An Exact Algorithm for Finding Minimum Oriented Bounding Boxes

Jukka Jylänki*
2015/06/01

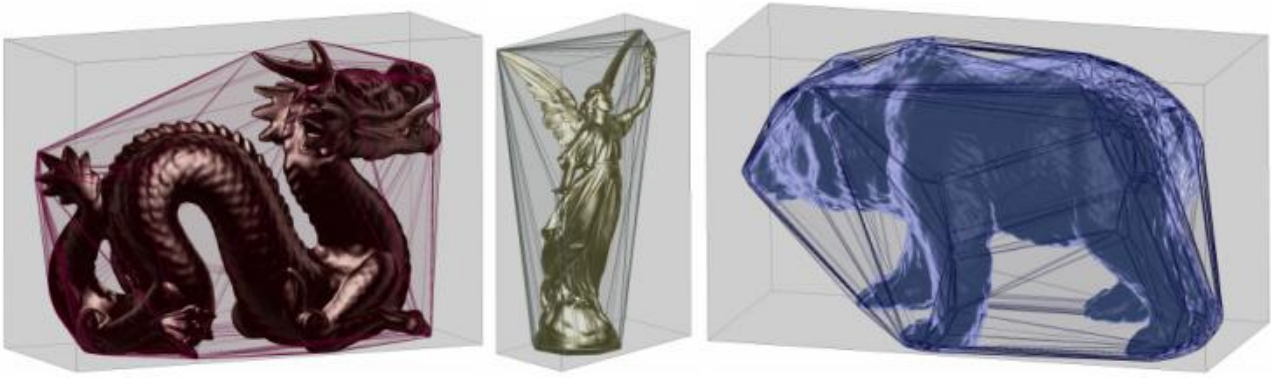

Figure 1: *Fast computation of the minimum volume OBB of a convex polyhedron. Left to right: Dragon: 2731149 vertices (2531 on the hull) in 1.3 seconds; Lucy: 14026670 vertices (5262 on the hull) in 5.6 seconds; a bear: 3105537 vertices (6628 on the hull) in 7.3 seconds. An optimized C++ implementation of the algorithm can be found here, or try it live on a web page here.*

Abstract

A new method is presented for computing tight-fitting enclosing bounding boxes for point sets in three dimensions. The algorithm is based on enumerating all box orientations that are uniquely determined by combinations of edges in the convex hull of the input point set. By using a graph search technique over the vertex graph of the hull, the iteration can be done quickly in expected $\mathcal{O}(n^{3/2}(\log n)^2)$ time for input point sets that have a uniform distribution of directions on their convex hull. Under very specific conditions, the algorithm runs in worst case $\mathcal{O}(n^3 \log n)$ complexity. Surprisingly, empirical evidence shows that this process always yields the globally minimum bounding box by volume, which leads to a conjecture that this method is in fact optimal.

CR Categories: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—Geometrical Problems and Computations; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Algorithms; Object representations

Keywords: computational geometry, bounding box, OBB

1 Introduction

Given a three-dimensional set of points, a fundamental problem in computational geometry is to find the smallest possible oriented bounding box (OBB) that contains all the points in the input data set. This problem arises widely in applications in computer graphics [Schneider and Eberly 2002], physical simulations [Ericson 2004] and spatial data structures [Gottschalk et al. 1996], as well as other areas, such as computer-aided design [Chan 2003] and even in medicine [Bartz et al. 2005]. Common uses in computer graphics include approximating a complex shape with a simpler OBB for the purposes of faster broad phase intersection testing, or for performing culling of renderable objects that need to be sent to the GPU.

OBBs have several desirable properties that make them useful for a variety of scenarios. They are convex and consist of three pairs of parallel planes, called slabs. In their local coordinate frame, they are representable as axis-aligned bounding boxes (AABB) or if scaled, as the unit cube. As such, intersection and distance tests between OBBs and other shapes can be implemented efficiently [Kay and Kajiya 1986] [Schneider and Eberly 2002, pp. 394, 624, 639]. OBBs also provide generally tighter fits than the simpler bounding spheres or AABBs. However, unlike spheres and AABBs, for which minimum volume enclosing shapes can be found in linear time [Megiddo 1982] [Welzl 1991], the major shortcoming with OBBs to date has been the difficulty of computing minimum volume OBB representations due to a lack of effective algorithms for the task.

In this paper, new analysis and progress is presented to help solve the minimum volume OBB problem. A new algorithm is introduced, which has considerable practical merit:

1. **Exact**: The algorithm runs through a discrete process without use of numerical optimization or iteration over the continuum. As a result, the method is stable and predictable.
2. **Simple**: Unlike previously published results, the new method is relatively easy to implement and it is possible to program it in different variants to balance a tradeoff between desired implementation complexity versus runtime complexity.
3. **Fast**: Both the time complexity and the constant time factors in the algorithm are low, which make it a practically viable solution.
4. **Enumerative**: The new algorithm is implementable as read-only sequential traversal over sets of edges, so it is categorizable as embarrassingly parallel and therefore suitable for processing large data sets as well.
5. **Optimal?** Based on extensive numerical searches over both synthetic and real world test cases, no example has yet sur-

*e-mail:jujjyl@gmail.com

faced where the new algorithm would not have found the optimal minimum volume box. However, at this point, no formal proof of the optimality of this algorithm is known.

The next section of this paper will first introduce the relevant earlier research. In sections 3 and 4, the main problem will then be gradually unwound in a bottom-up manner by presenting mathematical analysis of each of the individual subproblems one at a time. These results will all be brought together in the main algorithm finally in section 5. Last, sections 6 and 7 conclude with experimental results and practical remarks.

2 Related Work

In the past, only few papers have been published with theoretical advances on the minimum volume OBB problem. In the 2D case, it is known that the minimum area rectangle enclosing a convex polygon must be flush with one of the polygon edges [Freeman and Shapira 1975]. The **rotating calipers method** [Toussaint 1983] is a linear time algorithm that builds on this characterization. For the 3D case, the only currently known property is the following:

Theorem 2.1. *A box of minimal volume circumscribing a convex polyhedron must have at least two adjacent faces flush with edges of the polyhedron. [O'Rourke 1985]* ■

The term of being *flush* with an edge here means that a face of the box contains every point of that edge. Based on this property, O'Rourke developed a $\mathcal{O}(n^3)$ time algorithm which performs a type of rotating calipers search for each pair of edges on the hull. However, their method involves solving roots of a sixth degree trigonometric polynomial numerically, and in addition the complexity is so high that the method is often considered impractical. Unfortunately it also looks like this work has later become misinterpreted in literature in several published books, which in turn has led to misinformation in online sources [GameDev.Net], [StackOverflow], [Troutman]. To illustrate and clear up the confusion, consider the following quotes:

- "*O'Rourke (1985) shows that one box face must contain a polyhedron face and another box face must contain a polyhedron edge or three box faces must each contain a polyhedron edge.*"" [Schneider and Eberly 2002, p. 806]
- "*[...] given a polyhedron either one face and one edge or three edges of the polyhedron will be on different faces of its bounding box.*" [Ericson 2004, p.107]
- "*The paper [O'R85] shows that the minimum-volume OBB for a convex polyhedron must (1) have a face that is coincident with a face of the polyhedron or (2) be supported by three mutually perpendicular edges of the polyhedron.*" [Eberly 2006, p. 624] (see also [Geometric Tools LLC])

The first quote is possibly correct, however this does not follow from theorem 2.1 or the treatise in [O'Rourke 1985]. In fact, if it did, then it would directly prove the optimality of the new algorithm presented in this paper. The second claim on the other hand is not correct. Note the very subtle difference between the first two quotes that makes them different. It is possible that two adjacent faces of a box are flush with the *same* edge of a polyhedron, in the special case that this polyhedron edge coincides with an edge of the box. This counterexample will be called category **D**, and it will be presented in more detail later in section 5. See also figure 2. Finally, the third statement is likewise a misinterpretation of O'Rourke, and mathematically incorrect. The correct analysis will be presented in section 3.5.

Given that the problem has resisted attempts of solving for several decades, developers have sought to utilize a variety of numerical and statistical optimization methods to find suboptimal but good enough approximations for practical use. A basic method is to use principal component analysis (PCA) to estimate the direction of largest spread in the point set and establish that as a cardinal direction for the OBB. This process must be done using a continuous set representation on the convex hull, or otherwise the approximation can be unboundedly bad [Dimitrov et al. 2009]. For obtaining near-optimal results, there exists a $(1+\epsilon)$-approximation scheme [Barequet and Har-Peled 2001], whereas for a completely opposite line of approach one can embrace a carefully defined heuristic to trade optimality for speed [Larsson and Källberg 2011]. There have also been attempts at providing optimal results by using sophisticated numerical techniques, such as particle swarm optimization [Borckmans and Absil 2010] and genetic search [Chang et al. 2011]. These have been proven to be effective, but they have an element of randomness in them and may not find the optimal box in all cases.

The only brute force approach so far has been the exhaustive search of all orientations over SO(3, $\mathbb{R}$). This space is large, but there is a commonly known trick to prune it that seems to be folklore. Given a candidate orientation for the OBB, a better fit can be searched incrementally by repeatedly projecting the model onto a plane corresponding to one of the main axes of the current candidate OBB and solving the resulting 2D problem in linear time using Toussaint's rotating calipers method. This reduces the brute force challenge from SO(3, $\mathbb{R}$) to a search of a good starting direction vector in the unit (hemi)sphere, which, when iterated in this manner, would "lock" into the orientation of the optimal OBB. However the fundamental problem is still that the search is conducted over a continuous space and therefore it is not exhaustible by sampling a discrete set of orientations. Because of this, a brute force search over orientation angles is not able to guarantee an optimal result.

In summary, all known methods so far use either approximation, numerics or heuristics, which is unsatisfying. There is a clear need to develop a fast and exact geometric algorithm to solve the problem, which will be the main motivation in the following sections.

3 Mathematical Aspects

In order to work through the details, some notation and concepts need to be first introduced. Recall that the internal points of the input set do not play a role in the problem and one can restrict to considering only the convex hull of the point set. In fact, for the new algorithm, computing the convex hull is the required first step, since the algorithm is built to operate on the edges of a convex polyhedron. The convex hull can be computed in expected $\mathcal{O}(n \log n)$ time for example by using the quickhull algorithm [Barber et al. 1996].

