AN O(N) APPROXIMATION FOR THE DOUBLE BOUNDING BOX PROBLEM

Jörg Roth
Univ. of Applied Sciences Nuremberg
Kesslerplatz 12, 90489 Nuremberg, Germany
Joerg.Roth@Ohm-Hochschule.de

ABSTRACT

Minimal bounding rectangles are a simple and efficient tool for approximating geometries, particularly for accelerating spatial queries. If a spatial object fills a rectangular shape to a certain extent, then the minimal bounding rectangle is a reasonable approximation. Unfortunately some geo objects, such as streets or rivers have a small area but large bounding rectangles. In this paper we suggest an approximation with two bounding rectangles instead of a single one. Since the corresponding shape provides a better approximation, we get a greater average benefit. However, the computation of two bounding boxes with minimal area requires O(n·log n) steps for n geometry points. That may be a crucial point for geometries with large amounts of geometry points. In this paper, we introduce an approximation that requires O(n) steps but only produces approx. 11% more false hits compared to the theoretical optimum.

KEYWORDS

Spatial data, geometric approximation, bounding box

1. INTRODUCTION

Bounding boxes are a simple and efficient tool to speed up geometric or spatial operations. For two dimensions, a *bounding box* (often called *Minimal Bounding Rectangle*) is the smallest rectangle that encloses a geometry such as a polygon or line string (fig. 1a). Although bounding boxes may have more dimensions, two dimensions are sufficient for typical spatial queries for geo objects on the Earth's surface used by reverse geocoding, map painting or queries in current location-based services.

Bounding boxes provide only a rough approximation, but there exist efficient a priori checks for spatial properties. For example, if two geometries overlap, their bounding boxes overlap as well. Obviously, the other direction is not always true. Thus, we can use bounding boxes to check quickly whether or not some properties could *potentially* be true. If the bounding boxes pass the check, the corresponding exact geometry checks may fail, so bounding boxes can only reduce the set of candidates. Since an exact geometric test (e.g. if polygons overlap) may be time consuming, one goal is to have a small candidate set. Unfortunately for some spatial objects the bounding box area is very large compared to the original geometry. Line objects such as rivers or roads, in particular, may generate very large bounding boxes.

One way to reduce the candidate set is to reduce the surface area of the approximating construction. The better the approximation of a geometry, the lower is the number of false positive hits by a priori checks. To both retain the idea of bounding boxes and to simultaneously reduce the candidate set, we suggest *Double Bounding Boxes* (*DBBs*, fig. 1b). They consist of two simple bounding boxes, which may overlap. The area of the double bounding boxes fully encloses the geometry. To distinguish the single bounding box from the double bounding boxes described in this paper, we call the traditional box *Single Bounding Box* (*SBB*).

The greatest benefit have DBBs with a minimal total area. For the construction of such DBBs, there exists an algorithm which requires O(n·log n) steps for n geometry points (Roth, 2011). DBBs with minimal areas reduce the amount of false positive hits to 47.8% of the single bounding box false hits for typical geo data queries. In this paper we introduce an approximation called *Quick DBB*, that only requires O(n) steps without a significant drawback concerning false positive hits for realistic data.

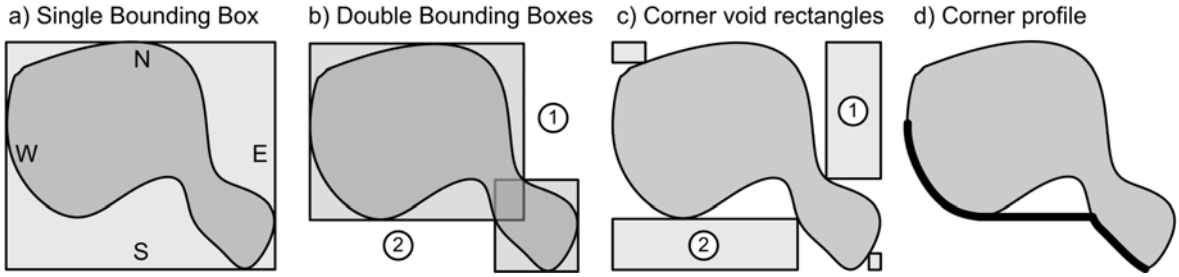

Fig. 1. Single and Double Bounding Boxes and construction ideas

2. RELATED WORK

Bounding boxes are used mainly for three purposes: 1. they can be used to speed up spatial comparisons; 2. they are used to create metadata of spatial objects; and 3. they may represent objects inside spatial indexes.

