Asia-Pacific Journal of Business
Vol. 9, No. 2, June 2018 (pp. 1-13)
https://imr.kangwon.ac.kr/apjb.7.7.201806.1.

APJB
ISSN 2233-5900 (Print)
ISSN 2384-3934 (Online)

Using Piecewise Circular Curves as a 2D Collision Primitive

Robert Ollington[a]

[a]Lecturer, Discipline of ICT, School of Technology, Environments and Design, College of Sciences and Engineering, University of Tasmania, Australia

Received 01 June 2018, Revised 15 June 2018, Accepted 25 June 2018

Abstract

Physics simulation is an important part of many interactive 2D applications and collision detection and response is key component of this simulation. While methods for reducing the number of collision tests that need to be performed has been well researched, methods for performing the final checks with collision primitives have seen little recent development.
This paper presents a new collision primitive, the n-arc, constructed from piecewise circular curves or biarcs. An algorithm for performing a collision check between these primitives is presented and compared to a convex polygon primitive. The n-arc is shown to exhibit similar, though slightly slower, performance to a polygon when no collision occurs, but is considerably faster when a collision does occur. The goodness of fit of the new primitive is also compared to a polygon. While the n-arc often gives a looser fit in terms of area, the continuous tangents of the n-arcs makes them a good choice for organic, soft or curved surfaces.

***Keywords*:** 2D, Biarc, Piecewise Circular Curve, Arc-spline
***JEL Classifications*:** C61, C63, O32

Ⅰ. Introduction

Physics simulation plays a very important role in modern computer games and many other interactive computer applications. A good physics simulation can make a game more enjoyable, more believable, and improve replayability. A key part of any physics simulation is collision detection – determining when two objects are intersecting. Depending on the geometry, this can be a very computationally expensive process.

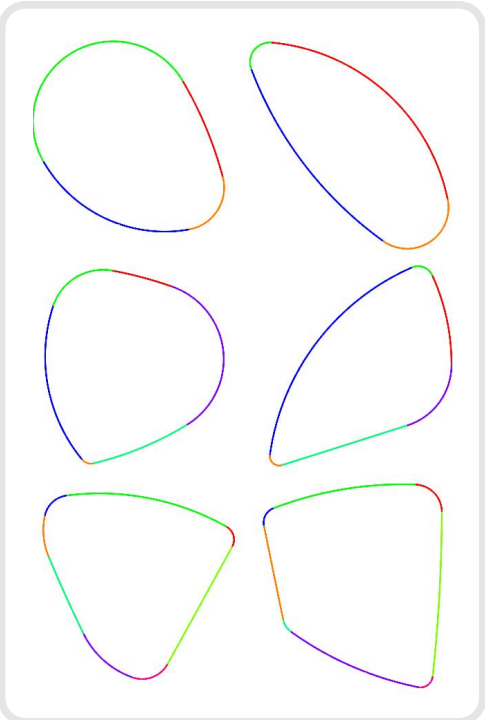
The typical approach to speeding up collision checks is to use a rough broad phase approach that can quickly cull po-

[a] First Author, E-mail: Robert.Ollington@utas.edu.au

tential collisions, followed by a narrow phase that performs a more precise collision check. Broad phase collision checking algorithms are an ongoing topic of research and typically involve the use of some form of space partitioning and/or hierarchical bounding volumes [Jimene´z et al. 2001; Lin and Gottschalk 1998].

The narrow phase consists of detailed checks between collision primitives. Often these primitives are approximations of the desired geometry and a trade-off must be made between the goodness of fit and the speed of the collision check. The development of primitives that provide a good fit and low computational cost is an important consideration. While some important research has been conducted for 3D primitives [Krishnan et al. 1998; Larsen et al. 2000; Krishnan et al. 1998], there has been little work on 2D primitives in recent times.

