幾何モデリング機能を備えた
マルチスレッド並列オブジェクト指向言語EusLisp

Multithread Concurrent Object-Oriented Language EusLisp
with Geometric Modeling Facilities

電子技術総合研究所　松井俊浩
Toshihiro Matsui (Electrotechnical Laboratory)

１．まえがき

ロボットの研究に、３次元のモデリング機能を備えたプログラミングシステムは欠くことはできない。コンピュータの中に物体の形を表現し、図形として表示しようとするプログラムの歴史は古く、現在、それらはソリッドモデラとしてCADやコンピュータグラフィックスに広く利用されている。これらのアプリケーションの利用者にとっては、モデルがどういう言語でどのように表現されるかは問題ではないかもしれない。しかし、ロボットの動作計画や画像理解をプログラムしようとする者にとっては、モデルに拡張を加えやすいか、複雑なアルゴリズムを書きやすいか、大きなシステムとして統合していけるか、などは重要な関心事である。既存のソリッドモデラを調査して分かったことは、オブジェクト指向という用語が登場する前から、これらのプログラムの内部はオブジェクト的に構成されており、オブジェクトの関係を記述するためにポインタが多用されていることである。このような目的にとって、オブジェクト指向型Lispは最適な実装言語となりうる。

EusLispは、オブジェクト指向をベースにしたLisp言語であり、３次元幾何モデリング機能、並列プログラミング、ウィンドウプログラミングを可能にする統合型のプログラミングシステムである[1,2,3]。本稿では、EusLispの使い方や特徴を解説する。

２．インストールの方法

EusLispは、1986年に開発を開始し、現在でも発展が続いている。CDROMに収めた版[8]もすでに古くなっているので、最新のアーカイブを[1]から入手されることをお勧めする。同URLにはリファレンスマニュアルも置かれている。EusLispは、公開ソフトウェアであるが、利用者には、登録をお願いしている。EusLispを利用できる環境は、SunOS4, SunOS5 (Solaris2)、SGI/IRIX, Alpha/OSF, PC/Linux, PC/Windowsである。installの詳細については、アーカイブ中のREADMEファイルを参照されたい。

３．簡単な使い方

EusLispは、テキストベースの会話型言語であり、キーボードからのコマンドにしたがって処理を進める。EusLispを起動すると、eusx$というプロンプトが現れる。その後、ユーザーが入力するコマンドは、LispのS式か、unixのコマンドのいずれかとして解釈され、その結果が表示される。このトップレベルの入力に限り、LispのS式の最も外側の括弧を省略することができる。load, print, listなどの関数を括弧なしでタイプすればよい。ls, cd, catなどのunixコマンドがそのまま使えるので、あたかも、unixのシェルにdcのような電卓機能やLispの機能が拡張されたような印象を受けるかも知れない。**図1**に簡単なセッションを示す。それぞれ、直方体、円柱、それらのCSG和、が作成され、その隠線消去表示が得られる。

```
% eusx
eus$ (append '(a b) '(c d e))
(a b c d e)
eusx$ + 1 2 3 4 5 6 7 8 9 10
55
eusx$ (load "lib/demo/view.l")
;; ウィンドウが一つ現れる。
eusx$ (setq cub1 (make-cube 500 400 100)
eusx$ (setq cyl2 (make-cylinder 100 200))
eusx$ (setq cubcyl3 (body+ cub1 cyl2))
eusx$ (hid cubcyl3)
```

図1　EusLispの起動と簡単なプログラムの例

４．オブジェクト指向言語としてのEusLisp

4.1　なぜLispか？

EusLispの言語としての構文や解釈の定義は、Common Lisp[9]に近い。したがって言語的な性質についての多くは、文献[9]を参照して頂きたい。Common Lispには、キーワード引数、プログラムやデータのS式による表記、無名関数(lambda式）、ハッシュ表など、CやC++にはない優れた機能が数多くある。たとえば、**図2**のように、bignumを用いると*e*を任意の精度で求めるプログラムが簡単に記述できる。