For each direction vector n, one or more vertices of the polyhedron can be identified as the most extreme points to the direction of n. These are called **supporting vertices**, and will be denoted by $Supp(n)$. If for two vertices v_1 and v_2 there exists a direction n such that $v_1 \in Supp(n)$ and $v_2 \in Supp(-n)$, then v_1 and v_2 are said to be **antipodal**. Geometrically described, two vertices are antipodal if it is possible to find an orientation for an enclosing OBB such that the two vertices of the hull lie on opposing faces of the OBB.

In this paper, a new concept is introduced that is in some ways similar to antipodality. If for two vertices v_1 and v_2 there exists two directions n_1 and n_2 such that $v_1 \in Supp(n_1)$ and $v_2 \in Supp(n_2)$ and $n_1 \cdot n_2 = 0$, then v_1 and v_2 are said to be **sidepodal**. The geometric meaning of this is that two vertices are said to be sidepodal if it is possible to find an orientation for an enclosing OBB where

the OBB touches both vertices v_1 and v_2 on two adjacent faces of the OBB.

The concepts of support, antipodality and sidepodality are naturally extended to also refer to edges (and faces) of the hull. For example, one may refer to an edge e being sidepodal to a vertex v, if it is possible to orient an OBB to be flush with e and v on two adjacent faces of the OBB.

If x is a vertex or an edge of the hull, then the set of all antipodal vertices in the hull to x will be denoted by $Anti^{\mathbb{V}}(x)$ and the set of all antipodal edges in the hull to x will be denoted by $Anti^{\mathbb{E}}(x)$. Likewise, the sets $Side^{\mathbb{V}}(x)$ and $Side^{\mathbb{E}}(x)$ will refer to the sets of all sidepodal vertices and respectively edges to x. Also, it will be necessary to examine set intersections of these sidepodal sets, so as a notational aid, $Side^{\mathbb{V}}(e_1, e_2)$ will be used as a shorthand to refer to the set intersection $Side^{\mathbb{V}}(e_1) \cap Side^{\mathbb{V}}(e_2)$. It is immediately clear that if edge $e : v_0 \to v_1$ is an antipodal or a sidepodal edge to a feature x, then both vertices v_0 and v_1 of e immediately have the same property. More precisely:

- If $e \in Anti^{\mathbb{E}}(x)$, then $\{v_0, v_1\} \subseteq Anti^{\mathbb{V}}(x)$,
- if $e \in Side^{\mathbb{E}}(x)$, then $\{v_0, v_1\} \subseteq Side^{\mathbb{V}}(x)$, and
- if $e \in Side^{\mathbb{E}}(x_1, x_2)$, then $\{v_0, v_1\} \subseteq Side^{\mathbb{V}}(x_1, x_2)$.

This may seem trivial, but it is worth explicitly noting since this makes traversing sidepodal and antipodal edges effectively identical to traversing the sets of vertices. Note that for the sidepodality relation, the opposite is not true, i.e. even if two neighboring vertices are sidepodal to a feature x, it does not necessarily follow that the edge connecting them would be sidepodal to x as well.

The notation $N(v)$ will be used to refer to the set of vertex neighbors of vertex v. Given an edge e, the angle measured from inside of the hull between the two adjacent faces connected to e is commonly called the **dihedral angle** of edge e. Because the hull is convex, all dihedral angles are in the range of $]0, 180[$ degrees. Finally, the face normals of the two adjacent faces to e pointing **outwards** of the hull will be denoted by $f^<(e)$ and $f^>(e)$. These normal vectors define the following simple property that will be useful later.

Lemma 3.1. *If a face of an OBB is flush with an edge e of the convex hull, then the (unnormalized) normal vector n of that face of the OBB is equal to $n = f^<(e) + \big(f^>(e) - f^<(e)\big) * \boldsymbol{t}$ for some $\boldsymbol{t} \in [0, 1]$.*

Proof. This formula for the normal follows directly from linear interpolation. The value of $\boldsymbol{t} = 0$ corresponds to the case when the OBB is flush with the face with normal $f^<(e)$, and the value of $\boldsymbol{t} = 1$ corresponds to the OBB being flush with the face with the normal $f^>(e)$. Since the dihedral angle of the edge is never 0, the linear interpolation will not produce a degenerate zero vector. □

While it would be possible to choose a parametrization based on angle, or another that would preserve the magnitude of the normal vector, either would greatly complicate later analysis, which is why this presentation is preferred instead.

There exists an interesting geometric way to relate sidepodality of a vertex and an edge to the support function.

Lemma 3.2. *Given a vertex v and an edge e on a convex polyhedron, the vertex $v \in Side^{\mathbb{V}}(e)$ if and only if there exists a direction vector n for which $v \in Supp(n)$ and $\big(n \cdot f^<(e)\big)\big(n \cdot f^>(e)\big) \le 0$.*

Proof. By lemma 3.1, the normal vector n_2 of the face of an OBB that is flush with an edge e has the parametrization $n_2 = f^<(e) + \big(f^>(e) - f^<(e)\big) * t$. The vertex v is sidepodal to edge e if and only if there exists a direction n such that $v \in Supp(n)$ and $n \cdot n_2 = 0$. Substituting the formula of the normal gives the equation

$$n \cdot \Big(f^<(e) + \big(f^>(e) - f^<(e)\big) * \boldsymbol{t}\Big) = 0, \quad \text{where } \boldsymbol{t} \in [0, 1]\,. \quad (1)$$

The left-hand side is an expression linear to $\boldsymbol{t}$, so its extremes are at $\boldsymbol{t} = 0$ and $\boldsymbol{t} = 1$, and they correspond to values $n \cdot f^<(e)$ and $n \cdot f^>(e)$. Therefore according to Bolzano's theorem, equation (1) has a solution if and only if one of these values is nonnegative and the other is nonpositive. In other words, either

1. $n \cdot f^<(e) \le 0$ and $n \cdot f^>(e) \ge 0$, or
2. $n \cdot f^<(e) \ge 0$ and $n \cdot f^>(e) \le 0$.

Then, in either case multiplying the two expressions together yields the desired result. □

Antipodality and sidepodality both form symmetric nontransitive relations for vertices and edges of the hull. The new algorithm will be based on computationally resolving these relations on the vertex graph of the convex hull. Therefore the rest of this section will focus on the mathematical study of these properties which will enable the development of suitable algorithms to enumerate these sets.

3.1 Antipodal Edge and Vertex

Given an edge e and a vertex v of a convex polyhedron, it is possible to formulate a precise test whether e and v are antipodal partners to each other. This involves finding a common normal vector n for which $e \in Supp(n)$ and $v \in Supp(-n)$, or concluding that such a vector does not exist. The first condition is fulfilled if and only if the direction n satisfies the parametrization of lemma 3.1, whereas the second condition is met if and only if all vertex neighbors w of vertex v are in the negative halfspace of the plane defined by v and $-n$. This gives the following set of conditions:

$$\begin{cases} n = f^<(e) + \big(f^>(e) - f^<(e)\big) * \boldsymbol{t}\,, & \text{where } \boldsymbol{t} \in [0, 1]\,, \\ (v - w) \cdot n \le 0 \quad \forall\, w \in N(v)\,. \end{cases}$$

The set of equations defined on the second line generates a set of intervals, one for each neighbor in $N(v)$, that the parameter $\boldsymbol{t}$ must lie in for the solution to be valid. When codified, it turns into a test of whether the range intervals have a non-degenerate overlap. This procedure is shown in algorithm 1.

Algorithm 1: AreAntipodal(v,e)

```
let t_- = 0
let t_+ = 1.0
for w ∈ N(v) do
    let p = f^<(e) · (w − v)
    let q = (f^<(e) − f^>(e)) · (w − v)
    if q > 0 then
        t_+ = Min(t_+, p/q)
    else if q < 0 then
        t_- = Max(t_-, p/q)
    else if p < 0 then
        return false
return t_- ≤ t_+
```

3.2 Antipodal Pair of Edges

Testing whether given two edges e_1 and e_2 of a convex polyhedron are antipodal is also straightforward. Geometrically reasoning, it is immediate that if an OBB is flush with two edges e_1 and e_2 on opposite faces of the box, then the normal vector n_1 of one of these faces is perpendicular to both e_1 and e_2, and therefore the direction of n_1 is given by the cross product of the two edge vectors. That is, $n_1 = c * e_1 \times e_2$ for some constant c. This direction is uniquely defined, and to see whether the configuration is physically valid, it remains only to test whether a box laid out in this orientation intersects the inside of the polyhedron. A special case occurs when e_1 is parallel to e_2, in which case the cross product is degenerate and the box is free to "hinge" around in contact with the two edges. Testing whether intersection occurs can be achieved by using the parametrization for the normal vectors of the two edges from lemma 3.1, which yields the following group of equations:

$$\begin{cases} n_1 = f^<(e_1) + \big(f^>(e_1) - f^<(e_1)\big) * \boldsymbol{t}\,, & \text{where } \boldsymbol{t} \in [0,1]\,, \\ n_2 = f^<(e_2) + \big(f^>(e_2) - f^<(e_2)\big) * \boldsymbol{u}\,, & \text{where } \boldsymbol{u} \in [0,1]\,, \\ n_1 = \boldsymbol{c} n_2\,, & \text{where } \boldsymbol{c} < 0\,. \end{cases}$$

This leads to a matrix equation of form $Mx = f^<(e_1)$, where $x = \begin{pmatrix} \boldsymbol{t} & \boldsymbol{c} & \boldsymbol{cu} \end{pmatrix}^T$ and M is represented as column vectors in the form

$$M = \begin{pmatrix} f^<(e_1) - f^>(e_1) & f^<(e_2) & f^>(e_2) - f^<(e_2) \end{pmatrix} \quad (2)$$

From here, the valid values for $\boldsymbol{t}$ and $\boldsymbol{u}$ are solved easily by generic 3x3 matrix techniques, and they directly define one of the face normal directions for the OBB. This method is presented in algorithm 2, which tests whether two edges are antipodal, and if so, returns the normal direction for the face of the OBB that is flush with edge e_1.