Fig. 1. Examples of 4-, 6-, and 8-Arcs



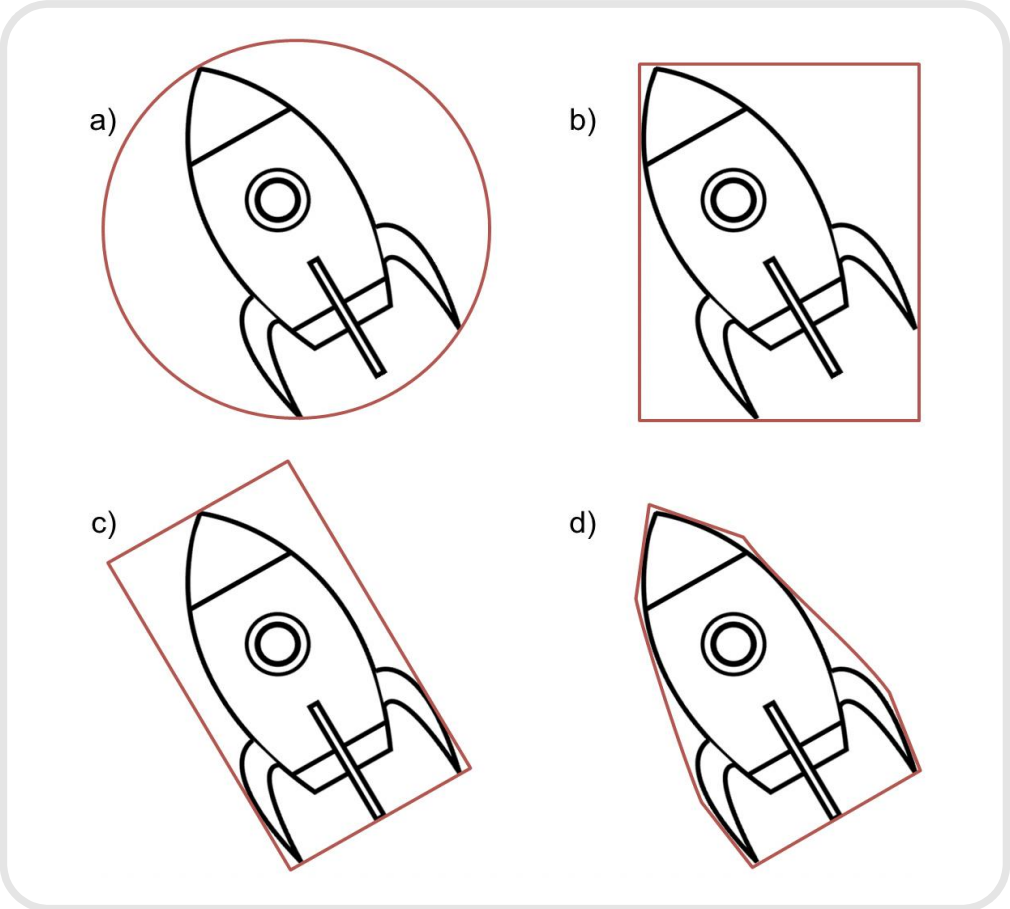
The 2D primitives most often used are circles, axis-aligned bounding boxes (AABBs), oriented bounding boxes (OBBs) and convex polygons as shown in Figure 2. Circle and AABB collisions are very fast to compute, but are a poor fit for many objects. OBBs provide a better fit for most objects, but are slower to compute. Polygons are a good fit for many objects, but are quite slow to compute.

While circles and AABBs generally provide a poor fit to the underlying geometry, it is often more practical to change the shape of the target geometry than use a slower collision primitive. Many physics based games (e.g. Angry Birds1) take this approach.

When a better fit is required, polygons are typically used and these can produce a very good fit for many shapes, especially man- made objects (e.g. buildings). However, many organic objects and objects with curved surfaces require a many-sided polygon to produce a good fit resulting in potentially poor performance. It would be useful then to have a primitive that provides similar (or better) performance to a polygon, but is better suited to organic/curved objects.

This paper presents a collision primitive and collision detection algorithm based on piecewise circular curves[Banchoff and Giblin 1994] or biarcs[Bolton 1975] and analyses the computational complexity of the algorithm and the goodness of fit of the primitive for typical shapes.

Fig. 2. Ing Box (AABB), c) Oriented Bounding Box (OBB), d) Convex Polygon

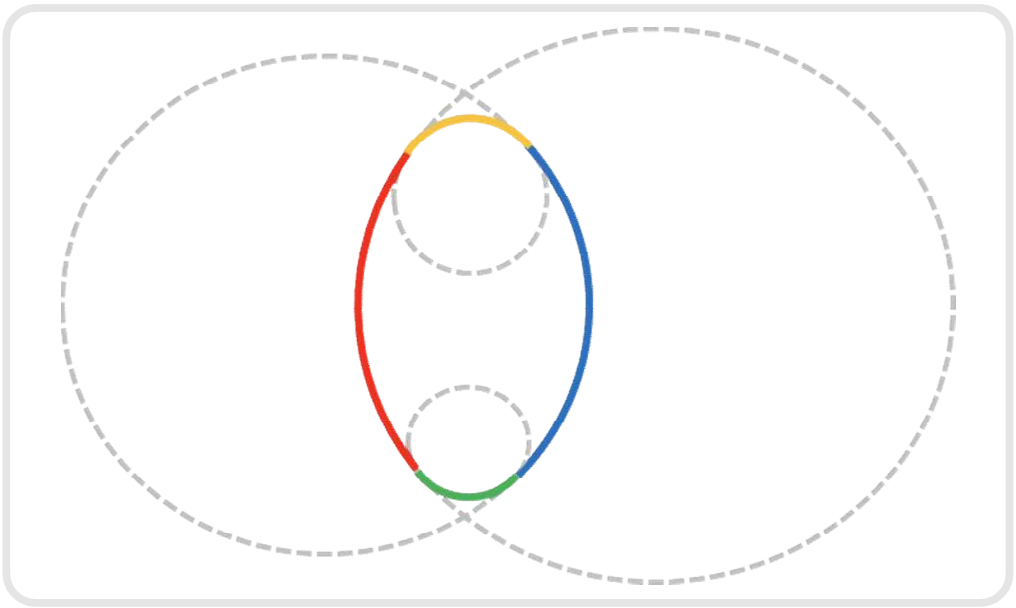


Ⅱ. N-arc Collision Primitive

Piecewise circular curves or biarcs became popular in the 70's when the ship building industry required a replacement for cubic splines [Bolton 1975]. These curves consist of a series of arcs arranged to produce a continuous tangent where they join. One of the primary reasons for using biarcs was that it was much easier to test for intersection between biarcs and other primitives (e.g. lines and circles). The requirement for fast and accurate collision checks is also an issue for interactive 2D applications.

