Exact Geometrical Predicate: Point in circle

Release 1.00

Bertrand Moreau, Alexandre Gouaillard

November 29, 2011

Singapore Immunology Network, Agency for Science, Technology and Research (A*STAR), Biopolis, Singapore

Abstract

This document describes the implementation in ITK of the ”point in circle” geometrical predicate. Based on Jonathan Shewchuk’s work which implements an exact version of the predicate using standard floating point types and arithmetic [**?**], the implementation consist of an ITK wrapper around the public domain C routines made available by the author of the precedent paper. [**?**] Wrapper using itk::PointSet, itk:CellInterface and itk:Mesh / itk:QuadEdgeMesh APIs are provided along with corresponding examples which should provide enough details for users to directly copy paste code in their application.

The application in mind for us is an exact and robust implementation of a delaunay triangulation / voronoi tesselation in ITK, and will be presented in a separate paper.

Contents

1 Description

Considering a counterclockwise oriented circle represented by three points a, b and c in a 2D space, a fourth point d lies inside the circle if and only if:

$$det \begin{bmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{bmatrix} > 0$$

The orientation of the points a, b and c can be checked by considering the sign of:

$$det \begin{bmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{bmatrix}$$

If this determinant is positive, the points a, b and c are counterclockwise oriented; if negative, they are clockwise oriented; and if equal to zero, they are collinear.

Points a, b, c and d coordinates are exact precision numbers, as described in [**?**]. This test can be generalized to 3 or more dimensions.

For more details and illustrations, see here: `http://www.cs.cmu.edu/%7Equake/robust.html`

2 Implementation

The predicate.c file is the unmodified public domain predicate from Prof. shewchuck. Note that only exact versions of incircle and orient2D are wrapped, but the entire file was left unmodified.

itkPointInCircleGeometricalPredicateFunctor is the implementation of the corresponding feature as a functor in ITK API. IsInside() is the basic functor and is templated over the PointType. It supposes that the PointType is defined with at least dimension=2, and in the case of dimension > 2 will silently ignore the dimensions superior to 2. A cocnept chekc should be used here, but the author did not have the knowledge to implement it.

IsInsideNotExact() is a naive implementation of the predicate using VNL standard matrix computation. It is only provided for illustration purpose and should be removed if this code was to be included in ITK (or moved directly to the test/example part.

In real filters, it is very likely that th euser will work not directly from points, but from triangles, i.e. they will have an itk::Mesh or itk::QuadEdgeMesh at their disposal and will want to test a point against a triangle directly. Getting the triangle from the cell container, then getting the point Ids frm the triangle, getting the point from the point container, all those operations that are needed before one can use IsInside() have been encapsulated for convenience in the TestPointInTriangleInMesh(). This function still use the non exact, naive implementation of the predicate that should not remain in the final itk version.

3 Usage

IsPointInCIrcle.cxx illustrate different usages of the functors.

If you already have the three points defining your circle, and the point to test, you can use the basic functor directly. In the following code we define three points, a,b,c such that the point (0,0) lies on the circle. Then we are going to define a poitn D whose coordinates will be (0+epsilon x, 0+epsilon y) and check that, even for very small values of the epsilons, the result of the test is correct. Note that, even though the function is templated over the point type, it can deduce it from the arguments, and the user does not need to explicitly state the template parameters, making the usage of template transparent to the user.

```
PointType mpa, mpb, mpc, mpd;
mpa[0] = mpc[0] = mpc[1] = mpb[1] = 1.0;
mpa[1] = mpb[0] =                   0.0;
mpd[0] = eps_x;
mpd[1] = eps_y;

return(
  IsInside(
    mpa[0], mpa[1],
    mpb[0], mpb[1],
    mpc[0], mpc[1],
    mpd[0], mpd[1] )
  );
```

In some cases, you want to work with a mesh, and let's say (in a very naive approach of testing a 2D mesh) test one point against all the triangles of a planar mesh (which correspond to the tesselation of some domain) to find in which triangle the points resides. The second functor has been made for that. In this example we create a full mesh wich contains one triangle with the same points as above, and proceed with the same test. To use this code to test for all triangles, just loop against the CellContainer to get the cells ID, and iterates untill the test returns true, or untill you have exhausted all the cells IDs. You should note that the CellContainer by default is a Map, and should use iterator (as opposed to Ids). You should also make sure that all cells in the container are triangles before proceeding. Note that the functor use an extra boolean to use the exact or naive version of IsInside() internally. This should be removed from the code if selected for ITK inclusion. Note also, that only the MeshType needs to be passed as a template to the functor, the other template parameters can be deduced from the arguments.

```
// create an empty mesh
typedef typename MeshType::Pointer MeshPointerType;
MeshPointerType mesh = MeshType::New();

// add points
mesh->SetPoint( 0, mpa );
mesh->SetPoint( 1, mpb );
mesh->SetPoint( 2, mpc );

// add cell
CellAutoPointer dummyAbstractCell;
TriangleCellType * dummyCell = new TriangleCellType();
dummyAbstractCell.TakeOwnership( dummyCell ); // polymorphism
PointIdentifier dummyCellPoints[3] = {0,1,2};
```

```
dummyAbstractCell->SetPointIds( dummyCellPoints );
mesh->SetCell( 0, dummyAbstractCell ); // invalidate the cell

// now this code below should be close to what the user will do
// just loop against cell Container to get Ids
CellIdentifier myTriangle = 0;
bool result = TestPointInTriangleInMesh< MeshType >( mesh, myTriangle, mpd, boolean(True) );
```

References

[1] Jonathan Richard Shewchuk. http://www.cs.cmu.edu/~quake/robust.html. (document)

[2] Jonathan Richard Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997. (document), 1