Algorithm 2: AntipodalDir(e_1, e_2)

compute M according to equation (2)
let $(\boldsymbol{t}, \boldsymbol{c}, \boldsymbol{cu})^T = M^{-1} f^<(e_1)$
if $\boldsymbol{c} < 0$ *and* $\boldsymbol{t} \in [0,1]$ *and* $\boldsymbol{u} \in [0,1]$ **then**
  let $n = f^<(e_1) + \big(f^>(e_1) - f^<(e_1)\big) * \boldsymbol{t}$
  return $n/||n||$
else
  return null

3.3 Sidepodal Pair of Edges

In a similar fashion, it can be analysed whether a pair of edges are sidepodal partners to each other. If two edges e_1 and e_2 are sidepodal, then the normal vectors defined on the two edges by lemma 3.1 must be perpendicular, which gives the following set of equations:

$$\begin{cases} n_1 = f^<(e_1) + \big(f^>(e_1) - f^<(e_1)\big) * \boldsymbol{t}\,, & \text{where } \boldsymbol{t} \in [0,1]\,, \\ n_2 = f^<(e_2) + \big(f^>(e_2) - f^<(e_2)\big) * \boldsymbol{u}\,, & \text{where } \boldsymbol{u} \in [0,1]\,, \\ n_1 \cdot n_2 = 0\,. \end{cases}$$

Solving this set of equations leads to a new equation of the form

$$a + b\boldsymbol{t} + c\boldsymbol{u} + d\boldsymbol{tu} = 0, \quad \text{where } \boldsymbol{t}, \boldsymbol{u} \in [0,1]\,, \quad (3)$$

where a, b, c and d are all scalar constants. Since the left-hand side is a bilinear, closed and continuous expression of the parameters $\boldsymbol{t}$ and $\boldsymbol{u}$, it attains its minimum and maximum values at one of the four extreme points of its domain, i.e. one of the four corners of the unit square. This leads to a quick test that samples all these four corner points of potential extrema and checks the signs of the result. If the signs are different, then again due to Bolzano's theorem, a solution must exist to equation 3. A programmatic way to test this is shown in algorithm 3.

Algorithm 3: AreSidepodal(e_1, e_2)

let $a = f^>(e_1) \cdot f^>(e_2)$
let $b = \big(f^<(e_1) - f^>(e_1)\big) \cdot f^>(e_2)$
let $c = \big(f^<(e_2) - f^>(e_2)\big) \cdot f^>(e_1)$
let $d = \big(f^<(e_1) - f^>(e_1)\big) \cdot \big(f^<(e_2) - f^>(e_2)\big)$
let $v_- = Min(a, a+b, a+c, a+b+c+d)$
let $v_+ = Max(a, a+b, a+c, a+b+c+d)$
return $v_- \leq 0$ and $v_+ \geq 0$

3.4 Sidepodal Edge and Vertex

If a vertex v is located sidepodal to an edge, then lemma 3.2 says that there must exist a normal n that satisfies the following conditions.

$$\begin{cases} \big(n \cdot f^<(e)\big)\big(n \cdot f^>(e)\big) \leq 0\,, \\ v \in Supp(n)\,. \end{cases}$$

The first inequality holds if and only if one of the factors is nonnegative, and the other is nonpositive, and the second one holds if and only if for each vertex neighbor $w \in N(v)\,, n \cdot (w - v) \leq 0$. Therefore the normal vector n satisfies either

$$\begin{cases} n \cdot f^<(e) \leq 0\,, \\ n \cdot f^>(e) \geq 0\,, \\ n \cdot (w - v) \leq 0 \quad \forall\, w \in N(v)\,, \end{cases} \quad (4)$$

or

$$\begin{cases} n \cdot f^<(e) \geq 0\,, \\ n \cdot f^>(e) \leq 0\,, \\ n \cdot (w - v) \leq 0 \quad \forall\, w \in N(v)\,. \end{cases} \quad (5)$$

The existence of a solution can be checked by proceeding with a Gaussian elimination type of approach, performing elementary row operations to reduce the inequalities to a pivoted form. However when doing this, two rows may only be added together when they have directions for their inequalities match, so the final pivotized form of conditions in (4) and (5) will both form a set of 3 to 6 inequalities. If neither of these sets of inequalities form a contradiction by forcing n to a zero vector, then the vertex v and edge e are sidepodal.

There is also another way to determine if $v \in Side^{\mathbb{V}}(e)$. This is based on the geometric observation that if a vertex is sidepodal to an edge, then starting from a box configuration that is flush with v and e on adjacent faces, it is always possible to spin the box around the normal of the box face that is flush with e until the face of the box that is flush with v meets with a second vertex $w \in N(v)$. This property can be stated in the following form.

Proposition 3.3. *Given a vertex v and an edge e of a convex polyhedron, $v \in Side^{\mathbb{V}}(e)$ if and only if there exists a vertex $w \in N(v)$ such that $(e_2 : v \to w) \in Side^{\mathbb{E}}(e)$.* ■

Exploiting this property seemed to be more convenient in practice rather than solving inequalities (4) and (5), however it is good to know that both options exist.

3.5 Basis from Three Edges

Fixing an OBB to be flush with only two edges on adjacent faces of the OBB does not yet uniquely define its orientation, but still leaves one degree of freedom. This can be seen clearly for example in the earlier analysis by [O'Rourke 1985]. The question then arises, if one is given three edges e_1, e_2 and e_3 of a polyhedron, is it possible to orient an OBB to be flush with these edges, so that each of them lies on separate, but mutually adjacent faces of the OBB, and if so, what should the orientation (face normals n_1, n_2 and n_3) of the OBB be? A hasty analysis might conclude that edges e_i would be in fact identical to normals n_i, and that there would be no solution if e_i were not mutually perpendicular. However this is not correct. Alternatively, one might think that the normals n_i are computable from edges e_i by some kind of sequence of cross products, but this is not true either.

To provide the correct analysis, one proceeds with setting up a group of equations like shown in previous sections. The three edges are all mutually sidepodal, which yields the following:

$$\begin{cases} n_1 = f^<(e_1) + \big(f^>(e_1) - f^<(e_1)\big) * \boldsymbol{t}\,, & \boldsymbol{t} \in [0,1]\,, \\ n_2 = f^<(e_2) + \big(f^>(e_2) - f^<(e_2)\big) * \boldsymbol{u}\,, & \boldsymbol{u} \in [0,1]\,, \\ n_3 = f^<(e_3) + \big(f^>(e_3) - f^<(e_3)\big) * \boldsymbol{v}\,, & \boldsymbol{v} \in [0,1]\,, \\ n_1 \cdot n_2 = 0\,, \\ n_2 \cdot n_3 = 0\,, \\ n_1 \cdot n_3 = 0\,. \end{cases} \tag{6}$$

Solving this is more complicated than before, but still doable. The first three conditions of equation set (6) may be renamed to the form

$$\begin{cases} n_1 = a + b\boldsymbol{t}\,, & \text{where } \boldsymbol{t} \in [0,1]\,, \\ n_2 = c + d\boldsymbol{u}\,, & \text{where } \boldsymbol{u} \in [0,1]\,, \\ n_3 = e + f\boldsymbol{v}\,, & \text{where } \boldsymbol{v} \in [0,1]\,, \end{cases} \tag{7}$$

for obvious substitutions of vector constants a, b, c, d, e and f. Expanding the three dot product conditions of equation set (6) with variables from equations (7) then yields

$$\begin{cases} a \cdot c + b \cdot c\boldsymbol{t} + \boldsymbol{u}(a \cdot d + b \cdot d\boldsymbol{t}) & = 0\,, \quad (8) \\ c \cdot e + c \cdot f\boldsymbol{v} + \boldsymbol{u}(d \cdot e + d \cdot f\boldsymbol{v}) & = 0\,, \quad (9) \\ a \cdot e + a \cdot f\boldsymbol{v} + \boldsymbol{t}(b \cdot e + b \cdot f\boldsymbol{v}) & = 0\,. \quad (10) \end{cases}$$

Multiplying equation (8) by $(d \cdot e + d \cdot f\boldsymbol{v})$ and equation (9) by $-(a \cdot d + b \cdot d\boldsymbol{t})$ and adding the resulting equations together gives

$$g + h\boldsymbol{v} + \boldsymbol{t}(i + j\boldsymbol{v}) = 0\,, \tag{11}$$

where

$$\begin{cases} g & := (a \cdot c)(d \cdot e) - (a \cdot d)(c \cdot e)\,, \\ h & := (a \cdot c)(d \cdot f) - (a \cdot d)(c \cdot f)\,, \\ i & := (b \cdot c)(d \cdot e) - (b \cdot d)(c \cdot e)\,, \\ j & := (b \cdot c)(d \cdot f) - (b \cdot d)(c \cdot f)\,, \end{cases} \tag{12}$$

and then multiplying equation (10) by $-(i + j\boldsymbol{v})$ and equation (11) by $(b \cdot e + b \cdot f\boldsymbol{v})$ and adding the resulting equations together yields

$$k\boldsymbol{v}^2 + l\boldsymbol{v} + m = 0\,, \tag{13}$$

where

$$\begin{cases} k & := h(b \cdot f) - j(a \cdot f)\,, \\ l & := h(b \cdot e) + g(b \cdot f) \\ & \quad - i(a \cdot f) - j(a \cdot e)\,, \\ m & := g(b \cdot e) - i(a \cdot e)\,. \end{cases} \tag{14}$$

From here on, the two possible values for the parameter $\boldsymbol{v}$ can be readily solved by applying a standard formula for roots of a second degree polynomial to equation (13), and the values for $\boldsymbol{t}$ and $\boldsymbol{u}$ can then be backtracked by using equations (9) and (10). If the second degree polynomial in equation (13) has no real roots or if the parameters $\boldsymbol{t}, \boldsymbol{u}$ or $\boldsymbol{v}$ are not in $[0, 1]$ range, then an orientation does not exist. As a special note, it is important to test that the final solution for $\boldsymbol{t}, \boldsymbol{u}$ and $\boldsymbol{v}$ satisfies the original set of equations (6) again, since equations (8) and (9) imply (11) only one way. That is, a solution to equation (11) might not be a solution to equations (9) and (10). Similarly, the equations (10) and (11) only imply (13) one way as well.

In conclusion, this derivation provides an exact test for determining whether given three edges of the convex hull can accommodate an OBB orientation where the three edges are all on mutually adjacent faces of the OBB, and if so, provides the possible coordinate frames (n_1, n_2, n_3) for the orientation. Algorithm 4 shows a programmatic example. Note that this function will return a set of zero to two solutions. The main algorithm will then loop over each of these solutions in turn and process each one as a separate candidate configuration.