For most applications the collision geometry is used to approximate a physical object, so a closed shape is usually required. To simplify collision checks it is also a common requirement that the shape be convex. In this paper a closed convex shape composed of biarcs is referred to as an n-arc (compare with n-gon for polygons). For example, a 4-arc can be constructed by starting with two

Fig. 3. Construction of a 4-Arc from Two Large Overlapping Circles and Two Smaller Inscribed Circles



large overlapping circles and then placing two smaller inscribed circles where they join, as shown in figure 3.

It is generally easier and more natural to construct an n-arc from a set of control points, such as the points where arcs intersect. Un- fortunately not every set of points can be used to construct a valid n-arc. For a set of points to be valid they must satisfy the following properties:

- There must be an even number of points (actually an odd number of points is possible provided two consecutive arcs have the same radius)
- The points must describe a convex polygon
- The polygon must be such that the sum of the even-numbered in- terior angles equals the sum of the odd-numbered interior angles

Theses constraints are difficult to satisfy manually, making it difficult to edit the control points. A method of converting an arbitrary polygon into an n-arc is required. Given an arbitrary polygon, an n-arc could be calculated using the method proposed by Maier and Pisinger [2013]. However, this method does not allow the number of arcs to be predeter- mined. We use a simpler method that makes use of the constraints above as shown in algorithm 1. When adjusting inner angles in the algorithm care must be taken to ensure that the overall sum of inner angles does not change. For example, decreasing one inner angle to make the polygon convex should be accompanied by an increase in one or more of the other angles to compensate. The precise method used for these adjustments is not important.

Once the vertices of the end-points of the arcs are known, the centres, and radii of the arcs need to be calculated. At least one radius must be known, from this the centre of the corresponding arc can be calculated. This radius also determines the radii of adjacent arcs since the arcs must be tangent continuous.

Owing to the careful construction, when checking for a collision between two n-arcs there will only ever be one pair of arcs that need to be tested, and this test can be performed as for circle collisions. The main task is to determine which pair of arcs should be tested.

Algorithm 1: Constructing an n-arc from an arbitrary n-gon

Input: $\mathbf{v_0}..\mathbf{v_n}$ the vertices of the n-gon ($\mathbf{v_0} = \mathbf{v_n}$)
Output: $\mathbf{v'_0}..\mathbf{v'_n}$ vertices defining the arc intersections of a valid n-arc

begin
  let $\theta_0..\theta_n$ be the inner angles of the polygon
  adjust $\theta_0..\theta_n$ so that the polygon is convex
  adjust $\theta_0..\theta_n$ so that $\sum_{i=0}^{\frac{n-1}{2}} \theta_{2i} = \sum_{i=0}^{\frac{n-1}{2}} \theta_{2i+1}$
  let L_0 be the line intersecting $\mathbf{v_0}$ and $\mathbf{v_1}$
  for *i in 1..n* **do**
    let L_i be the line found by rotating L_{i-1} by θ_i so that it passes through $\frac{\mathbf{v_i}+\mathbf{v_{i+1}}}{2}$
    let $\mathbf{v'_i}$ be the intersection of L_{i-1} and L_i
  end
end

Algorithm 2: n-Arc collision test

Input: $\mathbf{c_{1,0}}..\mathbf{c_{1,n}}, \mathbf{d_{1,0}}..\mathbf{d_{1,n}}, r_{1,0}..r_{1,n}, \mathbf{C_1}$ the arc centres, directions, and radii of the first n-arc and it's centroid
$\mathbf{c_{2,0}}..\mathbf{c_{2,n}}, \mathbf{d_{2,0}}..\mathbf{d_{2,n}}, r_{2,0}..r_{2,n}, \mathbf{C_2}$ the arc centres, directions, and radii of the other n-arc and it's centroid
Output: boolean: collision occurred?

begin
  $\mathbf{C_{12}} = \mathbf{C_2} - \mathbf{C_1}$
  for *each pair of arcs i,j* **do**
    $\mathbf{c_{ij}} = \mathbf{c_{2,j}} - \mathbf{c_{1,i}}$
    if $\mathbf{c_{ij}} \times \mathbf{d_{1,i+1}} \geq 0$ *and* $\mathbf{c_{ij}} \times \mathbf{d_{1,i}} < 0$ *and* $\mathbf{c_{ij}} \times \mathbf{d_{2,i+1}} < 0$ *and* $\mathbf{c_{ij}} \times \mathbf{d_{2,i}} \geq 0$ *and* $\mathbf{c_{ij}} \times \mathbf{C_{12}} > 0$ **then**
      return $(r_i + r_j)^2 > |c_{ij}|^2$
    end
  end
end

