A SIMPLE AND FAST HARDWARE-ACCELERATED POINT-IN-POLYGON TEST

F. Martínez, A. J. Rueda, F. R. Feito
Departamento de Informática, University of Jaén, Campus Las Lagunillas, Jaén, Spain

Keywords: Point-in-polygon test, hardware-accelerated algorithms.

Abstract: The new generations of GPUs bring us a set of new basic operations or tools that allow us to design new solutions to traditional problems in Computer Graphics. In this paper we present two approaches for the point-in-polygon problem based on the occlusion query extensions supported by modern GPUs. Both approaches are fast and their execution times do not depend on the number of edges of the polygon. Besides, one of the tests allows multiple point-in-polygon queries to be done in parallel with CPU computations.

1 INTRODUCTION

The 2D point-in-polygon test is a basic geometric operation in Computer Graphics, GIS and other areas. We can roughly classify point-in-polygon approaches into two categories. In the first one the algorithms work with the set of edges of the polygon, neither preprocessing nor additional data structure are needed. An example of these methods is the crossings test of Shimrat (Shimrat, 1962) as corrected by Hacker (Hacker, 1962). In the second category, a preprocessing of the polygon —normally a decomposition— is done, and an alternative data structure to the set of edges of the polygon is built. This alternative data structure implies additional storage requirements. However, the approaches into this category are very fast. Examples of this kind of algorithms are the grid method (Antonio, 1992) and the triangle fan methods, as the Spackman test (Spackman, 1993).

Traditionally, point-in-polygon test algorithms have been implemented in the CPU because it is a relatively simple and efficient operation. However, the new generation of GPUs allows us to design new solutions to the point-in-polygon problem. In (Hanrahan and Haeberli, 1990) graphics hardware is used to support picking, an operation quite similar to the point-in-polygon test.

The occlusion query mechanism of modern GPUs is used in occlusion culling algorithms (Akenine-Moller and Haines, 2002). These algorithms try to avoid drawing objects that are hidden by other objects in the scene. In this paper we show how this occlusion query mechanism can also be used to develop two original and efficient point-in-polygon tests. The main advantages of the tests are: they are easy to understand and they have a straightforward implementation, they work with any kind of polygon, they are fast and their execution times do not depend on the number of edges of the polygon. Besides, one of the tests allows multiple point-in-polygon queries to be done in parallel with CPU computations.

The remainder of the paper is structured as follows. Section 2 explains the picking algorithm based on graphics hardware mentioned in this introduction. Section 3 describes the first point-in-polygon test, which is based on the OpenGL's HP_occlusion_test extension. In Section 4 the second point-in-polygon test, which is based on the OpenGL's NV_occlusion_query extension, is described. In Section 5 we compare our point-in-polygon tests with other well known tests. Finally, Section 6 brings conclusions.

2 GRAPHICS HARDWARE BASED PICKING

In this section we describe a picking algorithm supported by graphics hardware developed by Hanrahan

Martínez F., J. Rueda A. and R. Feito F. (2007).
A SIMPLE AND FAST HARDWARE-ACCELERATED POINT-IN-POLYGON TEST.
In *Proceedings of the Second International Conference on Computer Graphics Theory and Applications - GM/R*, pages 161-165
DOI: 10.5220/0002072501610165
Copyright © SciTePress

and Haeberli (Hanrahan and Haeberli, 1990). At a "preprocessing" stage a scene is rendered in an off-screen buffer, with each polygon having a unique color that will be used as an identifier. When the user wants to pick an object clicks on a pixel, and the object can be efficiently determined looking up the pixel's color identifier in the off-screen buffer.

Obviously, the same idea can be used to solve the the point-in-polygon problem. The polygon can be drawn with a color *c* different from the background color in an off-screen buffer in 2D. We can say that point *p* falls inside the polygon if the color of its associated pixel in the off-screen buffer is *c*.

