で実装した折線を図 5.11 に示す．

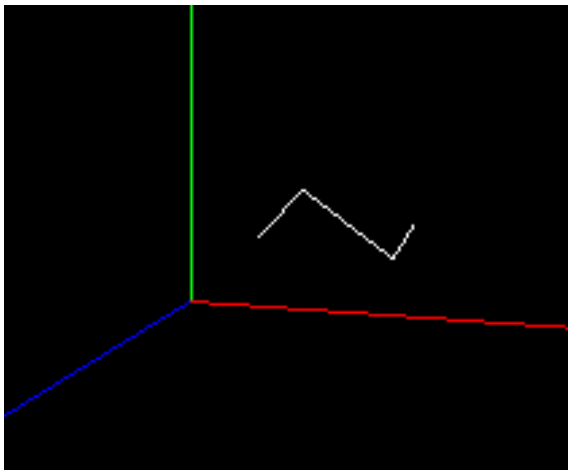


図 5.11　本研究で実装した折線

5.5.3　円

円は，中心から等距離にある点の集合でできる曲線である．円の概要を図 5.12 に示す．

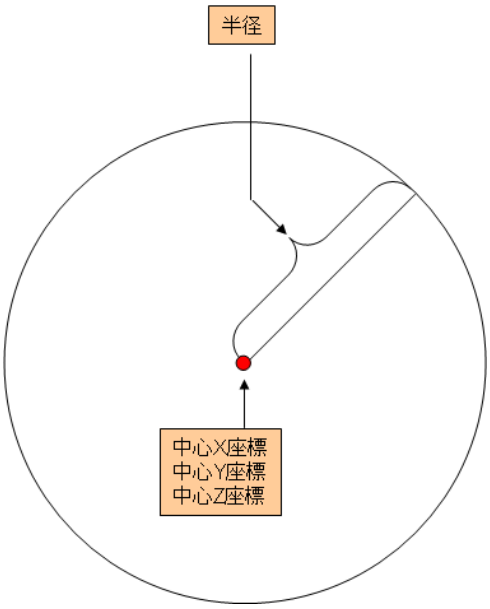


図 5.12　円の概要

本研究では，中心点の X 座標，Y 座標，Z 座標と半径の値を基に円を描画する．円のパラメータを表 5.7 に示す．

表 5.7　円のパラメータ

パラメータ	型	説明
m_dCenterX	double	中心点の X 座標
m_dCenterY	double	中心点の Y 座標
m_dCenterZ	double	中心点の Z 座標
m_dRadius	double	半径
m_CircleTopology	CircleTopology	円の位相情報

(1) 実装方法

円の実装方法は，マーカコード２の点マーカと同様に中心点の X 座標，Y 座標，Z 座標と半径を基に円周上の頂点を算出する．そして，折線と同様に glBegin 関数と glEnd 関数の間に頂点数分，glVertex3d 関数を指定することで描画する．ただし，円は閉じた曲線であるため，glBegin 関数の引数には，GL_LINE_LOOP を指定する．円の実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINE_LOOP);
```

```
        // 円を描画するための繰り返し
        for(int i=0;i<alX->Count;i++){
            // 描画の実行
            glVertex3d( (double)alX[i], (double)alY[i], (double)alZ[i]);
        }
    // 描画の終了
    glEnd();
```

また，円周上の頂点の算出方法を次に示す.

```
    // 円の頂点座標を算出するための繰り返し
    for(int i=0;i<=division_num;i++){
        // 分割の指定
        step = (i/((division_num)*1.0))*(end_angle - start_angle);
        if(step >= 360){
            step = step-360;
        }
        // X 座標値の算出
        x->Add(radius_x*cos((start_angle + step) /180*PAI)+ center_x);
        // Y 座標値の算出
        y->Add(radius_y*sin((start_angle + step) /180*PAI)+ center_y);
        // Z 座標値の算出
        z->Add(center_z);
    }
```

(2) 描画例

本研究で実装した円を図 5.13 に示す.

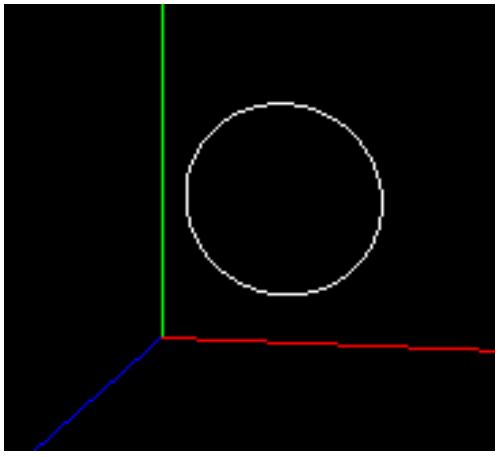


図 5.13　本研究で実装した円

5.5.4　円弧

円弧は，中心から等距離にある点の集合の一部を表す曲線である．円弧の概要を図 5.14 に示す．

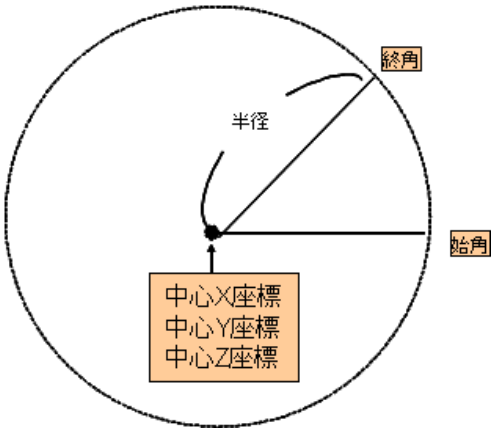


図 5.14　円弧の概要

本研究では，中心点の X 座標，Y 座標，Z 座標，半径，描画方向，始角，終角の値を基に円弧を描画する．描画方向は，円弧を時計まわりに描画するか，反時計まわりに描画するかを指定する．そして，円弧の範囲は，始角と終角の値により決定する．円弧のパラメータを表 5.8 に示す．

表 5.8　円弧のパラメータ

パラメータ	型	説明
m_dCenterX	double	中心点の X 座標
m_dCenterY	double	中心点の Y 座標
m_dCenterZ	double	中心点の Z 座標
m_dRadius	double	半径
m_iDirection	int	描画方向
m_dStartAngle	double	始角
m_dEndAngle	double	終角
m_ArcTopology	ArcTopology	円の位相情報

(1) 実装方法

円弧の実装方法は，円と同様に円周上の各頂点を算出して近似的に描画する．ただし，円弧の場合は，始角と終角があるため，円周上の全ての頂点を算出するのではなく，始角と終角の間だけ算出する．また，円弧は閉じた曲線でないため，glBegin 関数の引数には，GL_LINE_STRIP を指定する．円弧の実装例を次に示す．

```
// 描画の開始
glBegin(GL_LINE_STRIP);
    // 円弧を描画するための繰り返し
    for(int i=0;i<alX->Count;i++){
        // 描画の実行
        glVertex3d( (double)alX[i], (double)alY[i], (double)alZ[i]);
    }
// 描画の終了
glEnd();
```

(2) 描画例

本研究で実装した円弧を図 5.15 に示す.

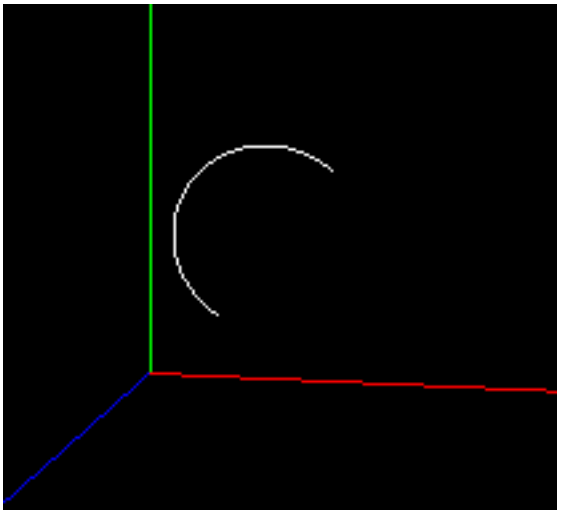


図 5.15　本研究で実装した円弧

5.5.5　楕円

楕円は，2 定点からの距離の和が一定となる点の集合でできる曲線である．楕円の概要を図 5.16 に示す.

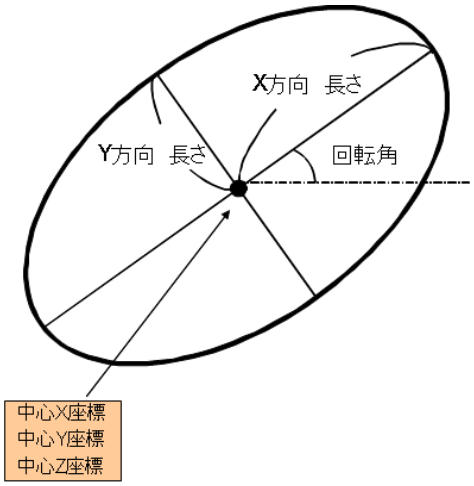


図 5.16　楕円の概要

本研究では，楕円の中心点の X 座標，Y 座標，Z 座標，X 方向半径，Y 方向半径と回転角の値を基に楕円を描画する．楕円のパラメータを表 5.9 に示す．

表 5.9　楕円のパラメータ

パラメータ	型	説明
m_dCenterX	double	中心点の X 座標
m_dCenterY	double	中心点の Y 座標
m_dCenterZ	double	中心点の Z 座標
m_dRadiusX	double	X 方向の半径
m_dRadiusY	double	Y 方向の半径
m_EllipseTopology	EllipseTopology	楕円の位相情報

(1) 実装方法

楕円の実装方法は，中心点の X 座標，Y 座標，Z 座標と X 軸方向半径の長さ，Y 軸方向半径の長さと回転角から楕円の円周上の各頂点の座標を算出し，その頂点を結び直線を描画する．楕円の円周上の各頂点の算出方法は，円と同様である．楕円の実装例を次に示す．

```
    // 楕円の描画の開始
    glBegin(GL_LINE_LOOP);
        // 楕円を描画するための繰り返し
        for(int i=0;i<alX->Count;i++){
            // 楕円の描画
            glVertex3d( (double)alX[i], (double)alY[i], (double)alZ[i]);
        }
    // 楕円の描画の終了
    glEnd();
```

(2) 描画例

本研究で実装した楕円を図 5.17 に示す.

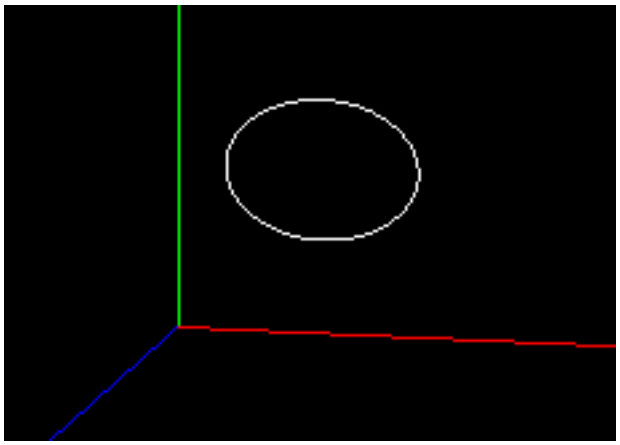


図 5.17　本研究で実装した楕円

5.5.6　楕円弧

楕円弧は，楕円の円周の一部の曲線である．楕円弧の概要を図 5.18 に示す.

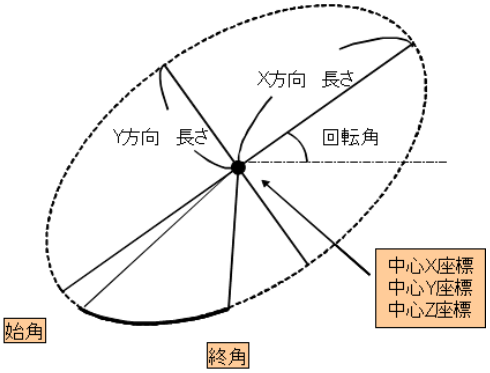


図 5.18　楕円弧の概要

本研究では，楕円弧の中心点の X 座標，Y 座標，Z 座標，X 方向半径，Y 方向半径，描画方向，始角，終角と回転角の値を基に楕円弧を描画する．楕円弧のパラメータを表 5.10 に示す.

表 5.10　楕円弧のパラメータ

パラメータ	型	説明
m_dCenterX	double	中心の X 座標
m_dCenterY	double	中心の Y 座標
m_dCenterZ	double	中心の Z 座標
m_dRadiusX	double	X 方向の半径
m_dRadiusY	double	Y 方向の半径
m_iDirection	int	描画方向

m_dStartAngle	double	始角
m_dEndAngle	double	終角
m_EllipseArcTopology	EllipseArcTopology	楕円弧の位相情報

(1) 実装方法

楕円弧の実装方法は，中心点の X 座標，Y 座標，Z 座標と X 軸方向半径の長さ，Y 軸方向半径の長さ，始角，終角と回転角を基に楕円上の各頂点の座標を算出し，頂点を結ぶ直線で描画する．各頂点の算出方法は円弧と同様である．楕円弧の実装例を次に示す．

```
// 描画の開始
glBegin(GL_LINE_STRIP);
    // 楕円弧を描画するための繰り返し
    for(int i=0;i<alX->Count;i++){
        // 描画の実行
        glVertex3d( (double)alX[i], (double)alY[i], (double)alZ[i] );
    }
// 描画の終了
glEnd();
```

(2) 描画例

本研究で実装した楕円弧を図 5.19 に示す．

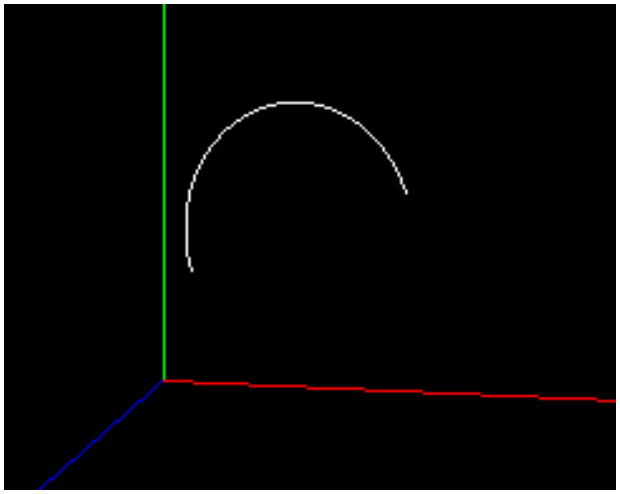


図 5.19　本研究で実装した楕円弧

5.5.7　ベジェ曲線

ベジェ曲線は，コンピュータ上で滑らかな曲線を描く際に使用される N 個の制御点から得られる曲線である．ベジェ曲線の概要を図 5.20 に示す．

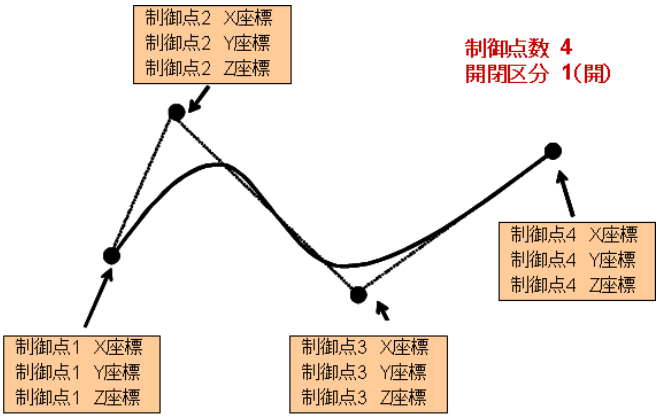


図 5.20　ベジェ曲線の概要

本研究では，頂点数，制御点数，X 座標の配列，Y 座標の配列と Z 座標の配列と開閉区分の値を基にベジェ曲線を描画する．開閉区分では，始点と終点を結ぶかどうかを指定する．ベジェ曲線のパラメータを表 5.11 に示す．

表 5.11　ベジェ曲線のパラメータ

パラメータ	型	説明
m_iOpenClose	int	開閉区分
m_iNumber	int	頂点数
m_iOrder	int	制御点数
m_alX	ArrayList	X 座標の配列
m_alY	ArrayList	Y 座標の配列
m_alZ	ArrayList	Z 座標の配列
m_BezierSuplineTopology	BezierSuplineTopology	ベジェ曲線の位相情報

(1) 実装方法

ベジェ曲線の実装方法は，glMap1d 関数，glEnable 関数，glEvalCoordld 関数を用いてベジェ曲線を描画する．まず，glMap1d 関数を用いて，描画するベジェ曲線のパラメータを設定する．次に，glEnable 関数を用いて，ベジェ曲線を有効化する．最後に，有効化したベジェ曲線を描画する．glMap1d 関数の定義を次に示します．

```
void glMap1d( GLenum target, GLdouble u1, GLdouble u2, Glint stride,GLint order,
              const GLdouble* points);
```

glMap1d 関数の引数の説明を表 5.12 に示す．

表 5.12　glMap1d 関数の引数

引数	型	説明
target	GLenum	エバリュエータで生成する値の種類
u1	GLdouble	パラメータの計算方法
u2	GLdouble	パラメータの計算方法
stride	Glint	曲線の始点と終点に用いる要素数
order	Glint	ベジェ曲面の階数
points	GLdouble*	制御点の配列

glMap1d 関数で設定した曲線を有効化するために glEnable 関数を使用する必要がある．glEnable 関数の定義を次に示す．

```
void glEnable(GLenum target);
```

glEnable 関数の引数の説明を表 5.13 に示す．

表 5.13　glEnable 関数の引数

引数	型	説明
target	GLenum	OpenGL の特定機能を示す定数

有効化したベジェ曲線を描画するために glEvalCoordld 関数を使用する必要がある．glEvalCoordld 関数の定義を次に示す．

```
void glEvalCoord1d(GLdouble u);
```

glEvalCoord1d 関数の引数の説明を表 5.14 に示す．

表 5.14　glEvalCoord1d 関数の引数

引数	型	説明
u	GLdouble	基底関数で算出する領域の座標

ベジェ曲線の実装例を次に示す．

```
// 開閉区分が 0（閉じる）
if(m_iOpenClose == 0){
    // 線分の描画開始
    glBegin(GL_LINE_STRIP);
        // 線分の始点座標
        glVertex3d((double)alX[0],(double)alY[0],(double)alZ[0]);
```

```
            // 線分の終点座標
            glVertex3d(
                (double)alX[m_iNumber-1],
                (double)alY[m_iNumber-1],
                (double)alZ[m_iNumber-1]);
        // 線分の描画終了
        glEnd();
    }

    // ベジェ曲線の座標を格納する 2 次元配列の宣言
    double **dCtlPoint;
    // メモリの確保 (1 次元目の配列のサイズ指定)
    dCtlPoint=new double*[m_iNumber/(m_iOrder-1)];
    // 2 次元目の配列のメモリを確保するための繰り返し
    for(int i=0;i<m_iNumber/(m_iOrder-1);i++){
        // メモリの確保 (2 次元目の配列のサイズ指定)
        dCtlPoint[i]=new double[(m_iOrder)*3];
    }

    // 配列に値を格納するための繰り返し
    for(int i=0;i<((m_iNumber)/(m_iOrder-1));i++){
        // 配列に値を格納するための繰り返し
        for(int j=0;j<(m_iOrder)*3;j++){
            // X 座標値を格納
            if(j%3 == 0) dCtlPoint[i][j] = (double)(alX[j/3+i*(m_iOrder-1)]);
            // Y 座標値を格納
            if(j%3 == 1) dCtlPoint[i][j] = (double)(alY[j/3+i*(m_iOrder-1)]);
            // Z 座標値を格納
            if(j%3 == 2) dCtlPoint[i][j] = (double)(alZ[j/3+i*(m_iOrder-1)]);
        }
    }

