

# J G C L

(Java Geometry Class Library)

プロジェクト

産業システム第二部第三グループ 山田 亮介

## 1. はじめに

JGCLとは、Java Geometry Class Libraryの略で、幾何形状処理のための、Javaのクラスライブラリを表します。幾何形状処理といえば、SRAではC言語で書かれたライブラリである、GHL(Geometry Handling Library)[1]という商品を扱っています。大雑把に言えば、JGCLは、このGHLの機能をJava用に置き換えたものです。

本稿では、幾何形状処理の概要を交えながらJGCLの紹介をします。なお、JGCLは、1998年12月から情報処理振興協会(IPA)による、「次世代デジタル応用基盤技術開発事業」[2]の一環として、精密形状処理研究所(PML)[3]と共同開発を行っており、2000年3月下旬現在、納品に向けて最後の詰めに入っています。

## 2. 幾何形状処理

「幾何形状処理のライブラリ」と言われても、ピンと来ない方が多いかも知れません。一体、JGCLを使ってどんな事が出来るのか？を理解していただくために、JGCLが提供する代表的な機能を紹介する事にします。

### 2.1 JGCLが提供する代表的な機能

#### ・ 幾何形状の定義と生成

幾何形状を計算機内で表現するためには、様々な形式があります。JGCLでは、形状要素の個々の定義とその要素ごとの階層を、STEP[4]と呼ばれる形式に基づいて定義します。

STEPは「製品データの記述と表現」に関する規格および関連する活動の総称のことで、ISOが標準化を進めている国際規格の一つです。他の

形式である、IGESやDXF形式と比較して、仕様に曖昧さが入り込まないようにしているのが特徴です。[5]

たとえば、STEPでは、直線を次のように定義しています。

P : 点  
 $\vec{u}$  : ベクトル  
 u : パラメータ  
 $(u) = P + u \vec{u}$

上の例からもわかるように、直線はある1点とその点を通るベクトルで表現できます。パラメータuのとりうる値によって、直線上の点が決まり、それら点の集合が直線といえるわけです。

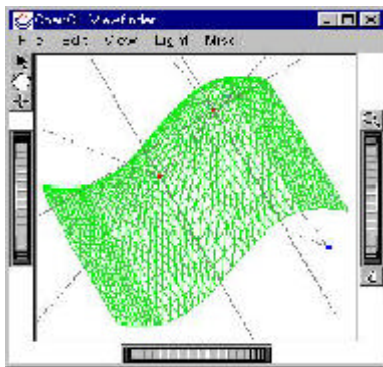
このようにJGCLでは、STEPに基づいた情報を与えてやる事で、幾何形状要素オブジェクトを生成できます。扱う事が出来る形状要素は以下の通りです。

点, ベクトル, 座標系,  
 線: 直線, 円錐曲線(円, 楕円, 放物線, 双曲線), ポリライン, 自由曲線(ベジエ曲線, Bspline曲線), トリム曲線, 複合曲線  
 面: 平面, 2次曲面(球面, 柱面, 円錐面), 自由曲面(ベジエ曲面, Bspline曲面), 回転面, 柱面, 矩形境界曲面, 曲線境界曲面  
 面以外は2次元, 3次元, 面は3次元のみです。

#### ・ 幾何形状の評価

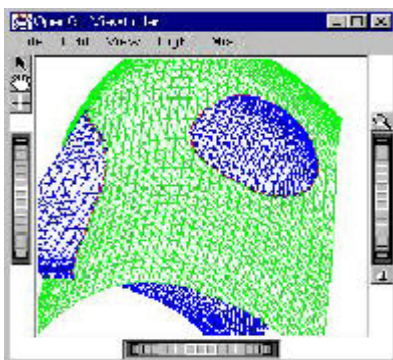
あるパラメータ値における座標値や、接線の傾き, 曲率などを求める基本的な機能を持ちます。これら単体でももちろん使えますが、様々な演算の下請けとして働きます。

- ・ 投影点



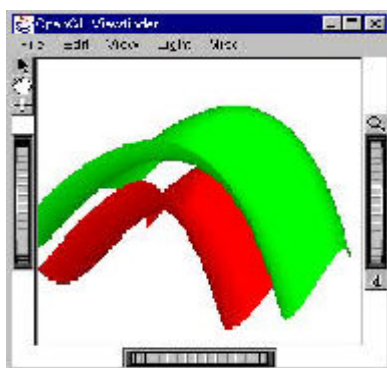
ある点から、ある曲線、曲面上に下ろした垂線の足を求めます。元の点と、計算結果で得られた点との距離を取る事で最短距離の計算にも利用できます。単体で使われるというよりも、他の機能の下部部分として多くの局面で使われます。