In addition to bounding boxes we have approximating shapes such as circles, ellipses or convex hulls (Gartner & Schonherr, 1997, Barequet & Har-Peled, 1999, Hill, 2006, Yang et al., 2008, Welzl, 1991). The main goal is to provide a quick a priori test for a spatial condition. If this a priori test is true then it makes sense to perform an additional check which is more exact but also more costly. If the a priori test was not true, then no further checks are needed.

A bounding box does not necessarily have to have aligned axes. If a box is rotated, it may be smaller and still enclose a geometry. This leads to the construction of *Oriented Bounding Boxes* (Yuan et al., 2006), often used in the area of game engines for collision detection or Optical Character Recognition (OCR). Such boxes can be computed using so-called *rotating calipers* (Toussaint, 1983). We did not follow that approach in this paper because it does not suit our intended usage.

To give an impression of how an aligned bounding box may speed up a test, consider a case where we have stored two polygonal geometries A and B. In addition to the polygon's vertices p_i, the bounding box is also stored, defined by two corners c_1, c_2 where $c_{1x}=\min(p_{ix})$, $c_{1y}=\min(p_{iy})$, $c_{2x}=\max(p_{ix})$, $c_{2y}=\max(p_{iy})$. We now could query whether or not A and B overlap, whether A is completely inside B (or vice versa), whether A and B only touch (only share a line or point), or whether A and B are disjoint. For each of these geometric properties there exists a bounding box test. To test whether the geometries overlap, for example, we check $overlap(SBB_A,SBB_B)=c_{A1x}\leq c_{B2x}$ AND $c_{B1x}\leq c_{A2x}$ AND $c_{A1y}\leq c_{B2y}$ AND $c_{B1y}\leq c_{A2y}$. Only if this expression is true does it make sense to perform the more complex check of whether the polygons overlap exactly. The drawback of this approach is the number of successful box checks where the exact test fails (false positives). This happens particularly when the actual geometry's area is small compared to the box area.

It is important to note that the a priori check for aligned bounding boxes (in contrast to oriented bounding boxes) can even be processed inside a standard database query. Thus, an approach to use non-spatial databases for spatial objects is to store the box parameters in table columns and the exact geometry as a binary large object. Only if the database passes the a priori check, and only then, the geometry is deserialised from the table and the exact check is performed (Roth, 2009).

Instead of a bounding box we could also use a bounding circle, defined by centre c_x, c_y and radius r. To compute an optimal centre (that leads to a minimal r) is not trivial (Elzinga, 1972) and requires a longer computation than a bounding box, but there exist approximations (Welzl, 1991, Hearn & Vijay, 1982). Once the bounding circle is computed, the overlap a priori test is $(c_{Ax}-c_{Bx})^2+(c_{Ay}-c_{By})^2\leq(r_A+r_B)^2$ which can also be processed inside a standard database query.

A third approximation shape is the convex hull, which is the minimal convex polygonal area containing the original geometry. A convex hull provides a better approximation than a rectangle or a circle. There exist algorithms with $O(n\cdot\log n)$ complexity to create convex hulls for a given geometry with n vertices, but there are also some drawbacks. First, the time to check geometric properties increases with the number of convex hull vertices whereas the time for rectangles and circles is always constant. Second, the number of convex hull vertices is limited only by the number of vertices of the given geometry and thus may by very high. Hastings presents an approach to split convex hulls into multiple convex hulls (called *multihulls*) to provide a

better approximation (Hastings, 2009). His main idea is to identify the vertex of the original geometry with the largest distance to the convex hull, which is then used to cut the convex hull into two parts. As a result, the union of hulls provides an increasingly accurate approximation of the original geometry, with the drawback that the time to check geometric properties tends to the time for exact checks.