Algorithm 4: ComputeBasis(e_1, e_2, e_3)

```
compute a - m from equations (7), (12) and (14).
if polynomial k v^2 + l v + m has no roots then
    return {}
let v_1, v_2 be the roots of k v^2 + l v + m
let O = ∅
for v ∈ {v_1, v_2} do
    let t = (g + h v)/(i + j v)
    let u = (c · e + c · f v)/(d · e + d · f v)
    let n_1 = a + b t
    let n_2 = c + d u
    let n_3 = e + f v
    if t, u, v ∈ [0, 1] and n_1 ⊥ n_2 and n_1 ⊥ n_3 and n_2 ⊥ n_3 then
        O = O ∪ {(n_1/||n_1||, n_2/||n_2||, n_3/||n_3||)}
return O
```

3.6 Basis from a Direction and an Edge

The last subroutine that will be needed is the following. Let n_1 be a predetermined normal direction for one of the faces of the OBB. Then, given an edge e, is it possible to complete the remaining orientation of the OBB (compute n_2 and n_3) such that edge e is flush with a face of the resulting OBB? The task is to identify if this is possible, and if so, complete n_1 to an orthonormal basis (n_1, n_2, n_3) where, say, n_2 is perpendicular to e. Note that like in the case of the three edges from the previous chapter, the correct answer is not the cross product $n_2 := n_1 \times e$, since if e is parallel to n_1, the cross product will come out zero. Instead, lemma 3.1 can be used again, which gives the following two conditions.

$$\begin{cases} n_2 = f^<(e) + \big(f^>(e) - f^<(e)\big) * \boldsymbol{u}\,, & \text{where } \boldsymbol{u} \in [0,1]\,, \\ n_1 \cdot n_2 = 0\,. \end{cases}$$

Combining these two gives

$$n_1 \cdot f^<(e) + n_1 \cdot \big(f^>(e) - f^<(e)\big) * \boldsymbol{u} = 0\,,$$

from where the solution is

$$\boldsymbol{u} = \big(n_1 \cdot f^<(e)\big) / \big(n_1 \cdot (f^<(e) - f^>(e))\big)$$

when the denominator is not zero, and when it is, any value of $\boldsymbol{u}$ will satisfy the equation if and only if $n_1 \cdot f^<(e) = 0$ as well. The

Algorithm 5: CompleteBasis(n_1, e)

let $p = n_1 \cdot f^<(e)$
let $q = n_1 \cdot \big(f^<(e) - f^>(e)\big)$
if $q \neq 0$ **then**
  let $\boldsymbol{u} = p/q$
  let $n_2 = f^<(e) + \big(f^>(e) - f^<(e)\big) * \boldsymbol{u}$
  $n_2 = n_2/||n_2||$
  let $n_3 = n_1 \times n_2$
  return $\big\{(n_1, n_2, n_3)\big\}$
else if $p = 0$ **then**
  let $n_2 = f^<(e)$
  let $n_3 = n_1 \times n_2$
  return $\big\{(n_1, n_2, n_3)\big\}$
else
  return $\emptyset$

remaining face normal n_3 is then solvable via a cross product. This process is shown in algorithm 5.

The equations and algorithms that were presented in this section provide the tools for identifying antipodal and sidepodal companions for edges, and allows construction of candidate orientations for an enclosing OBB once a set of candidate edges have been chosen. The enumeration of these edges will follow a graph search approach, but in order to make it feasible, these sets must first be analyzed in some spatial detail. The next section will focus on that task, which will then enable building a search strategy for the algorithm overall.

4 Analysis of Antipodal and Sidepodal Sets

For each convex polyhedron C, one can define the **Gaussian sphere** representation of C, denoted by $G(C)$. This representation is a type of a dual representation of C, formed by a decomposition of the unit sphere into disjoint regions, one for each vertex of C. A point n in the Gaussian sphere belongs to the region of vertex v of C if $v \in Supp(n)$. This has the effect that vertices of C are mapped to faces of $G(C)$, and faces of C are mapped to single vertices on $G(C)$. An edge e on C is mapped to an arc on a great circle of $G(C)$ that is perpendicular in direction to e. Each face f on $G(C)$ is convex, because if $v \in Supp(n_1)$ and $v \in Supp(n_2)$, then $v \in Supp\big(\boldsymbol{t}n_1 + (1-\boldsymbol{t})n_2\big)$ as well for $\boldsymbol{t} \in [0, 1]$.

Using the Gaussian sphere representation, the following can be said.

Theorem 4.1. *Given an edge e of C, the set $Anti^{\mathbb{V}}(e)$ forms a single connected component in the vertex neighbor graph of C.*

Proof. The viable normal directions n for an OBB flush with an edge e are explicitly parametrized by lemma 3.1, thus

$$Anti^{\mathbb{V}}(e) = \left\{ Supp\Big(-f^<(e) - \big(f^>(e) - f^<(e)\big) * \boldsymbol{t}\Big) : \boldsymbol{t} \in [0,1] \right\}.$$

On the Gaussian sphere $G(C)$, this corresponds to a closed and continuous arc. Since this arc forms a single connected subset of points on the Gaussian sphere, the set of vertices on C whose convex face regions on $G(C)$ the arc overlaps with is also connected. □

Thinking about the set $Anti^{\mathbb{V}}(e)$ being defined by an arc on the Gaussian sphere is also very useful for another reason. The length of the arc is directly defined by the dihedral angle of edge e, and the closer the angle is to 180 degrees, the smaller the arc becomes, meaning also that the smaller the set $Anti^{\mathbb{V}}(e)$ corresponding to edge e will be in general.

Likewise, the sets $Side^{\mathbb{V}}(e)$ and $Side^{\mathbb{E}}(e)$ have the same property.

Theorem 4.2. *Given an edge e of C, the sets $Side^{\mathbb{V}}(e)$ and $Side^{\mathbb{E}}(e)$ both form a single connected component in the vertex neighbor graph of C.*

Proof. By lemma 3.2, the set of antipodal vertices to edge e is defined in terms of the support function in the form

$$Side^{\mathbb{V}}(e) = \left\{ Supp(n) : \big(f^<(e) \cdot n\big)\big(f^>(e) \cdot n\big) \leq 0 \right\}.$$

The set of normal vectors satisfying this inequality condition for n forms a single closed and connected subset of $G(C)$, so therefore the set of vertices on C whose convex face regions on $G(C)$ this subset corresponds with is also connected. Adding the geometric observation from proposition 3.3, it follows that the set $Side^{\mathbb{E}}(e)$ is also connected. □

The reason that connectedness for these sets is important is that with this property, if one first obtains any element v_0 in, say, $Anti^{\mathbb{V}}(e)$, then the rest of the elements in that set can be enumerated quickly in $\mathcal{O}\big(|Anti^{\mathbb{V}}(e)|\big)$ time by performing a graph search starting from the neighborhood of v_0, which will be much faster than having to resort to a full $\mathcal{O}\big(|V|\big)$ search over all vertices of the convex polyhedron. However, this property is only true if the number of neighbors for each vertex of the polyhedron is bounded by a constant. This condition will be implicitly assumed in all the analysis that follows, but one should keep this technicality in mind.

Finally, a similar result exists for the structure of set intersections of pairs of sidepodal vertex sets.

Theorem 4.3. *Given two edges e_1 and e_2 of C, the set $Side^{\mathbb{V}}(e_1, e_2)$ forms at most two separate connected components in the vertex neighbor graph of C.*

Proof. Expanding on the proof of lemma 3.2, the boundary of $Side^{\mathbb{V}}(e_1)$ is defined by two equations $f^<(e_1) \cdot n = 0$ and $f^>(e_1) \cdot n = 0$. These conditions map out two great circles on the Gaussian sphere. Since the same holds for $Side^{\mathbb{V}}(e_2)$, the boundary of $Side^{\mathbb{V}}(e_1, e_2)$ is defined by an intersection of two pairs of great circles. Given that the intersection of two circles can have two distinct solutions, the boundary of $Side^{\mathbb{V}}(e_1, e_2)$ can be split into two separate regions, both of which are themselves connected. □

For the set $Side^{\mathbb{V}}(e_1, e_2)$ it is therefore necessary to find, or "bootstrap" to two separate elements on the opposing sides of the Gaussian sphere and perform a graph search from both starting points in order to ensure that the whole set $Side^{\mathbb{V}}(e_1, e_2)$ gets enumerated in full.

This bootstrapping process is fortunately simple. Each of the sets $Anti^{\mathbb{V}}(e)$, $Side^{\mathbb{V}}(e)$ and $Side^{\mathbb{V}}(e_1, e_2)$ are defined by one or two support vector directions, so the task of bootstrapping is the same as to find an extreme vertex of the hull. That can be done in linear time by iterating over the whole of V, but that is not very interesting. A more advanced method is to utilize a **Dobkin-Kirkpatrick hierarchy** to enable finding supporting vertices in $\mathcal{O}(\log n)$ time [Dobkin and Kirkpatrick 1990]. Therefore the total time complexity to enumerate these sets is

- $Anti^{\mathbb{V}}(e)$: $\mathcal{O}\big(\log n + |Anti^{\mathbb{V}}(e)|\big)$.
- $Side^{\mathbb{V}}(e)$: $\mathcal{O}\big(\log n + |Side^{\mathbb{V}}(e)|\big)$.

- $Side^{\mathbb{V}}(e_1, e_2)$: $\mathcal{O}\big(\log n + |Side^{\mathbb{V}}(e_1, e_2)|\big)$.

Unfortunately it seems to be too difficult to give strict limits for the sizes of antipodal and sidepodal sets in the general case. In the following sections, the sizes of these sets are analyzed in two special cases that are the most interesting.

4.1 Analysis of Sphere(n)

Consider a set of n distinct points taken uniformly random on the surface of the unit sphere and denote by Sphere(n) the convex hull of that set. Clearly Sphere(n) consists of the n vertices itself. As n grows, the shape of Sphere(n) grows to resemble the unit sphere. This set has two particularly important properties. The first is that the dihedral angle of each edge e in Sphere(n) tends towards 180 degrees. This is another way of saying that the "sharp" edges of Sphere(n) all disappear. The second property is that the number of vertices on each face of Sphere(n) is bounded by a constant. In fact, each face is defined by exactly three vertices with probability 1.

In this scenario, the areas of the convex regions in the Gaussian sphere corresponding to each vertex in Sphere(n) tend to zero size. Therefore any region R of the Gaussian sphere corresponds to a number of vertices of Sphere(n) that is **linearly proportional** to the **surface area** of R in Sphere(n). In particular, this means the following.