An OpenGL implementation of a point-in-polygon test that uses this strategy can be seen next.

```
bool inclusiontest (Point p)
{
  GLfloat pixels[1][1][1];
  GLint x = p.x * X_BUFFER_SIZE/X_IMAGE_SIZE;
  GLint y = p.y * Y_BUFFER_SIZE/Y_IMAGE_SIZE;
  glReadPixels(x, y, 1, 1, GL_RED, GL_FLOAT,
               pixels);
  return pixels[0][0][0] == 1;
}
```

At the preprocessing stage the polygon is drawn with RGB color (1,0,0), i.e. red, in an black off-screen buffer, i.e. with RGB color (0,0,0), using a 2D orthographic projection. To test whether point *p* is inside the polygon, the coordinates (*x*, *y*) of its associated pixel in the off-screen buffer are first computed. Then, the red component of pixel (*x*, *y*) is read from the off-screen buffer, if its value is 1 then *p* falls inside the polygon.

3 A POINT-IN-POLYGON TEST BASED ON THE HP_OCCLUSION_TEST EXTENSION

The OpenGL's HP_occlusion_test extension is intended for testing the visibility of an object, where "visible" means that at least one of its pixels passes the stencil and depth tests. An obvious application of this extension is occlusion culling. Before rendering an object in a scene, an occlusion test can be done with the object bounding volume, if the test fails the object does not need to be rendered and some performance can be gained.

This extension can also be used to develop a point-in-polygon test as follows. At a preprocessing stage the filled polygon is drawn in an off-screen buffer on a plane orthogonal to the observer's line of sight, using an orthographic projection —see Figure 1.a). To test whether a 2D point p = (x,y) is inside the polygon, an occlusion test with a 3D point p2 = (x,y,z) is done, where *z* is such that *p2* is positioned in a plane farther, from the observer's point of view, than the plane where the polygon was drawn. If point *p2* fails the occlusion test, then it is occluded by the polygon and it can be concluded that *p* falls into the polygon —see Figure 1.b). If *p2* passes the occlusion test, it is not occluded by the polygon, so *p* is outside the polygon —see Figure 1.c).

A possible implementation of the test can be seen next.

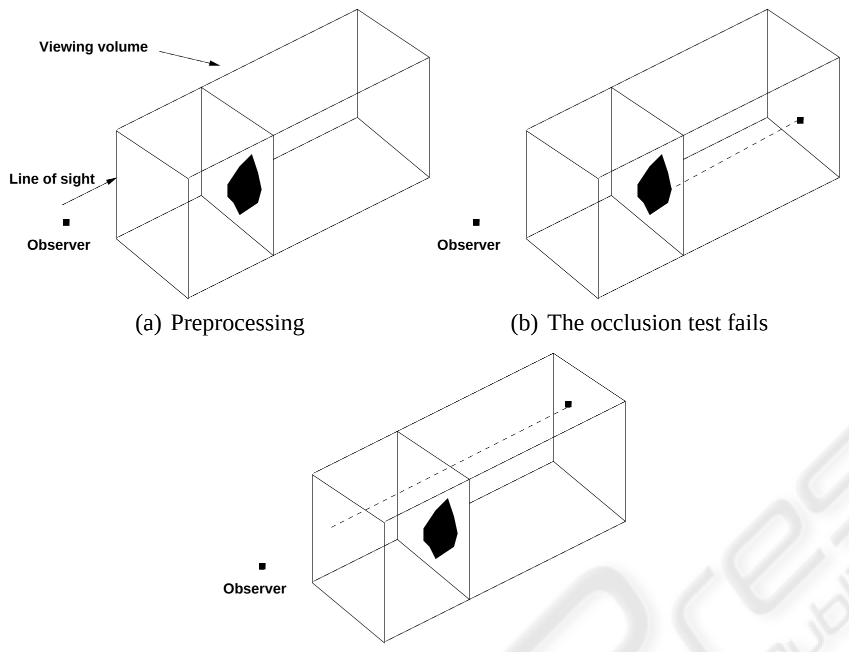
```
bool inclusiontest (Point p)
{
  GLboolean result;
  glEnable (GL_OCCLUSION_TEST_HP);
  glBegin (GL_POINTS);
    glVertex3f (p.x, p.y, -0.5);
  glEnd ();
  glDisable (GL_OCCLUSION_TEST_HP);
  glGetBooleanv (GL_OCCLUSION_TEST_RESULT_HP,
                 &result);
  return !result;
}
```

In this implementation the polygon is drawn in a P-buffer. In order to draw an arbitrary polygon we have considered two approaches: the first one is to use the OpenGL's tessellation algorithm, and the second one is based on using the stencil buffer (A. Rueda and Ruiz, 2004). We have chosen the latter because it is faster for complex polygons.