The second purpose of bounding boxes is to have a short representation of a spatial object used, for example, in metadata records. There often exists a simple string representation of this box that can be used, for example, in web pages for URLs or sent by email to identify spatial objects. Some examples: The *Dublin Core Metadata Initiative* (*DCMI*) (Kunze & Baker, 2007) is an open organization that defines metadata standards. DCMI provides a box encoding scheme for spatial objects (Cox et al., 2006). The *vertical and horizontal spatial extents* (which are our bounding boxes) are part of the ISO 19115 Metadata Standard for geographic information (ISO 19115, 2003). The bounding box (called *envelope*) is part of the Geography Markup Language (GML) (Portele, 2007).

The last significant role of bounding boxes is to represent a geometry inside spatial indexes. A common approach to quickly access geometries is to first approximate the shape of an object by a bounding box that is inserted into a tree structure. Common spatial indexes are *Quadtrees* (Finkel & Bentley, 1974) or variations of R-Trees (Guttman, 1984). They mainly differ in how a tree is efficiently built and maintained when objects are inserted, changed or removed. A query goes down through the tree until an appropriate tree node is found. The corresponding bounding box then can be used to identify a (hopefully small) set of candidates that must then undergo further geometric checks.

Very often the shapes of spatial objects do not fill axis-aligned rectangles, thus there is a significant loss of shape information. In the following we suggest an approach that approximates arbitrary shapes with the help of *two* bounding boxes.

3. THE QUICK DBB IDEA

Using two bounding boxes first seems to be a disadvantage. E.g., for the a priori test if two geometries A, B overlap, a single test *overlap*(SBB_A,SBB_B) is required. Using DBBs we have *four* checks:
overlap(DBB_{A1},DBB_{B1}) OR *overlap*(DBB_{A2},DBB_{B1}) OR *overlap*(DBB_{A1},DBB_{B2}) OR *overlap*(DBB_{A2},DBB_{B2})
Compared to the exact checks the time for box checks can be neglected as they only require few simple number comparisons. We later show that they significantly reduce the number of candidates for exact checks. Before we present the quick algorithm to create DBBs we briefly discuss the theoretical optimum.

3.1 The exact algorithm

Fig. 1a shows a traditional SBB. In the following, north (N), south (S), east (E) and west (W) denote the four main directions. Our goal is to find *two* axis-aligned bounding boxes that enclose the geometry (fig. 1b). To compute a DBB, it is reasonable to look at the 'inverse' areas. So, we do not look at the covered area of a geometry, but rather at the empty area in the corners (e.g. the areas (1) and (2) in fig. 1b). For this, we introduce so-called *void rectangles* (fig. 1c): a void rectangle is aligned to the DBB's axes and one corner is an SBB's corner. In addition, it touches the geometry, but it does not overlap. DBB and void rectangles have aligned borders. For example, the north/west rectangle's east border is aligned to the north/east void rectangle's west border. If we respect some additional conditions, we thus can infer DBBs from the void rectangles.

Fig. 1d demonstrates the idea of generating a void rectangle. The north/east corners of all south/west void rectangles reside on a special line string we call the *corner profile*. To find appropriate void rectangles, we have to check corners that touch their corner profile. Note that even if corner profiles have a finite number of vertices, we get an infinite number of possible void rectangles, as they may touch the profile at an arbitrary position inside a line segment.

To get a complete list of DBB types, we must answer the following question: *Which layout of void rectangles allows the completion of the remaining area by exactly two more rectangles (our DBB)*? The systematic way to list all cases is to iterate through the possible numbers of void rectangles (fig. 2):

- 1 void rectangle: For any single void rectangle, two more rectangles can always be added to fill the entire space (case A), we have 4 sub-cases: the single void corner can be at NW, NE, SW, SE.