- ・ 交点 交線



形状同士が交わった際にできる、点、線を求めます。曲線同士、または曲線と曲面の場合は交点を、曲面同士の場合は交線を求めます。2直線の交点を求めるプログラムを付録として添付しました。

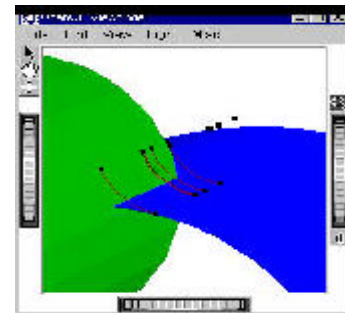
- ・ オフセット



オフセットとは、元の形状を指定した幅の分だけずらした形状を生成する演算の事です。単純に

移動させるという意味でなく、元の形状を包み込むような、または、その逆の形状を生成させる機能です。

- ・ フィレット



フィレットとは、ある形状の角を丸める操作の事です。丸めたい角に円または球を当てはめるようなイメージです。円または球の半径を指定することで、生成させるフィレットの大きさを変える事が出来ます。

## 2.1 形状処理ライブラリの適用分野

JGCLの代表的な機能について説明をしましたが、これらの計算ができてなにが嬉しいの？と思われる方も多いかと思えます。機能だけを見れば、何に使うのだろと思われるかも知れませんが、こういった機能は、3次元形状をデザインするCADの中の計算エンジンや、フォントデータの作成などに使われています。

CADなどに使用する場合は、要求する「精度」「誤差」などを意識しないとイケないのですが、JGCL、GHLでは、それらの部分を意識した作りになっています。それらを意識して作られたライブラリは、実はあまり多くありません。

JGCLの元となっているGHLは、実際にCAD製品の中の計算エンジンとして組み込まれています。また、フォントデータ作成、管理のためにも使われています。滑らかなフォントを実現するためのアウトラインフォントは、B-spline曲線などで表現されているからです。

GHLの利用は、身近なところでは、オープンソースソフトウェア部の青木さんが中心となって開発されている「じゅん for Smalltalk」[6]と、それを結合させるためのライブラリである「けいじょう for Smalltalk」[7]を使う事で、生命研プロジェクトの中でも、利用されていました。今回GHLのJava版として、「JGCL」が完成すれ

ば、「じゅん」のJava版である「じゅん for Java」[8]と連携することで、そのような事もJavaの世界だけで記述できることになります。今後はこれらを利用して、なにか面白い事が出来るのではないかと考えています。

### 3. 開発環境

JGCLの開発は以下の環境で行いました。

- OS Windows95/98/NT, FreeBSD, Linux, MacOS, Solaris for intel
- Java JDK 1.1 以降

OSについては、Javaが動けばなんでもよいということで、各個人の好みで分かれています。ただし、発注仕様書にはWindows95/98/NTとUNIX系のOSで動作させると記述されていますので、最終的にそれらのOS上での動作は必須です。

### 4. 設計と実装

このプロジェクトでは、主に以下の3つのものを実装しました。

- JGCL 本体
- Judan (じゅん for Javaを用いたテスト環境)
- NH (GHLとの演算結果の比較を行うためのテスト環境)

私の担当がJGCL本体とJudanの開発だったので、この2つを中心お話しします。

#### 4.1 JGCL の設計と実装

標準ライブラリ以外は使用しないという方針を立てました。これにより、JGCLの利用、JGCLを使って作られたアプリケーションの実行に、他ライブラリは不要となり、ライブラリの独立性が高まります。GHLも標準のCライブラリしか使用しないことで、様々なプラットフォームでの利用を容易にしています。

##### 4.1.1 クラスの設計

クラス階層は、STEPの定義に基づき設計しました。また、クラスの内部の設計では、STEPで定義されている形状要素の定義情報をクラスの属性として持たせています。コンストラクタは、それらの形状要素の定義情報を引数に持つものをpublicなものとして備えています。

メソッドについては、基本的にGHLの関数をそれに対応するJGCLのクラスのpublicメソッ

ドとしてそのまま移植する方針で行いました。C言語特有のポインタ操作など、Javaではそのまま記述できない部分や、GHLの元々のソースコードの見通しが悪い部分など、そのような部分は、適宜修正しました。

##### 4.1.2 実装アルゴリズム例

幾何形状処理を理解していただくために、交点を求める方法をご紹介します。交点、交線計算は形状の組み合わせによって計算方法が異なり、すべての方法をここでご紹介するのは無理です。ここで述べるもの以外の機能の実装方法などは、参考文献[9][10][11]などに任せ、ここでは、ベジエ曲面とベジエ曲線の交点を求める方法を単純化してご紹介します。大まかには以下の処理を行っています。