The first criterion to be met is that the centre of each arc should lie within the extended sector defined by the other arc. In our implementation this is achieved by storing direction vectors for the radii of each arc and checking to make sure the opposing centre is on the correct sides of the two radii defining an arc. We use the 2D "cross product" for this purpose as shown in equation 1. This will return a positive value if B is to the left of A.

$$A \times B = A_x B_y - A_y B_x \quad (1)$$

The arc centres may (and frequently will) lie outside of the object, therefore using this test alone will result in false positives and negatives. To avoid this problem we also check that an arc is "facing" towards the centroid of the other n-arc. This is achieved by adding a further constraint so that no arc has a centre angle greater than 180 and checking that the dot product of the vector between arc centres and the vector between object centroids is positive. Psuedo- code for the collision algorithm is shown in algorithm 2. Note that the exhaustive search to find the correct pair of arcs may be replaced with a directed search, but this is unlikely to be more efficient unless there are a large number of arcs.

This collision test is based on the separating axes theorem (SAT). Polygon collision tests are also commonly implemented using SAT. The polygon SAT method checks each vertex against each edge of the opposing polygon as shown in algorithm 3.

Algorithm 3: n-Gon collision test using SAT

Input: $\mathbf{v_{1,0}}..\mathbf{v_{1,n}},\mathbf{p_{1,0}}..\mathbf{p_{1,n}}$ the vertices and perpendiculars of the first polygon; $\mathbf{v_{2,0}}..\mathbf{v_{2,n}},\mathbf{p_{2,0}}..\mathbf{p_{2,n}}$ the vertices and perpendiculars of the second polygon

Output: boolean: collision occurred?

```
begin
    s_min = ∞
    for each edge i in poly 1 do
        s_max = max_j[(v_1,i − v_2,j • p_2,j]
        if s_max < 0 then
            return False
        end
        s_min = min(s_min, s_max)
    end
    for each edge j in poly 2 do
        s_max = max_i[(v_2,j − v_1,i • p_1,i]
        if s_max < 0 then
            return False
        end
        s_min = min(s_min, s_max)
    end
    return True
end
```

Both SAT algorithms run in worst-case O(mn) time where m and n are the number of arcs or edges in each object. The polygon algorithm has an opportunity to exit early if a separating axis is found, but if no separating axis is found, the algorithm must run to completion. The n-arc algorithm may exit as soon as the correct pair of arcs is located regardless of whether there is a collision or not. This should give the n-arc algorithm an advantage in situations where collisions are frequent (e.g. stacked objects). However, the n- arc algorithm must perform a square root operation if a collision is found and the penetration of the two objects needs to be calculated.

An alternative method for polygon collision test is the Gilbert- Johnson-Keerthi (GJK) algorithm, shown in algorithm 4. This method is based on calculating the Minkowski difference of two convex shapes. The method searched for a simplex within the Minkowski difference that contains the origin, indicating that a collision has occurred.

To perform the search the algorithm makes use of a support function that returns a point on a shape that is furthest in the supplied search direction. For polygons, this means calculating the dot product of each vertex with the search direction vector. For n-arcs it is also very easy to implement a support function. The support function for n-arcs simply searches for the arc corresponding to the search direction and then calculates the point on the arc in that direction. The theoretical worst case running time of the GJK algorithm is $O(n^2)$ however in practice it is rare that more than a few iterations are needed giving near linear time (this can be improved further to near constant time if the simplex used from the previous frame is used to seed the algorithm).

The GJK algorithm only determines if a collision has occurred or not, it does not provide any information about the point of collision or the amount of penetration. This data is needed in order to calculate the collision response. Therefore, when a collision response is required, the GJK algorithm is enhanced by using the expanding polytopes algorithm (EPA) to calculate the collision point.

EPA works by expanding the simplex calculated using GJK. The goal is to find the nearest point on the Minkowski difference to the origin, hence this algorithm also makes use of the same support function used in GJK.

For polygons EPA requires at most O(n) iterations. Unfortunately, EPA does not work well for n-arcs since it will not terminate at an edge that cannot be ex-

panded, since the Minkowski difference of two n-arcs will have a curved perimeter. Instead a threshold must be chosen and the algorithm terminated if further expansion in the search direction is less than threshold. The performance of EPA with n-arcs is highly dependant on the size of the termination threshold.

When using either SAT of GJK, efficient implementation of the n-arc algorithm requires the centres, directions, and radii of each arc to be stored as well as the object centroids. This is slightly more than for polygons, which require storage of the vertices and perpendiculars only. The polygon perpendiculars can be easily calculated but they must be normalised, which is a significant additional cost.