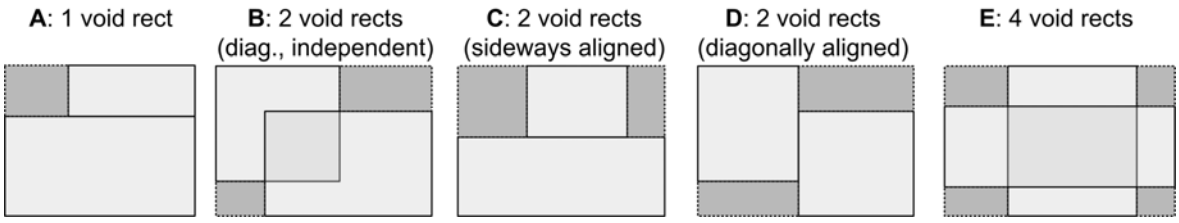

Fig. 2. Void rectangle cases

- 2 void rectangles: There exist three cases B, C, and D. Case B has 2 sub-cases: the void rectangles can be at NW/SE or NE/SW. Case C has 4 sub-cases: the common border can be at N, S, E, or W. Case D has 4 sub-cases: the void rectangles can be at NW/SE or NE/SW and they can share the border in direction NS or EW.
- 3 void rectangles: There is no way to generate DBBs from three given void rectangles.
- 4 void rectangles: The only chance to generate DBBs from 4 given void rectangles requires every pair of neighbor void rectangles to have the same height (east-west neighbors) or the same width (north-south neighbors). This is case E.

In summary, there exist 15 cases. The exact algorithm described in (Roth, 2011) iterates through these cases and computes the respective largest void rectangles. The overall maximum leads to the minimal DBB. Note that for cases C, D, and E, the void rectangles are aligned (e.g. have same heights), thus the algorithm must compute the maximum sum area for two or four aligned void rectangles.

The algorithm uses a plane sweep algorithm that sorts all line segments by their x-coordinate of the left vertex. Efficient sorting requires O(n·log n) steps. It requires additional O(n) steps to ensure monotony since the sorted line list can be iterated element by element again. In summary, it requires O(n·log n) steps to generate the four corner profiles. To compute the respective maximum void rectangles then can be performed in O(n) steps, thus the overall runtime complexity is O(n·log n).

3.2 The O(n) approximation

We can significantly reduce the amount of time to compute void rectangles if we introduce an additional constraint: for an SBB with a width to height ratio of s/t we only consider void rectangles with the same ratio of $w/h = s/t$. This means, the void rectangles inner corners reside on the SBB's diagonals (fig. 3a). We call a double bounding box based on these void rectangles the *Quick Double Bounding Box* (*QDBB*).

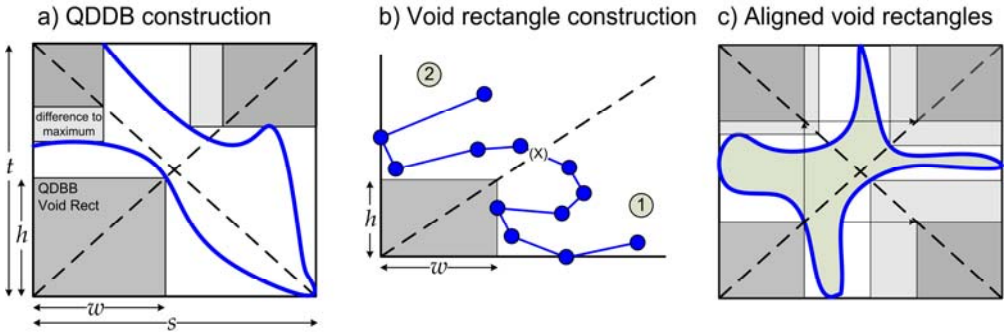

Fig. 3. Quick approximation idea

Obviously, the void rectangles usually are not maximal. E.g. two of the three void rectangles in fig. 3a) can be extended. We show later that for real geo data this difference to the optimum is acceptable.

The benefit is that we can compute void rectangles very quickly (fig. 3b). E.g., for the SW void rectangle we use the following approach: for each point (p_x, p_y) of a geometry do the following:

- if (p_x, p_y) resides in area (1) and $p_x<w$ assign $w = p_x$, $h = w{\cdot}t/s$;
- if (p_x, p_y) resides in area (2) and $p_y<h$ assign $h = p_y$, $w = h{\cdot}s/t$.

For each intersection $X=(X_x, X_y)$ of a geometry's line segment with the diagonal:

- if $X_x<w$ assign $w = X_x$, $h = X_y$.