CやC++に慣れた人は、よくLispを見て括弧の多さに驚くが、それは外見上の問題に過ぎない。括弧は、begin, end, {, }, カンマ、セミコロンなどの

```
(defun compute-e (digits)
   (let ((x (expt 10 digits))
         (e (expt 10 digits))  (i 0))
      (while  (> x 1)
        (setq x (/ x (incf i)))
        (setq e (+ e x)))
      e))
```

図２　自然対数の底を多倍長で求めるプログラム

何種類もの構造化要素を一種類に統一した構文要素であり、簡単な原則にしたがっていくらでも複雑な構造が記述できる万能の構文要素である。それよりも、LispとCの比較で重要なのは、Lispには自動メモリ管理、すなわち不要になったメモリを見つけて回収する機構が組み込まれていることである。Cでもmallocの使用頻度は増加の一途であるが、Lispのようなリスト処理言語やC++のようなオブジェクト指向言語では、メモリを動的に割り当てることが多い。メモリの量は有限なので不要メモリの回収が不可欠になる。cfreeを呼べば良い、というのは当たらない。システムが複雑になり多数のモジュールで構成されるようになると、あるデータが不要か否かをプログラムの字面から判断することは不可能になるからである。実際、Cのプログラムは巨大になるにつれてこの自動メモリ管理を自前で用意することになり、ポインタ管理の不備からたびたびフォールトを起こす。Lispでは、自動メモリ管理のおかげでリストなどの不定長の構造を自由に利用することができ、プログラミングからメモリ量や表現法に関する不安を取り除ける。

次に、インタプリタとコンパイラを併用でき、特に前者によって会話的、漸近的にプログラミングできる点、さらにプログラムとデータが共にリストで表現され、プログラムを生成するプログラムが簡単に書ける点が重要である。これは、これまで単におもしろいが役に立たない性質と見られていた。昨今インタネットが普及し、異なるマシン同士がプログラムを送り合う形態が一般化しつつある状況では、プログラムをネットワーク可搬型にする役に立つ性質として注目され始めている。

一方、Lispの問題もいくつか指摘できる。

(1) 実行時に必要とするコードが大きいため、機能をライブラリ化してCのプログラムから利用したり、機器に組み込むことが難しい。
(2) 各々の関数の機能が高水準なので実行の様子が想像し難く、効率向上のためのチューニングが難しい。
(3) 自動メモリ管理に伴うゴミ集めのため、予測不能の計算停止時間が挿入される。

したがって、現状では、デバイスを制御するためのサーボにまでLispを適用するのは困難である。しかし、コンパクトで高速かつ大容量メモリを備えた計算機が普及しつつあり、マルチスレッドを利用した実時間ゴミ集めも可能になりつつあるので、これらの問題も次第に解決されていくことが期待できる[6]。

EusLispは、言語の習得を容易にするために、極力Common Lisp[9]との互換を保とうとしている。しかし、EusLispが単一継承のオブジェクト指向をベースにしてLispを実現したことによる型階層の不整合と、幾何モデリングを高速に実行させるための実用性を重視したことに起因して、*多値*、*複素数型*、*文字型*、*deftype*、*progv, compiler-let, macrolet, persistent closure*、などが非互換・未実装となっている。