    // ベジェ曲線を複数描画するための繰り返し
    for(int i=0;i<(m_iNumber/(m_iOrder-1));i++){
        // i 番目のベジェ曲線を設定
        glMap1d(GL_MAP1_VERTEX_3, 0, 1, 3, (m_iOrder), dCtlPoint[i]);
        // ベジェ曲線の有効化
        glEnable(GL_MAP1_VERTEX_3);

        // ベジェ曲線の描画の実行
        glBegin(GL_LINE_STRIP);
            // ベジェ曲線を描画するために分割回数繰り返し
            for(int t = 0; t <= iStep; ++t){
                // ベジェ曲線の描画
                glEvalCoord1d(t/(iStep*1.0));
            }
        // ベジェ曲線の描画の終了
        glEnd();
        glDisable(GL_MAP1_VERTEX_3);
    }
```

ベジェ曲線の描画では，まず，開閉区分が 0（閉じる）の場合，始点と終点を結び，曲線を閉じる．次に，制御点の座標を配列に格納する．これは，glMap1d 関数では，制御点の座

標を配列で渡す必要があるためである．そして，glMap1d 関数を用いてベジェ曲線を描画する．glMap1d 関数の引数には，3 次元空間上にベジェ曲線を描くため，target は GL_MAP1_VERTEX_3 を指定する．また，指定したパラメータをそのまま適用するため，u1 と u2 は 0，1 を指定する．stride は次の頂点は何要素先なのかを指定するもので，本研究では，3 次元空間上に図形を描画するため，３を指定する．そして，order は制御点数を指定する．最後に，glEnable 関数と glEvalCoord1d 関数を用いて，設定したベジェ曲線を描画する．

(2) 描画例

本研究で実装したベジェ曲線を図 5.21 に示す．

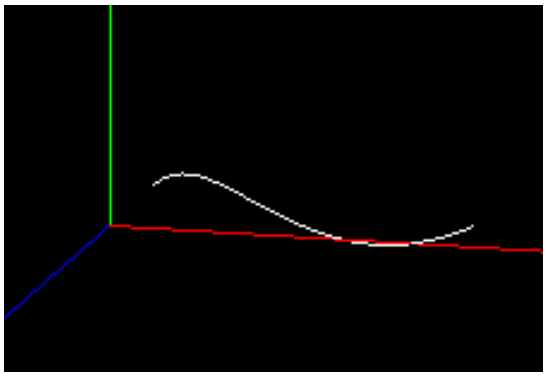


図 5.21　本研究で実装したベジェ曲線

5.5.8　NURBS曲線

NURBS 曲線は, 非一様有利スプライン曲線ともいい, B-Spline 曲線から派生したもので, 柔軟性に優れた曲線である．NURBS 曲線の概要を図 5.22 に示す．

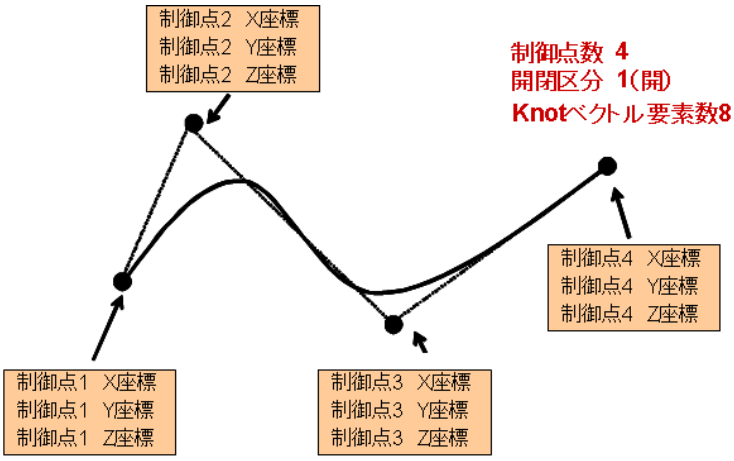


図 5.22　NURBS 曲線の概要

本研究では，頂点数，制御点数，knot ベクトルの配列と X 座標の配列，Y 座標の配列，Z 座標の配列，ウェイトと開閉区分の値を基に NURBS 曲線を描画する．knot ベクトルは，曲線を調整するためのもので，要素数が（次数+制御点数+1）である．また，ウェイトは，各頂点の頂点座標に対応し，各制御点に重みを持たせる．NURBS 曲線のパラメータを表 5.15 に示す．

表 5.15　NURBS 曲線のパラメータ

パラメータ	型	説明
m_iOpenClose	int	開閉区分
m_iNumber	int	頂点数
m_iOrder	int	制御点数
m_alKnot	ArrayList	knot ベクトルの配列
m_alX	ArrayList	X 座標の配列
m_alY	ArrayList	Y 座標の配列
m_alZ	ArrayList	Z 座標の配列
m_alWeight	ArrayList	重みの配列
m_NurbsSplineTopology	NurbsSplineTopology	NURBS 曲線の位相情報

(1) 実装方法

NURBS 曲線の実装方法は，gluNurbsSurface 関数と gluNurbsProperty 関数を使用する．gluNurbsSurface 関数の定義を次に示す．

```
gluNurbsCurve( GLUnurbs* nobj, GLint nknots, GLfloat* knot, GLint stride,
               GLfloat* ctlarray, GLint order, GLenum type);
```

gluNurbsSurface 関数の引数の説明を表 5.16 に示す.

表 5.16　gluNurbsSurface 関数の引数

引数	型	説明
nobj	GLUnurbs*	NURBS オブジェクト
nknots	Glint	節点数
knot	GLfloat	節点の配列
stride	GLint	曲線の始点と終点に用いる要素数
otlarray	GLfloat	制御点の配列
order	GLint	NURBS 曲線の階数
type	GLenum	曲線の形式

gluNurbsProperty 関数の定義を次に示す.

```
void gluNurbsProperty ( GLUnurbs *nobj, GLenum property, GLfloat value);
```

gluNurbsProperty 関数の引数の説明を表 5.17 に示す.

表 5.17　gluNurbsProperty 関数の引数

引数	型	説明
nobj	GLUnurbs*	NURBS オブジェクト
property	GLenum	設定するプロパティ
value	GLfloat	設定したプロパティの値

NURBS 曲線の実装例を次に示す.

```
// 開閉区分が 0 (閉じる)
if(m_iOpenClose == 0){
    System::Collections::ArrayList^ allineX = gcnew System::Collections::ArrayList();
    System::Collections::ArrayList^ allineY = gcnew System::Collections::ArrayList();
    System::Collections::ArrayList^ allineZ = gcnew System::Collections::ArrayList();
    NurbsToPolyline(
            allineX,allineY,allineZ,m_alKnot,alX,alY,alZ,m_alWeight,m_iOrder,2);

    // 線分の描画開始
    glBegin(GL_LINE_STRIP);
        // 線分の始点座標
        glVertex3d((double)allineX[0],(double)allineY[0],(double)allineZ[0]);
        // 線分の終点座標
        glVertex3d(
            (double)allineX[allineX->Count-1],
            (double)allineY[allineY->Count-1],
            (double)allineZ[allineZ->Count-1]);
    // 線分の描画終了
```

```
        glEnd();
    }

    // Nurbs 曲線の座標を格納する配列の生成
    float *fCtlPoint;
    // メモリの確保
    fCtlPoint=new float[m_iNumber*4];

    // 配列に値を格納するための繰り返し
    for(int i=0;i<m_iNumber*4;i++){
        // 配列に値を格納
        if(i%4 == 0) fCtlPoint[i] = (double)alX[i/4];
        if(i%4 == 1) fCtlPoint[i] = (double)alY[i/4];
        if(i%4 == 2) fCtlPoint[i] = (double)alZ[i/4];
        if(i%4 == 3) fCtlPoint[i] = (double)m_alWeight[i/4];
    }

    // knot ベクトルの値を格納する配列の生成
    float *fKnot;
    // knot ベクトルのメモリ確保
    fKnot=new float[m_alKnot->Count];
    // knot ベクトルに値を格納するための繰り返し
    for(int i=0;i<m_alKnot->Count;i++){
        // knot ベクトルに値を格納
        fKnot[i] = (double)(m_alKnot[i]);
    }

    // Nurbs オブジェクトの宣言
    GLUnurbsObj *NurbsObj;
    // Nurbs オブジェクト作成
    NurbsObj = gluNewNurbsRenderer();
    // サンプリング範囲の指定(小さいほど滑らか)
    gluNurbsProperty(NurbsObj, GLU_SAMPLING_TOLERANCE,0.1);

    gluNurbsCurve(
        NurbsObj, m_alKnot->Count, fKnot, 4, fCtlPoint, m_iOrder, GL_MAP1_VERTEX_4);

    // 行列スタックの解放
    glPopMatrix();

    delete fKnot;
    delete fCtlPoint;
```

NURBS 曲線を描画するには，まず，開閉区分が 0（閉じる）の場合，始点と終点を結び，NURBS 曲線を閉じる．次に，制御点の座標，knot ベクトルを配列に格納する．これは，gluNurbsCurve 関数では，制御点の座標や knot ベクトルを配列で渡す必要があるためである．そして，NURBS オブジェクトを作成し，gluNurbsProperty 関数の引数とする．gluNurbsProperty 関数の引数には，まず，先ほど作成した NURBS オブジェクトを指定する．次に，property は GLU_SAMPLING_TOLERANCE を指定し，value は 0.5 を指定する．

最後に，gluBeginCurve 関数，gluNurbsCurve 関数，gluEndCurve 関数を用いて NURBS 曲線を描画する．gluBeginCurve 関数や gluEndCurve 関数は，NURBS オブジェクトを指定し，gluNurbsCurve 関数は，nobj に NURBS オブジェクト，nknots にノットベクトルの要素数，knot にノットベクトルの配列を指定する．stride は次の頂点は何要素先なのかを指定する．本研究では，本研究では，3 次元空間上に図形を描画するため，３を指定する．ctlarray は作成した制御点の配列を指定する．order には，制御点数を指定する．type には，3 次元空間上に NURBS 曲線を描くため，GL_MAP1_VERTEX_3 を指定する．

(2) 描画例

本研究で実装した NURBS 曲線を図 5.23 に示す．

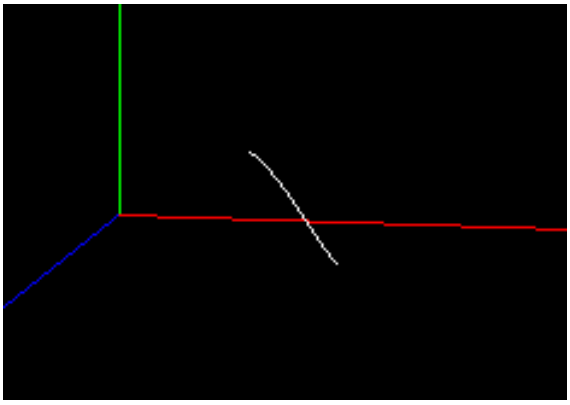


図 5.23　本研究で実装した NURBS 曲線

5.5.9　クロソイド

クロソイドとは，曲率が一定の比率で変化する特徴を持った曲線である．クロソイドは，直線と円弧を滑らかにつなぐことができるため，道路や鉄道の路線形状の設計に使用される．また，緩和曲線の 1 種であり，車の速度を一定とし，ハンドルを一定の角速度で回したときに車が描く曲線である．クロソイドの概要を図 5.24 に示す．

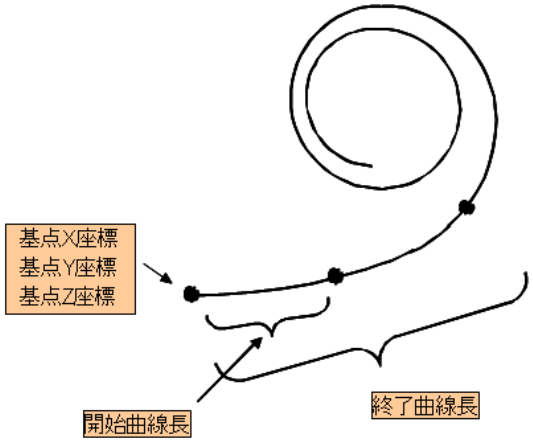


図 5.24　クロソイドの概要

本研究では，基点の X 座標，Y 座標，Z 座標，パラメータ，描画方向，開始曲線長と終了曲線長の値を基にクロソイドを描画する．パラメータとは，渦状の線の半径を算出する際に必要となる値である．描画方向は，クロソイドを時計まわりに描画するか，反時計まわりに描画するかを指定する．クロソイドのパラメータを表 5.18 に示す．

表 5.18　クロソイドのパラメータ

パラメータ	型	説明
m_dBaseX	double	基点の X 座標
m_dBaseY	double	基点の Y 座標
m_dBaseZ	double	基点の Z 座標
m_iParameter	double	パラメータ
m_iDirection	int	描画方向
m_dStartLength	double	開始曲線長
m_dEndLength	double	終了曲線長
m_ClothoidTopology	ClothoidTopology	クロソイドの位相情報

(1) 実装方法

クロソイドの実装方法は，クロソイドを折線に近似し，折線の各頂点の座標を算出してその頂点を結び描画する．各頂点の座標の算出方法は，まず，クロソイドでは，曲線長，半径から次の関係式が成立する．

$$\tau = L / 2R$$

なお，τ は X 軸とクロソイド上の点の接線とのなす角を表す．次に，クロソイド上の座標（X，Y，Z）は次式から算出する．

$$x = A\sqrt{2\tau}\left(1 - \frac{1}{2!\times 5}\tau^2 + \frac{1}{4!\times 9}\tau^4 - \frac{1}{6!\times 13}\tau^6 + \ldots\right)$$

$$y = A\tau\sqrt{2\tau}\left(\frac{1}{3} - \frac{1}{3!\times 7}\tau^2 + \frac{1}{5!\times 11}\tau^4 - \frac{1}{7!\times 15}\tau^6 + \ldots\right)$$

ただし，A はパラメータを表す．

具体的には，まず，上記の式からからτを算出する．次に，上記の式を下記のように一般式として，k を「1,2,3,」と変え，その時の座標を算出する．

$$A\sqrt{2\tau}((-1)^{k-1}\times\frac{1}{(2(k-1))!\times(4k-3)})\times\tau^{2(k-1)}$$

$$A\tau\sqrt{2\tau}((-1)^{k-1}\times\frac{1}{(2k-1)!\times(4k-1)})\times\tau^{2(k-1)}$$

そして，k と k-1 の時の座標の差が 0.001 になった時点で計算を終了する．これらの処理を曲線長の値をΔL ずつ加算し，クロソイド上の全ての座標を算出する．

クロソイドの実装例を次に示す．

```
// 頂点座標を算出
ClothoidToPolyline( alX, alY, alZ, m_iParameter, m_dStartLength, m_dEndLength,
              m_dBaseX, m_dBaseY, m_dBaseZ, m_iDirection, 1080);

// クロソイドの描画の開始
glBegin(GL_LINE_STRIP);
    // クロソイドを描画するための繰り返し
    for(int i=0;i<alX->Count;i++){
        // クロソイドの描画
        glVertex3d(
            (double)alX[i]-m_dBaseX,
            (double)alY[i]-m_dBaseY,
            (double)alZ[i]-m_dBaseZ);
    }
// クロソイドの描画の終了
glEnd();
```

各頂点座標の算出の実装例を次に示す．

```
// 分割の指定
double dStep = abs(dEndLength-dStartLength)/(iDivision_num*1.0);
int iDirectionFlagX;
// X 座標を格納する変数の生成
double* dX;
// Y 座標を格納する変数の生成
double* dY;
// Z 座標を格納する変数の生成
double* dZ;
// 半径を格納する変数の生成
double dRadius;
// 半径の設定
dRadius = (iParameter * iParameter)/abs(dEndLength-dStartLength);

// 開始曲線長＜終了曲線長の場合
if( dStartLength < dEndLength ){
    // 向きフラグの設定
    iDirectionFlagX = 0;
}else{
    // 開始曲線長を保存する変数の生成
```

```
        double dChange;
        // 開始曲線長の値を保存
        dChange = dStartLength;
        // 終了曲線長の値を開始曲線長に代入
        dStartLength = dEndLength;
        //  保存した値を終了曲線長に代入
        dEndLength = dChange;
        // 向きフラグの設定
        iDirectionFlagX = 1;
    }

    // X 座標値を格納する配列の生成
    dX = new double[iDivision_num];
    // Y 座標値を格納する配列の生成
    dY = new double[iDivision_num];
    // Z 座標値を格納する配列の生成
    dZ = new double[iDivision_num];

    // 繰り返し数を格納する変数の生成
    int iCount = 0;

    // クロソイドの頂点の算出
    for( double i = 0;i < dEndLength ;i = i + dStep ){
        // 変数 i が開始曲線長以上の場合
        if( i >= dStartLength ){
            // 座標値を格納する変数の生成
            double dTempX,dTempY;
            // クロソイド曲線の座標を算出
            Calculator(iParameter,i,dRadius,&dTempX,&dTempY);
            // 向きフラグ(iDirection)が 1 の場合
            if( iDirection == 1 ){
                dTempY = -dTempY;
            }
            // 向きフラグ(iDirectionFlagX)が 1 の場合
            if( iDirectionFlagX == 1 ){
                dTempX = -dTempX;
            }
            // 頂点 X の算出
            dX[iCount] = dTempX + dBaseX;
            // 頂点 Y の算出
            dY[iCount] = dTempY + dBaseY;
            // 頂点 Z の算出
            dZ[iCount] = dBaseZ;
            // カウンタの値の加算
            iCount++;
        }
    }
```

頂点座標の算出の実装例を次に示す.