Theorem 4.4. *With probability 1, on the set Sphere(n), for each edge e, the size of antipodal vertices $|Anti^{\mathbb{V}}(e)| \in \mathcal{O}(1)$.*

Proof. As n grows, the dihedral angle of e tends to 180 degrees. In other words, $f^{<}(e)$ tends towards $f^{>}(e)$ and hence the arc traced by $-n = -f^{<}(e) - \big(f^{>}(e) - f^{<}(e)\big) * \boldsymbol{t}$ degenerates towards a single point, which has zero surface area. □

A similar property exists for sidepodal edges.

Theorem 4.5. *With probability 1, on the set Sphere(n), for each edge e, the size of sidepodal vertices $|Side^{\mathbb{V}}(e)| \in \mathcal{O}(\sqrt{n})$.*

Proof. Since $f^{<}(e)$ tends towards $f^{>}(e)$, the two boundary circles that define support directions for $Side^{\mathbb{V}}(e)$ tend to a single overlapping great circle on the Gaussian sphere. The length of circumference of this great circle is $2\pi r$, whereas the total surface area of the sphere is $4\pi r^2$. Relating areas to vertices, $n \approxeq 4\pi r^2$, so therefore $\sqrt{n} \approxeq 2\pi r$, or $|Side^{\mathbb{V}}(e)| \in \mathcal{O}(\sqrt{n})$. □

And finally,

Theorem 4.6. *With probability 1, on the set Sphere(n), for each pair of edges e_1 and e_2, the size of sidepodal vertices $|Side^{\mathbb{V}}(e_1, e_2)| \in \mathcal{O}(1)$.*

Proof. Continuing from proof of 4.5, the support directions for $Side^{\mathbb{V}}(e_1, e_2)$ is the intersection formed by two great circles. When $e_1 \neq e_2$, the great circles are distinct and the intersection has exactly two solutions that are points. Like in the case of proof of 4.4, points have zero area, and hence $Side^{\mathbb{V}}(e_1, e_2) \in \mathcal{O}(1)$ as well. □

Putting these three results together looks like the following.

Theorem 4.7. *On the set Sphere(n), given an edge e or a pair of edges e_1, e_2, the total time taken to enumerate*

- $Anti^{\mathbb{V}}(e)$ *is* $\mathcal{O}(\log n) + \mathcal{O}(1) = \mathcal{O}(\log n)$.
- $Side^{\mathbb{V}}(e)$ *is* $\mathcal{O}(\log n) + \mathcal{O}(\sqrt{n}) = \mathcal{O}(\sqrt{n})$.
- $Side^{\mathbb{V}}(e_1, e_2)$ *is* $\mathcal{O}(\log n) + \mathcal{O}(1) = \mathcal{O}(\log n)$.

Proof. The total time to iterate over these sets is proportional to the time taken to bootstrap to the first vertex, plus the size of the set itself. □

This result concludes the analysis in positive light that in a suitably uniform scenario, the sizes of antipodal and sidepodal sets are considerably smaller and faster to enumerate than linear time. In the next section, the same analysis is done without the assumption of uniformity in place.

4.2 A Singularity in Cylinder(n)

If the requirement of uniform distribution from the case of Sphere(n) is lifted, do the properties of theorem 4.7 still hold in the general case? Unfortunately not always, and it is possible to construct a scenario where the sizes of these sets are linear. Curiously, the only currently found case occurs when the input data set is a cylinder, which is defined by

$$\text{Cylinder}(n) := \left\{ \big(\cos\frac{2\pi i}{n}, \pm 1, \sin\frac{2\pi i}{n} \big) : i = 1, 2, \ldots, n \right\}.$$

This set represents a cylindrical shape with n points on both end caps. An examination of this set shows that each edge e that is part of the cylinder end caps of the convex hull of Cylinder(n) has a linear number of vertices on average in each of the sets $Anti^{\mathbb{V}}(e)$, $Side^{\mathbb{V}}(e)$ and $Side^{\mathbb{V}}(e_1, e_2)$. Therefore enumerating over these sets is no faster than just enumerating over all vertices of the hull.

Examining the Gaussian sphere of this shape gives a clue to why this happens. One problem stems from the fact that the end faces of the cylinder lie on parallel planes, and have both $n \in \Omega(n)$ vertices. For the Gaussian sphere, this means that there are two vertices corresponding to those faces exactly at the opposite poles of the Gaussian sphere, and they both have $n \in \Omega(n)$ incident edges connecting them. The existence of such a "singularity" point that connects a linear number of vertices forces the size of $Anti^{\mathbb{V}}(e)$ to be linear for each edge on the cylinder end caps. If the number of vertices on each face of the convex hull was bounded by a constant, this would not happen. However, even if imposing such a restriction, the sizes of the sets $Side^{\mathbb{V}}(e)$ and $Side^{\mathbb{V}}(e_1, e_2)$ will still remain linear, so this restriction does not completely capture the worst case behavior. Therefore, at least in for Cylinder(n), the following appears to be the case.

Proposition 4.8. *In the worst case,*

$$|Anti^{\mathbb{V}}(e)|, |Side^{\mathbb{V}}(e)| \text{ and } |Side^{\mathbb{V}}(e_1, e_2)| \in \Omega(n). \quad \blacksquare$$

It remains an open question whether it is possible to characterize the conditions when this occurs in a more precise manner to understand this behavior better. So far this worst case behavior has only been observed on shapes that are very cylindrical, and even a small deviation from this generally causes the issue to vanish.

5 The New Algorithm

With the help of the mathematical analysis in sections 3 and 4, it is now possible to present the main algorithm. The overall strategy is straightforward: instead of performing a brute force search over all potential orientation directions in the sphere, the plan is to enumerate all OBB orientations that are uniquely fixed by combinations of

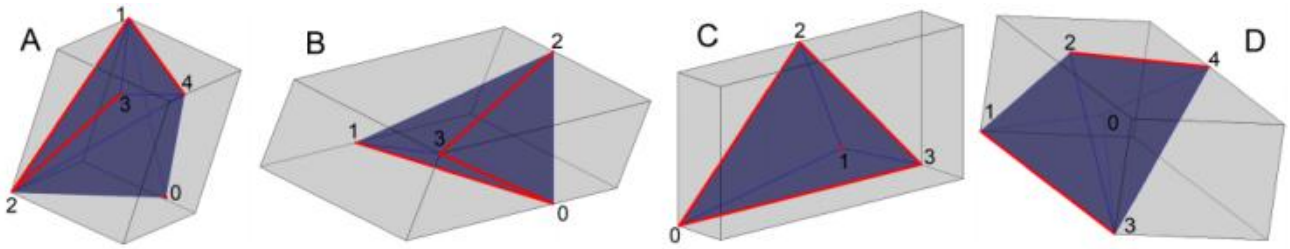

Figure 2: *Examples of the four different edge contact categories **A**-**D** for a convex polyhedron and its enclosing OBB. The features in contact with the OBB are highlighted in red. **A**: Three edges flush with three mutually adjacent faces of the OBB. **B**: Three edges flush with three faces of the OBB, of which two are opposite. **C**: Three edges flush with two adjacent faces. **D**: Two edges flush with three faces, of which two must be opposite.*

Algorithm 6: FindMinOBB(V,F,E)

for $e_1 \in E$ **do**
  for $e_2 \in Side^{\mathbb{E}}(e_1)$ **do**
    /* Test category A */
    for $e_3 \in Side^{\mathbb{E}}(e_1, e_2)$ **do**
      $B = ComputeBasis(e_1, e_2, e_3)$
      if $B \neq \emptyset$ **then**
        $O = ComputeOBB(V, B)$
        $RecordOBB(O)$
    /* Test category B */
    for $e_3 \in Anti^{\mathbb{E}}(e_1)$ **do**
      $n = AntipodalDir(e_1, e_3)$
      $B = CompleteBasis(n, e_2)$
      if $B \neq \emptyset$ **then**
        $O = ComputeOBB(V, B)$
        $RecordOBB(O)$
/* Test category C */
for $f \in F$ **do**
  let e_1 , e_2 be two edges in f.
  let $n = e_1 \times e_2$.
  for $e_3 \in Side^{\mathbb{E}}(e_1)$ **do**
    $B = CompleteBasis(n, e_3)$
    if $B \neq \emptyset$ **then**
      $O = ComputeOBB(V, B)$
      $RecordOBB(O)$

edges of the convex hull. These configurations are divided into four categories:

A The OBB is flush with three (or more) edges of the hull on three (or more) mutually adjacent faces of the OBB. Example case with minimal contact of the hull and the OBB:
$0 : (1, 0, 2)$, $2 : (4, 0, 4)$, $4 : (3, 2, 0)$
$1 : (1, 4, 3)$, $3 : (4, 2, 1)$
The minimal volume OBB is flush with edges $1 \to 2$, $1 \to 4$ and $2 \to 3$ on three mutually adjacent faces of the OBB and touches vertex 0 on the face opposite to edge $1 \to 2$.

B The OBB is flush with three (or more) edges of the hull on three (or more) faces of the OBB, of which two are opposite to each other. Example case with minimal contact of the hull and the OBB:
$0 : (0, 2, 0)$, $2 : (0, 4, 0)$
$1 : (0, 2, 2)$, $3 : (2, 0, 2)$
The minimal volume OBB is flush with edges $0 \to 1$, $0 \to 2$ and $2 \to 3$. The edges $0 \to 1$ and $2 \to 3$ lie on the opposite faces of the OBB.

C The OBB is flush with three (or more) edges of the hull on only two adjacent faces of the OBB, or in other words, the OBB is flush with a face and an edge of the hull. Example case with minimal contact of the hull and the OBB:
$0 : (0, 0, 0)$, $2 : (5, 5, 0)$
$1 : (5, 2, 2)$, $3 : (10, 0, 0)$
The minimal volume OBB is flush with the face formed by vertices $0 \to 2 \to 3$, and the opposite face of the OBB is in contact with the vertex 1. Note that the contact edge on an adjacent face of the OBB may or may not be the same as one of the edges of the face. In this case, the adjacent edge is edge $0 \to 3$, which is also one of the face edges.

D The OBB is flush with only two edges of the hull on three different faces of the OBB. This is a special case where an edge of the hull coincides with an edge of the OBB. When this occurs, there must exist two opposite faces on the OBB that contain edges of the convex hull, or otherwise the box and the polyhedron could be projected to a 2D plane along the common shared edge (reducing the shared edge to a point), and the resulting 2D rectangle would not be flush with any edge of the 2D polygon and therefore would not be optimal. Example case with minimal contact of the hull and the OBB:
$0 : (0, 4, 2)$, $2 : (2, 4, 2)$, $4 : (1, 4, 0)$
$1 : (0, 4, 4)$, $3 : (3, 0, 1)$
In this example, the minimal volume OBB is flush with only two edges of the polyhedron, but these edges are in contact with three different faces of the OBB. The vertex 0 is an internal vertex that is not in contact with the OBB.