- ベジエ曲面を平面とみなせるまで分割する
- ベジエ曲線を直線とみなせるまで分割する
- 分割した平面と分割した直線が交わりを持つかどうか調べる
- 分割した平面と分割した直線の交点を求める(おおまかな解)
- 求めた解を初期値として、収束演算を使って、厳密な解を求める
- 求めた解を誤差を考慮して、異なった解とみなされた場合、解に追加する。

##### 4.1.3 許容誤差

JGCLでは、点の座標値など、内部で扱う数値は倍精度の実数であり、その倍精度実数内での誤差を考慮した演算を行う事が特徴です。

単純にある点在同一の点であるかどうかを判定する場合を考えてみます。なんらかの計算結果の後、以下に示す点A, Bが得られたとします。

A(1.0, 1.0, 1.0)

B(1.0000001, 1.0, 1.0)

それぞれの(x, y, z)座標の値を単純に比較してしまえば、点A, Bは別物と言う判定がなされます。しかし、有限精度の実数計算を行う場合、その演算結果には誤差がつきものであり、どの程度の誤差を許容範囲とするかを考える必要があります。すなわち、点は許容誤差分の半径を持った円の内側部分と考える必要があります。直線の場合は、±許容誤差分の幅を持った線と考える事が出来ます。

許容誤差を大きく取り過ぎると異なった解も同じ物とみなされ、小さく取り過ぎると、同一解となるはずのものが別の解として現れます。交点計算などでは、演算の繰り返しによる蓄積誤差を取り戻すために収束演算を行い、解を誤差内に収めるようにします。収束演算の結果、誤差内に収まらなかったものは、別の解となります。

このように誤差の扱いは非常に微妙な問題であり、JGCLでは計算スレッド毎に許容誤差を個別に設定できる機能を持ちます。

## 4.2 Judan

Judanとは、JGCLのためのテスト環境(ドライバ)であり、JGCLを内部で利用したアプリケーションの一つとも言えます。ここでは、Judanの持つ機能とその実装について簡単にご紹介します。

### 4.2.1 Judanの機能

#### - グラフィック表示機能

計算結果のグラフィック表示には、オープンソースビジネス部で開発中の「じゅん for Java」を使用しております。JGCLは単なる計算エンジンであり、演算結果を画面に表示するための機能は持ちません。計算結果を数値的に表示する事は可能ですが、点の座標値などの、数値の羅列を見ても、計算結果が正しいのかが非常にわかりにくいです。

動作確認を行う際、実行結果の正当性の検証に時間がかかってしまうため、直感的に理解するための仕掛けが必要となります。例えば、曲面同士の交わりの計算結果を確かめるのに、2曲面の交わっている境界に、正しく交線が現れているかどうかを目視出来たほうが、テスト結果の確認も容易です。Judanを作成した理由の一つはここにあります。

#### - パーザ機能

Judanでは、あるフォーマットにしたがって記述された幾何形状の定義情報、その形状要素に対する操作情報を読み込むことで、その幾何形状オブジェクトを生成し、メソッドを実行する機能を持ちます。JGCLはクラスライブラリなので、メソッドのテストには、そのメソッドを呼び出しを記述したmainメソッドが必要となります。ある機能を実現するためのメソッドのテストをするた

めに、一つ一つmainメソッドを書くのは、かなり面倒な作業であり、mainメソッドの手直しのたびにコンパイルする時間も必要となります。この機能を提供する事で、上に述べた煩雑な作業を無くす事ができ、テストにかかる時間を大幅に短縮できます。

### 4.2.2 Judanの実装

Judanは大きく以下の2つの部分に分ける事が出来ます。

#### - フロントエンド部(解析部)

パーザ機能の実装には、JavaCC[12]を使用しました。JavaCCは、C言語でいうところのlex & yaccを合わせた機能を持ちます。アクション部と呼ばれる部分に、構文が合致したときに行う処理を記述して、オブジェクトの生成や以下に述べる命令実行部に渡す情報を記憶させておきます。

#### - バックエンド部(解釈実行と結果表示)

JavaのReflection機能を利用して、指定されたメソッドの呼び出しを行います。フロントエンド部で解析された情報を元に、Methodオブジェクトを生成し、実行させます。

結果の表示においては、メソッドの実行結果を、「じゅん for Java」の幾何オブジェクトに変換して、表示させます。はじめにJGCLの機能を用いて、曲線の場合は細かい線分の集合として、曲面なら、平面をなす三角形の集合に変換します。それらを、じゅんのポリラインとポリゴンに変換し、表示させています。