```
    void FFeatureBase::Calculator(
        double dParam, double dLength, double dRadius, double* dX, double* dY)
    {
```

```
        // クロソイド曲線の計算に利用する変数の生成
        double ds,dpk,dqk,dtau,dk,dxk,dxo,dsg,dx,dy;

        if( dRadius >= 0 ){
             dsg = 1;
        }else{
             dsg = -1;
        }

        // X 座標の算出
        ds = 1;
        dpk = 1;
        dk = 1;
        dtau = dLength / (2 * abs(dRadius));
        dxk = dParam * sqrt(2 * dtau);
        dxo = 0;
        dx = dxk;

        // 座標の差が 0.001 になった時点で計算終了
        while (abs(dx - dxo) >= 0.001){
             dxo = dx;
             dk = dk + 1;
             dpk = dpk * (-1) / ((2 * dk - 2) * (2 * dk - 3)) * dtau * dtau;
             dqk = dpk / (4 * dk - 3);
             ds = ds + dqk;
             dx = dxk * ds;
        }
        (*dX) = Rund(dx, 3);

        // Y 座標の算出
        ds = (double)(1.0 / 3.0);
        dpk = 1.0;
        dk = 1.0;
        dxk = dParam * dtau * sqrt(2 * dtau);
        dxo = 0.0;
        dy = dxk * ds;
        // 座標の差が 0.001 になった時点で計算終了
        while (abs(dy - dxo) >= 0.001){
             dxo = dy;
             dk = dk + 1;
             dpk = dpk * (-1) / ((2 * dk - 1) * (2 * dk - 2)) * dtau * dtau;
             dqk = dpk / (4 * dk - 1);
             ds = ds + dqk;
             dy = dxk * ds;
        }
        (*dY) = dsg * Rund(dy, 3);
   }
```

(2) 描画例

本研究で実装したクロソイドを図 5.25 に示す.

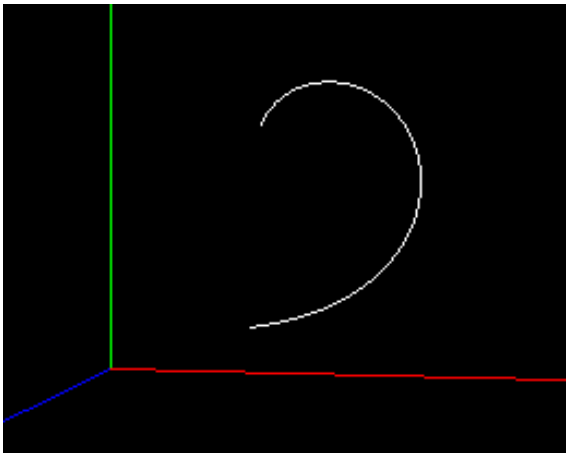


図 5.25　本研究で実装したクロソイド

5.6　面要素の表現方法

本節では，面要素として，直方体，円錐，円柱，球，トーラス，ウェッジ，ベジェ曲面，NURBS 曲面，掃引体，回転体について OpenGL での表現方法について纏める．

5.6.1　直方体

直方体は，6 個の面から構成される立体の図形である．直方体の概要を図 5.26 に示す．

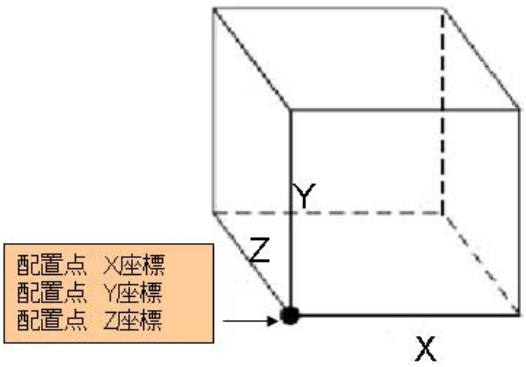


図 5.26　直方体の概要

本研究では，配置点の X 座標，Y 座標，Z 座標，X 軸方向への大きさ，Y 軸方向への大きさと Z 軸方向への大きさの値を基に直方体を描画する．直方体のパラメータを表 5.19 に示す．

表 5.19　直方体のパラメータ

パラメータ	型	説明
m_dCenterX	double	配置点 X 座標
m_dCenterY	double	配置点 Y 座標
m_dCenterZ	double	配置点 Z 座標
m_dLengthX	double	X 軸方向への大きさ
m_dLengthY	double	Y 軸方向への大きさ
m_dLengthZ	double	Z 軸方向への大きさ
m_BlockTopology	BlockTopology	直方体の位相情報

(1) 実装方法

OpenGL には直方体を描画する関数が用意されていないため，直方体の実装方法は，glVertex3d 関数を用いて正面，奥面，上面，下面，右面と左面の 6 つの面を作成し，直方体を描画する．直方体の実装例を次に示す．

```
// 正面
// 描画の開始
glBegin(GL_POLYGON);
    // 描画の実行
    glVertex3d(0, 0, 0);
    glVertex3d(m_dLengthX, 0, 0);
    glVertex3d(m_dLengthX, m_dLengthY, 0);
    glVertex3d(0, m_dLengthY, 0);
// 描画の終了
glEnd();

// 奥面
// 描画の開始
glBegin(GL_POLYGON);
    // 描画の実行
    glVertex3d(m_dLengthX, 0, m_dLengthZ);
    glVertex3d(0, 0, m_dLengthZ);
    glVertex3d(0, m_dLengthY, m_dLengthZ);
    glVertex3d(m_dLengthX, m_dLengthY, m_dLengthZ);
// 描画の終了
glEnd();

// 下面
// 描画の開始
glBegin(GL_POLYGON);
    // 描画の実行
    glVertex3d(0, 0, m_dLengthZ);
    glVertex3d(m_dLengthX, 0, m_dLengthZ);
    glVertex3d(m_dLengthX, 0, 0);
    glVertex3d(0, 0, 0);
// 描画の終了
glEnd();

// 上面
```

```
// 描画の開始
glBegin(GL_POLYGON);
     // 描画の実行
     glVertex3d(0, m_dLengthY, 0);
     glVertex3d(m_dLengthX, m_dLengthY, 0);
     glVertex3d(m_dLengthX, m_dLengthY, m_dLengthZ);
     glVertex3d(0, m_dLengthY, m_dLengthZ);
// 描画の終了
glEnd();

// 右面
// 描画の開始
glBegin(GL_POLYGON);
     // 描画の実行
     glVertex3d(m_dLengthX, 0, 0);
     glVertex3d(m_dLengthX, 0, m_dLengthZ);
     glVertex3d(m_dLengthX, m_dLengthY, m_dLengthZ);
     glVertex3d(m_dLengthX, m_dLengthY, 0);
// 描画の終了
glEnd();

// 左面
// 描画の開始
glBegin(GL_POLYGON);
     // 描画の実行
     glVertex3d(0, 0, m_dLengthZ);
     glVertex3d(0, 0, 0);
     glVertex3d(0, m_dLengthY, 0);
     glVertex3d(0, m_dLengthY, m_dLengthZ);
// 描画の終了
glEnd();

// 変換行列の復元
glPopMatrix();
// 描画実行の強制
glFlush();
```

(2) 描画例

本研究で実装した直方体の描画例を図 5.27 に示す.

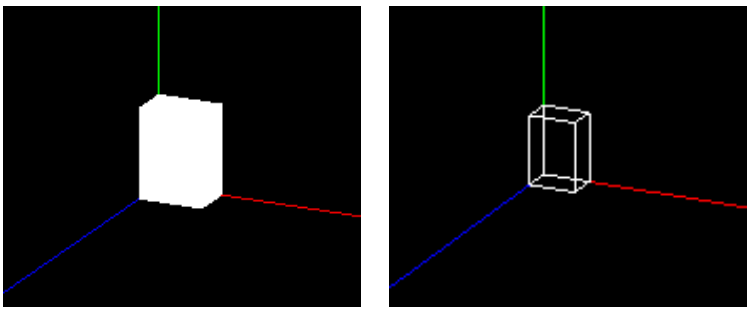


図 5.27　本研究で実装した直方体

5.6.2　円錐

円錐は，円を底面に持つ錐状の立体の図形である．円錐の概要を図 5.28 に示す．

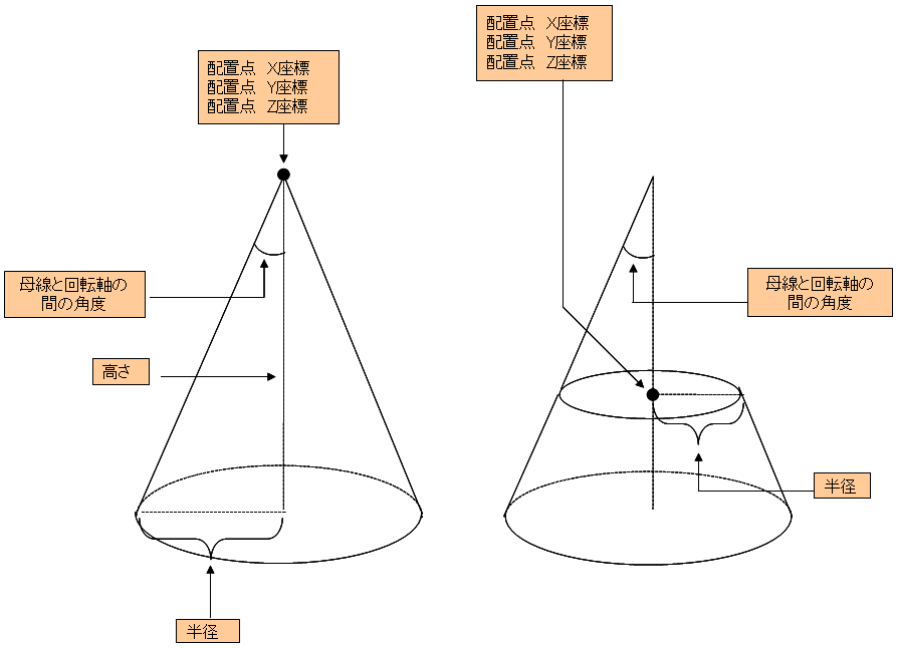


図 5.28　円錐の概要

本研究では，配置点 X 座標，Y 座標，Z 座標，高さ，半径と母線と回転軸の間の角度の値を基に円錐を描画する．円錐のパラメータの表 5.20 に示す．

表 5.20　円錐のパラメータ

パラメータ	型	説明
m_dCenterX	double	配置点 X 座標
m_dCenterY	double	配置点 Y 座標
m_dCenterZ	double	配置点 Z 座標
m_dHeight	double	高さ
m_dRadius	double	半径
m_dSemiAngle	double	母線と回転軸の間の角度
m_ConeTopology	ConeTopology	円錐の位相情報

(1) 実装方法

円錐の実装方法は，OpenGL の gluCylinder 関数を使用して描画する．gluCylinder 関数の定義を次に示す．

```
void gluCylinder(GLUquadricObj* qobj,GLdouble baseRadius,GLdouble topRadius, GLdouble height
                 GLint slices ,GLint stacks)
```

gluCylinder 関数の引数の説明を表 5.21 に示す.

表 5.21　gluCylinder 関数の引数

引数	型	説明
qobj	GLUquadricObj*	2 次曲面オブジェクト
baseRadius	GLdouble	底面の半径
topRadius	GLdouble	上面の半径
height	GLdouble	高さ
slices	GLint	Z 軸を中心に放射状の細分数
stacks	GLint	Z 軸に沿った輪切りの細分数

gluCylinder 関数は，円錐の底面を描画することができないため，底面は円を描画することで実現する．円錐の実装例を次に示す.

```
// 図形描画時の分割数
int iStep = 1080;

glBegin(GL_POLYGON);
     for(int i=0;i<iStep;i++)
     {
          // 変数の宣言
          double dX, dY, dZ;

          dX = m_dRadius*cos(2.0*PAI*(double)i/(double)iStep);
          dY = m_dRadius*sin(2.0*PAI*(double)i/(double)iStep);
          dZ = 0;
          // 頂点座標の設定
          glVertex3d( dX, dY, dZ);
     }
glEnd();
// 描画の実行
gluCylinder(m_Cone,m_dRadius,TopRadius,m_dHeight,Slices,Stacks);
```

(2) 描画例

本研究で実装した円錐の描画例を図 5.29 に示す.

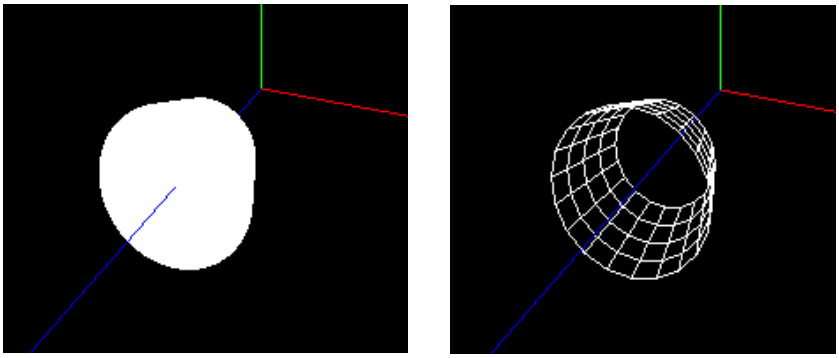


図 5.29　本研究で実装した円錐

5.6.3　円柱

円柱は，円を上面と底面に持つ筒状の立体の図形である．円柱の概要を図 5.30 に示す．

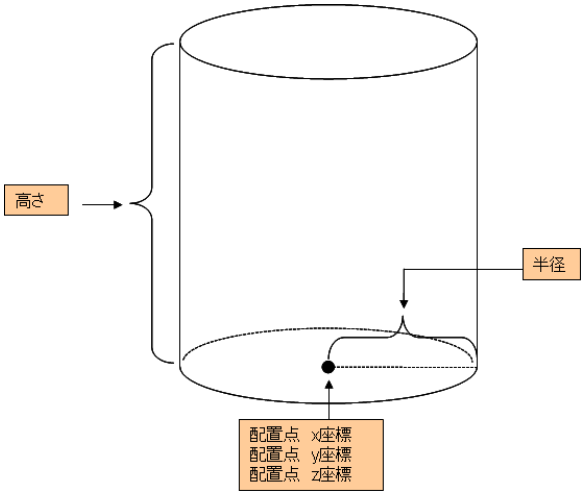


図 5.30　円柱の概要

本研究では，配置点 X 座標，Y 座標，Z 座標，高さと半径の値を基に円柱を描画する．円柱のパラメータを表 5.22 に示す．

表 5.22　円柱のパラメータ

パラメータ	型	説明
m_dCenterX	double	配置点 X 座標
m_dCenterY	double	配置点 Y 座標
m_dCenterZ	double	配置点 Z 座標
m_dHeight	double	高さ
m_dRadius	double	半径
m_CylinderTopology	CylinderTopology	円柱の位相情報

(1) 実装方法

円柱の実装方法は，円錐の実装に使用した gluCylinder 関数を用いて描画する．ただし，円錐と同様で gluCylinder 関数は底面を描画しないため，円を描画することで底面を描画する．円柱の実装例を次に示す．

```
// 図形描画時の分割数
int iStep = 1080;

glBegin(GL_POLYGON);
     for(int i=0;i<iStep;i++)
     {
          // 変数の宣言
          double dX, dY, dZ;

          dX = m_dRadius*cos(2.0*PAI*(double)i/(double)iStep);
          dY = m_dRadius*sin(2.0*PAI*(double)i/(double)iStep);
          dZ = 0;
          // 頂点座標の設定
          glVertex3d( dX, dY, dZ);
          glVertex3d( dX, dY, dZ +m_dHeight);
     }
glEnd();
// 描画の実行
gluCylinder(m_Cylinder,m_dRadius,m_dRadius,m_dHeight,iSlices,iStacks);
```

(2) 描画例

本研究で実装した円柱の描画例を図 5.31 に示す．

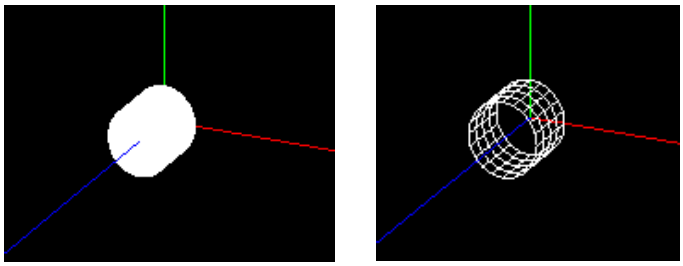


図 5.31　本研究で実装した円柱

5.6.4　球

球は，空間上の任意の点を中心に等間隔に配置された点の集合体である．球の概要を図 5.32 に示す．

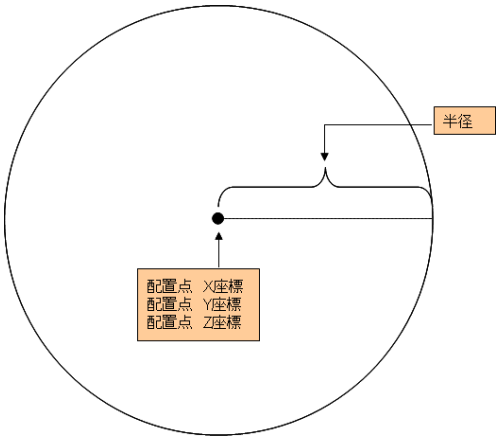


図 5.32　球の概要

本研究では，配置点 X 座標，Y 座標，Z 座標と半径の値を基に球を描画する．球のパラメータを表 5.23 に示す．

表 5.23　球のパラメータ

パラメータ	型	説明
m_dCenterX	double	配置点 X 座標
m_dCenterY	double	配置点 Y 座標
m_dCenterZ	double	配置点 Z 座標
m_dRadius	double	半径
m_SphereTopology	SphereTopology	球の位相情報

(1) 実装方法

球の実装方法は，gluSphere 関数を用いて描画する．gluSphere 関数の定義を次に示す．

```
void gluSphere(GLUquadricObj *qobj,GLdouble radius,GLint slices,GLint stacks)
```

gluSphere 関数の引数の説明を表 5.24 に示す．

表 5.24　gluSphere 関数の引数

引数	型	説明
qobj	GLUquadricObj*	2 次曲面オブジェクト
radius	GLdouble	球の半径
slices	GLint	Z 軸を中心に放射状の細分数
stacks	GLint	Z 軸に沿った輪切りの細分数

球の実装例を次に示す．

```
// 球の描画
gluSphere(m_Sphere, m_dRadius, Slices, Stacks);
```

(2) 描画例

本研究で実装した球の描画例を図 5.33 に示す.

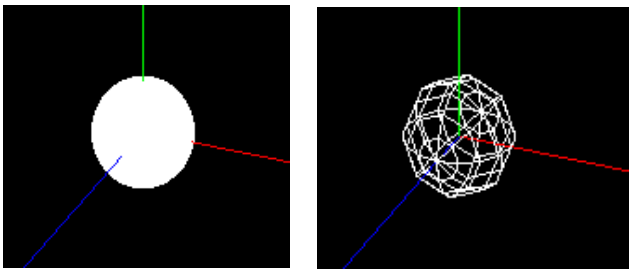


図 5.33　本研究で実装した球

5.6.5　トーラス

トーラスは，閉曲面から構成されるドーナツ状の立体である．トーラスの概要を図 5.34 に示す.

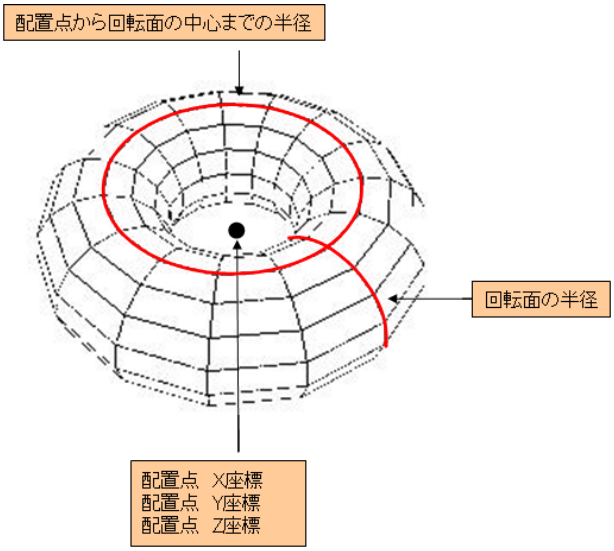


図 5.34　トーラスの概要

本研究では，配置点 X 座標，Y 座標， Z 座標，配置点から回転面の中心までの半径と回転面の半径の値を基にトーラスを描画する．トーラスのパラメータを表 5.25 に示す.

表 5.25　トーラスのパラメータ

パラメータ	型	説明
m_dCenterX	double	配置点 X 座標
m_dCenterY	double	配置点 Y 座標
m_dCenterZ	double	配置点 Z 座標
m_dMajorRadius	double	配置点から回転面の中心までの半径
m_dMinorRadius	double	回転面の半径
m_TorusTopology	TorusTopology	トーラスの位相情報

(1) 実装方法

トーラスの実装方法は, glutWireTorus 関数もしくは glutSolidTorus 関数を用いて描画する. glutWireTorus 関数は，ワイヤーフレームモデルを描画する場合に使用し，glutSolidTorus 関数は，サーフェスモデルの場合に使用する．glutWireTorus 関数の定義を次に示す.

```
void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings)
```

glutWireTorus 関数の引数の説明を表 5.26 に示す．また，glutSolidTorus 関数の定義は, glutWireTorus 関数と同様である.

表 5.26　glutWireTorus 関数の引数

引数	型	説明
innerRadius	GLdouble	トーラスの内径
outerRadius	GLdouble	トーラスの外径
nsides	GLint	断面部分の細分数
rings	GLint	管の沿った細分数

トーラスの実装例を次に示す.