4 A POINT-IN-POLYGON TEST BASED ON THE NV_OCCLUSION_QUERY EXTENSION

The point-in-polygon test presented in this section is basically the same that the one presented in the previous section. The only difference is that this test uses the OpenGL's NV_occlusion_query extension to query whether the point is occluded by the polygon. However, this extension allows multiple point-in-polygon queries to be done in parallel with CPU computations.

The NV_occlusion_query extension provides an alternative occlusion query that overcome two limitations of the HP_occlusion_test extension. First, instead of returning a boolean value, it returns the number of pixels of the object that pass the stencil and depth tests. This is useful in occlusion culling because provides more information about the visibility



(a) Preprocessing (b) The occlusion test fails

(c) The occlusion test succeeds

Figure 1: Example of HP_occlusion_test.

of the object. For instance, an application could decide not to draw an object if less than a threshold number of pixels of its bounding volume pass the test, or to draw it with a low-detail model. However, our point-in-polygon test will not benefit from this feature because we only test the visibility of a point —which it is drawn with only one pixel—, so we get the same information with both occlusion queries.

The second improvement of this extension is that allows for issuing an occlusion query before asking for the result of a previous query, so that multiple occlusion queries, and therefore point-in-polygon tests, can be solved in parallel with CPU computation.

Next we show an implementation of a point-in-polygon test based on the NV_occlusion_query extension that allows multiple tests to be done in parallel with CPU computations.

```
vector<Point> points; // points to be tested
vector<bool> result;  // test results
GLuint pixelCount;
GLuint *occlusionQueries = (GLuint *)
         malloc(points.size ()*sizeof(GLuint));

// preprocessing (the polygon is drawn)
...
// occlusion tests start
glGenOcclusionQueriesNV (points.size (),
                         occlusionQueries);
for (int i = 0; i < points.size (); i++) {
  glBeginOcclusionQueryNV(occlusionQueries[i]);
  glBegin (GL_POINTS);
    glVertex3f (points[i].x, points[i].y, -0.5);
  glEnd ();
  glEndOcclusionQueryNV();
}
// CPU computation can be done here
...
// occlusion results are asked
for (int i = 0; i < points.size (); i++) {
  glGetOcclusionQueryuivNV(occlusionQueries[i],
              GL_PIXEL_COUNT_NV, &pixelCount);
  result[i] = pixelCount == 0;
}
```

In the first cycle the occlusion queries are issued, each query tests how many pixels of a query point pass the depth test. In the second cycle the results of the queries are processed, if no pixels of the point pass the test —really at most one pixel will pass— the point falls inside the polygon. It is important to note that between the first and second cycle code can be inserted that will execute in parallel with the occlusion queries.

5 COMPARISON OF THE TESTS

In this section we compare the tests based on graphics hardware explained in this paper with the following tests: 1) The crossings test, the most used method and the fastest algorithm without any preprocessing. 2) The spackman test, a representative sample of triangle fan methods. 3) The grid test, one of the fastest methods, which is based on a regular space decomposition.

To make the comparison we have used the implementation of Eric Haines (Haines, 1994) in Graphics Gems IV. The tests consist in 1000 consecutive point tests performed on 5 different random polygons, with sizes from 10 to 10000 edges. The target hardware consists of a Intel Pentium IV processor at 1.6 GHz, with 512 MB of RAM memory. The graphics card is an NVIDIA GeForce 6600. In the tests based on graphics hardware the size of the P-buffer where the polygon is drawn is 400x400. Table 1 shows the execution times of the different tests.

As it can be seen, the results obtained by the approaches based on the occlusion query mechanism are very good. The approach based on the NV_occlusion_query extension beats the crossings and Spackman methods for polygon from 100 edges.