## 5. おわりに

本稿では、幾何形状処理演算を行うためのクラスライブラリJGCLについて、幾何形状処理の概要とともに述べました。Javaで記述する事により、STEPで定義されている階層構造など、素直に反映する事ができました。構造の整理にもなり、GHLよりもコードの保守性が向上したと思われれます。しかし、JGCLが現在抱えている問題点もあるので、それについて、JGCLの今後とともに以下に挙げておきます。

### 5.1 問題点

#### - パッケージの分割

現在、すべてのクラスが同一パッケージにあり、非常に見通しが悪くなっています。これらに

ついて、カテゴリを分けるべくパッケージを分割し、見通しをよくする必要があると考えています。

#### - 重複している機能をまとめる

設計時に、形状要素以外のクラスを正しく設計しなかったため、似たような機能を持つ別のクラスが複数存在しています。実装の終盤になって、より便利なクラスが実装され、以前に実装した部分をそれで置き換えたいなどの欲求が生れる事がしばしば出てきました。そのような部分のコード統一を図り、コードの整理をすべきであると考えています。

#### - パフォーマンスの向上

現時点で、パフォーマンスはCとは比べ物にならないくらい悪いものになっています。現時点では、正確にプロファイルを取ったわけでもなく、チューニングする余地は残されていると考えています。今後、計算上のボトルネック部分を検出し、今よりも十分なパフォーマンスが得られるようにチューニングすべきだと思います。

### 5.2 J G C L の今後

J G C L の扱いはまだ決まっていません。現バージョンは、ベータサイトとして、ソースをオープンにして、ユーザに使っていただき、問題点や改良点をフィードバックしてもらおうといった案も出ています。これについては、今後、P M L とミーティングを重ねつつ、決定していく事になると思います。

問題点でも述べたように現時点では実用には耐えられない部分が多々あります。そのまま埋もれさせるには訳にはいかないため、J G C L を使う場面を考え、開発継続のためのネタ探し、スポンサー探しなどを行っていく予定です。このようなJavaレベルでの形状処理ライブラリは、皆無に近い状態だと思いますので、いろいろな局面に入り込む場所があると睨んでいます。

### 6. 参考文献と URL

- [1] <http://www.sra.co.jp/GHL/>
- [2] <http://www.ipa.go.jp/NBP/digital98-7.html>
- [3] <http://www.pml.co.jp/>
- [4] Industrial automation systems and integration  
- Product data representation and exchange -

Integrated generic resources: Geometric and topological representation, ISO 10303-42,(1994)

- [5] 小堀研一, 春日久美子: 基礎から学ぶ図形処理, 工業調査会,(1996)
- [6] <http://www.sra.co.jp/people/aoki/Jun/>
- [7] <http://www.sra.co.jp/people/aoki/Kjo/>
- [8] <http://www.sra.co.jp/people/nisinaka/Jun4Java/>
- [9] 柿下尚武: ジオメトリックエンジン入門, C MAGAZINE 1999年11月号, SOFTBANK,(1999)
- [10] 鳥谷浩志: 千代倉弘明, 3次元CADの基礎と応用, 共立出版株式会社,(1991)
- [11] Gerald Farin: Curves and Surface for Computer Aided Geometric Design A Practical Guide Second Edition, Academic Press, Inc,(1990), 木村文彦監修, 山口泰監訳: CAGDのための曲線・曲面理論 実践的利用法, 共立出版株式会社,(1991)
- [12] <http://www.metamata.com/JavaCC/>

### 7. 付録

```

/*
 * 2直線の交点を求めるプログラムの例
 */
import jp.go.ipa.jgcl.*;
class IntersectLineLine {
    public static void main(String[] args) {
        // 点とベクトルを与えて直線を生成
        JgclLine3D line1 =
            new JgclLine3D(
                new JgclCartesianPoint3D(0.0, 10.0, 20.0),
                new JgclLiteralVector3D(0.0, 20.0, 30.0));
        // 点とベクトルを与えて直線を生成
        JgclLine3D line2 =
            new JgclLine3D(
                new JgclCartesianPoint3D(0.0, 50.0, -10.0),
                new JgclLiteralVector3D(0.0, -30.0, 20.0));
        // 交点を求める
        try {
            JgclIntersectionPoint3D[] sol =
                line1.intersect(line2);
            // 2直線は平行である。
            if (sol == null) {
                System.out.println("2 lines are parallel");
                return;
            }
            // 解の情報を出力する。
            for (int i = 0; i < sol.length; i++)
                sol[i].output(System.out);
        } catch (JgclIndefiniteSolution e) {
            // 2直線はオーバーラップしている
            System.out.println("2 lines are overlap")
        }
    }
}

```