```
// モデル表示
switch(m_FeatureModelType){
// ワイヤーフレーム
case FEATURE_MODEL_TYPE::WIRE:
     glutWireTorus(m_dMinorRadius, m_dMajorRadius, iNSides, iRings);
     break;
// サーフェス
case FEATURE_MODEL_TYPE::SURFACE:
     glutSolidTorus(m_dMinorRadius, m_dMajorRadius, iNSides, iRings);
     break;
}
```

(2) 描画例

本研究で実装したトーラスの描画例を図 5.35 に示す.

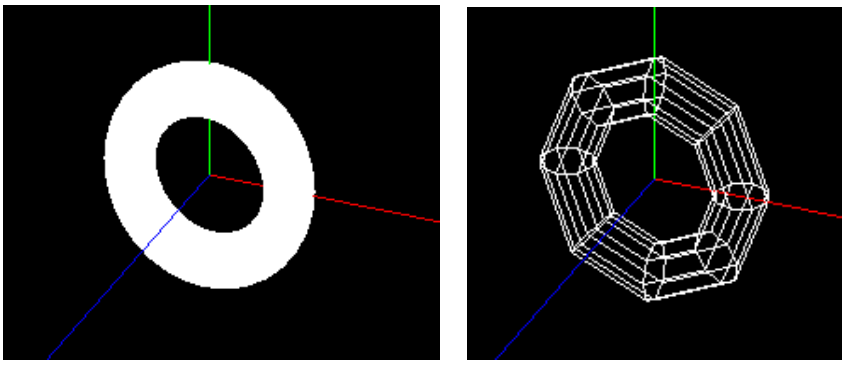


図 5.35　本研究で実装したトーラス

5.6.6　ウェッジ

ウェッジは，4 個，または，6 個の面から構成される立体の図形である．ウェッジの概要を図 5.36 に示す.

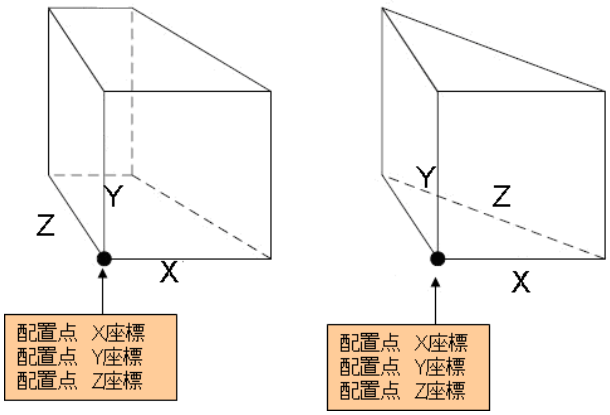


図 5.36　ウェッジの概要

本研究では，配置点 X 座標，Y 座標，Z 座標，X 軸方向への大きさ，Y 軸方向への大きさ，Z 軸方向への大きさと X 軸方向の短い辺の大きさの値を基にウェッジを描画する．各面の大きさは，X 軸方向への大きさ，Y 軸方向への大きさ，Z 軸方向への大きさ，X 軸方向の短い辺の大きさを指定することにより決定する．ウェッジのパラメータを表 5.27 に示す.

表 5.27　ウェッジのパラメータ

パラメータ	型	説明
m_dCenterX	double	配置点 X 座標
m_dCenterY	double	配置点 Y 座標
m_dCenterZ	double	配置点 Z 座標
m_dLengthX	double	X 軸方向への大きさ
m_dLengthY	double	Y 軸方向への大きさ
m_dLengthZ	double	Z 軸方向への大きさ
m_dltX	double	X 軸方向の短い辺の大きさ
m_WedgeTopology	WedgeTopology	ウェッジの位相情報

(1) 実装方法

OpenGL には，ウェッジを描画する関数が用意されていないため，ウェッジの実装方法，は．直方体と同様に正面，奥面と 4 つの側面の 6 つの面を作成し，描画する．ウェッジの実装例を次に示す．

```
// 底面
// 描画の開始
glBegin(GL_POLYGON);
    // 描画の実行
    glVertex3d(0, 0, 0);
    glVertex3d(m_dLengthX, 0, 0);
    glVertex3d(m_dLtX , m_dLengthY , 0);
    glVertex3d(0, m_dLengthY, 0);
// 描画の終了
glEnd();

// 上面
// 描画の開始
glBegin(GL_POLYGON);
    // 描画の実行
    glVertex3d(0, 0, m_dLengthZ);
    glVertex3d(m_dLengthX, 0, m_dLengthZ);
    glVertex3d(m_dLtX, m_dLengthY, m_dLengthZ);
    glVertex3d(0, m_dLengthY, m_dLengthZ);
// 描画の終了
glEnd();

// 側面１
// 描画の開始
glBegin(GL_POLYGON);
    // 描画の実行
    glVertex3d(0, 0, 0);
    glVertex3d(m_dLengthX, 0, 0);
    glVertex3d(m_dLengthX, 0, m_dLengthZ);
    glVertex3d(0, 0, m_dLengthZ);
// 描画の終了
glEnd();
```

```
    // 側面２
    // 描画の開始
    glBegin(GL_POLYGON);
        // 描画の実行
        glVertex3d(m_dLengthX, 0, 0);
        glVertex3d(m_dLengthX, 0, m_dLengthZ);
        glVertex3d(m_dLtX, m_dLengthY, m_dLengthZ);
        glVertex3d(m_dLtX, m_dLengthY, 0);
    // 描画の終了
    glEnd();

    // 側面３
    // 描画の開始
    glBegin(GL_POLYGON);
        // 描画の実行
        glVertex3d(m_dLtX, m_dLengthY, m_dLengthZ);
        glVertex3d(0, m_dLengthY, m_dLengthZ);
        glVertex3d(0, m_dLengthY, 0);
        glVertex3d(m_dLtX, m_dLengthY, 0);
    // 描画の終了
    glEnd();

    // 側面４
    // 描画の開始
    glBegin(GL_POLYGON);
        // 描画の実行
        glVertex3d(0, m_dLengthY, m_dLengthZ);
        glVertex3d(0, 0, m_dLengthZ);
        glVertex3d(0, 0, 0);
        glVertex3d(0, m_dLengthY, 0);
    // 描画の終了
    glEnd();
```

(2) 描画例

本研究で実装したウェッジの描画例を図 5.37 に示す.

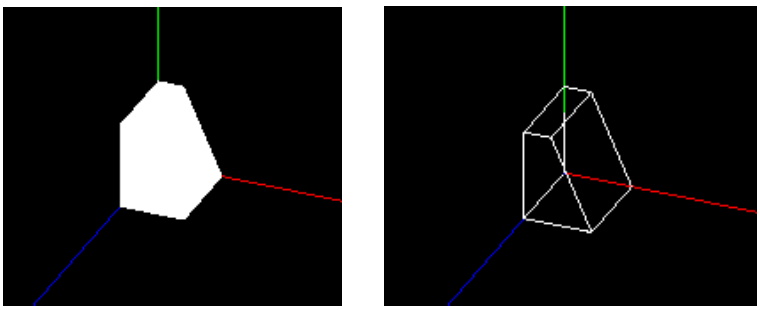


図 5.37　本研究で実装したウェッジ

5.6.7　ベジェ曲面

ベジェ曲面は，ベジェ曲線を拡張したもので，ベジェ曲線と同じ性質をもつ自由曲面である．ベジェ曲面の概要を図 5.38 に示す．

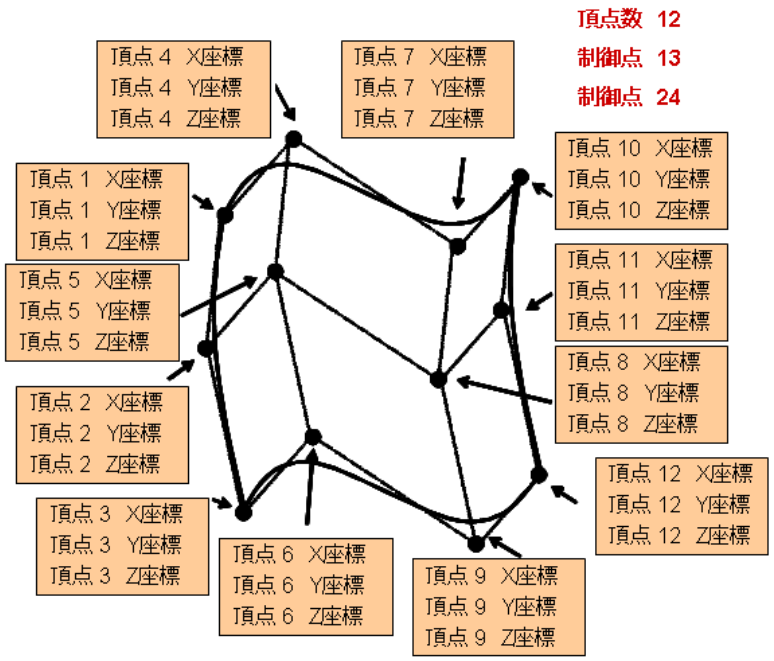


図 5.38　ベジェ曲面の概要

本研究では，頂点数，平面の制御点数，奥行の制御点数，X 座標の配列，Y 座標の配列と Z 座標の配列の値を基にベジェ曲面を描画する．ベジェ曲面のパラメータを表 5.28 に示す．

表 5.28　ベジェ曲面のパラメータ

パラメータ	型	説明
m_iNumber	int	頂点数
m_iOrder1	int	平面の制御点数
m_iOrder2	int	奥行の制御点数
m_alX	ArrayList	X 座標の配列
m_alY	ArrayList	Y 座標の配列
m_alZ	ArrayList	Z 座標の配列
m_BeziarSurfaceTopology	BeziarSurfaceTopology	ベジェ曲線の位相情報

(1) 実装方法

ベジェ曲面の実装方法は，glMap2d 関数を用いて描画する．glMap2d 関数の定義を次に示す．

```
void glMap2d( GLenum target, GLdouble u1, GLdouble u2, GLint ustride, GLint uorder,
              GLdouble v1, GLdouble v2, GLint vstride, GLint vorder, const GLdouble * points)
```

glMap2d 関数の引数の説明を表 5.29 に示す．

表 5.29　glMap2d 関数のパラメータ

引数	型	説明
target	GLenum	エバリュエータで生成する値の種類
u1	GLdouble	平面方向のパラメータの計算方法
u2	GLdouble	平面方向のパラメータの計算方法
ustride	GLint	平面方向の曲線の始点と終点に用いる要素数
uorder	GLin	平面方向の階数
v1	GLdouble	奥行き方向のパラメータの計算方法
v2	GLdouble	奥行き方向のパラメータの計算方法
vstride	GLint	奥行き方向の曲線の始点と終点に用いる要素数
vorder	GLint	奥行き方向の階数
points	GLdouble*	制御点の配列

ベジェ曲面の実装例を次に示す．

```
// ベジェ曲面を複数描画するための繰り返し
for(int i=0;i<(((m_iNumber/m_iOrder1)-1)/(m_iOrder2-1));i++){
    // i 番目のベジェ曲面を設定
    glMap2d(GL_MAP2_VERTEX_3, 0, 1, 3, m_iOrder1, 0, 1, 3*(m_iOrder1),
            m_iOrder2, dCtlPoint[i]);
    // ベジェ曲面の有効化
    glEnable(GL_MAP2_VERTEX_3);
    // 分割の設定
    double slice=0.05;

glBegin(GL_QUADS);
// 奥行方向描画の繰り返し
for(double v = 0; v < 1; v += slice){
    // 平面方向の描画の繰り返し
    for(double u = 0; u < 1; u += slice){
        // 四角形 ABCD の四隅を設定
        glEvalCoord2d(u,v);              // 頂点 A
        glEvalCoord2d(u+slice,v);        // 頂点 B
        glEvalCoord2d(u+slice,v+slice);  // 頂点 C
        glEvalCoord2d(u,v+slice);        // 頂点 D
    }
```

```
    }
    // ベジェ曲面（サーフェスモデル）の描画の終了
    glEnd();
```

glEvalCoord2d 関数の u と v には 0～1 までの数字が入る. それぞれ, 0 を設定すると始点, 1 を設定すると終点を描画する. また, glMap2d 関数で設定した曲線を有効化するため, glEnable 関数を使用する.

(2) 描画例

本研究で実装したベジェ曲面の描画例を図 5.39 に示す.

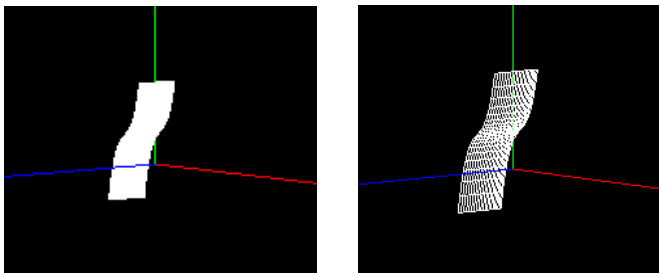


図 5.39　本研究で実装したベジェ曲面

5.6.8　NURBS曲面

NURBS 曲面は，NURBS 曲線を拡張したもので，NURBS 曲線と同じ性質をもつ自由曲面である．NURBS 曲面の概要を図 5.40 に示す.

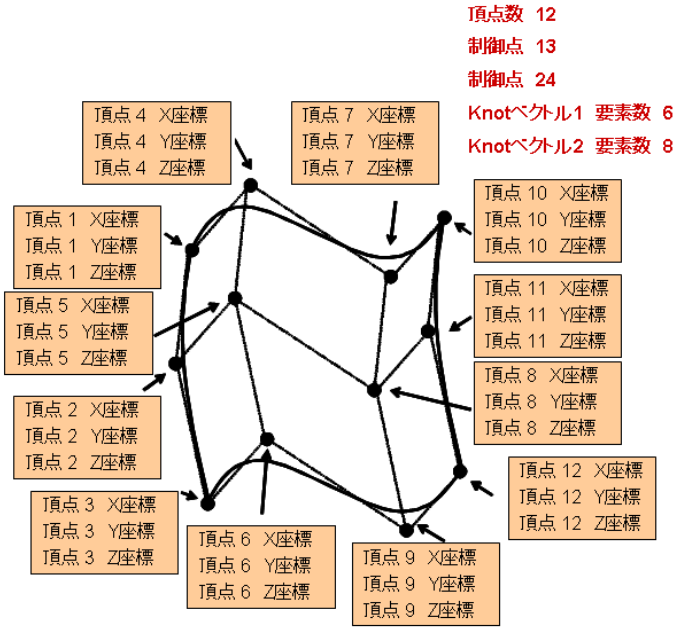


図 5.40　NURBS 曲面の概要

本研究では，頂点数，平面制御点数，奥行制御点数，平面 knot ベクトル，奥行 knot ベクトル，X 座標の配列，Y 座標の配列，Z 座標の配列とウェイトの配列の値を基に NURBS 曲面を描画する．NURBS 曲面のパラメータを表 5.30 に示す．

表 5.30　NURBS 曲面のパラメータ

パラメータ	型	説明
m_iNumber	int	頂点数
m_iOrder1	int	平面 制御点数
m_iOrder2	int	奥行 制御点数
m_alKnot1	ArrayList	平面 knot ベクトル
m_alKnot2	ArrayList	奥行 knot ベクトル
m_alX	ArrayList	X 座標の配列
m_alY	ArrayList	Y 座標の配列
m_alZ	ArrayList	Z 座標の配列
m_alWeight	ArrayList	ウェイトの配列
m_NurbsSurfaceTopology	NurbsSurfaceTopology	NURBS 曲線の位相情報

(1) 実装方法

NURBS 曲面の実装方法は，gluNurbsSurface 関数を用いて描画する．gluNurbsSurface 関数の定義を次に示す．

```
gluNurbsCurve( GLUnurbs* nobj, GLint uknots_count, GLfloat* uknot, GLint vknot_count,
               GLfloat vknot,GLint u_stride,GLint v_stride,GLfloat ctlarray,
               GLint uorder,GLint vorder,Glenum type);
```

gluNurbsSurface 関数の引数の説明を表 5.31 に示す．

表 5.31　gluNurbsSurface 関数の引数

引数	型	説明
nobj	GLUnurbsObj	NURBS オブジェクト
uknot_count	GLint	平面方向の節点数
uknot	GLfloat	平面方向の knot ベクトル
vknot_count	GLint	奥行き方向の節点数
vknot	GLfloat	奥行き方向の knot ベクトル
u_stride	GLint	平面方向の曲線の始点と終点に用いる要素数
v_stride	GLint	奥行き方向の曲線の始点と終点に用いる要素数
ctlarray	GLfloat	曲面の制御点の配列
uorder	GLint	平面方向の NURBS 曲面の階数
vorder	GLint	奥行き方向の NURBS 曲面の階数
type	GLenum	曲面の形式

NURBS 曲面の実装例を次に示す.

```
// u 方向のサンプリング範囲の設定
gluNurbsProperty(theNurb, GLU_U_STEP,0.1);
// v 方向のサンプリングの設定
gluNurbsProperty(theNurb, GLU_V_STEP,0.1);