Figure 2 shows a rendering to illustrate each of the example cases.

The main search will proceed by testing all valid configurations of each of these categories. This is done in the following manner:

A + B For each edge $e_1 \in E$, find all edges $e_2 \in Side^{\mathbb{E}}(e_1)$ that can be placed on an adjacent side of the OBB. Then, for category **A**, complete the basis by searching for all edges $e_3 \in Side^{\mathbb{E}}(e_1, e_2)$ to be placed on a third face of the OBB that is mutually adjacent to both earlier faces. Correspondingly for category **B**, complete the basis by searching for all edges $e_3 \in Anti^{\mathbb{E}}(e_1)$ to be placed on a third face of the OBB that is opposite to the face that is in contact with edge e_1.

C Loop over all faces $f \in F$. This fixes one of the face normals n_1 of the OBB. Then loop over all edges $e_3 \in Side^{\mathbb{E}}(e_1)$, where e_1 is one of the edges of F. Complete the basis for each using algorithm 5.

D This category gets implicitly handled during the search for

category **B** by allowing the assumption that $e_1 = e_2$, so no separate code for testing category **D** is needed. Effectively this is recognizing that an edge may be sidepodal to itself.

The full code to perform this iteration is presented in algorithm 6. This code contains two new functions that have not been introduced before. The first function $ComputeOBB$ takes a basis for the orientation of an OBB and computes six extreme vertices of the hull, one for each face of the OBB. With the help of the Dobkin-Kirkpatrick structure, this takes at most $\mathcal{O}(\log n)$ time. The second one is the function $RecordOBB$. This is a constant time operation which tests the newly created OBB for volume against the best previously found one, and records the smaller of the two.

Algorithm 6 contains two separate top level loops. The first examines categories **A**, **B** and **D**, followed by a second one which examines category **C**. In the first loop structure, an outer loop over each edge of the hull fixes the first edge. This is a $\mathcal{O}(n)$ operation. The second loop iterates over all sidepodal edges of the first edge. This takes $\mathcal{O}\big(\log n + |Side^{\mathbb{E}}(e_1)|\big)$ time.

The first innermost loop searches through all configurations in category **A** in $\mathcal{O}\big(\log n + |Side^{\mathbb{E}}(e_1, e_2)|\big)$ time. To establish the orientation of the OBB, algorithm 4 is invoked, which computes a basis set (n_1, n_2, n_3) for the given three edges. The functions $ComputeOBB$ and $RecordOBB$ then process the found basis.

The second innermost loop enumerates all configurations in category **B** in $\mathcal{O}\big(\log n + |Anti^{\mathbb{E}}(e_1)|\big)$ time. If the iterated edges are compatible, then one of the OBB face directions is first computed with algorithm 2, which is completed to a full basis by the method 5. This basis is then again tested.

Last, the second loop body iterates through each configuration in category **C**. Each face of the hull directly defines a face direction for the OBB. The inner loop iterates over the sidepodal edges of e_1 in $\mathcal{O}\big(\log n + |Side^{\mathbb{E}}(e_1)|\big)$ time to find the candidate orientations for the second axis of the OBB. Like in the previous case, the full basis is completed and tested.

The overall number of steps in the first loop structure is $|E|\big(\log n + |Side^{\mathbb{E}}(e_*)|\big)\big(\log n + |Side^{\mathbb{E}}(e_*, e_*)| + |Anti^{\mathbb{E}}(e_*)|\big)\log n$, whereas the second loop runs in $|F|\big(\log n + |Side^{\mathbb{E}}(e_*)|\big)\log n$ steps. On the Sphere(n) data set, based on theorem 4.7, the overall algorithm runtime is therefore $\mathcal{O}\big(n^{3/2}(\log n)^2\big)$. In the worst case, like in the case of the Cylinder(n) data set suggested by proposition 4.8, the algorithm runs in $\mathcal{O}(n^3 \log n)$ time.

5.1 Implementing the Graph Search

The main search algorithm in listing 6 contains for loops over several different sets. Implementing these loops carefully is the key to obtaining correct performance. While the for loops $e_1 \in E$ and $f \in F$ are trivial iterations of all stored edges and faces of the hull, the other loops will require a two-phase operation that first bootstraps to a first element of the set, and then traverses the vertex graph of the hull to compute the remaining ones. This means that the data structure storing the convex polyhedron will need to have vertex adjacency information available. The actual iteration of each set then works out to

For $e_2 \in Anti^{\mathbb{E}}(e_1)$**:**

1. Compute the first antipodal vertex $v_0 \in Supp\big(-f^{<}(e_1)\big)$.
2. Since the set of antipodal vertices is connected due to theorem 4.1, find all remaining antipodal vertices $\{v_i\}_i \subseteq Anti^{\mathbb{V}}(e_1)$ by a flood fill that starts from v_0 and proceeds to each of its neighbors that are tested for antipodality with respect to edge e_1 using algorithm 1.
3. For each edge e_2 that exists in the subgraph induced by the antipodal vertices $\{v_i\}_i$, test each for antipodality with respect to edge e_1 using algorithm 2. This set of edges generates the final set $Anti^{\mathbb{E}}(e_1)$.

For $e_2 \in Side^{\mathbb{E}}(e_1)$**:**

1. Compute the first sidepodal vertex $v_0 \in Supp(e_1)$.
2. Since the set of sidepodal vertices is connected due to theorem 4.2, find all remaining sidepodal vertices $\{v_i\}_i \subseteq Side^{\mathbb{V}}(e_1)$ by a flood fill that starts from v_0 and proceeds to each of its neighbors that are tested for sidepodality with respect to e_1 using methods from section 3.4.
3. For each edge e_2 that exists in the subgraph induced by the sidepodal vertices $\{v_i\}_i$, test each for sidepodality with respect to edge e_1 by using algorithm 3. This set of edges generates the final set $Side^{\mathbb{E}}(e_1)$.

For $e_3 \in Side^{\mathbb{E}}(e_1, e_2)$**:**

1. Let $n = n_1 \times n_2$, where n_1 and n_2 are chosen using lemma 3.1 with respect to edges e_1 and e_2 in such a way that the cross product is not denegerate.
2. Compute first representatives from both of the two disjoint sets of connected components by searching $v_0 \in Supp(n)$ and $v_1 \in Supp(-n)$.
3. Since the set of sidepodal vertices to a pair of edges is connected inside the two disjoint sets, due to theorem 4.3, find all remaining sidepodal vertices $\{v_i\}_i \subseteq Side^{\mathbb{V}}(e_1, e_2)$ by two separate flood fills that start from v_0 and v_1 respectively, and proceed to each of their neighbors that are tested for sidepodality with respect to e_1 and e_2 using methods from section 3.4.
4. For each edge e_3 that exists in the subgraph induced by the sidepodal vertices $\{v_i\}_i$, use algorithm 5 to compute the set of edges that are sidepodal to both e_1 and e_2. This set of edges generates the final set $Side^{\mathbb{E}}(e_1, e_2)$.

This strategy ensures that the sets $Side^{\mathbb{V}}(\cdot)$ and $Side^{\mathbb{E}}(\cdot)$ are related to each other in a numerically stable way.

5.2 Practical Considerations

Whenever implementing any piece of real-world code, there are always special considerations that are important to take into account for the implementation to be successful and as efficient as possible. In this section, such practical matters are discussed.

The first one is the fact that the algorithm relies on the precomputation of the convex hull for the input point set. It is well known that robustly computing an accurate convex hull for arbitrary input can be hard due to floating point imprecision. A good treatise to the stability issues with convex hull implementation that the reader should be aware of has been presented at GDC [Gregorius 2014].

The OBB computation method itself has been found to be quite stable. It should however be noted that in places where the algorithms presented in this paper compare floating point values for range or equality, the code should in practice contain a very small epsilon threshold for these comparisons. This is because otherwise the graph search may terminate early when enumerating for example

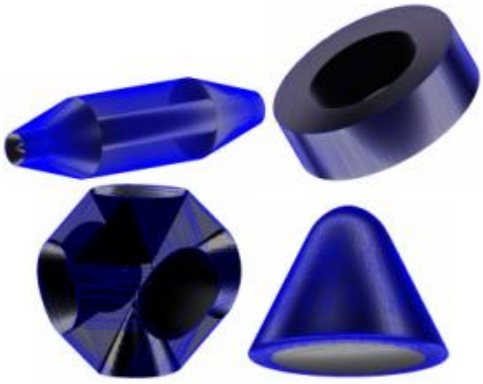

Figure 3: *Four models from the GAMMA Group real world data set that exhibited worst case performance behavior. In clockwise order starting from top left: arrow4.obj, taperoll.obj, candy05.obj and kummer.obj. Each object has a strong cylindrical component to their convex hulls.*

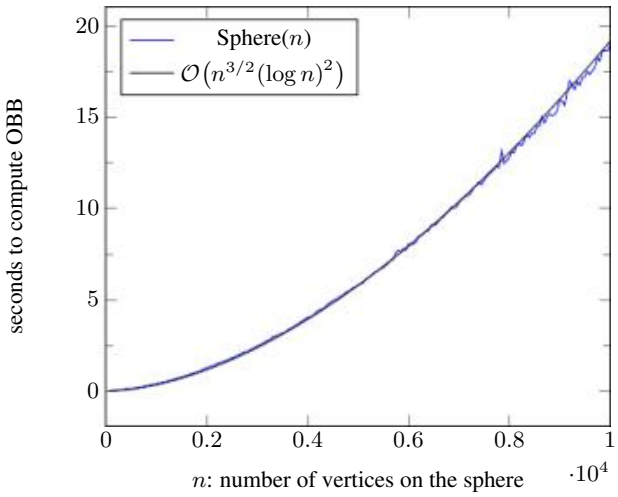

Figure 4: *Time taken to compute minimum volume OBB for Sphere(n).*

the sets $Side^{\mathbb{E}}(e_*)$, and not find an important configuration for the OBB.