For the other three void rectangles we perform respective steps. As a great benefit, these steps do not require any sorting of the original geometry points, thus can be performed in O(n) steps.

Fig. 3c) shows the case E of 4 aligned void rectangles. Even though three corners have their own maximum void rectangle, the smallest rectangle (here SW) determines all corners for case E. In this example, the optimal area is not achieved as the two east rectangles are too small. Again, for real data this usually is acceptable (see section 4).

3.3 Justification for the void rectangle ratio, worst case considerations

At first view, the width to height ratio of $w/h = s/t$ seems to be an arbitrary choice, as other ratios e.g. $w/h = 1$ could also be considered. The justification for the used ratio is illustrated in fig. 4.

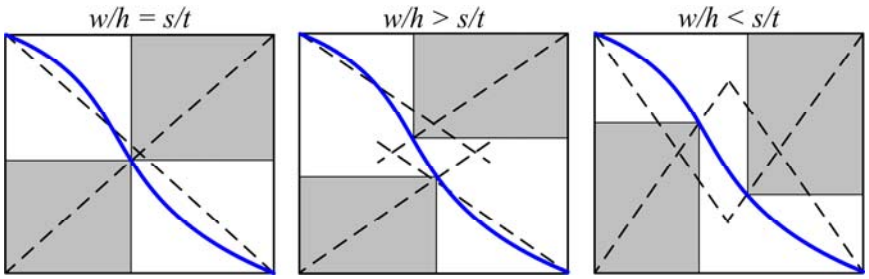

Fig. 4. Justification for the specific void rectangle ratio

The crucial cases are monotonic line geometries that often represent streets or rivers. For $w/h = s/t$ the two void rectangles share a corner (fig. 4, left). For this scenario the QDBB that is constructed according to case D is very close to the theoretical optimum. If we deviate from this ratio (fig. 4 middle or right), the void rectangles do not construct a DBB as the remaining area cannot be covered by *two* rectangles, but would require three. As a result, ratios of $w/h \neq s/t$ often produce void rectangles that cannot be represented by any DBB.

Before the quick algorithm is empirically compared to the exact algorithm we discuss the theoretical worst case. We consider case A to illustrate the worst result of the quick DBB algorithm (fig. 5).

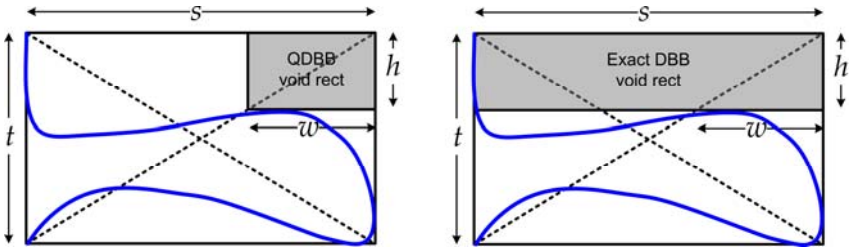

Fig. 5. Worst case scenario

The worst case occurs, if the largest void rectangle nearly covers the SBB's width, whereas the QDBB's void rectangle is limited by the diagonal corner. The QDBB's area then is

$$A_Q = s \cdot t - A_{VoidQ} = s \cdot t - h^2 \frac{s}{t} = s \cdot \left(t - h^2 / t\right)$$

whereas the exact DBB area is only

$$A_E = s \cdot \left(t - h\right).$$

The ratio between the QBB area and the optimum is thus

$$\frac{A_Q}{A_E} = \frac{s \cdot \left(t - h^2 / t\right)}{s \cdot \left(t - h\right)} = \frac{h + t}{t}.$$

For $h \to t$ the ratio tends to 2. This means, in worst case the QDBB's area has double the size of the optimum. As such, this result is disappointing, but

- the QDBB's size is smaller compared to the SBB's size $A_S = s \cdot t$; actually A_S/A_Q tends to ∞ for $h \to t$;
- such a scenario virtually never occurs in real data as it requires a very specific pattern.

In the following we provide an evaluation of how good the quick approximation is for real data.