// Nurbs オブジェクトを設定し描画
gluBeginSurface(theNurb);
// Nurbs オブジェクトの設定
gluNurbsSurface(theNurb,m_alKnot1->Count,fKnot1,m_alKnot2->Count,
                fKnot2,4,4*(m_iOrder1),fCtlPoint,m_iOrder1,m_iOrder2,GL_MAP2_VERTEX_4);
// Nurbs オブジェクトの描画の終了
gluEndSurface(theNurb);
```

まず，gluNurbsProperty 関数を使用して，平面方向，奥行方向のサンプリング範囲の設定を行う．property に GLU_U_STEP と GLU_V_STEP を設定し，value は 0.1 を設定します．value は，Nurbs 曲線の描画に使用する多角形の線分や辺の最大の長さを定義する．そのため，value が小さいほど滑らかな図形の描画が可能である．

次に，gluBeginSurface 関数，gluNurbsSurface 関数，gluEndSurface 関数を使用して NURBS 曲面を描画する．gluBeginSurface 関数と gluEndSurface 関数は，NURBS 曲面の描画の開始と終了を宣言するもので，引数の NURBS オブジェクトを指定する．そして，gluNurbsSurface 関数の引数に NURBS 曲面の形状を決定する情報を指定することで NURBS 曲線を描画する．

(2) 描画例

本研究で実装した NURBS 曲面の描画例を図 5.41 に示す.

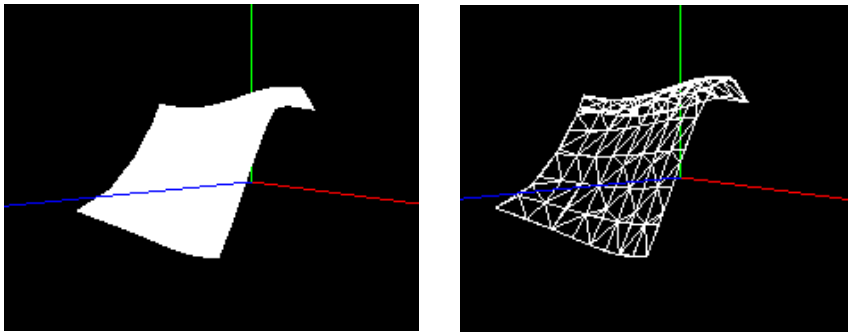


図 5.41　本研究で実装した NURBS 曲面

5.7　使用すべき API

本節では，本章で説明した点，曲線，面要素を描画するために使用した API について整理し，モデルの描画に使用すべき API について纏める．モデルの描画に使用すべき API を表 5.32 に示す.

表 5.32　モデルの描画に使用すべき API

API	用途
glEnable	設定内容の有効化
glBegin	描画処理の区間の開始
glEnd	描画処理の区間の終了
gluBeginCurve	NURBS 曲線の描画処理の開始
gluEndCurve	NURBS 曲線の描画処理の終了
gluBeginSurface	NURBS 曲面の描画処理の開始
gluEndSurface	NURBS 曲面の描画処理の終了
glVertex3d	点の描画
glMap1d	曲線の情報の設定
glEvalCoord1d	glMap1d で設定された曲線の描画
glMap2d	曲面の情報の設定
glEvalCoord2d	glMap2d で設定した曲面の描画
gluNewNurbsRenderer	NURBS オブジェクトの生成
gluNurbsProperty	NURBS 曲面の情報の設定
gluNurbsCurve	NURBS 曲線の描画
gluNurbsSurface	NURBS 曲面の描画
gluCylinder	円錐（円柱）の描画
gluSphere	球の描画
glutSolidTorus	トーラスの描画（ソリッドモデル）
glutWireTorus	トーラスの描画（ワイヤーフレームモデル）

5.8　OpenGLで描画可能なモデル

本節では，本章で説明した点，曲線，面要素の描画方法を基に OpenGL のネイティブな API で描画可能なモデルについて整理する．モデル描画用のネイティブな API の有無を表 5.33 に示す．

表 5.33　モデル描画用のネイティブな API の有無

モデル	API の有無	モデル	API の有無
点マーカ	×	直方体	×
線分	○	円錐	△
折線	○	円柱	△
円	×	球	○
円弧	×	トーラス	○
楕円	×	ウェッジ	○
楕円弧	×	ベジェ曲面	○
ベジェ曲線	○	NURBS 曲面	○
NURBS 曲線	○		
クロソイド	×		

「○」は，OpenGL のネイティブな API で描画可能であることを表し，「×」は，ネイティブな API で描画できないものを表す．「△」は，ネイディブな API で描画可能であるが，本研究のモデルの内容と異なるため，描画時に工夫する必要がある．

5.9 おわりに

本章では, OpenGL を利用した 3 次元モデルの表現方法について調査した. ここでは, 点, 曲線，面要素に分け，サンプルプログラムを交えてそれぞれの幾何要素の表現方法について纏めた．また，本章の調査結果から今回，指定した幾何要素については，すべて描画することができたが，押出面や掃引面などの複雑な幾何要素を描画するための関数が OpenGL には用意されていないため，図形近似を行って描画する必要があることがわかった．

6　OpenGLによるCADの機能の実装方法

6.1　はじめに

本章では，前章で説明した 3 次元モデルの作成以外の 3 次元 CAD で必要となる機能について，OpenGL での実装方法について調査する．まず，作成したモデルを確認するため，描画空間の設定と視点の切り替え方法について調査する．次に，モデルの操作として，平行移動，回転，尺度変更などの機能の実装方法について調査する．最後に，文字や寸法線などの注釈の描画方法について調査する．

6.2　描画空間と視点に関する機能

本節では，CAD エンジンを必要となる描画空間の設定と視点の切り替えについて，OpenGL での実装方法について説明する．

6.2.1　描画空間の設定

OpenGL には，描画空間の設定として，隠面処理とビューポートを設定するための関数が用意されている．

まず，隠面処理とは，描画領域において，最も視点に近い位置にあるモデルを描画し，それによって隠されるモデルを描画しないようにする処理である．OpenGL では，glEnable 関数を利用することで陰面処理を実装することができる．glEnable 関数について第 5.5.7 項で説明済みである．陰面処理では，cap に GL_DEPTH_TEST を指定することで，Z バッファ法による隠面処理を行うことができる．Z バッファ法とは，奥行き情報のみを記憶する Z バッファという領域を保持し，モデルを描画する領域における各画素の色情報を設定する際，同じ座標の奥行き情報を比較し，最も視点に近い位置にあるモデルの色情報を設定する方法のことである．

次に，ビューポートとは，描画領域におけるモデルを表示する領域のことである．ビューポートを設定することで，モデルを表示する位置や大きさを設定できる．OpenGL では，glViewport 関数を利用することでビューポートを実装することができる．glViewport 関数の定義を次に示す．

```
void glViewport( GLint x, GLint y, GLsizei width, Glsizei height)
```

glViewport 関数の引数の説明を表 6.1 に示す．

表 6.1　glViewport 関数の引数

引数	型	説明
x	GLint	描画領域の左下の X 座標
y	GLint	描画領域の左下の Y 座標
width	GLint	描画領域の幅
height	GLint	描画領域の高さ

(1) 実装例

glEnable 関数を用いた陰面処理の実装例を次に示す.

```
// 描画空間の初期化
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
```

ここでは，glEnable 関数の引数に Z バッファ法による隠面消去を行うため，GL_DEPTH_TEST を指定する．また，glDepthFunc 関数により，Z バッファの値の比較方法を設定する．glDepthFunc 関数の引数に GL_LEQUAL を指定することで，比較対象の値が Z バッファに保持している値以下である場合に Z バッファの値を更新する.

glViewport 関数を用いたビューポートの設定の実装例を次に示す.

```
// ビューポートの設定
glViewport(0, 0, DManager->m_VManager->m_dWidth, DManager->m_VManager->m_dHeight);
```

ビューポートの設定では，glViewport 関数に対して描画領域と同じ座標を指定することで，ビューポートと描画領域と同じ大きさに設定する.

6.2.2　投影法の切り替え

OpenGL には，平行投影と透視投影の 2 種類の投影法が実装されている．平行投影を設定するには glOrtho 関数を使用し，透視投影を設定するには gluPerspective 関数を使用する．glOrtho 関数の定義を以下に示す.

```
void glOrtho(GLdouble xleft,  GLdouble xright,  GLdouble ybottom,  GLdouble ytop,
             GLdouble znear,  GLdouble zfar)
```

glOrtho 関数の引数の説明を表 6.2 に示す.

表 6.2　glOrtho 関数の引数

引数	型	説明
left	GLdouble	描画領域の左側の座標
right	GLdouble	描画領域の右側の座標
bottom	GLdouble	描画領域の下側の座標
top	GLdouble	描画領域の上側の座標
near	GLdouble	視点から投影面までの距離
far	GLdouble	視点から描画領域の奥行きの距離

透視投影を設定するための gluPerspective 関数の定義を以下に示す.

```
void gluPerspective(GLdouble fovy,  GLdouble aspect,  GLdouble znear,  GLdouble zfar)
```

gluPerspective 関数の引数の説明を表 6.3 に示す.

表 6.3　glPerspective 関数の引数

引数	型	説明
fovy	GLdouble	幅
aspect	GLdouble	縦横比
znear	GLdouble	視点から投影面までの距離
zfar	GLdouble	視点から描画領域の奥行きの距離

(1) 実装例

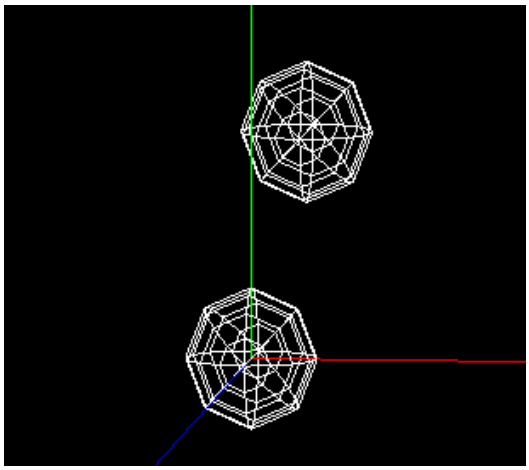
投影法の実装例を次に示す.

```
if (m_ProjectionType == ENVIRONMENT::PROJECTION_TYPE::ORTHO) {
    // 投影法を平行投影に設定
    glOrtho( m_dLeft, m_dRight, m_dBottom, m_dTop, m_dZNear, m_dZFar);
} else {
    // 投影法を透視投影に設定
    gluPerspective(40, m_dWidth / m_dHeight, m_dZNear, m_dZFar);
}
```

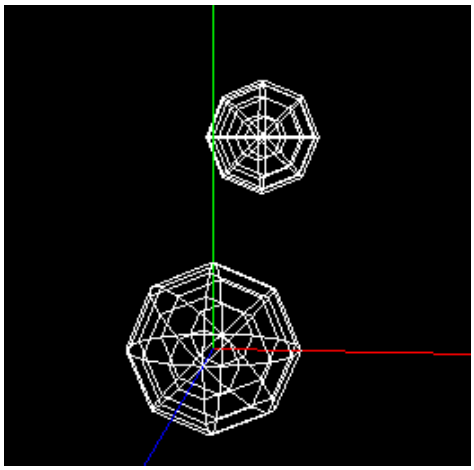
本研究では，m_ProjectionType の値によって，投影法を設定する．m_ProjectionType の値が ENVIRONMENT::PROJECTION_TYPE::ORTHO である場合，glOrtho 関数によって平行投影に設定し，ENVIRONMENT::PROJECTION_TYPE::FRUSTUM である場合，gluPerspectve 関数によって透視投影に設定する.

(2) 実行結果

OpenGL で実装した平行投影と透視投影の描画例を図 6.1 に示す.



平行投影



透視投影

図 6.1　投影法の描画例

6.2.3　視点の切り替え

視点の切り替えは，平行移動と回転といった幾何変換処理を行うことにより実現する．視点の平行移動には glTranslated 関数を使用する．glTranslated 関数の定義を次に示す．

```
void glTranslated(GLdouble x, GLdouble y, GLdouble z)
```

glTranslated 関数の引数の説明を表 6.4 に示す．

表 6.4　glTranslated 関数の引数

引数	型	説明
X	GLdouble	X 軸方向の移動量
Y	GLdouble	Y 軸方向の移動量
Z	GLdouble	Z 軸方向の移動量

視点を回転させるには，glRotated メソッドを使用する．glRotated メソッドの定義を次に示す．

```
void glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z)
```

glRotated 関数の引数の説明を表 6.5 に示す．

表 6.5　glRotated 関数のパラメータ

引数	型	説明
Angle	GLdouble	回転角度
X	GLdouble	ベクトルの X 座標
Y	GLdouble	ベクトルの Y 座標
Z	GLdouble	ベクトルの Z 座標

(1) 実装例

視点の切り替えの実装例を次に示す.

```
// 注視点からの距離を設定
glTranslated(0.0, 0.0, -m_dDistance);
// 注視点からの高さを設定
glRotated(-m_dElevation, 1.0, 0.0, 0.0);
// 注視点からの方向を設定
glRotated(-m_dAzimuth, 0.0, 1.0, 0.0);
```

本研究では，変数 m_dDistance，　m_dElevation と m_dAzimuth の値によって，視点の位置と方向を設定する．また，投影法を平行投影に設定した場合には，視点からの距離によってグローバル座標系のスケールを変換することで，視点から見たモデルの大きさを設定する．視点の位置，方向，距離の関係を図 6.2 に示す.

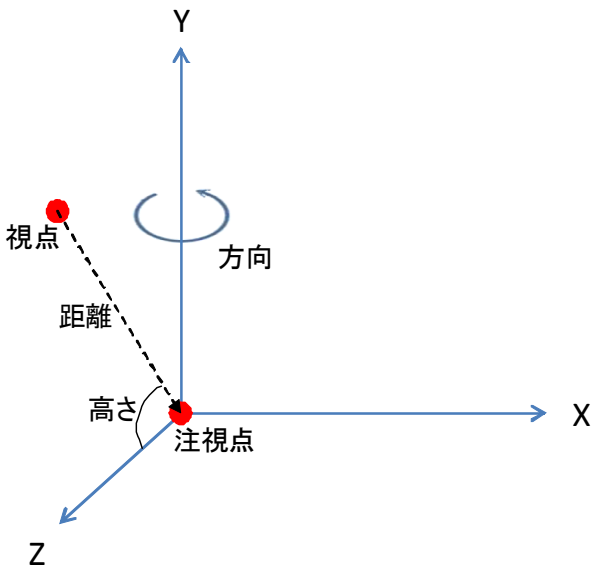


図 6.2　視点の位置，方法，距離の関係

6.3　モデルの操作

本節では，モデルの操作として，モデルの色，線種，線幅の設定について説明する．

6.3.1　色の設定

色の設定では，各モデルを描画する場合の色を設定する．本研究では，各モデルが保持している色コードを基にモデルの色を決定する．本研究で設定できる色を表 6.6 に示す．

表 6.6　設定できる色

定数名	色コード	色名
BLACK	1	黒
RED	2	赤
GREEN	3	緑
BLUE	4	青
YELLOW	5	黄
MAGENTA	6	マジェンタ
CYAN	7	シアン
WHITE	8	白
DEEPPINK	9	牡丹
BROWN	10	茶
ORANGE	11	橙
LIGHTGREEN	12	薄緑
LIGHTBLUE	13	薄青
LAVENDER	14	青紫
LIGHTGRAY	15	明灰
DARKGRA	16	暗灰

OpenGL では，glColor3d や glColor4d などの glColor*関数を使用することで図形の色を設定できる．本研究では，glColor3d 関数を使用して図形の色を設定する．glColor3d 関数の定義を次に示す．

```
void glColor3d(GLdouble r, GLdouble g, GLdouble b)
```

glColor3d 関数の引数の説明を表 6.7 に示す．

表 6.7　glColor3d 関数の引数

引数	型	説明
r	GLdouble	赤色成分の値
g	GLdouble	緑色成分の値
b	GLdouble	青色成分の値

光の 3 原色である RGB の値を 0 から 1 の範囲で指定する．「WHITE」の色の実装例を次に示す．

```
// 白
glColor3d(1.0, 1.0, 1.0);
```

本研究で設定できる色と glColor3d 関数の引数の値の関係を表 6.8 に示す．

表 6.8　色と glColor3d 関数の引数の値との関係

定数名	色コード	(r, g, b)
BLACK	1	(0.0,0.0,0.0)
RED	2	(1.0,0.0,0.0)
GREEN	3	(0.0,1.0,0.0)
BLUE	4	(0.0,0.0,1.0)
YELLOW	5	(1.0,1.0,0.0)
MAGENTA	6	(1.0,0.0,1.0)
CYAN	7	(0.0,1.0,1.0)
WHITE	8	(1.0,1.0,1.0)
DEEPPINK	9	(192/255,0.0,128/255)
BROWN	10	(192/255,128/255,64/255)
ORANGE	11	(1.0,128/255,0.0)
LIGHTGREEN	12	(128/255,192/255,128/255)
LIGHTBLUE	13	(0.0,128/255,1.0)
LAVENDER	14	(128/255,64/255,1.0)
LIGHTGRAY	15	(192/255,192/255,192/255)
DARKGRA	16	(128/255,128/255,128/255)

6.3.2　線種の設定

線種の設定では，各モデルを描画する場合の線の形状を設定する．本研究では，各モデルが保持している線種コードを基にモデルの線種を決定する．本研究で設定できる線種を表 6.9 に示す．

表 6.9　設定できる線種

定数名	色コード	色名
COTINUOUS	1	実線
DASHED	2	破線
DASHEDSPACED	3	跳び破線

OpenGL では，glLineStipple 関数を使用することで図形の線種を設定できる．glLineStipple 関数の定義を次に示す．

```
void glLineStipple(GLint factor, GLushort pattern)
```

glLineStipple 関数の引数の説明を表 6.10 に示す．

表 6.10　glLineStipple 関数の引数

引数	型	説明
factor	GLint	破線の間隔
pattern	GLushort	破線のパターン

ただし，glLineStipple 関数では，描画するピクセルと描画しないピクセルを 2 ビット表現する．また，引数として指定できるビット数に限りがあるため，glLineStipple 関数では,複雑な線種を表現することが困難である．そのため，本研究では，実線，破線と跳び破線のみとした．線種の実装例を次に示す．

```
// 破線
glLineStipple(1 , 0x7FF8);
```

glLineStipple 関数を使用して線種を設定するには，まず，描画するピクセルと描画しないピクセルを 2 ビット表現する．次に，2 ビット表現したデータを 16 ビットに変換し，glLineStipple 関数の引数である pattern に指定する．破線の設定方法を図 6.3 に示す．

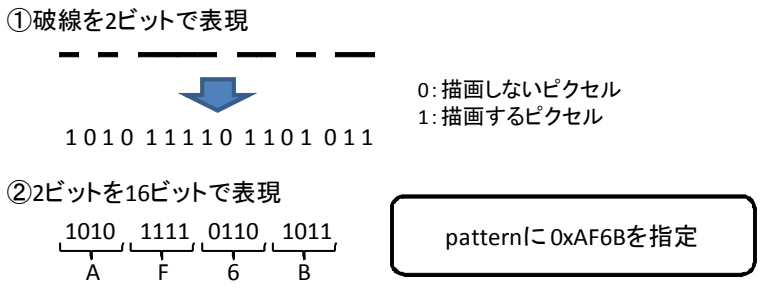


図 6.3　破線の実装方法

6.3.3　線幅の設定

線幅の設定では，各モデルを描画する場合の線の太さを設定する．本研究では，各モデルが保持している線幅コードを基にモデルの線幅を決定する．本研究で設定できる線幅を表 6.11 に示す.

表 6.11　本研究で設定できる線幅

定数名	線幅コード	線幅
W013	1	0.13
W018	2	0.18
W025	3	0.25
W035	4	0.35
W050	5	0.5
W070	6	0.7
W100	7	1.0
W140	8	1.4
W200	9	2.0

OpenGL では，glLineWidth 関数を使用することで図形の線幅を設定できる．glLineWidth 関数の定義を次に示す.

```
void glLineWidth(GLfloat width)
```

glLineWidth 関数の引数の説明を表 6.12 に示す.

表 6.12　glLineWidth 関数の引数

引数	型	説明
width	GLfloat	線幅の値

線幅の実装例を次に示す.