Algorithm 4: The GJK algorithm

Input: $\mathbf{S_1}, \mathbf{S_2}$ the shapes to be tested;
Output: boolean: collision occurred?

begin
  $simplex = \emptyset$
  $d = Centre(S_2) - Centre(S_1)$
  $p = (Support(S_1, d) - Support(S_2, -d))$
  $simplex = simplex \cup p$
  $d = -d$
  while *True* **do**
    $p = (Support(S_1, d) - Support(S_2, -d))$
    $simplex = simplex \cup p$
    if $p \bullet d \leq 0$ **then**
      return *False*
    else
      if *ContainsOrigin(simplex)* **then**
        return *True*
      else
        $simplex = NearestSimplex(simplex)$
        $d = GetDirection(simplex)$
      end
    end
  end
end

Ⅲ. Evaluation

3.1 Computational Efficiency

To test the performance of the n-arc collisions, both the SAT and GJK algorithms for n-arcs and polygons were implemented in the Python programming language. The algorithms were tested in two scenarios, one where collisions never occurred and one where collisions always occurred. The time for 1000 collision checks was measured using the cProfile library and averaged over five trials and the results are shown in figure 5.

For polygons, both algorithms are faster when no collision occurs as expected. In both the no-collision and the collision case, the SAT algorithm is slightly faster when there are fewer polygon edges, but does not scale as well as the GJK algorithm. Scaling appears to be approximately linear in both cases for GJK, but quadratic for SAT.

For n-arcs, the no-collision case behaves similarly to polygons, but is slightly slower for both the SAT and GJK algorithms. How- ever, when collisions do occur, the n-arc SAT algorithm is significantly faster than either of the polygon algorithms for the number of arcs tested and in fact performance is similar to the no-collision case for n-arcs.

The polygon SAT algorithm is slower when a collision occurs because every vertex has to be tested against every edge - there is no early exit option for the algorithm when a collision occurs. For n-arcs early exit from the algorithm is still possible. In fact, with 10 and 12

arcs we see that the n-arc SAT algorithm is slightly faster when there is a collision compared to when there is no collision. When two n-arcs overlap significantly, there may be two or more pairs of arcs that are an (almost) equally good choice for the inter- section test. The n-arc algorithm will choose the first valid pair of arcs encountered giving multiple options for an early exit. This will be more likely to occur when there are small (corner) arcs present and when there is a large interpenetration between objects.

The polygon GJK algorithm is slower when a collision occurs because of the EPA calculations required to determine the collision point and penetration. This is significantly slower than the simple calculations required for the n-arc SAT algorithm, although it is likely that for larger numbers of arcs/edges the polygon GJK algorithm becomes faster due to improved scaling.

In contrast, the n-arc GJK algorithm exhibits very poor performance in the collision case particularly for the 4-arc shapes. This is due to the EPA part of the algorithm as explained above. The EPA algorithm stops when it cannot expand the nearest simplex edge to the origin. Since the simplex edge is a straight line and the Minkowski difference of n-arcs is a curve, the edge can be expanded indefinitely and a stopping threshold must be used. When there are more arcs GJK stops with the simplex edge closer to the origin on average, hence fewer EPA iterations are required giving the unusual