4. EVALUATION

We evaluated our approach with the help of the OpenStreetMap database (Ramm & Topf, 2010). We used the file 'Germany' which contains approx. 4 mio. areas and line objects (file of Sept. 2010). The first step was to measure the creation time for quick DBBs compared to exact DBBs. The DBB algorithms run in Java (SE 1.6) on a XEON L5506 with 2.13 GHz (Windows Server 2008). Fig. 6 shows the results.

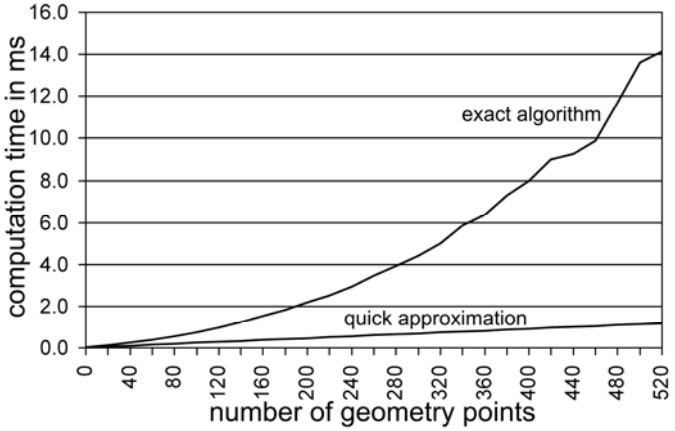

Fig. 6. Processing time to create DBBs

As expected, the measurements approve the runtime behavior of O(n) for the quick approximation and O($n \cdot \log n$) for the exact algorithm. Even though the exact algorithm runs in acceptable time, for geometries with large amounts of vertices (e.g. of city or district boundaries), the runtime benefit of the quick algorithm is significant. As a result, QDBBs are efficient even for large and fine-grained geometries.

We now show that the approximation does not lead to significant loss of information. For this, we checked the approach for typical spatial queries. Our test queries checked whether stored geo objects overlap with randomly generated geometries. We generated five different types of query geometries: lines with four points, lines with two points, triangles, rotated rectangles and circles. The circle diameters and line, triangle and rectangles sizes randomly ranged from 0 to 5000m. The center, size and orientation of the geometries were randomly generated using a uniformly distributed random number. Fig. 7 shows the results.

For each of the five geometry types we produced 100,000 random geometries. Each of them undergoes an overlap check against all geo objects stored in the database. For each geometry type we plot the average number of hits against the query geometry size

- using SBBs for query and geo object geometries,
- using QDBBs for query and geo object geometries,
- using exact DBBs for query and geo object geometries, and
- using an exact overlap check that compares the original geometries.

Line segments produce the lowest numbers of exact hits with a considerable amount of SBB and DBB hits. As expected, line segments usually have a poor box approximation, however (Q)DBBs show significantly better values than SBBs. For circles the difference between SBB, (Q)DBB and exact hits is small. This is because an SBB area is only factor $4/\pi \approx 1.27$ and a DBB is only $(4\sqrt{2}-2)/\pi \approx 1.16$ (case E shape) larger than the corresponding circle area. Thus boxes already provide a good approximation for circles.