Next, we describe the main advantages of the tests presented in the paper based on graphics hardware:

- Their execution times are almost constant, they do not depend on the number of vertices of the polygon.
- The tests, as well as their preprocessings, are very simple, easy to understand, and they have an straightforward implementation.
- They work with the GPU memory, saving RAM memory.
- As “alternative data structure” an off-screen buffer is used, whose size is constant, not depending on the size of the polygon.
- They work with any kind of polygon: concave, with holes, with intersecting edges, etc.

Furthermore, the test based on the NV_occlusion_query extension has the advantage that multiple queries can be done in parallel with CPU execution.

And next we describe the main drawbacks of these tests:

- As other methods they need preprocessing —to draw the polygon—. Therefore, they are mainly suitable when a high number of tests is needed. Table 2 compares the preprocessing times of the different tests.
- The precision of the tests depends on the resolution of the P-buffer, and it is lower than software based tests.

Finally, another drawback of the tests based on the occlusion query mechanism is that they cannot be extended to solve picking.

6 CONCLUSION

The occlusion query extensions of modern GPUs were originally intended for supporting occlusion culling algorithms. This paper has shown that these extensions can also be used to develop two original, efficient point-in-polygon tests. The tests, as well as their preprocessings, are easy to understand and they have a straightforward implementation. Besides, they work with any kind of polygon, they are fast and their execution times do not depend on the number of edges of the polygon. Finally, one of the tests allows multiple point-in-polygon queries to be done in parallel with CPU execution.

However, the tests have some drawbacks: their precision is lower than software based tests, they need preprocessing, and they cannot be extended to solve picking.

ACKNOWLEDGEMENTS

This work has been partially granted by the Ministerio de Ciencia y Tecnología of Spain and the European Union by means of the ERDF funds, under the research project TIN2004-06326-C03-03 and by the Conserjería de Innovación, Ciencia y Empresa of the Junta de Andalucía, under the research project P06-TIC-01403.

REFERENCES

A. Rueda, R. Segura, F. F. and Ruiz, J. (2004). Rasterizing complex polygons without tesselations. In *Graphical Models*. 66(3) 127-132.

Akenine-Moller, T. and Haines, E. (2002). *Real–Time Rendering*. A.K. Peters, Massachusetts, 2nd edition.

Antonio, F. (1992). *Faster line segment intersection*. David Kirk (Ed.) Graphics Gems III Academic Press, Boston, 1st edition.

Hacker, R. (1962). Certification of algorithm 112: position of point relative to polygon. In *Communications of the ACM*. Vol. 5 pp. 606.

Haines, E. (1994). *Point in polygon strategies*. Paul Heckert (Ed.) Graphics Gems IV Academic Press, New York, 1st edition.

Hanrahan, P. and Haeberli, P. (1990). Direct wysiwyg painting and texturing on 3d shapes. In *Computer Graphics (SIGGRAPH ’90 Proceedings)*. 24(4) 215-223.

Shimrat, M. (1962). Algorithm 112: position of point relative to polygon. In *Communications of the ACM*. Vol. 5 pp. 434.

Spackman, J. (1993). Simple, fast triangle intersection, part ii. In *Ray Tracing News*. 6(2).

Table 1: Execution times of the 1000 tests (in microseconds).

Algorithm	Number of edges				
	10	100	1000	5000	10000
Grid (400 cells)	0.12	0.2	0.7	2.49	4
Crossings	0.5	2.77	25.86	147.92	594.35
Spackman	0.27	2.83	26.55	243.35	613.9
NV_occlusion_query	2.62	2.54	2.56	2.58	2.56
HP_occlusion_test	10.3	10.4	10.44	10.2	10
Picking algorithm	26.86	26.87	26.83	26.94	26.6

Table 2: Execution times of the preprocessing (in milliseconds).

Algorithm	Number of edges				
	10	100	1000	5000	10000
Grid (400 cells)	0.13	0.4	2.64	10.37	21.74
Crossings	–	–	–	–	–
Spackman	0.02	0.08	0.2	0.84	1.89
graphics hardware tests	0.75	1.1	2.37	5.78	11.1