Fig. 4. Performance of N-Arcs and N-Gons When no Collision Occurs

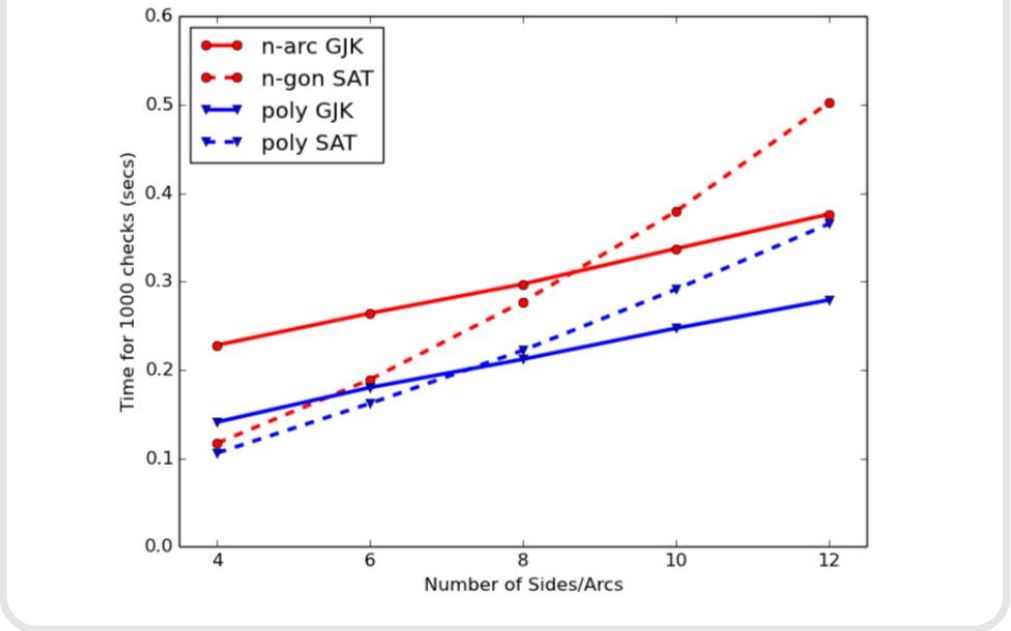


Fig. 5. Performance of N-Arcs and N-Gons When a Collision Occurs

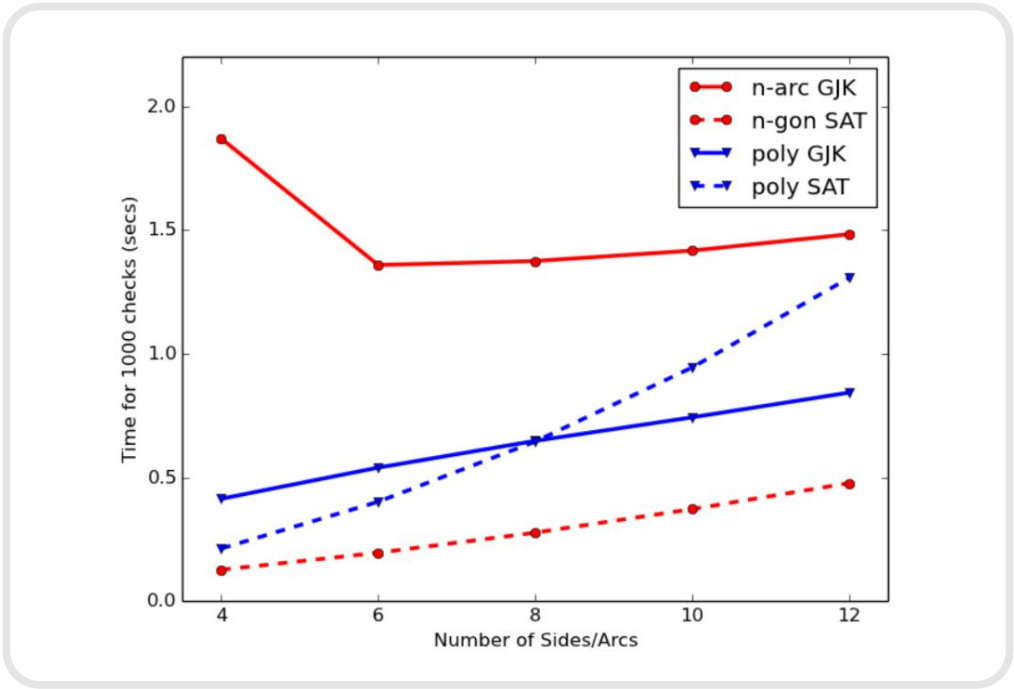
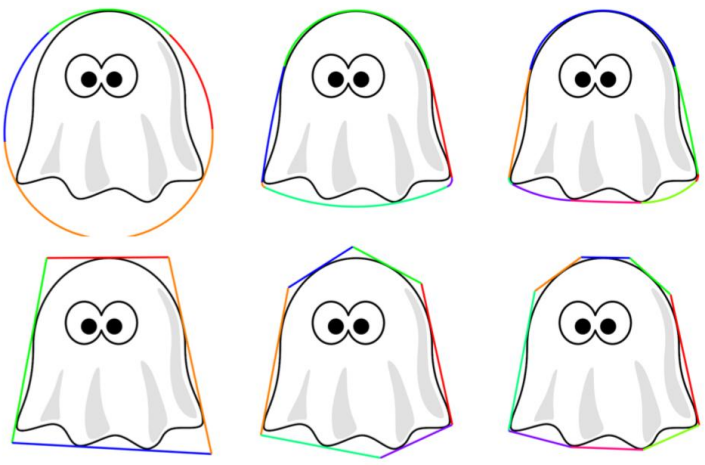


Fig. 6. Bounding Colliders and Bounded Areas for the Ghost Image. Top Row: 4-Arc, 6-Arc, 8-Arc; Bottom Row: 4-Gon, 6-Gon, 8-Gon



	4	6	8
n-arc	250709	193206	190113
n-gon	205817	193536	188046

performance curve for the n-arc GJK algorithm.

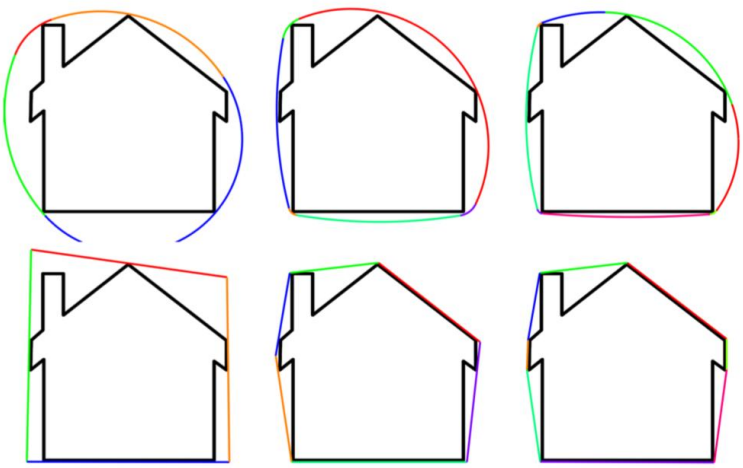
3.2 Goodness of Fit

To test the ability of n-arcs to fit a given shape, four images were chosen from the Pixabay website2. n-Arcs and n-gons were then manually matched to find a minimal bound for the image.