```
// 線幅 0.13
glLineWidth(0.13f);
```

glLineWidth メソッドを使用して線幅を設定するには,線の太さをピクセル単位で指定する．例えば，線の太さを 0.13 ピクセルに設定する場合は，width に 0.13f を指定する.

6.4　注釈の挿入

本節では，注釈の挿入として，文字要素，寸法線（直線，角度，弧長，半径，直径），引出線，バルーンの実装方法について解説する．

6.4.1　文字の描画

すべての注釈には，必ず文字を描画する必要がある．そこで，本項では，OpenGL での文字の描画方法について説明する．

OpenGL では，２バイト文字を描画する関数が用意されておらず，本研究では，マイクロソフト社が Windows 環境上で使用するためのライブラリである WGL を使用する．WGL 関数を利用して文字を描画するメソッドには，wglUseFontOutlines メソッドがある．wglUseFontOutlines メソッドの定義を次に示す．

```
void wglUseFontOutlines(IntPtr hdc, int Code, int List, uint baseID, float gap, float push,
                        OutlineFontFormat FontModel, GlyphmetricicsFloat gmf);
```

wglUseFontOutlines メソッドの引数の説明を表 6.13 に示す．

表 6.13　wglUseFontOutlines メソッドの引数

引数	型	説明
hdc	IntPtr	デバイスコンテキスト
Code	int	文字コード
List	int	リストの総数
baseID	uint	指標番号
gap	float	3 次元フォントとのずれ
push	float	Z 方向の幅
FontModel	OutlineFontFormat	描画モデル
gmf	GlyphmetricicsFloat	文字寸法の配列

wglUseFontOutlines メソッドを利用した文字の描画の実装例を次に示す．

```
// 1 文字ずつ文字を描画
for(int i = 0;i < barray->Length;i++){
    // 文字コードを取得
    unsigned char txt = static_cast<unsigned char>(barray[i]);
    long code = static_cast<long>(txt);
    // 文字のバイト数の判定
    if (IsDBCSLeadByte(barray[i]) ){
        // 2 バイト文字の場合
        // 文字コードの変換
        txt = static_cast<unsigned char>(barray[i + 1]);
        code = code * 256 + static_cast<long>(txt);
        // 変数 i の加算
```

```
            i++;
            // フォントサイズの設定
            iFontSize = 1;
        }else{
            // 1バイト文字の場合
            // フォントサイズの設定
            iFontSize = 2;
        }
        // 文字の描画
        wglUseFontOutlines(hDC,code,1,i,0.0f,0.1f,WGL_FONT_POLYGONS, &agmf);
        // i番目のオブジェクトの呼出し
        glCallList(i);
    }
```

OpenGL では，文字列を 1 文字ずつ描画する．まず，wglUseFontOutlines メソッドに文字コードを指定するため，文字列から 1 文字ずつ文字コードを取得する．その際，２バイト文字と 1 バイト文字で文字コードの取得方法が異なる．次に，wglUseFontOutlines メソッドを用いて取得した文字コードの文字を描画する．この処理を文字列分繰り返すことで文字を描画する．

6.4.2　矢印の描画

文字要素以外の注釈要素は，矢印を保持している．そのため，矢印の描画方法について解説する. 本研究で扱う矢印の形状は, ISO10303 および SXF で定義されているものとした. 本研究で扱う矢印を図 6.4 に示す.

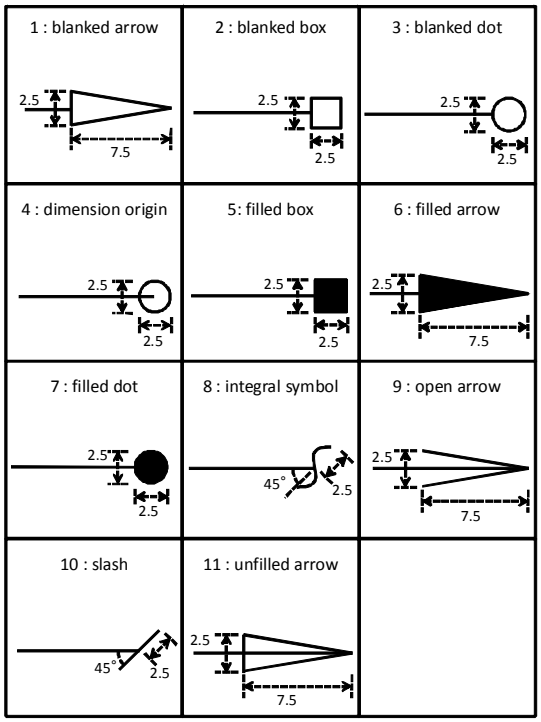


図 6.4　本研究で扱う矢印

OpenGL には, 点を描画するための関数が用意されているため, glVertex3d 関数や glMap1d 関数などの描画関数を使用して点マーカを描画する．次に各矢印の描画方法について解説する．

- blanked arrow

blanked arrow は, glVertex3d 関数を使用して 3 本の直で三角形を描画することで実現する．blank arrow の実装例を次に示す．

```
// 描画の開始
glBegin(GL_LINE_LOOP);
    // 描画の実行
    glVertex3d(0,0,0);
    glVertex3d(iDirection*7.5,-1.25,0);
    glVertex3d(iDirection*7.5, 1.25,0);
// 描画の終了
glEnd();
```

glBegin 関数は，頂点データの始まりを宣言する関数であり，glEnd は，頂点データの終了を宣言する関数である．blank arrow は，三角形（閉じた多角形）であるため，glBegin 関数の引数に GL_LINE_LOOP を指定する．

- blanked box

blanked box は，glVertex3d 関数を使用して 4 本の直線で四角形を描画することで実現する．blank box の実装例を次に示す．

```
// 描画の開始
glBegin(GL_LINE_LOOP);
    // 描画の実行
    glVertex3d(-1.25,-1.25,0);
    glVertex3d(-1.25, 1.25,0);
    glVertex3d( 1.25, 1.25,0);
    glVertex3d( 1.25,-1.25,0);
// 描画の終了
glEnd();
```

glBegin 関数の引数に blanked arrow と同様で GL_LINE_LOOP を指定する．

- blanked dot

blanked dot は，glVertex3d 関数を使用して円を折線に近似して描画することで実現する．blank dot の実装例を次に示す．

```
// 描画の開始
glBegin(GL_LINE_LOOP);
    // 円を描画するための繰り返し
    for(int i=0;i<iStep;i++){
        // X 座標値の算出
        dX = 1.25 *cos(2.0*PAI*(float)i/(float)iStep) + 0;
        // Y 座標値の算出
        dY = 1.25 *sin(2.0*PAI*(float)i/(float)iStep) + 0;
        // Z 座標値の算出
        dZ = 0;
        // 描画の実行
        glVertex3d(dX,dY,dZ);
    }
// 描画の終了
glEnd();
```

- dimension origin

dimension origin は，blank dot と同様に glVertex3d 関数を使用して円を折線に近似して描画することで実現する．dimension origin の実装例を次に示す．

```
// 描画の開始
glBegin(GL_LINE_LOOP);
    // 円を描画するための繰り返し
    for(int i=0;i<iStep;i++){
        // X 座標値の算出
        dX = 1.25 *cos(2.0*PAI*(float)i/(float)iStep) + 0;
        // Y 座標値の算出
        dY = 1.25 *sin(2.0*PAI*(float)i/(float)iStep) + 0;
        // Z 座標値の算出
        dZ = 0;
        // 描画の実行
        glVertex3d(dX,dY,dZ);
    }
// 描画の終了
glEnd();
```

- filled box

filled box は，blanked box と同様に glVertex3d 関数を使用して 4 本の直線で四角形を描画することで実現する．ただし，filled box は，塗りつぶされた四角形であるため，glBegin 関数の引数には，GL_POLYGON を使用する．filled box の実装例を次に示す．

```
// 描画の開始
glBegin(GL_POLYGON);
    // 描画の実行
    glVertex3d(-1.25,-1.25,0);
    glVertex3d(-1.25, 1.25,0);
    glVertex3d( 1.25, 1.25,0);
    glVertex3d( 1.25,-1.25,0);
// 描画の終了
```

```
    glEnd();
```

- filled arrow

filled arrow は，blanked arrow と同様に glVertex3d 関数を使用して 3 本の直線で三角形を描画することで実現する．ただし，filled arrow は，filled box と同様，塗りつぶされた図形であるため，glBegin 関数の引数には，GL_ POLYGON を使用する．filled arrow の実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_POLYGON);
        // 描画の実行
        glVertex3d(0,0,0);
        glVertex3d(iDirection*7.5, 1.25,0);
        glVertex3d(iDirection*7.5,-1.25,0);
    // 描画の終了
    glEnd();
```

- filled dot

filled dot は，blank dot と同様に glVertex3d 関数を使用して円を折線に近似して描画することで実現する．ただし，filled dot は，filled box と同様，塗りつぶされた図形であるため，glBegin 関数の引数には，GL_ POLYGON を使用する．filled arrow の実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_POLYGON);
        // 円を描画するための繰り返し
        for(int i=0;i<iStep;i++){
            // X 座標値の算出
            dX = 1.25 *cos(2.0*PAI*(float)i/(float)iStep) + 0;
            // Y 座標値の算出
            dY = 1.25 *sin(2.0*PAI*(float)i/(float)iStep) + 0;
            // Z 座標値の算出
            dZ = 0;
            // 描画の実行
            glVertex3d(dX,dY,dZ);
        }
    // 描画の終了
    glEnd();
```

- integral symbol

integral symbol は，glMap1d 関数を使用してベジェ曲線を描画することで実現する．integral symbol を描画するためのベジェ曲線の制御点を図 6.5 に示す．

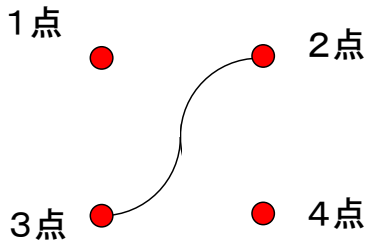


図 6.5　ベジェ曲線の制御点

integral symbol の実装例を次に示す.

```
double dIntegral[13];
dIntegral[0] = iDirection*(-2.5/sqrt(2.0));
dIntegral[1] = (2.5/sqrt(2.0));
dIntegral[2] = 0;
dIntegral[3] = iDirection*(2.5/sqrt(2.0));
dIntegral[4] = (2.5/sqrt(2.0));
dIntegral[5] = 0;
dIntegral[6] = iDirection*(-2.5/sqrt(2.0));
dIntegral[7] = (-2.5/sqrt(2.0));
dIntegral[8] = 0;
dIntegral[9] = iDirection*(2.5/sqrt(2.0));
dIntegral[10] =(-2.5/sqrt(2.0));
dIntegral[12] = 0;
// エバリュエータの定義
glMap1d(GL_MAP1_VERTEX_3, 0, 1, 3, 4, dIntegral);
glEnable(GL_MAP1_VERTEX_3);

// 描画の開始
glBegin(GL_LINE_STRIP);
    // 矢印描画のための繰り返し
    for(int t = 0; t <= iStep; ++t){
        // 描画の実行
        glEvalCoord1d(t/(iStep*1.0));
    }
// 描画の終了
glEnd();
```

- open arrow

open arrow は, glVertex3d を利用して 2 本の直線を描画することで実現する. ただし, open arrow は，開いた図形であるため，glBegin 関数の引数には，GL_LINE_STRIP を指定する. open arrow の実装例を次に示す.

```
// 描画の開始
glBegin(GL_LINE_STRIP);
    // 描画の実行
    glVertex3d(iDirection*7.5,-1.25,0);
```

```
        glVertex3d(0,0,0);
        glVertex3d(iDirection*7.5, 1.25,0);
    // 描画の終了
    glEnd();
```

- slash

slash は，glVertex3d を利用して直線を描画することで実現する．slash の実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINE_STRIP);
        // 描画の実行
        glVertex3d(iDirection*(-2.5/sqrt(2.0)),(2.5/sqrt(2.0)),0);
        glVertex3d(iDirection*(2.5/sqrt(2.0)),(-2.5/sqrt(2.0)),0);
    // 描画の終了
    glEnd();
```

- unfilled arrow

unfilled arrow は，filled arrow と同様で glVertex3d を利用して 3 本の直線で三角形を描画することで実現する．ただし，unfilled arrow は，塗りつぶされた図形でないため，glBegin 関数の引数には，GL_LINE_LOOP を指定する．Unfilled arrow の実装例を次に示す．

```
    // 描画の開始
    glBegin(GL_LINE_LOOP);
        // 描画の実行
        glVertex3d(0,0,0);
        glVertex3d(iDirection*7.5,-1.25,0);
        glVertex3d(iDirection*7.5, 1.25,0);
    // 描画の終了
    glEnd();
```

6.4.3　文字要素

文字要素は，アルファベットや日本語などの文字の集まりである．文字の概要を図 6.6 に示す．

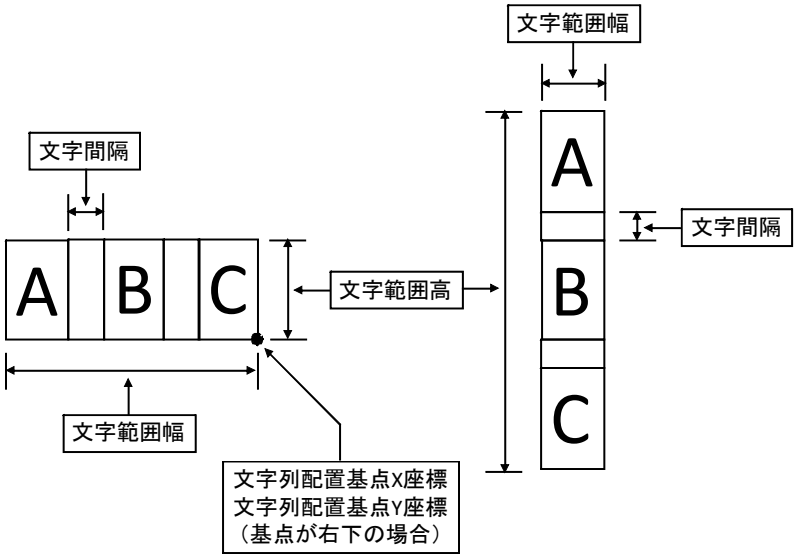


図 6.6　文字要素の概要

本研究の文字は，配置点である X 座標，Y 座標と Z 座標と文字列配置位置を指定する．また，文字の大きさを決めるために，文字範囲高，文字範囲幅と文字間隔を指定する．文字を斜体にするために，スラント角度を指定する．そして，文字方向を指定することで縦書きと横書きの切り替えができる．

(1) 実装方法

文字要素の実装方法は, 第 6.4.1 項で解説した. wglUseFontOutlines メソッドを使用する．（詳しくは第 6.4.1 項を参照）.

(2) 描画例

本研究で実装した文字要素の描画例を図 6.7 に示す.

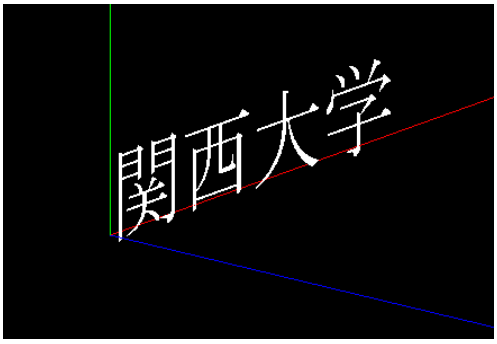


図 6.7　本研究で実装した文字要素

6.4.4　直線寸法

直線寸法は, モデルに平行な長さを表す寸法線である. 直線寸法の概要を図 6.8 に示す.

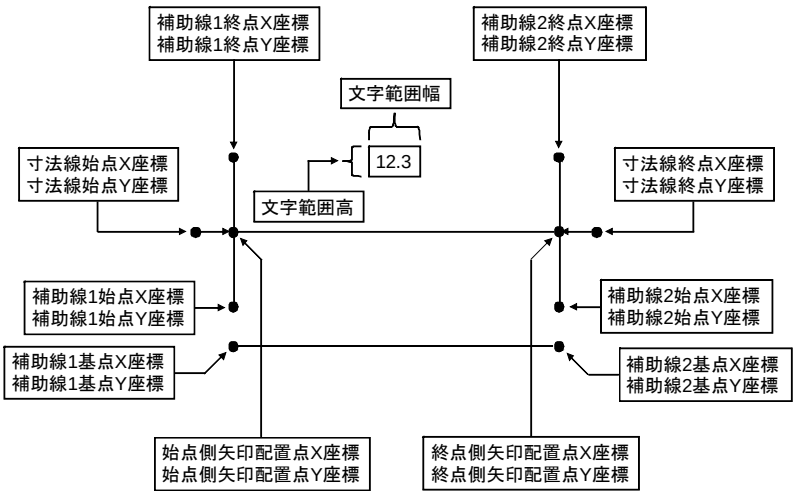


図 6.8　直線寸法の概要

直線寸法は，3 本の直線，2 つの矢印と文字によって構成される．

(1) 実装方法

直線寸法の実装方法は，直線，矢印，文字を描画することで実現する．直線寸法の実装方法を図 6.9 に示す．

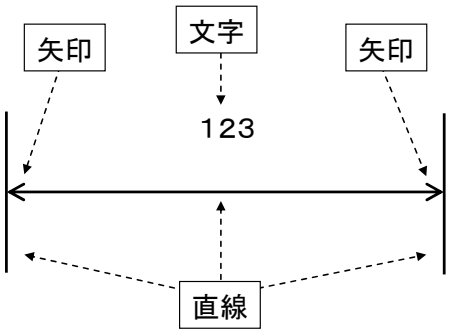


図 6.9　直線寸法の実装方法

図 6.9 に示すように，直線寸法では，3 本の直線，２つの矢印と文字を描画する．直線寸法の実装例を次に示す．ただし，文字と矢印の実装例については，第 6.4.1 項と第 6.4.2 項に記載済みであるため，ここでは，寸法の実装例のみを記載する．

```
    // 描画の開始
    glBegin(GL_LINE_STRIP);
        // 描画の実行
        glVertex3d(m_dSunX1,m_dSunY1,m_dSunZ1);
        glVertex3d(m_dSunX2,m_dSunY2,m_dSunZ2);
        // 描画の終了
```

```
    glEnd();

    // 補助線１が有りの場合
    if(m_iFlg2 == 1){
         // 描画の開始
         glBegin(GL_LINE_STRIP);
              // 描画の実行
              glVertex3d(m_dHo1X1,m_dHo1Y1,m_dHo1Z1);
              glVertex3d(m_dHo1X2,m_dHo1Y2,m_dHo1Z2);
         // 描画の終了
         glEnd();
    }

    // 補助線２が有りの場合
    if(m_iFlg3 == 1){
    // 描画の開始
         glBegin(GL_LINE_STRIP);
              // 描画の実行
              glVertex3d(m_dHo2X1,m_dHo2Y1,m_dHo2Z1);
              glVertex3d(m_dHo2X2,m_dHo2Y2,m_dHo2Z2);
         // 描画の終了
         glEnd();
    }
```

(2) 描画例

本研究で実装した直線寸法の描画例を図 6.10 に示す.

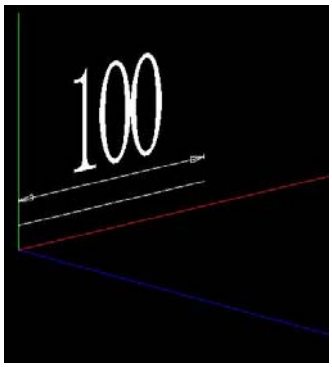


図 6.10　本研究で実装した直線寸法

6.4.5　角度寸法，弧長寸法

角度寸法と弧長寸法は，円弧または折線の円弧部分に長さを表す寸法線である．角度寸法と弧長寸法の概要を図 6.11 に示す.

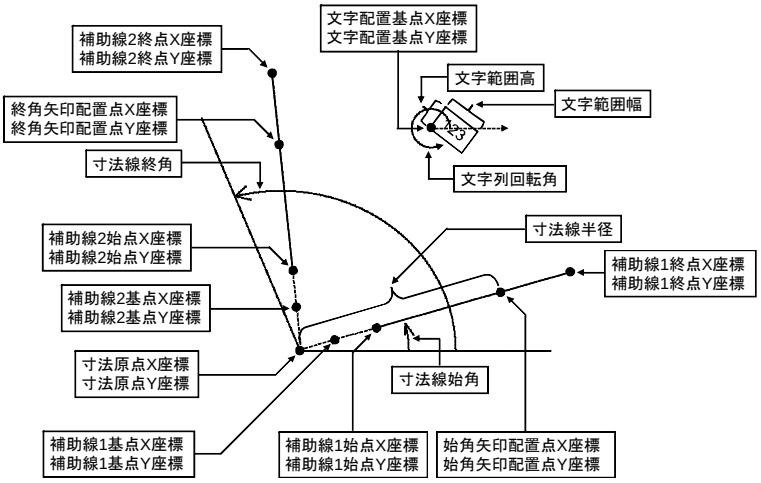


図 6.11　角度寸法と弧長寸法の概要

角度寸法と弧長寸法は，2 本の直線，円弧，2 つの矢印と文字で構成させる．

(1) 実装方法

角度寸法と弧長寸法の実装方法は，円弧，矢印，文字を描画することで実現する．角度寸法と弧長寸法の実装方法を図 6.12 に示す．

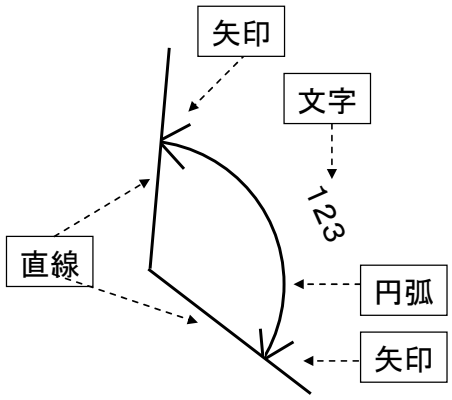


図 6.12　角度寸法と弧長寸法の実装方法

図 6.12 に示すように，角度寸法と弧長寸法では，1 つの円弧，2 本の直線，2 つの矢印と文字を描画する．角度寸法と弧長寸法の実装例を次に示す．ただし，文字と矢印の実装例については，第 6.4.1 項と第 6.4.2 項に記載済みであるため，ここでは，寸法の実装例のみを記載する．

```
    // 補助線フラグ有の場合
    if(m_iFlg2 == 1){
        // 描画の開始
```

```
        glBegin(GL_LINE_STRIP);
        // 描画の実行
        glVertex3d(m_dHo1X1,m_dHo1Y1,m_dHo1Z1);
        glVertex3d(m_dHo1X2,m_dHo1Y2,m_dHo1Z2);
        // 描画の終了
        glEnd();
    }
    // 補助線フラグ有の場合
    if(m_iFlg3 == 1){
        // 描画の開始
        glBegin(GL_LINE_STRIP);
            // 描画の開始
            glVertex3d(m_dHo2X1,m_dHo2Y1,m_dHo2Z1);
            glVertex3d(m_dHo2X2,m_dHo2Y2,m_dHo2Z2);
        // 描画の終了
        glEnd();
    }

    // 図形描画時の分割数(360*3)
    int iStep = 1080;
    // X座標値を格納する配列の生成
    System::Collections::ArrayList^ alX = gcnew System::Collections::ArrayList();
    // Y座標値を格納する配列の生成
    System::Collections::ArrayList^ alY = gcnew System::Collections::ArrayList();
    // Z座標値を格納する配列の生成
    System::Collections::ArrayList^ alZ = gcnew System::Collections::ArrayList();

    // 頂点座標の算出
    ArcToPolyline(alX,alY,alZ,0,0,0,
        m_dSunRadius,m_dSunRadius,0,m_dSunAngle0,m_dSunAngle1,iStep);

    // 描画の開始
    glBegin(GL_LINE_STRIP);
        // 円弧を描画するための繰り返し
        for(int i=0;i<alX->Count;i++){
            // 描画の実行
            glVertex3d( (double)alX[i], (double)alY[i], (double)alZ[i]);
        }
    // 描画の終了
    glEnd();
```

(2) 描画例

本研究で実装した角度寸法と弧長寸法の描画例を図 6.13 に示す.

