Šárka Voráčová

COMPUTATIONAL GEOMETRY WITH MAPLE

Abstract

The paper presents some elementary methods for computational geometry and their further studies of the running time complexity and their dependence on the various input size parameters. The goal is to demonstrate the utilization of Maple package in to the Computational geometry. Pedagogical benefits can be found in the large number of Maple programs, some of which are analogous to C++ programs, including those for convex hulls of a point set in small dimension, planar polygonal partitioning and triangulations.

Keywords

Convex Hull, Triangulation, Computational algorithms

1 Introduction

Computational geometry is concerned with the design and analysis of algorithms for geometric problems. Algorithms arise in many practical areas such as computer graphics, robotics and engineering design. The basic techniques used in computational geometry are: polygon triangulation, convex hulls, Voronoi diagrams, arrangements, geometric searching and motion planning.

Actually,there exist several software packages which are of general interest to the discrete and computational geometry community [9]. Majority of these softwares are distributed as source code written in C, Java or C++ [7, 2], but there are also available the packages supporting the algorithms of computational geometry.

However Computer Algebra Systems Maple and Mathematica only offer 2-dimensional convex hulls, the higher dimensional convex hulls and other basic tasks can be computed via free Maple package Convex [5]. Matlab uses Qhull [6] for their computational geometry functions: `convhulln, delaunayn, griddata3, voronoin`. Qhull computes very fast arbitrary-dimensional convex hull. It uses floating-point arithmetic with many parameters for tolerancing. There are

also available a number of free Matlab package for mesh generating, and linear optimization[8].

2 Data Representation

Geometric algorithms involve the manipulation of objects, which are not handled at the machine language level, so we must organize the complex objects by means of the simpler data types directly representable by the computer. The most common complex objects encountered in the design of geometric algorithms are sets and lists (ordered sequences). Data structures used in Computational Geometry are well describe in [3]. Let S be a set represented in a data structure and A is an arbitrary element.

```
> S:={A,B,C,D};
```

The fundamental operations occurring in set manipulation are adding A to the set S , removing A from S and test for membership in a set. These tasks are in Maple covered by functions:

```
>S:={op(S),A};
>S:=(subsop(i=NULL),S);
>member(A,S)
```

2.1 Data Structure

However, the nature of geometric problems has led to the development of specific data structures, for programming in Maple it is convenient to use simply lists of coordinates of vertices or simplified the doubly-connected edge lists (DCEL).

The most easiest way to represent edges and polygons is by using an lists or sets. All points are represented by lists of the appropriate number of coordinates. These representation are attractive for code clarity. For example, triangle is given by the list of vertices ordered in the counterclockwise manner. It is efficient to distinguish between the right and reverse side of the polygon. The structure of loops and index increments are somewhat clearer with lists than with arrays and standard quad-edge data structure.

```
> F[1]:=[[0,0,0],[0,1,0],[0,0,1]];
```

DCEL is suited to represent a connected planar graph embedded in the plane. Edge is given by its nodes and with information about incident edges and faces.

```
>A[1]:=[0,0]:A[2]:=[0,2]:A[3]:=[1,3]:A[4]:=[2,2]: ...
>e[1]:=[A[1],A[2],0,1,10,5]:e[2]:=[A[2],A[3],0,2,1,3]: ...
```

By using the for loops to iterate over the coordinates we can transform the DCEL to the simple ordered list of faces and draw the two-dimensional polygons `F[i]`.

```
> for i to nops(F) do
>  F[i]:={}:
>  for j to nops(e) do
>   if list_e[j][3]=i or list_e[j][4]=i then
     F[i]:={op(F[i]),list_e[j][1],list_e[j][2]}end if:
>  end do:
> end do:
> listF:=[seq(convert(F[i],list),i=1..5)];
>plots[polygonplot](listF);
```

2.2 Arithmetics

We will represent the coordinates with integer rather than with floating point numbers wherever possible.This will permit us to avoid the issue of floating-point round-off error and allow us to write code that is verifiably correct within a range of coordinate values. Maple performs the arithmetic computation in the floating-point environment. For arithmetic operations if one of the operands is a floating-point number then floating-point arithmetics takes place automatically. The global name `digits` which has 10 as its default, determines the number of digits in the significant which Maple uses when calculating with floating-point number.

3 Maple programming language

Writing a Maple program can be very simple. It may only involve putting a command `proc()` and `end proc` around a sequence of commands. We can write useful Maple programs in a few hours, rather then a few weeks that it often takes with other languages. This

efficiency is partly due to the fact that Maple is interactive. This interaction makes it easier to test and correct programs. Coding in Maple does not require expert programming skills. We can use special commands which allow us to perform complicated tasks with a single command instead of pages of code.