4.2 EusLispのオブジェクト指向

EusLispは、数値以外のデータ型をクラスによって定義しており、各々のデータをオブジェクトと呼ぶ。クラスには、`cons, symbol, string, stream, vector, coordinates`（座標系）, `body`（物体）, `face, edge, viewing`（視野変換）、`xwindow, hashtable, readtable`など、数十が定義されている。クラスの階層は、単一継承だけが許される。ユーザーは、これらのシステム組込クラスにサブクラスを定義することによってシステム機能を拡張できる。Lispの下にオブジェクト指向を置くことにより、インスタンスの作成、メッセージ送信、スロット変数へのアクセスなどの少数のプリミティブだけを基に、統一的なシステムの実装が可能となる。

```
(defclass room :super bounding-box :slots (walls))
(defmethod room
 (:init (rs &optional (bs 30))
  (send-super :init2 (float-vector (- rs) (- rs) (- rs))
                     (float-vector rs rs rs))
  (setq walls (make-cube (- rs bs) (- rs bs) (- rs bs)))
    self)
 (:check-collision (b)
    (send self :inner (send b :worldpos)))
 (:drawners () walls))

(defclass moving-body :super body :slots (velocity))
(defmethod moving-body
  (:init (shape initial-vel)
     (replace-object self shape)
     (setq velocity (copy-seq initial-vel))
     self )
  (:tick (&optional (n 1))
    (send self :translate (scale n velocity)))  )

(defun move-until-hit (b r)
   (while (null (send r :check-collision b))
        (draw b) (send b :tick)) )

(setq ball (instance moving-body :init (make-icosahedron 40) #f(500 0 0)))
(setq box (instance room :init 300))
(draw box ball)
```

図3　クラス、メソッドの定義の例：roomの中でmoving-bodyが運動する

クラスには、メソッドを定義し、関数sendによってメソッドを起動する。クラスからオブジェクトを作成するには、

関数を用いる。**図3**に、クラス定義、メソッド定義、関数定義、インスタンス作成の例として、直方体の部屋の中でボールが壁に衝突するまでの運動を表示するプログラムを示す。

EusLispでは、問題の解法の記述に関数とクラス（メソッド）を選択することができる。一般に、(1) 変化しうる状態を持つ物、(2) 概念（クラス）に対して個性の異なるインスタンスが生成される場合、(3) 抽象化された概念から限定された概念への階層が見いだされ、一般的な概念の再利用が期待できる物、では、クラスを用いるのが適当である。一方、対等な関係にある複数のオブジェクトの間の関係や演算を記述したい場合は関数によるのが適当である。特に前者は、幾何モデルのような「もの」を記述する場合はまさにオブジェクトに対応させるのが直感的にわかりやすく、既存の機能をベースに段階的に大規模なシステムを構築していくのに都合がよい。

5．幾何モデリング機能

EusLispには、表1に示すような任意次元のベクタ、マトリックス演算プリミティブがLisp関数として組み込まれている。形状オブジェクトとして２次元および３次元の幾何モデルを扱うことができる。

図4は、モデルの表現のために定義されているクラスの継承の構造である。モデルは、座標系(coordinates)およびその連結(cascaded-coords)を中心にして定義されており、平行移動、回転、拡大などの変換は、:translate, :rotate, :scaleなどのメッセージを送ることで、モデルや視点に共通に処理される。また、座標系の実装方法と計算インタフェースを独立にすることができる。

図形・物体の形状は、多角形および多面体で近似表現する。**図5**に示すように、プリミティブとして直方体、円柱、回転体、推、トーラスなどがあり、和・差・積などの集合演算によって複雑な形状を合成する。

図6は、複数の部品からなるエンジンモデルとそのカットモデル、ETA3マニピュレータモデルの隠線消

v+, v-, v. v*, v.*	ÉxÉNÉ^ÇÃòaÅAç ÅAì‡êœÅAäOêœ
v<, v>, vmin, vmax	ÉxÉNÉ^ÇÃíÂär
scale,norm,distance	ÉXÉJÉâî{ÅAÉmÉãÉÄÅAãóó£
make-matrix,unit-matrix	É}ÉgÉäÉNÉXçÎê¨
matrix-row,-column	É}ÉgÉäÉNÉXÇÃçsÅAóÒÇÃéÊìÈoÇµ
m*,transform	É}ÉgÉäÉNÉXêœÅAÉ}ÉgÉäÉNÉXïœä
transpose,rotate-matrix	ì]ìÅAâÒì]
rpy-matrix,rpy-angle	roll-pitch-yawäp
lu-decompose,lu-solve	LUï™â