To improve overall performance, a developer is advised to pay attention to the symmetry of the problem to avoid redundant (but constant factor) work. One such case occurs for example when enumerating over all edge pairs e_i and e_j that are sidepodal to each other. Since the order in which these are searched does not matter, one can cut the work in half by assuming a canonical ordering that $i \leq j$. Note that depending on the context, equality can be important, or otherwise category **D** from section 5 might be missed. Another such place where symmetry may occur is if traversing over vertex pairs to find edges on the graph. In that case, the edges $v_i \to v_j$ and $v_j \to v_i$ are the same, and should not be considered twice.

The version of the algorithm that was implemented for the benchmarks in the next section does not actually use a Dobkin-Kirkpatrick hierarchy to locate extreme vertices. To explain why, consider the problem of finding k supporting vertices $\big(Supp(n_1), Supp(n_2), \ldots, Supp(n_k)\big)$ corresponding to a sequence of k direction vectors $(n_1, n_2, \ldots, n_k)$. In the implementation, one is able to organize these queries so that each search direction is often only a very small perturbation away from the previous search direction. Then the first query for $Supp(n_1)$ will amount to an uninformed search over V, but when searching for $Supp(n_2)$, it is expected to be found in close vicinity to $Supp(n_1)$ and can be located in only a few steps by performing a greedy hill climbing search starting from the previously found vertex. Therefore $Supp(n_2)$ through $Supp(n_k)$ can be found very quickly if one caches the previous answer and always begins the next search from that cached vertex. In this kind of implementation, there exists a subroutine of depth-first search code which quickly establishes a good spatially coherent ordering for the edge graph, and all loops in the algorithm are implemented to follow this spatial order. In practice this was seen to perform extremely well, to the point of almost having the confidence to assume amortized $\mathcal{O}(1)$ time extreme vertex queries. However, there are technicalities involved and practical benchmarks still showed a very slow logarithmic growth. A rigorous exploitation of spatial coherency might have to consider a fast constant factor approximation to what is called the Chinese Postman Problem [Filho and de Avila Ribeiro Junqueira 2010] in order to establish a strict upper bound.

When benchmarking the implementation, it was found to be practical to precompute up front the first representatives for the sets $Anti^{\mathbb{V}}(e)$ and $Side^{\mathbb{E}}(e)$ for each edge e. Since this can be done outside the hot inner loops in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ storage space, it will speed up the implementation noticeably. In fact, precomputing full contents of the antipodal and sidepodal sets outside the hot loops was measured to be a performance gain. That however is an implementation tradeoff since storing full sets will require more than a linear amount of memory storage.

For the graph searches, the implementation will probably follow a flood fill type of approach to detect whether a given vertex has been visited before. An efficient implementation of this should not use a set or an associative container that would dynamically mutate to track the visited vertices, but instead it is much faster to preallocate a flat array of static size n, where each index of the array stores the information whether that vertex has been visited or not. This scheme gives the ability to clear the structure for new searches in constant time, by giving each search operation its own unique ID, and storing at each index this ID if the given vertex has been visited. Starting a new search then corresponds to simply incrementing the ID counter by one, which effectively resets all vertices to unvisited state. Also, when performing the actual search, one should not use recursive function calls, because unless optimized into tail calls, these may have unbounded depth and stack space in most runtime environments is fixed.

5.3 Dealing with the Singularity

Since the performance gap between best and worst cases is so large, it is natural to ask if the worst case performance can be improved. Unfortunately this is yet unknown, but there are a number of things that can be done to mitigate the issue in practice. In this section three such strategies are presented.

The algorithms presented in section 3 deal with computing scalars to obtain normal vectors for OBB faces from the parametrization given in lemma 3.1. In the difficult case of the cylinder, it can be observed that the solutions for this parametrization often come out at the extreme ends when $\boldsymbol{t} = 0$ or $\boldsymbol{t} = 1$, since several edges of the cylinder have a dihedral angle of exactly 90 degrees. This corresponds to the special case when one of the OBB faces is flush with a face of the polyhedron, and because these configurations will all be searched in the loop for category **C**, there is no reason to handle them at all in categories **A** and **B**. Therefore the implementation is

file	vertices	faces	edges	t_{exact}	t_{merge}	t_{extrude}	t_{approx}	$V_{\text{extrude}}/V_{\text{exact}}$	$V_{\text{approx}}/V_{\text{exact}}$
arrow4.obj	1024	2044	3066	84.7	29.6	16.4	0.54	1.0005x	1.006x
taperoll.obj	839	1674	2511	67.5	54.3	8.63	2.05	1.005x	1.011x
candy05.obj	20044	40084	60126	84.7	83.9	29.8	0.1	1.00007x	1.001x
kummer.obj	624	1244	1866	66.9	11.8	3.45	0.7	1.007x	1.012x

Figure 5: *Results of applying preprocessing methods on the four worst case models from the data set obtained from GAMMA Group mesh database. The columns t_x show the number of seconds taken to compute the minimum volume OBB in different cases. The column t_{exact} corresponds to the case of the original unprocessed model. Last two columns denote ratios of the found OBBs to the volume of the optimal box, when approximation methods were used.*

free to restrict valid values of $\boldsymbol{t}$ to the range $[\varepsilon, 1-\varepsilon]$ for a small constant $\varepsilon > 0$ in order to cut down redundant work.

Another observation is that the convex hull algorithm that is employed might by convention generate hulls that consist of triangular faces only. This can create neighboring triangles that are aligned in a plane. The edges joining such adjacent planar triangles have a dihedral angle of 180 degrees each, so they are degenerate and do not have any actual effect and an implementation should remove these edges in a preprocess step. This can be done either by merging all neighboring planar triangles to form convex polygons, or by ignoring all internal edges on the faces during the search. This scheme will be later called the **merge** strategy. In the case of the cylinder, the overhead from redundant internal edges can be considerable since there are a linear number of them, and if not removed, each of these internal edges will in turn have a linear number of sidepodal partners.

The abovementioned optimizations are unfortunately only improvements by a constant factor. In order to completely remove the problem in practice, two specific approximation methods were employed with the goal of preprocessing the convex hull in a subtle manner that should have only minimal effect on the resulting OBB, but would considerably cut down the work in practice. Assuming that the basis of the OBB obtained as a result from this approximation does not change much, then the same basis can be used on the original unprocessed convex hull to compute the exact tight-fitting OBB.

The first method builds on the idea that because the end caps of the cylinder are so problematic due to the large number of vertices in them, these faces should be removed. This can be done by extruding a new vertex outward to the direction of the face normal from the centroid of each face polygon that has too many vertices. This breaks each polygonal face up into a number of triangles, which each have their unique face normal direction. If the new vertex is located very close to the plane of the polygon, the orientation of the resulting OBB will be very close to the original. This method will be called the **extrude** strategy. For convex hull algorithms that produce only triangular data, this strategy will first require running the **merge** step to find the faces with large number of vertices.

The second method operates by constructing a simplified approximation of the convex hull by repeatedly merging two adjacent faces of the hull that are almost coplanar, until an error threshold is met or the number of faces is reduced down to a number that is small enough to process. When two faces are merged in this manner, the new face does not even need to enclose all the points of the original hull, since the approximation will only be used to establish an orientation for the OBB and the tight fit can be computed on the original hull after the basis has been chosen. The process can be done for disjoint pairs of faces, after which the convex hull is recomputed again to ensure proper convexity. For the case of the cylinder, this method was found to be very efficient, since it successfully removes large numbers of edges on the side of the cylinder, where the finest detail typically is. This will be called the **approx** strategy. Running **merge** or **extrude** steps before this strategy was not seen beneficial, so the results for this strategy consist of only running the **approx** step alone.

The next section reports results from practical tests, where the last conducted test shows comparisons of adding the three optimizations **merge**, **extrude** and **approx** that were introduced in this section.

6 Experimental Results

The new algorithm proposed in section 5 was implemented using the open source MathGeoLib software package [Jylänki 2011–2015]. The code was then tested on two different data sets. The first one was the synthetic case of Sphere(n) from section 4.1 for values $n \in \{50, 100, 150, \ldots, 10000\}$. The second data set consisted of 2088 different 3D models obtained from the GAMMA Group online 3D mesh research database [GAMMA Group 2008]. From this database, all files in the .obj file format were chosen from five different categories:

- 55 models from the dataset **2001**.
- 381 models from the dataset **ANIMALS**.
- 530 models from the dataset **ARCHITEC**.
- 437 models from the dataset **GEOMETRY**.
- 685 models from the dataset **MECHANICAL**.

In the first test, an OBB was generated for each input object in both of the two data sets using the exact version of new algorithm that does not include approximations from the previous section, and the result was compared for optimality against the smallest volume OBB found via a brute force search. The brute force search consisted of subdividing the unit hemisphere to a search grid of $256 * 256$ starting search directions, and using the rotating calipers method to improve the volume of the box until reaching a local minimum for each start orientation. This test confirmed that in each case, the new algorithm found the same or a better bounding OBB than the brute force method did, up to the precision determinable by floating point computation.

The second test consisted of benchmarking the performance of the exact version of the new algorithm on the same two data sets. The performance tests were run on a MacBook Pro laptop with a 2.7GHz Intel Core i7 processor and 16GB of RAM running Windows 8.1. The test program was built by using the Microsoft Visual Studio 2013 Update 4 compiler, with 64-bit compilation and MathGeoLib AVX SIMD performance optimizations enabled. Execution was single-threaded in all benchmarks. The numbers reported in each case do not include the time taken to run the quickhull algorithm. The resulting runtime performance for the case of Sphere(n) is shown in figure 4, where the actual data is graphed in blue. A regression curve of the expected performance of $n^{3/2}(\log n)^2$ provided by earlier analysis was fit to this data in the range $n \leq 5000$ and is shown for reference in black. Extrapolating this curve for