4 Example - Algorithms for 2-dimensional Convex Hull

A computing a convex hull is a vehicle for the solution of a number of unrelated questions in computational geometry, such as pattern recognition, image processing.

Many convex hull algorithms are known, it's behavior depends greatly on the specific combinatorial properties of the polytope on which it is working. However, there is currently no algorithm for computing the convex hull which is polynomial in the combined input and output size, unless the dimension is considered constant. Maple and Mathematica only offer 2-dimensional convex hulls. Higher dimensional convex hulls can be computed via the Maple package convex.

We will show the programm code in Maple with the simple algorithm - Graham's scan. The nature of Graham's scan algorithm is revealed by the following theorems: Consecutive vertices of a convex polygon occur in sorted angular order about any interior point (Figure. 1).

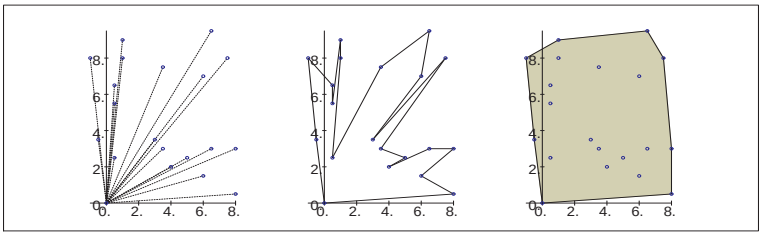

Figure 1: GrahamScan

First we find the rightmost lowest point and trivially transformed the coordinates of the others so that this point is at the origin.

```
>origin:=X[1]:
```

```
> for i from 2 to nops(X) do
>   if X[i][2]<=origin[2] then origin:=X[i]:end if:
> end do;
> for i from 1 to nops(X) do
>   Xnew[i]:=X[i]-origin:
> end do;
```

We now sort the n points by polar angle and the distance from the origin.

```
> angle:=(x,y)->evalb(x[1]*y[2]-y[1]*x[2]>0);
> Q:=sort([seq(Q,i=1..nops(Q))],angle);}
```

The essence of Graham algorithm is a single scan around the ordered points, during which the internal points are eliminated. Computation of the signed area of triangle given by vertices X, Y, Z.

```
> signArea:=proc(X::list,Y::list,Z::list)
> linalg[det](array(1..3,1..3,[[X[1],X[2],1],[Y[1],Y[2],1],
  [Z[1],Z[2],1]]));
> end:}\\
```

Suppose, that the list of points is ordered by polar angle. We repeatedly examine triples of consecutive points in counterclockwise order to determine whether or not they define a reflex angle.

```
> convhull:=proc(X::{list,set})
> local S,i,t,P;
> P:=trans(X);
> S[1]:=P[1];S[2]:=P[2];t:=2;i:=3;
> while i<=nops(P) do
>   if signArea(S[t-1],S[t],P[i])>0 then
>   t:=t+1; S[t]:=P[i]; i:=i+1;
>   else t:=t-1;
>   end if;
> end do:
> [seq(S[i],i=1..t)];
> end:}
```

5 Conclusion

Maple programming language is designed for the development of mathematical subroutines and custom applications. The syntax is similar to that of C, or Fortran. If you have used any of these languages, you can easily take advantage of the programming capabilities of Maple. Maple can generate code that is compatible with programming language C, so we could develop a mathematical model using Maple and then use Maple to generate C code corresponding to the model. It is possible to call routines written in C by using Maple's external calling facility.

References

[1] F. P. Preparata, M. Shamos: *Computational Geometry, an Introduction,* Springer-Verlag, New York, 1985

[2] J. O'Rourke: *Computational Geometry in C, second edition,* Cambridge University Press, 1998

[3] A.V. Aho, J.E. Hopcroft, J.D. Ullman: *Data Structures and Algorithms,* Addison-Wesley 1983

[4] J.E. Goodman, J. O'Rourke: *Handbook of Discrete and Computational Geometry,* Chapman and Hall/CRC, 2004

[5] M. Franz: Convex - a Maple package for convex geometry, version 1.1, 2004, available at `http://www-fourier.ujf-grenoble.fr/ franz/convex/`

[6] *Quickhull* algorithm for computing the convex hulls, Delaunay triangulations and Voronoi diagrams, 2004, available at `http://www.qhull.org`

[7] J.R. Schewchuk: *Triangle* C program for two-dimensionalmesh generation and construction of Delaunay triangulations, constrained Delaunay triangulations, and Voronoi diagrams, 2004, available at `http://www.cs.cmu.edu/ quake/triangle.html`

[8] S.A. Mitchell: Computational Geometry Triangulation, mesh generation in Matlab, available at `http://endo.sandia,gov/ samitch/csstuff/csguide.html`

[9] N. Amenta: Directory of Computational geometry Software, available at `http://www.geom.umn.edu/software/cglist/welcome.html`