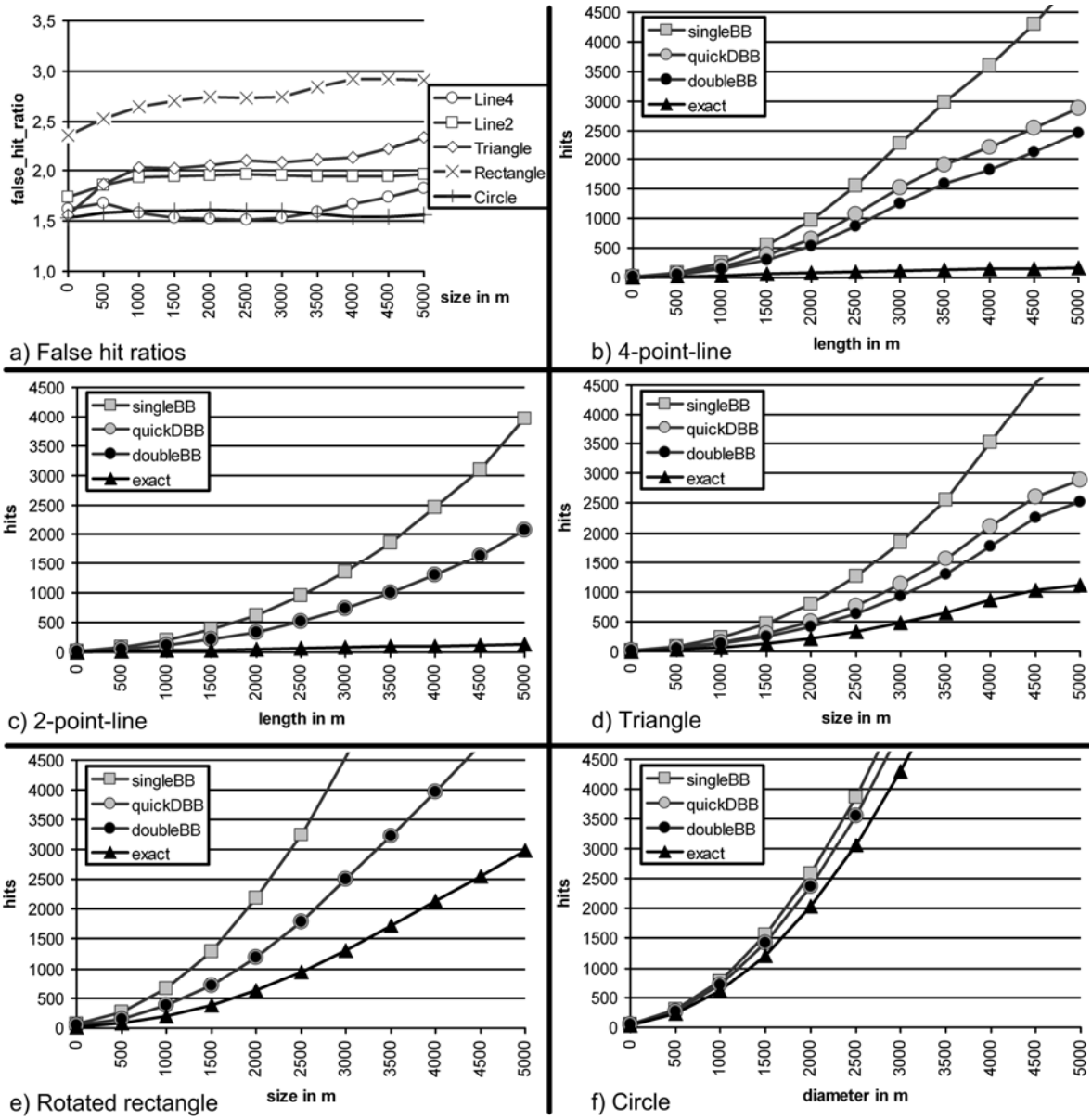

Fig. 7. Evaluation of approximation quality

To explore the overall benefit of QDBBs compared to SBBs we introduce the false hit ratio

$$false\_hit\_ratio = \frac{SBB\_hits - exact\_hits}{QDBB\_hits - exact\_hits}$$

that describes the amount of false SBB candidates compared the false QDBB candidates. A false hit ratio of, e.g., 2 means: we get twice as much false SBB candidates compared to QDBBs.

Fig. 7a) presents the false hit ratios for the five geometry types. The improvement ranges from approx. 1.5 to 2.9 (average 2.03). This is a great benefit, as QDBBs significantly reduced the number of false candidates compared to SBBs. We roughly get half as much false candidates.

As in important observation, exact and quick DBBs have very similar hit ratios. In the cases 2-point-lines, rectangles and circles, the plots are virtually identical. Only for 4-point-lines and triangles we can detect a small loss. In our experiments, QDBBs only produce 10.9% more false hits on average than the exact algorithm. As a major result, QDBBs can replace the exact DBBs in real scenarios without significant drawbacks.