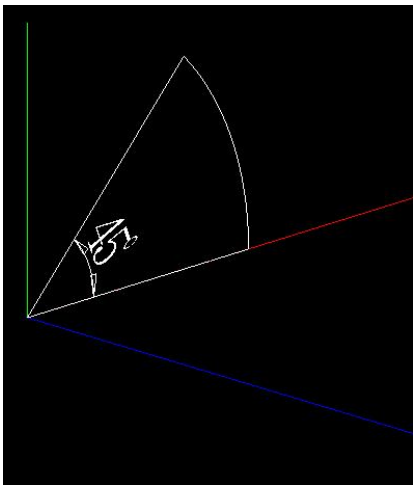


図 6.13　本研究で実装した角度寸法と弧長寸法

6.4.6　半径寸法

半径寸法は，円，円弧，楕円や楕円弧の半径の長さを表す寸法線である．半径寸法の概要を図 6.14 に示す．

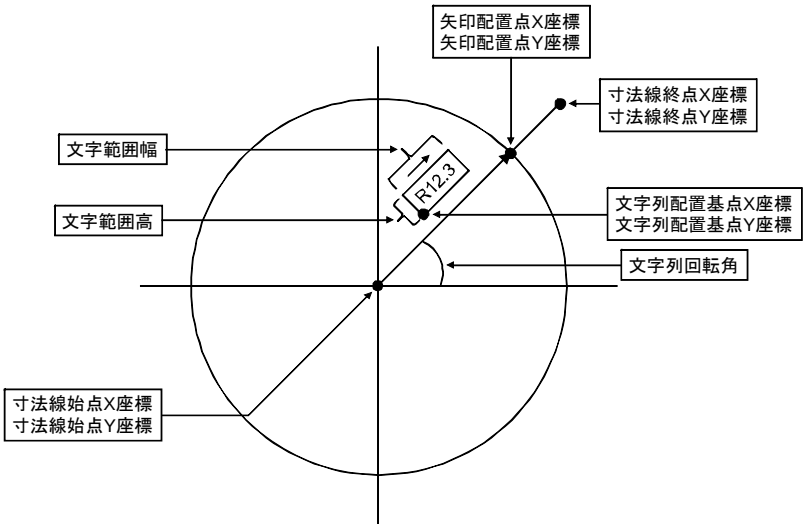


図 6.14　半径寸法の概要

半径寸法は，直線，矢印と文字によって構成される．

(1) 実装方法

半径寸法の実装方法は，直線，矢印，文字を描画することで実現する．半径寸法の実装方法を図 6.15 に示す．

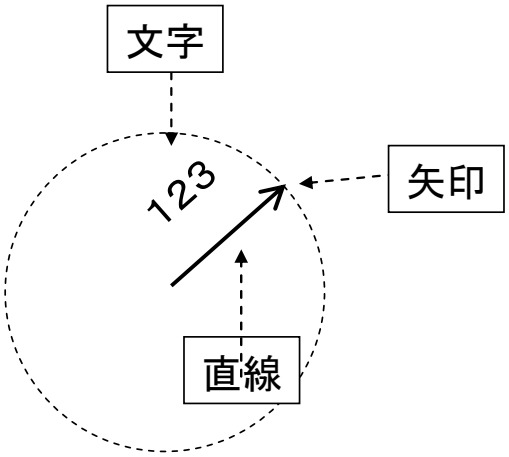


図 6.15　半径寸法の実装方法

図 6.15 に示すように，半径寸法では，１本の直線，１つの矢印と文字を描画する．半径寸法の実装例を次に示す．ただし，文字と矢印の実装例については，第 6.4.1 項と第 6.4.2 項に記載済みであるため，ここでは，寸法の実装例のみを記載する．

```
// 描画の開始
glBegin(GL_LINE_STRIP);
    // 描画の実行
    glVertex3d(m_dSunX1,m_dSunY1,m_dSunZ1);
    glVertex3d(m_dSunX2,m_dSunY2,m_dSunZ2);
// 描画の終了
glEnd();
```

(2) 描画例

本研究で実装した半径寸法の描画例を図 6.16 に示す．

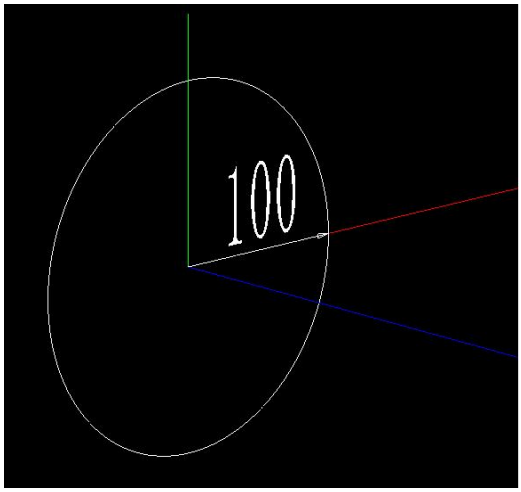


図 6.16　本研究で実装した半径寸法

6.4.7　直径寸法

直径寸法は，円，円弧，楕円や楕円弧の直径の長さを表す寸法線である．直径寸法の概要を図 6.17 に示す.

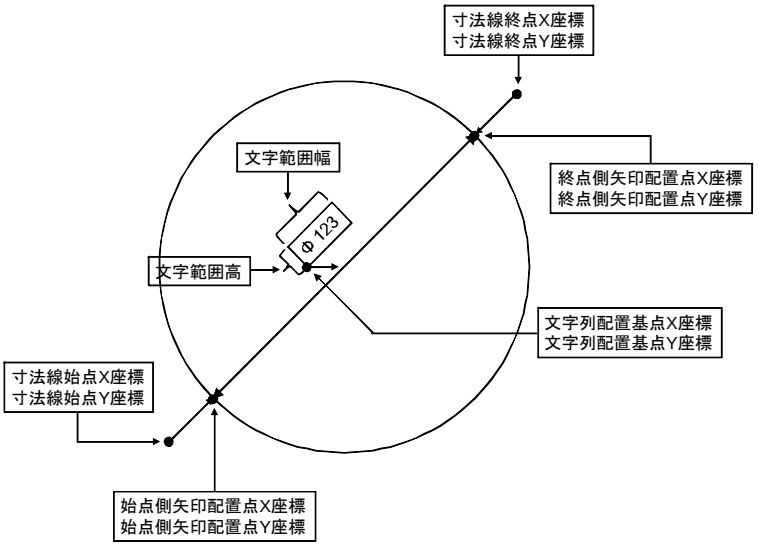


図 6.17　直径寸法の概要

直径寸法は，直線，2 つの矢印と文字によって構成される.

(1) 実装方法

直径寸法の実装方法は，直線，矢印，文字を描画することで実現する．直径寸法の実装方法を図 6.18 に示す.

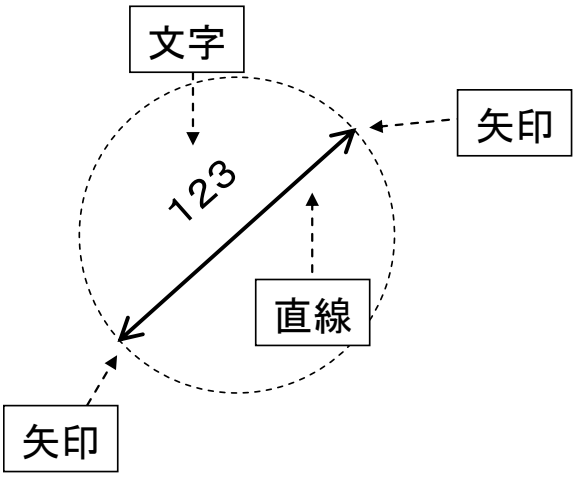


図 6.18　直径寸法の実装方法

図 6.18 に示すように，直径寸法では，１本の直線，２つの矢印と文字を描画する．直径寸法の実装例を次に示す．ただし，文字と矢印の実装例については，第 6.4.1 項と第 6.4.2 項に記載済みであるため，ここでは，寸法の実装例のみを記載する．

```
    // 描画の開始
    glBegin(GL_LINE_STRIP);
        // 描画の実行
        glVertex3d(m_dSunX1,m_dSunY1,m_dSunZ1);
        glVertex3d(m_dSunX2,m_dSunY2,m_dSunZ2);
    // 描画の終了
    glEnd();
```

(2) 描画例

本研究で実装した直径寸法の描画例を図 6.19 に示す.

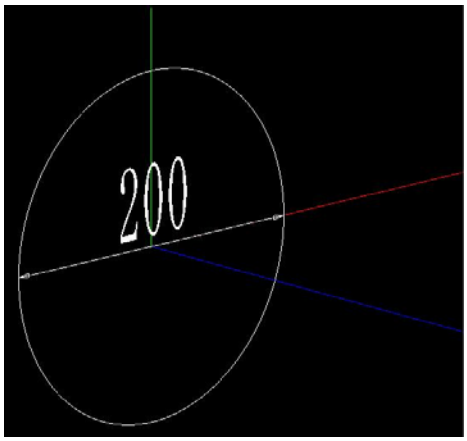


図 6.19 本研究で実装した直径寸法の描画例

6.4.8 引出線

引出線は,図面に説明をいれる際に使用する注釈である.引出線の概要を図 6.20 に示す.

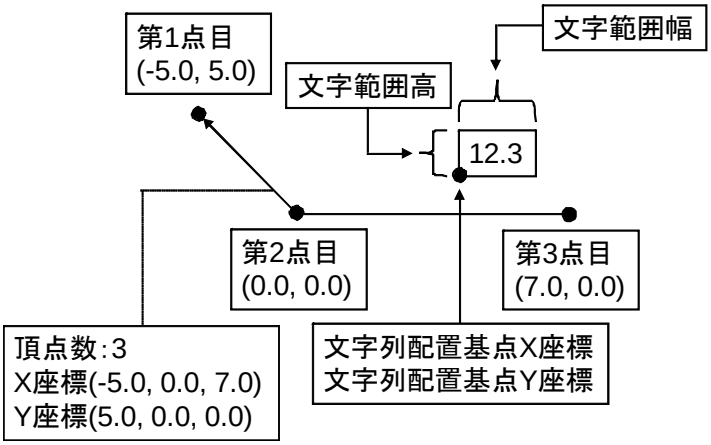


図 6.20　引出線の概要

引出線は，折線，矢印と文字によって構成される．

(1) 実装方法

引出線の実装方法は，折線，１つの矢印と文字を描画することで実現する．引出線の実装方法を図 6.21 に示す．

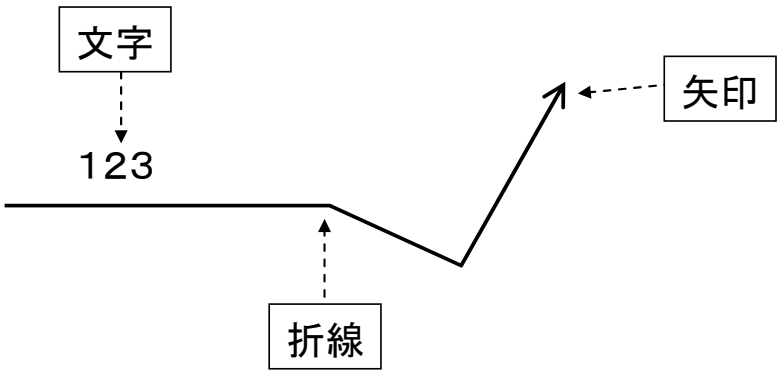


図 6.21　引出線の実装方法

図 6.21 に示すように，引出線では，１本の折線，１つの矢印と文字を描画する．引出線の実装例を次に示す．ただし，文字と矢印の実装例については，第 6.4.1 項と第 6.4.2 項に記載済みであるため，ここでは，寸法の実装例のみを記載する．

```
// 描画の実行
glBegin(GL_LINE_STRIP);
    // 折線を描画するための繰り返し
    for(int i=0;i<m_iNumber;i++){
        // 描画の実行
        glVertex3d((double)m_alX[i],(double)m_alY[i],(double)m_alZ[i]);
    }
```

```
    // 描画の終了
    glEnd();
```

(2) 描画例

本研究で実装した引出線を図 6.22 に示す.

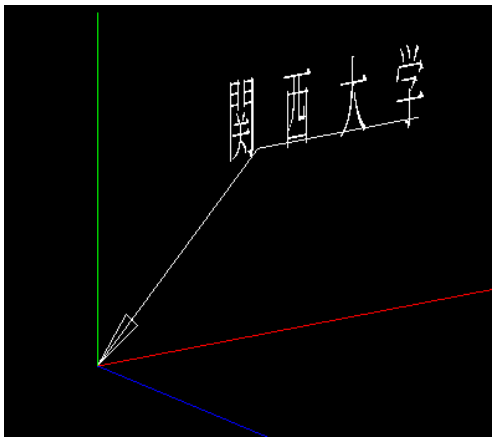


図 6.22　本研究で実装した引出線

6.4.9　バルーン

バルーンは，引出線と同様で図面に説明をいれるために使用する注釈である．バルーンの概要を図 6.23 に示す.

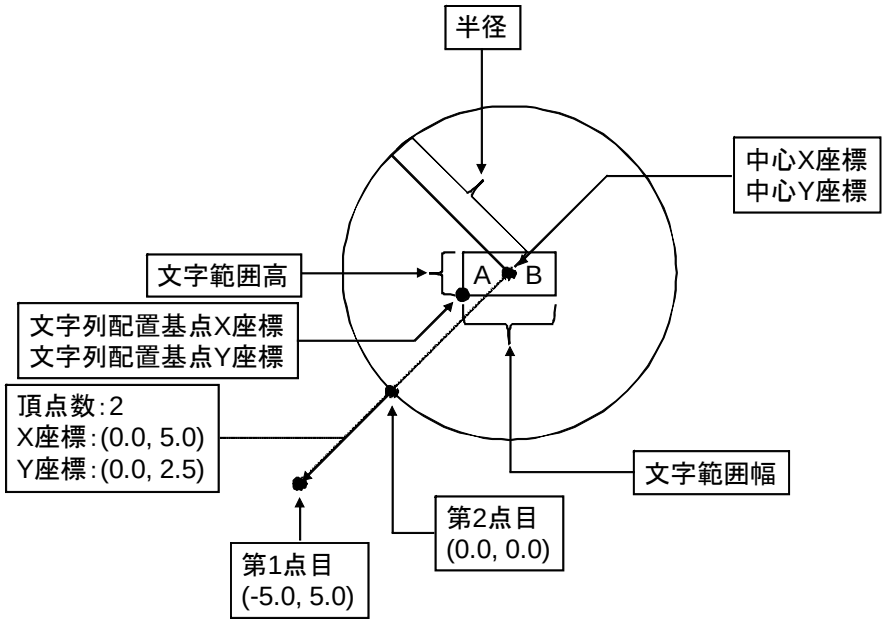


図 6.23　バルーンの概要

バルーンは，直線，円，矢印と文字によって構成される．

(1) 実装方法

バルーンの実装方法は，折線，1 つの円，1 つの矢印と文字を描画することで実現する．バルーンの実装方法を図 6.24 に示す．

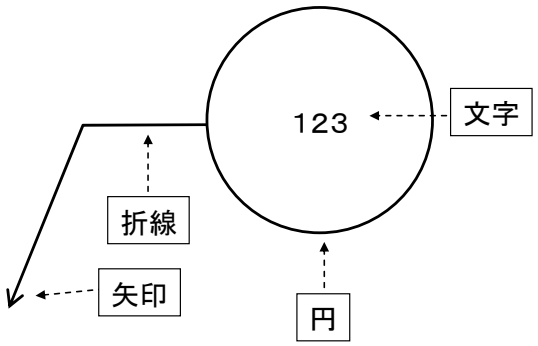


図 6.24　バルーンの実装方法

図 6.24 に示すように，バルーンでは，1 本の折線，1 つの円，1 つの矢印と文字を描画する．バルーンの実装例を次に示す．ただし，文字と矢印の実装例については，第 6.4.1 項と第 6.4.2 項に記載済みであるため，ここでは，寸法の実装例のみを記載する．

```
// バルーンの折線の描画
glBegin(GL_LINE_STRIP);
    // 折線を描画するための繰り返し
    for(int i=0;i<m_iNumber;i++){
        // 描画の実行
        glVertex3d((double)m_alX[i],(double)m_alY[i],(double)m_alZ[i]);
    }
// 描画の終了
glEnd();

// バルーンの円の描画
// 図形描画時の分割数
int iStep = 1080;

// 円の頂点 X 座標を格納する ArrayList の生成
System::Collections::ArrayList^ alCircleX = gcnew System::Collections::ArrayList();
// 円の頂点 Y 座標を格納する ArrayList の生成
System::Collections::ArrayList^ alCircleY = gcnew System::Collections::ArrayList();
// 円の頂点 Z 座標を格納する ArrayList の生成
System::Collections::ArrayList^ alCircleZ = gcnew System::Collections::ArrayList();

// 円の頂点座標算出
```

```
    ArcToPolyline(alCircleX,alCircleY,alCircleZ,
              0,0,0,m_dRadius,m_dRadius,0,0,360,iStep);

    // 描画の開始
    glBegin(GL_LINE_LOOP);
         // 円を描画するための繰り返し
         for(int i=0;i<alCircleX->Count;i++){
              // 頂点の指定
              glVertex3d( (double)alCircleX[i],(double)alCircleY[i],(double)alCircleZ[i]);
         }
    // 描画の終了
    glEnd();
```

(2) 描画例

本研究で実装したバルーンの図 6.25 に示す.