表1　幾何計算プリミティブ関数

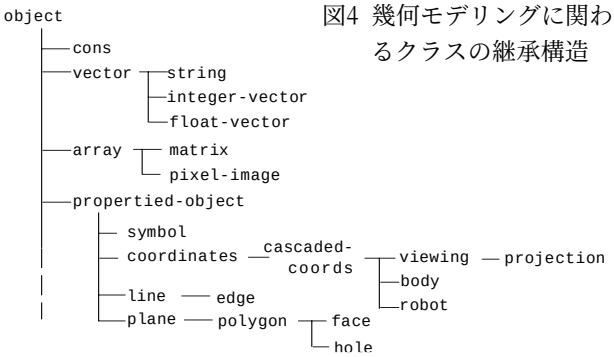

図4 幾何モデリングに関わるクラスの継承構造

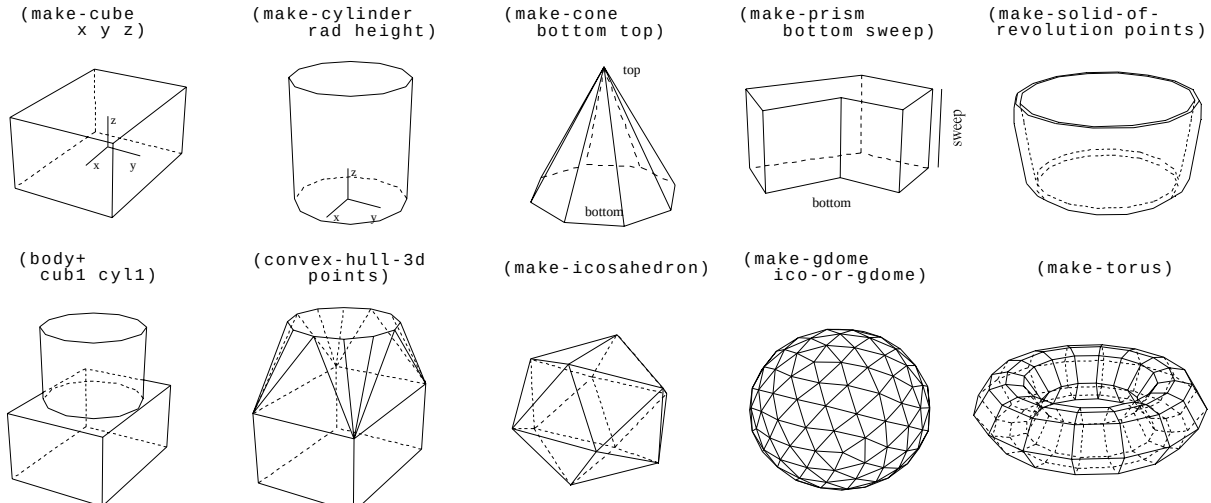

図5　幾何モデルの素立体と集合演算による合成

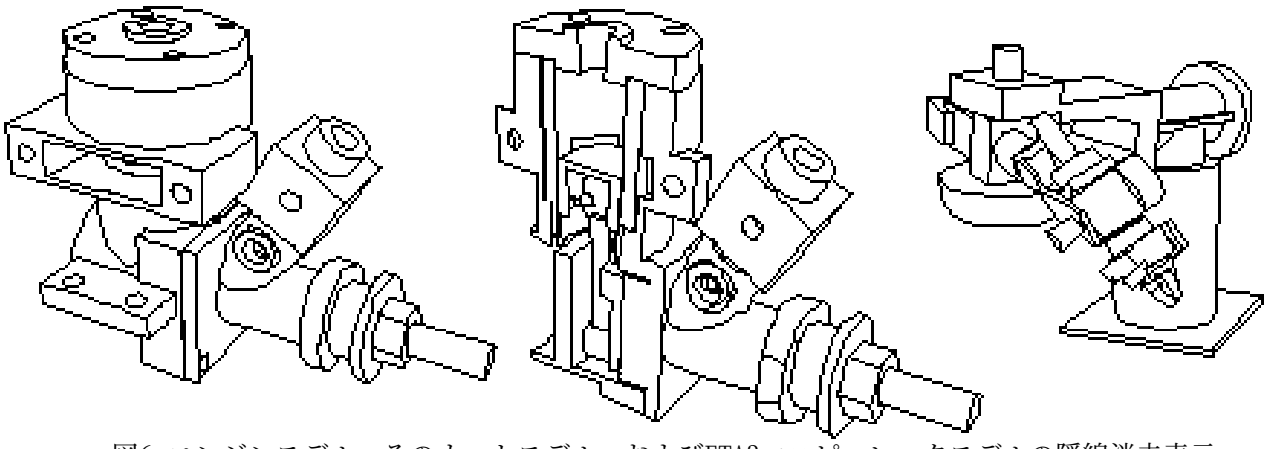

図6 エンジンモデル、そのカットモデル、およびETA3マニピュレータモデルの隠線消去表示

去表示である。図には現れないが、物体同士の干渉、多数の点の凸包、最短距離領域を表すボロノイ図、表面積や体積を計算することができる。それらを基に、安定把握位置の計画、接触物体の及ぼす動作拘束の導出[10]、組立作業計画[11]、動作シミュレーション[12]などを行うことができる。幾何モデル計算では、接触状態等の特異点において数値計算誤差のために状態の判定が不安定になり、位相構造が乱れることが問題となる。面同士の接触を取り扱える程度には安定化を図ったので、市販のパソコン用モデリングツールなどよりは信頼がおけるようになっている。

6. 並列・分散プログラミング

EusLispは、マルチスレッドを用いた共有メモリ型の並列プログラムを走らせることができる[4,5]。近年数個から数十個のCPUを備えたマルチプロセッサワークステーションが一般的になりつつある。一つの問題を複数のスレッドに分割し、異なるCPUで実行させることで実行効率の向上を図ることができる。また、スレッドは独立したシステムコールを発行できることを利用して、画像、音響など異なったチャネルから発生する非同期の事象を取り扱うプログラミングが容易になる。

スレッドは、プロセスに比べると簡単に生成できるのが特徴であるが、Lispのような動的な言語ではスタックに大きなメモリを割り当てるためのオーバヘッドが大きい。そのため、スレッドはあらかじめ必要な数を生成しておき、計算の途中で必要に応じてスレッドプールから割り当てる方法を取る。

並列に計算を行うために、関数の評価を`thread`関数によってスレッドに割り付ける。スレッドから計算の結果を受け取るには、`wait-thread`関数を用いる。スレッドはコンテキスト（スタック）を共有しないが、ヒープ中に割り付けられたオブジェクトを共有し、`symbol-value`などを通じて通信する。共有資源へのアクセスの排他制御などはプログラマの責任である。同期プリミティブとして、相互排除ロック， セマフォ、条件変数、リーダー・ライター・ロック，同期メモリポート、バリア同期が用意されている。

並列処理は、一般にスレッドの間の独立性に応じた台数効果が得られる。たとえば共有データの全くないFibonacci関数であれば、4台のCPUで3倍以上の性能向上が得られる。しかし、後述のように、共有資源へのロックが頻発するプログラム、特にメモリ管理への負荷が大きいプログラムでは、期待するような並列度が得られないこともある。