5. CONCLUSION AND FUTURE WORK

In this paper we presented an approach to approximate two-dimensional geometries with the help of two bounding rectangles. In addition to the optimal algorithm that runs with complexity of $O(n \cdot \log n)$, we presented a quick approximation that requires $O(n)$ steps for n geometry points. For typical geo data the quick algorithm only produces 10.9% more false hits. Compared to traditional single bounding boxes, quick DBB produce only less than half the amount of false hits, thus the QDBB is a good candidate to replace the traditional bounding box as the major tool to approximate geometries for a priori geometric checks.

In the future we want to investigate the benefits of using more than two boxes for approximation with so-called *Multiple Bounding Boxes* (*MBBs*). A first approach reuses QDBBs to recursively segment a geometry. A second approach avoids recursion, but requires a quick segmentation algorithm that ideally also runs with $O(n)$ steps.

REFERENCES

Barequet, G. & Har-Peled, S. (1999). Efficiently Approximating the Minimum-Volume Bounding Box of a Point Set in 3D, Proc. 10th ACM-SIAM Sympos. Discrete Algorithms (1999), 82-91

Cox, S., Powell, A., Wilson, A. & Johnston, P. (2006). DCMI Box Encoding Scheme: specification of the spatial limits of a place, and methods for encoding this in a text string, http://dublincore.org

Elzinga, J. & Hearn D.W. (1972). The Minimum Covering Sphere Problem, Management Science, Vol. 19, 96–104 (1972)

Finkel, R. & Bentley J.L. (1974). Quad Trees: A Data Structure for Retrieval on Composite Keys. Acta Informatica 4 (1): 1974, 1–9

Gartner, B. & Schonherr. S. (1997). Smallest enclosing ellipses - fast and exact, Tech. Report B 97-03, Free Univ. Berlin, Germany (1997)

Guttman, A. (1984). R-Trees: A Dynamic Index Structure for Spatial Searching. Proc. of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, June 1984, 47-57

Hastings, J. T. (2009). Multihulls: A Technique for Spatial Representation and Processing with Variable Shape Fidelity, Applied to Gazetteers, Transactions in GIS, 2009, 13(5-6), 465-480

Hearn D.W. & Vijay J. (1982). Efficient Algorithms for the (Weighted) Minimum Circle Problem, Operations Research, Vol. 30, No. 4, 777–795 (1982)

Hill, L. L. (2006) Georeferencing: The Geographic Associations of Information MIT Press, Cambridge, MA

ISO 19115 (2003). Geographic Information - Metadata. International Organization for Standardization (ISO), 2003

Kunze, J. & Baker, T. (2007). The Dublin Core Metadata Element Set, Request for Comments 5013 (IETF), Aug. 2007

Portele C. (ed.) (2007). OpenGIS Geography Markup Language (GML) Encoding Standard, Open Geospatial Consortium, 2007

Ramm, F. & Topf, J. (2010). OpenStreetMap: Using and Enhancing the Free Map of the World, UIT Cambridge Ltd. 2010

Roth, J. (2009). The Extended Split Index to Efficiently Store and Retrieve Spatial Data With Standard Databases. IADIS International Conference Applied Computing 2009, Rome (Italy), Nov. 19-21, 2009, Vol. I, 85-92

Roth, J. (2011). The Approximation of Two-Dimensional Spatial Objects by Two Bounding Rectangles, Spatial Cognition and Computation: An Interdisciplinary Journal, in press

Toussaint, G. T. (1983). Solving geometric problems with the rotating calipers. Proceedings of IEEE MELECON'83, Athens, Greece, May 1983

Welzl, E. (1991). Smallest enclosing disks (balls and ellipsoids) in New Results and New Trends in Computer Science, Lecture Notes in Computer Science, Vol. 555 (1991), 359-370

Yang, M., Kpalma, K. & Ronsin, J. (2008). A Survey of Shape Feature Extraction Techniques. In Pattern Recognition Techniques, Technology and Applications, Peng-Yeng Yin (ed.) , Nov. 2008, I-Tech, Vienna, Austria, pp. 626

Yuan, B., Kwoh, L. K., Tan, C. L. (2006). Finding the Best-Fit Bounding Boxes, Springer LNCS 3872, 268-279