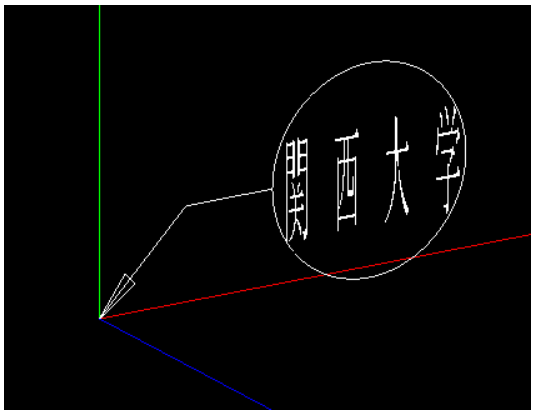


図 6.25　本研究で実装したバルーン

6.5　使用すべきAPI

本節では，本章で説明した CAD 機能を実現するために使用した API について整理し，使用すべき API について纏める．CAD の機能の実装に使用すべき API を表 6.14 に示す.

表 6.14　CAD の機能の実装に使用すべき API

API	用途
glViewport	モデルの描画領域の設定
glDepthFunc	陰面処理の比較方法の設定
glOrtho	平行投影の設定
gluPerspective	透視投影の設定
glTranslated	モデルの平行移動
glRotated	モデルの回転
glScaled	モデルの尺度変更
glColor3d	モデルの色の設定
glLineStipple	モデルの線種の設定
glLineWidth	モデルの線幅の設定

wglUseFontOutlines	文字の描画（Windows アプリケーション用）

6.6 おわりに

本章では，モデルの作成以外の 3 次元 CAD エンジンで必要と機能について OpenGL での実装方法について調査した. 描画空間の設定, 視点の切り替えや移動などの処理は, OpenGL の関数を利用することで実現できることがわかった．ただし，注釈について，直接，作成するための方法が用意されていないため，5 章で調査したモデル（線分，折線，円，楕円）を組み合わせることで実現した.

7　マイクロソフトが提供するグラフィックスライブラリの動向調査

7.1　はじめに

OpenGL 以外にマイクロソフト社がグラフィックスライブラリを提供している．そこで，本章では，マイクロソフトが提供しているグラフィックスライブラリについて調査する．まず，マイクロソフト社のグラフィックスライブラリについて整理する．次に，モデルを描画するための関数や方法について調査する．

7.2　3次元グラフィックスライブラリ

本節では，マイクロソフト社が提供する最新のグラフィックスライブラリについて調査した．調査対象のライブラリとして，Direct3D と WPF（XAML）を対象とした．以下に調査結果について述べる．

7.2.1　Direct3D

Direct3D は，マイクロソフト社が提供するマルチメディアライブラリ，DirectX の一種である．Windows の GDI（Graphics Device Interface）では不可能な高速で高品質なグラフィックス表示を行うライブラリである．Direct3D は，主に複雑な 3 次元図形を表示することを目的としているが，2 次元図形も扱うことができる．DirectX 9 以降のバージョンからはマイクロソフト社の提供する統合開発環境 Visual Studio との連携により，より容易で複雑な描画処理が可能となった．

(1) 利用可能なプリミティブ

Direct3D で利用可能な基本プリミティブを図 7.1 に示す．

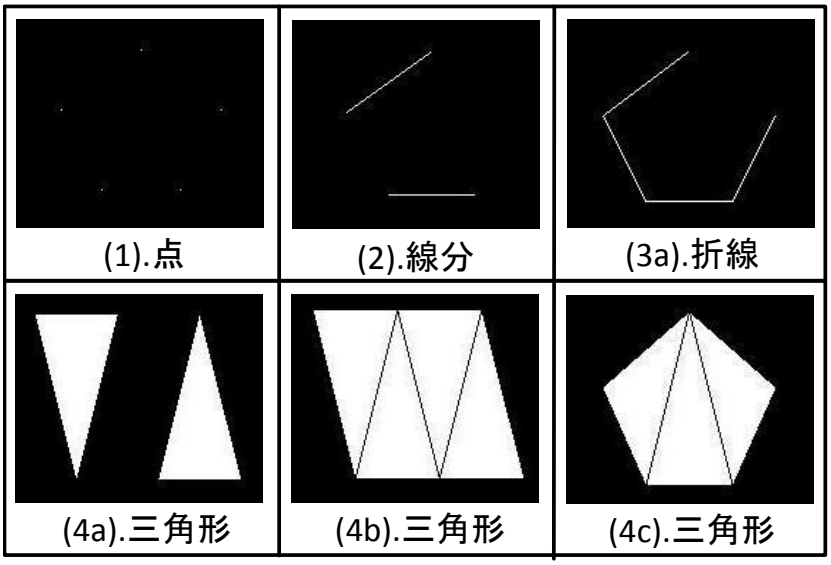


図 7.1　Direct3D で利用可能な基本プリミティブ

Direct3D では，点，線分，折線，三角形の基本プリミティブを利用することができる．一方，OpenGL では，第 5.2 節で解説した通り，上記の基本プリミティブに加え，凸多角形，四角形を利用することができる．そのため，複雑な図形の描画を考えた場合，OpenGL の方が適していると考えられる．

Direct3D で利用できる高次プリミティブを図 7.2 に示す．

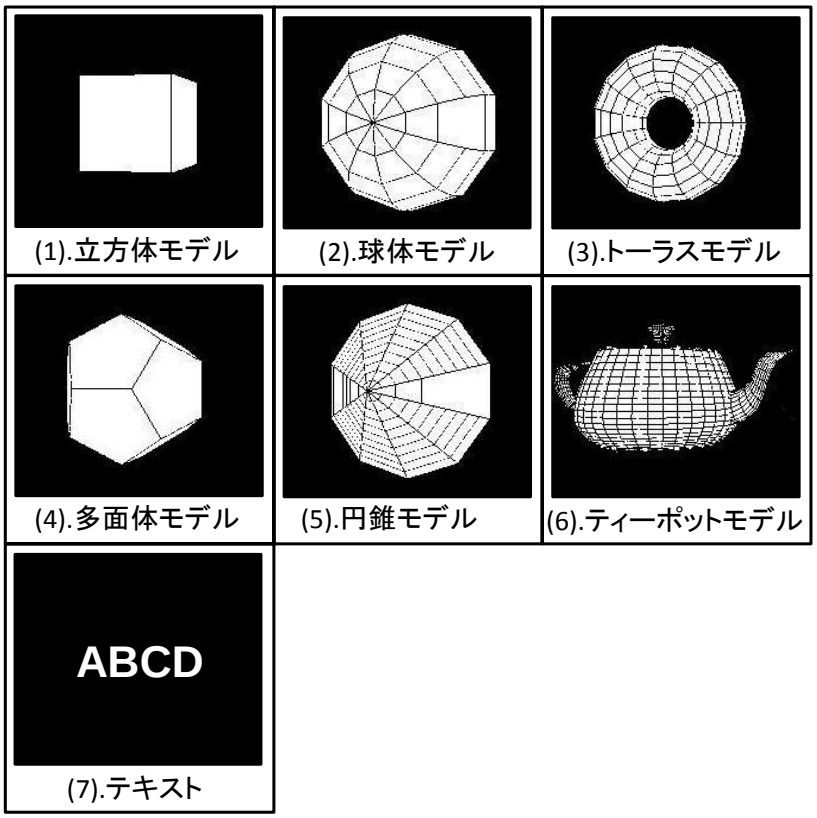


図 7.2　Direct3D で利用可能な高次プリミティブ

Direct3D では，立方体モデル，球体モデル，トーラスモデル，多面体モデル，円錐モデル，ティーポットモデルとテキストの高次プリミティブを利用することができる．これは，OpenGL で利用できる高次プリミティブと同じであるが，テキストを利用することができると点で Direct3D に分があると考えられる．しかし，OpenGL でも補助ライブラリを利用することでテキストを利用することができるため，高次プリミティブに関しては，OpenGL と Direct3D で差がないと考える．

7.2.2　WPFとXAML（eXtensible Application Markup Language）

WPF（Windows Presentation Foundation）とは，Windows の最新 OS である Vista のために開発されたグラフィックスサブシステムで，2D グラフィックス，3D グラフィックス，ビデオなどを統一して扱うことができるといった特徴を持っている．しかし，WPF は，高速でリアルな 3D アプリケーションを作成するための API でないため，OpenGL や Direct3D のような 3 次元グラフィックスを扱うための多様な API が存在しない．

XAML とは，WPF が提供する GUI を定義する XML（eXtensible Markup Language）ベースのマークアップ言語である．マークアップ言語とは，HTML（Hyper Text Markup Language）のように，文章をタグと呼ばれる文字列で囲むことで，文章の構造や見栄えに関する指定を文章と共にテキストファイルに記述する言語である．WPF では，マネージコードのみで

GUI を作成することもできるが，階層構造を簡単に表現できると共に階層構造を視覚的にも理解しやすいマークアップ言語である XAML を一般的に利用する．アプリケーションの処理は，別ファイルに C#や VB.NET を使用して記述する．

(1) WPFによる３次元モデルの作成

WPF では，OpenGL や Direct3D と異なり，立方体，円柱，球体モデルなどの３次元モデルを作成するためのプリミティングが用意されていない．そのため，WPF で３次元モデルは，点，線，面で構成される．具体的には，構成する頂点情報から稜線を作成し，稜線から面を作成することで立体形状を作成する．そのため，複雑な図形を描画することは困難である．WPF による立方体の描画方法を図 7.3 に示す．

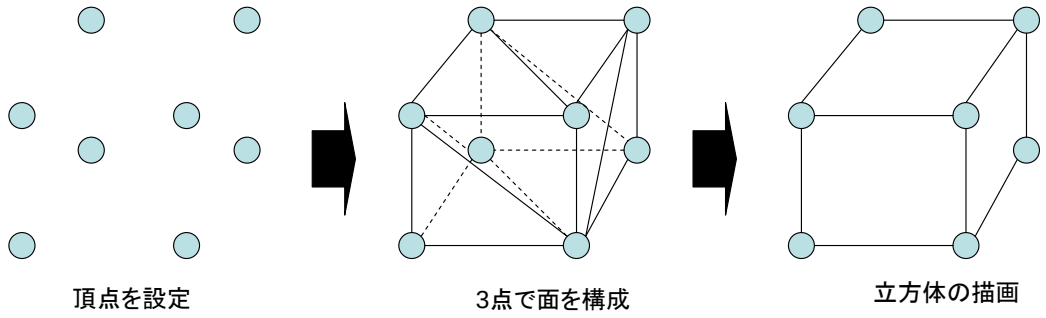


図 7.3　WPF による立方体の描画方法

7.3　おわりに

本章では，マイクロソフト社が提供するグラフィックスライブラリについて調査した．ここでは，Direc3D と WPF（XAML）を調査対象とした．調査結果から WPF は，3 次元 CAD エンジンを作成する際には適さないものであることがわかった．しかし，Direct3D に関しては，使用できる OS や開発環境が限られるが，OpenGL と同等の描画関数が用意されていることがわかった．

8　結論

本研究では，グラフィックスアプリケーション開発のための OpenGL の可能性について調査した．本研究では，現在，国産で安価なものが期待されている汎用 3 次元 CAD エンジンをターゲットとして，OpenGL を利用した際の実現方法について調査した．本研究で調査した項目は以下の通りである．

- 市販 3 次元 CAD の実態調査
- OpenGL の OS や開発言語との相性調査
- 3 次元 CAD エンジンのプロトタイプの開発に向けての機能調査
- OpenGL による 3 次元モデルの表現方法
- OpenGL による CAD の機能の実装方法
- マイクロソフトが提供するグラフィックスライブラリの動向調査

各調査によって得られた成果と結論を以下に述べる．

- **市販3次元CADの実態調査**

ここでは，既存の 3 次元 CAD の機能や取り扱う要素を整理し，市販の 3 次元 CAD の機能比較を行った．この調査により，3 次元 CAD エンジンを開発する際，モデル表現には,ソリッドモデルとワイヤーフレームを実装し，座標系は，右手直交座標系を実装すべきであることがわかった．

- **OpenGLのOSや開発言語との相性調査**

ここでは，3 次元 CAD エンジンを開発する上での開発環境を決定するため，OpenGL を利用した際の各 OS と開発言語について調査した．この調査により，提供されている補助ライブラリや統合開発環境の充実性から開発環境には, OS を Windows, 開発言語を Visual C++ にすべきであるという結論に達した．また，グラフィックボードには，NVIDIA 社の Quadro が最も適しているという結論に達した．

- **3次元CADエンジンのプロトタイプの開発に向けての機能調査**

ここでは, 本研究で開発する 3 次元 CAD エンジンのプロトタイプで実装する機能について整理するため，市販の汎用 3 次元 CAD が保持する機能について調査した．そして，調査結果から本研究で実装する機能について整理した．

- **OpenGLによる3次元モデルの表現方法**

ここでは，上記の機能調査で整理した機能の内，3 次元モデルの描画についてフォーカスを当てて実現方法について調査した．この調査により，3 次元モデルを表現する際に使用すべき API を把握することができた．モデル表現については，3 次元 CAD エンジンで使用されているモデルの内，点，曲線要素に関して，ネイティブな API だけでは表現することが困難であることがわかった．そのため，各モデルを折線などに近似することでモデルを表現する必要があることがわかった．また，面要素に関しては，ネイティブな API で表現することができたが，モデルによっては，図形近似による表現が必要なものがあった．

- **OpenGLによるCADの機能の実装方法**

ここでは，3 次元モデル以外の 3 次元 CAD エンジンの機能の実装方法について，調査した．この調査により，モデルの表現以外の 3 次元 CAD エンジンで必要と機能を実装する際に，使用すべき API を把握することができた．また，3 次元 CAD エンジンで必ず必要となる機能の実装方法について把握することができた．

- **マイクロソフトが提供するグラフィックスライブラリの動向調査**

ここでは，OpenGL 以外のグラフィックスライブラリについて把握するため，マイクロソフト社が提供するグラフィックスライブラリの Direct3D と WPF（XAML）について調査した．この調査により，マイクロソフト社のグラフィックスライブラリについて把握でき，Direct3D が OpenGL と同等の関数を保持していることが把握できた．ただし，Direct3D や WPF は Windows アプリケーションで動作するものであるため，Mac や Linux などでは利用できないことが把握できた．これらを踏まえて，汎用的な 3 次元 CAD エンジンの開発に適したグラフィックスライブラリは OpenGL であることがわかった．

様式－３－３

RESEARCH ON POSSIBILITY OF "OPENGL" FOR DEVELOPING GRAPHICS APPLICATION

Tanaka, S.[1] Shibasaki, R.[2] Kitagawa, E.[3] Kubota, S.[4] Monobe, K.[5] Yoshida, H.[6]
[1]Faculty of Informatics, Kansai University [2]Center for Spatial Information Science, The University of Tokyo
[3]Faculty of Management Information, Hannan University [4]Faculty of Software and Information Science, Iwate Prefectural University [5]School of Project Design, Miyagi University [6]Graduate School of Information Technology, Kobe Institute of Computing

The plan concerning use of 3D CAD is proposed in CALS/EC Action Program 2005 (Ministry of Land, Infrastructure and Transport). However, the majority of 3D CAD engines are foreign products, with no domestic products at all now. 3D CAD data has not been used in the design and the construction phases of the construction fields, because there is no cheap and domestic 3D CAD engine. Therefore, it is to be hoped that a cheap and domestic 3D CAD engine will be developed. In the present research, the current state of graphics libraries for the development of 3D CAD engine is understood. Especially, "OpenGL" that has produced actual results in developing graphics applications is focused on, and the possibility of "OpenGL" is investigated. The investigation content is as follows in the present research.

- Investigation of 3D CAD using OpenGL
- Investigation of concept of phase and geometry about OpenGL
- Investigation of API that should be used in OpenGL
- Investigation of compatibility of OpenGL with other language and systems
- Investigation of the function of developing prototype of 3D CAD using OpenGL
- Investigation of the trend of graphics libraries offered by Microsoft

In the present research, for understanding the possibility of "OpenGL" for development of graphics application, we investigated into the above six matters. The conclusion obtained by the present research is as follows.

- Functions that should be implemented when 3D CAD is developed
- The model representation and coordinate that should be used
- API that should be used when 3D CAD is developed
- The model that can be expressed using native API in OpenGL
- APIs that should be used when a model is made
- OS, development environment, and graphics boards that should be used
- The most suitable graphics library for developing 3D CAD

We think that the report in the present research can be a reference when a CAD vender in our country develops a 3D CAD engine. In the future, we will develop a pilot system of 3D CAD engine based on the results of investigation in the present research. In addition, we will evaluate whether "OpenGL" is excellent in terms of functions, speed and cost.

KEYWORDS : *3D CAD, OpenGL, graphics library, Model representation*

様式－３－２

研　究　成　果　の　要　約

助成番号	助　成　研　究　名	研　究　者・所　属
第2008-2号	グラフィックスアプリケーション開発のための OpenGLの可能性に関する調査研究	関西大学総合情報学部 先端科学技術推進機構 教授　田中成典

1. はじめに

国土交通省CALS/ECアクションプログラム2005では,3次元CADの利活用に関する構想が提案されている．しかし，現在，利用されている3次元CADは,AutoCADやSolidWorksをはじめとした国外の高価なソフトウェアが大半であり，国産は皆無である．このことが，建設事業の設計・施工フェーズにおける3次元CADデータの利活用の弊害となっている．そのため，国産の安価な3次元CADエンジンの早急な開発が切望されている．

そこで，本研究では，3次元CADエンジンの開発に欠かせないグラフィックスライブラリの現状を把握することを目的とし，グラフィックスアプリケーションの構築に実績のあるOpenGLを対象として，OpenGLの可能性に関する調査研究を実施する．

2. 研究内容

本研究では，OpenGLの可能性を把握するために，次の6項目について調査を行った．

2.1.市販3次元CADの実態調査

本調査では，既存の3次元CADの実態を把握するため，市販の汎用3次元CADの特徴，モデリングカーネル，モデル表現や座標系について調査した．

2.2.OpenGLが保持する位相と幾何の概念調査

本調査では,OpenGLが保持するAPI群について調査し,モデルの描画に利用可能なAPIについて整理した．また，OpenGLの保持するAPIを利用して汎用3次元CADで必要となる各モデルの描画方法について調査した．

2.3.最も利用すべきOpenGLのAPIの調査

本調査では，3次元CADエンジンの機能の実現方法について調査し,利用すべきAPI群について整理した．本研究で対象とした機能は，汎用3次元CADが保持すべき機能の内，OpenGLと関連があるものとした．

2.4.OpenGLと他言語や他システムとの相性調査

本調査では，OpenGLと各OSや開発言語との相性について調査した．また，3次元CADエンジンの開発時に使用すべきグラフィックボードについても調査した．

2.5.OpenGL による3次元CADエンジンのプロトタイプの開発に向けての調査

本調査では，3次元CADエンジンの開発に向けて実装すべき機能について纏めるため，汎用3次元CADが保持する機能について調査し，本研究で実装する機能について整理した．

2.6.マイクロソフトが提供するグラフィックスライブラリの最新動向調査

本調査では，OpenGL以外のグラフィックスライブラリについて把握するため，マイクロソフト社が提供しているグラフィックスライブラリであるDirext3DとWPFについて調査した．

3. おわりに

本研究では，3次元CADエンジン開発のためのOpenGLの可能性を把握することを目的として調査を行った．本調査研究によって得られた結論は次の通りである．

- 3次元CADエンジンの作成において実装すべき機能
- 使用すべきモデル表現および座標系
- 3次元CADエンジンを開発する際に使用すべきAPI
- OpenGLのネイティブなAPIで表現可能なモデル
- 使用すべきOS，開発環境，グラフィックスボード
- 3次元CADエンジン開発に最も適しているグラフィックスライブラリ

今後は，本研究の調査結果を基に，3次元CADエンジンのパイロットシステムを開発し，機能面，速度面，コスト面においてOpenGLが優れているかどうかを検証する．