EusLispは、Unixのほとんどのシステムコールへのインタフェースを備え、TCPおよびUDPを用いたプロセス間通信だけでなく、pipe，FIFO，メッセージキュー、共有メモリ、シグナル等、を利用することができる。ネットワークを用いて分散計算を行う枠組みとして、MPI (Message Passing Interface) が提案されている[7]。MPIを用いると各マシンのアーキテクチャは同一である必要はなく、ベクタ型のスーパコンピュータ、グラフィクスワークステーション、パソコンなどを必要なだけ接続してマシンの特性に合った計算を行わせることができる。EusLispからMPIにアクセスするためのインタフェースが追加されている。

7. グラフィックスインタフェース

コンパイルされたオブジェクトコードのリンクには、system5のdl (dynamic linking)機能を用いている。dlによって、Lispに限らずあらゆる言語実行コードをリンクする他言語インタフェースが実現されており、Cのプログラムや既存のライブラリを実行時に結合することができる。数値計算、画像処理、

信号処理等のプログラム等には、Lispにこだわらず実績のあるライブラリを使うべきである。

この方法でXlibがリンクされている。Xlibは、MT-safeにはできていないので、複数のスレッドがXwindowに同時にアクセスすることは危険である。Xlibの上にXwindowツールキットが、EusLisp自身で記述されている。

3次元グラフィックスは、EusLisp自身で書かれた隠線消去、レイトレーシングプログラムがある。ハードウェアを使ってより高速の画像が得られるよう、OpenGLとのインタフェースが取られている。Mesaを用いればOpenGL互換の機能が通常のワークステーションで利用できる。

8. 終わりに

EusLispは、現在までに東大、阪大等国内の数十の研究機関と国外約10研究機関に配布されている。これまで主にロボットのモデリングや作業計画に応用されてきたが、移動ロボットの行動制御の記述等にも応用が始まっている。今後、プログラミング環境も含めて、並列機能の拡充に傾力する予定である。また、インタプリタとS式を用いれば可搬性の高いプログラミングができるので、JAVAのようにインタネット上でプログラムを交換し合うシステムを計画中である。

参考文献

[1] T. Matsui: Object-Oriented Concurrent Lisp with Solid Modeling Facilities, http://ci.etl.go.jp/~matsui/eus/euslisp.html.

[2] 松井俊浩, 原功: EusLisp version 8.0 Reference Manual, 電子技術総合研究所研究速報, ETL-TR-95-19, 1995.

[3] Matsui, T. and Inaba, M : EusLisp: an Object-Based Implementation of Lisp", Journal of Information Processing, vol. 13, no. 3, pp.327-338, 1990.

[4] 松井俊浩, 関口智嗣: マルチスレッドを用いた並列EusLispの設計と実現, 情報処理学会論文誌, vol. 36, no. 8, pp.1885-1896(1995).

[5] 松井俊浩, 関口智嗣: マルチスレッドEusLispのスレッド分割型メモリ管理, 情報処理学会プログラミング研究会、95-PRO-3、pp.9-14 , 1995.

[6] 田中良夫、松井俊浩: 並列ゴミ集めのEusLispへの実装, 情報処理学会プログラミング研究会 ,95-PRO-6, pp.1-6, (1996).

[7] Gropp, W., et al., Using MPI, The MIT Press , 1994.

[8] ロボット工学ソフトウェアライブラリ、日本ロボット学会誌、vol. 14, no. 1, 付録、1996.

[9] Guy L. Steele Jr.: Common Lisp The Language, 2nd Ed.," Digital Press, 1990.

[10] Hirukawa, Matsui, T. and Takase, T: Automatic Determination of Possible Velocity and Applicable Force of Frictionless Objects in Contact from Geometric Models," IEEE Trans. Robotics and Automation, vol. 10, no.3, pp.309-322, 1994.

[11] 松井俊浩、坂根茂幸、比留川博久：EusLispの幾何モデリング機能を用いた組立状態推論、電子技術総合研究所彙報、59巻1号、pp.57-66, 1995.

[12] 松井俊浩：オブジェクト指向型モデルに基づくロボットプログラミングシステムの研究、電子技術総合研究所研究報告、第926号、1991.