The first image was the ghost image shown in figure 6. This image is bounded by curves and should be a good match for n-arcs. Subjectively the 6-arc and 8-arc appear to provide a very good fit for this shape. The 6-arc produces a slightly tighter fit (smaller area) than the 6-gon, but the 8-gon gives a tighter fit than the 8-arc. How- ever, for this shape the 6- and 8-arcs may be a better choice since a tangent-continuous curve should produce a more natural collision response.

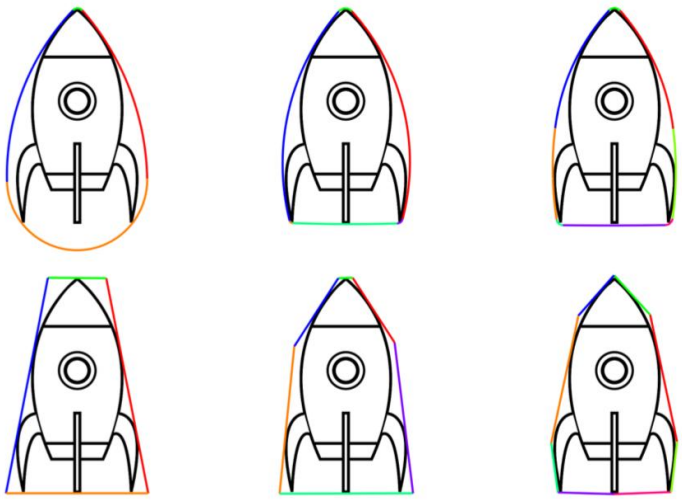
The second image was the house image shown in figure 7. This image is a closed polygon, the type of image that should be a poor choice for n-arcs. The 8-gon provides a perfect (convex) fit for this image and the areas of all the n-gon colliders are considerably less than for the corresponding n-arcs as expected. Theoretically, a better fit could be achieved for the n-arcs, but the construction of these colliders was very difficult due to the necessity for the arc control points to be very close together and precisely positioned at the corners. An automated matching procedure would probably provide a slightly better fit.

Fig. 7. Bounding Colliders and Bounded Areas for the House Image. Top Row: 4-Arc, 6-Arc, 8-Arc; Bottom Row: 4-Gon, 6-Gon, 8-Gon



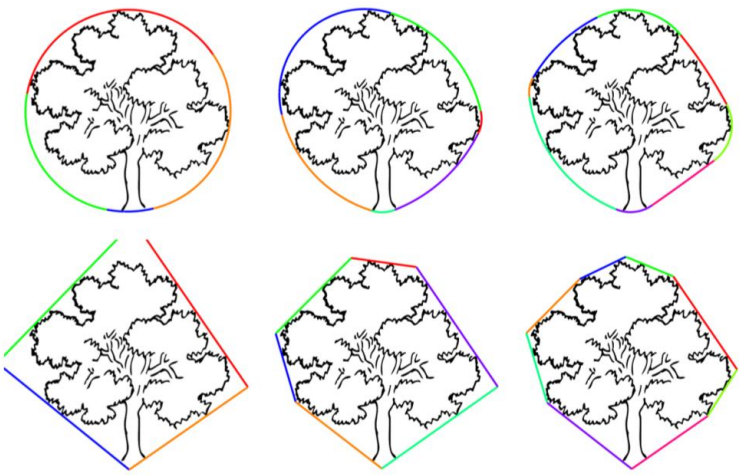
	4	6	8
n-arc	299157	255518	247934
n-gon	261466	224099	223328

Fig. 8. Top Row: 4-Arc, 6-Arc, 8-Arc; Bottom Row: 4-Gon, 6-Gon, 8-Gon. Bounding Colliders and Bounded Areas for the Spaceship Image



	4	6	8
n-arc	139253	120478	118935
n-gon	122248	120166	113624

Fig. 9. Bounding Colliders and Bounded Areas for the Tree Image.
Top Row: 4-Arc, 6-Arc, 8-Arc; Bottom Row: 4-Gon, 6-Gon, 8-Gon



	4	6	8
n-arc	212399	196603	187574
n-gon	203283	185140	180916

The third image was a spaceship constructed mostly from curves as shown in figure 8. Again, the n-arcs produce a subjectively good fit for this image, particularly the 6-arc and 8-arc. However, the areas of these colliders are slightly larger than for the 6-gon and 8-gon respectively. Despite this the n-arcs may produce a more desirable collision response depending on the application. For example, the nose of the spaceship is very well matched and this may be especially important for some applications. For other applications, the base may be more important and the n-gons give a better match here.