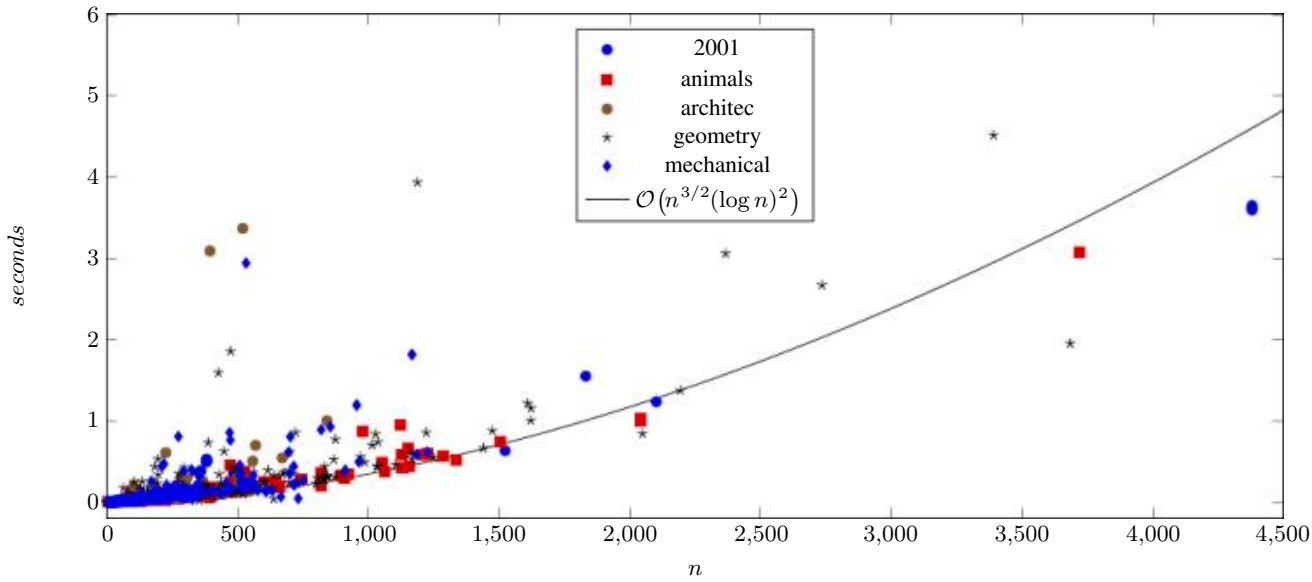

Figure 6: *A scatter plot showing the time taken to compute the minimum volume OBB for five different data sets from the GAMMA Group 3D mesh database. Each model is located on the x-axis based on the number of vertices on its convex hull. The y-coordinate specifies the number of seconds taken to compute the OBB for that model. The black curve shows the regression function from the previous Sphere(n) case. The four models shown in figure 3 were complete outliers that did not fit in this plot.*

values of $5000 \leq n \leq 10000$ shows a pleasing match with the recorded data.

Figure 6 shows the performance of the exact version of the algorithm on the real world data sets from the GAMMA Group. In the scatter plot, the x-coordinate of each point gives the number of points on the convex hull of that object, whereas the vertical axis shows the time taken to compute the OBB, in seconds. The results from this benchmark show that fortunately the performance for most real world objects is very similar to the case of Sphere(n). Of the 2088 models that were tested, 2084 exhibited performance close to the best case behavior. The remaining four models were outliers that took more than a minute to process. Taking a closer look revealed that each of these four had convex hulls that have a strong cylindrical shape. Figure 3 shows a rendering of each.

The last test consisted of measuring the effects of including the **merge**, **extrude** and **approx** mitigation strategies on these four difficult test models from the GAMMA Group database that suffered from degenerate behavior. The results are presented in figure 5 and they look promising. For all models, merging planar triangles was an improvement. Since it does not change the optimality of the result or slow down the computation of the OBB, it would be a mistake to ignore the **merge** preprocessing step. The **extrude** approximation method was also effective and it was able to speed up the computation considerably in each case. Even while this method is not theoretically optimal, it is something to consider in practice, since the resulting OBB volumes were only 0.5% larger than the optimal in worst case. Finally, the results of the **approx** method shows that it is a promising way to speed up the search to a few seconds at most, while only introducing about one percent increase in the resulting volume.

7 Conclusion and Future Work

A new algorithm has been introduced for finding tight-fitting enclosing bounding boxes for three-dimensional point data. The method operates by quickly iterating over all possible OBB orientations that are uniquely determined by combinations of edges of the convex hull of the input. This was made possible with the help of new mathematical insight on the problem. The algorithm is able to accurately find the optimal minimum volume enclosing OBB in all conducted tests. Unfortunately the optimality of this method depends on an unproven assumption, but given the infallible practical evidence, it is appropriate to posit the following.

Conjecture 7.1. *A box of minimal volume circumscribing a convex polyhedron has at least three distinct edge-face contacts (of which at least two occur on adjacent faces of the OBB).*

In this statement, an edge-face contact means that a specific face f of the OBB is flush with a specific edge e of the polyhedron. Being distinct means that in any pair of the contacts, either the edges or the faces in question are different. This statement captures all of the categories **A** - **D** enumerated in section 5 (figure 2), so if proven, it would assert the optimality of the new algorithm. The second part in parentheses is repeated from theorem 2.1. If conjecture 7.1 holds, the presented method will be the first algorithm in its own class in comparison to all previously published results: an exact and deterministic geometric enumeration method for solving the minimum volume OBB problem.

The algorithm has been implemented in practice and the real world results validate the theoretical aspects. On the tested real world models that are not cylinder-like, the algorithm runs in the predicted $\mathcal{O}(n^{3/2}(\log n)^2)$ time, which is a large improvement over the previously published best $\mathcal{O}(n^3)$ time complexity. For models that have only a few hundred points on their convex hulls, the algorithm can find optimal OBBs at interactive rates. In the worst case, which is currently only known to happen on cylindrical shapes and shapes that can have vertices with $\Omega(n)$ number of neighbors, the algorithm exhibits degenerate behavior that causes it to run in $\mathcal{O}(n^3 \log n)$ time. However this performance cliff can be mitigated in practice by performing a preprocessing step on the input. It is an

open question if this worst case behavior can be strictly improved. On the other end, I believe that $\Omega(n^{3/2})$ is a lower bound that cannot be circumvented by any algorithm that is based on enumerating edge configurations of the hull.

In some applications it is more useful to find an enclosing OBB that minimizes the surface area of the box rather than the volume. The presented algorithm can be trivially adapted to optimize the surface area instead of the volume, but it is unclear however whether this will result in optimal minimum surface area OBBs. Due to the already lengthy treatise presented in this paper, this kind of examination was left out for later.

Solving conjecture 7.1 will probably need to examine the volume of an OBB as a function of a given basis. O'Rourke suspected that this function might have local minima in configurations where the OBB is flush with exactly two edges and four vertices of the polyhedron, but noted that he had not found evidence of this in practice. If such a configuration exists that is a global minimum, it will disprove the optimality of the new algorithm presented here. Despite attempts at producing such a configuration via computer search, I have not been able to find a counterexample. In fact, such a configuration does not seem to exist even as a local minimum. Therefore it raises a question whether conjecture 7.1 could even be strengthened to apply to all local minima of the volume function.

Acknowledgements

I would like to thank Kalle Leppälä and Ilari Vallivaara for each of the many pleasing and insightful conversations on the topic.

References

BARBER, C. B., DOBKIN, D. P., AND HUHDANPAA, H. 1996. The quickhull algorithm for convex hulls. *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE 22*, 4, 469–483.

BAREQUET, G., AND HAR-PELED, S. 2001. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *J. Algorithms 38*, 1 (Jan.), 91–109.

BARTZ, D., KLOSOWSKI, J. T., STANEKER, D., INTERAKTIVE SYSTEME, G., AND TÜBINGEN, U., 2005. Visual computing for medicine.

BORCKMANS, P. B., AND ABSIL, P.-A. 2010. Oriented bounding box computation using particle swarm optimization. In *ESANN*.

CHAN, C. 2003. *Minimum Bounding Boxes and Volume Decomposition of CAD Models*. University of Hong Kong.

CHANG, C.-T., GORISSEN, B., AND MELCHIOR, S. 2011. Fast oriented bounding box optimization on the rotation group SO(3,$\mathbb{R}$). *ACM Trans. Graph. 30*, 5 (Oct.), 122:1–122:16.

DIMITROV, D., KNAUER, C., KRIEGEL, K., AND ROTE, G. 2009. Bounds on the quality of the pca bounding boxes. *Comput. Geom. Theory Appl. 42*, 8 (Oct.), 772–789.

DOBKIN, D. P., AND KIRKPATRICK, D. G. 1990. Determining the separation of preprocessed polyhedra: A unified approach. In *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, Springer-Verlag New York, Inc., New York, NY, USA, 400–413.

EBERLY, D. H. 2006. *3D Game Engine Design, Second Edition: A Practical Approach to Real-Time Computer Graphics (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

ERICSON, C. 2004. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

FILHO, M. G., AND DE AVILA RIBEIRO JUNQUEIRA, R. 2010. Chinese postman problem (cpp): solution methods and computational time. *Int. J. of Logistics Systems and Management*.

FREEMAN, H., AND SHAPIRA, R. 1975. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM 18*, 7 (July), 409–413.

GAMEDEV.NET. How to create oriented bounding box.

GAMMA GROUP, 2008. 3d meshes research database of the group génération automatique de maillages et méthodes d'adaptation, inria, france. [software, website]. Available at `https://www-roc.inria.fr/gamma/gamma/download/download.php`.

GEOMETRIC TOOLS LLC. Mathematics: Computational geometry.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '96, 171–180.

GREGORIUS, D., 2014. The 3d quickhull algorithm. Game Developers Conference. Available at `http://www.gdcvault.com/play/1020141/Physics-for-Game-Programmers`.

JYLÄNKI, J., 2011–2015. MathGeoLib: A C++ library for 3D mathematics and geometry manipulation. Available at `http://clb.confined.space/MathGeoLib/` and `https://github.com/juj/MathGeoLib/`.

KAY, T. L., AND KAJIYA, J. T. 1986. Ray tracing complex scenes. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '86, 269–278.

LARSSON, T., AND KÄLLBERG, L. 2011. Fast computation of tight-fitting oriented bounding boxes. In *Game Engine Gems 2*, E. Lengyel, Ed. A K Peters, 3–19.

MEGIDDO, N. 1982. Linear-time algorithms for linear programming in r3 and related problems. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*, 329–338.

O'ROURKE, J. 1985. Finding minimal enclosing boxes. *International Journal of Computer & Information Sciences 14*, 3, 183–199.

SCHNEIDER, P. J., AND EBERLY, D. 2002. *Geometric Tools for Computer Graphics*. Elsevier Science Inc., New York, NY, USA.

STACKOVERFLOW. Creating oobb from points.

TOUSSAINT, G. 1983. Solving geometric problems with the rotating calipers. *In Proc. IEEE MELECON '83*, 10—02.

TROUTMAN, N. Calculating minimum volume bounding box.

WELZL, E. 1991. Smallest enclosing disks (balls and ellipsoids). In *Results and New Trends in Computer Science*, Springer-Verlag, 359–370.