The final image was a tree as seen in figure 9. This is a difficult shape for both n-arcs and n-gons to match. The n-gons produce a better fit in terms of area than the corresponding n-arcs, but again for this organic shape, the n-arcs may provide a more natural collision response.

Ⅳ. Conclusion and Further Work

A novel 2D collision primitive, the n-arc, was introduced along with two algorithm for performing collision checks based on the SAT and GJK algorithms. For a small number of arcs, the SAT algorithm outperformed the GJK algorithm, especially in the advent of a collision.

The n-arc SAT algorithm exhibited slightly poorer performance to the polygon SAT and GJK algorithms when no collision occurred, but was significantly faster in the advent of a collision. Furthermore, unlike the polygon algorithms, the n-arc SAT algorithm had almost identical performance irrespective of whether a collision occurred or not. This is particularly important when a stable frame-rate is required or when there is an expectation of frequent collisions.

The ability of the new primitive to fit a range of underlying geometries was also investigated. While n-arcs often produce a slightly worse fit than a polygon of the same degree (in terms of the respective areas for the primitives), the n-arc fit may be qualitatively more satisfying for curved and organic shapes. It is also likely, though difficult to measure, that the collision response produced by a tangent-continuous curve will be more realistic and believable than that for a polygon approximating a curved surface. Furthermore, since artists often work within the bounds of the collision primitives available, having more primitives to choose from should allow greater artistic freedom.

A method for constructing n-arcs from a set of control points (algorithm 1) was also presented, this method was not particularly intuitive. An adaptation of the method proposed by Maier and Pisinger [2013] may be preferable. Alternatively, a method for converting a Beziér curves into biarcs, similar to that proposed by Risˇ kus [2006], could also be used — leveraging the existing familiarity many artist have editing Beziér curves.

Thus far only closed, convex n-arcs have been implemented. With minor modifications to the algorithm, it should be possible to test for collisions with open and/or concave objects. This is likely to result in reduced performance since multiple collision points are possible, but the advantages may outweigh the performance cost. Further work is under way to

test the feasibility of implementing these extensions to the algorithm.

References

Banchoff, T. and P. Giblin (1994), On the Geometlg of Piecewise Circular Curves, *The American Mathematical Monthly 101,* 5, 403 – 416.

Bolton, K. (1975), Biarc Curves, *Computer-Aided Design 7,* 2 (Apr.), 89 – 92.

Fu, P., O. R Walton and J. T. Harvey (2012), Polyarc Discrete Element for Efficiently Simulating Arbitrarily Shaped 2D Particles, *International Journal for Numerical Methods in Engineering*, *89*(5), 599-617.

Jimene´Z, P., F. Thomas and C. Torras (2001), 3D Collision Detection : A Survey, *Computers & Graphics 25,* 2, 269 – 285.

Krishnan, S., M. Gopi, M. Lin, D. Manocha and A. Pattekar (1998), Rapid and Accurate Contact Determination between Spline Models using ShellTrees, *Computer Graphics Forum 17,* 3 (Aug.), 315 – 326.

Krishnan, S., A. Pattekar, M. C. Lin, and D. Manocha (1998), Spherical Shell : A Higher Order Bounding Volume for Fast Proximity Queries, In *3rd Workshop on the Algorithmic Foundations of Robotics,* 177 – 190.

Larsen, E., S. Gottschalk, M. Lin and D. Manocha (2000), Fast Distance Queries with Rectangular Swept Sphere Volumes, In *IEEE International Conference on Robotics and Automation,* Ieee, 3719 – 3726.

Lin, M. C. and S. Gottschalk (1998), Collision Detection between Geometric Models : A Survey, In *Proc. of IMA Conference on Mathematics of Surfaces,* 602 – 608.

Maier, G. and G. Pisinger (2013), Approximation of a Closed Polygon with a Minimum Number of Circular Arcs and Line Segments, *Computational Geometry 46,* 3 (Apr.), 263 – 275.

Maier, G., A. Schindler, F. Janda and S. Brummer (2013), Optimal Arc-spline Approximation with Detecting Straight Sections, In *International Conference on Computational Science and Its Applications* (June), Springer, Berlin, Heidelberg, 99-112.

Risˇkus, A. (2006), Approximation of a Cubic Be´zier Curve by Circular Arcs and Vice Versa, *Information Technology and Control 35,* 4, 371 – 378.

Rosin, P. L. (1999), A Survey and Comparison of Traditional Piecewise Circular Approximations to the Ellipse, *Computer Aided Geometric Design*, *16*(4), 269-286.

Rosin, P. L. and M. L. Pitteway (2001), The Ellipse and the Five-centred arch, *The Mathematical Gazette*, *85*(502), 13-24.

Tong, Z., P. Fu, Y. F. Dafalias and Y. Yao (2014), Discrete Element Method Analysis of Non Coaxial Flow under Rotational Shear, *Inter- national Journal for Numerical and Analytical Methods in Geomechanics*, *38*(14), 1519-1540.