Collision Detection Between Two Moving Convex Objects In 2D (task overview)

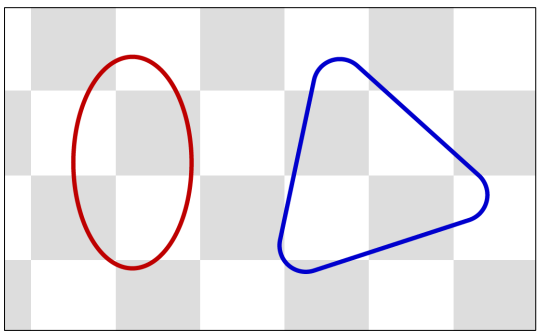
Foreword. Collision detection between two moving objects in 3D is a task arising in areas such as robot motion planning, physics simulation and other. We approach the task by first considering similar tasks in 2D which appear less complex because of lower dimensionality and less variables.

Our result: it was developed an algorithm for collision detection between two moving convex objects in 2D, including a case where one of the objects is not strictly convex.

It is going to be useful in the future work in three-dimensional setting.

Algorithm usage. In an application where objects move, it first performs computationally inexpensive tests such as bounding circles overlap. If such test does not exclude possible overlap then our collision detection algorithm runs.

Definition of convex object.



Strictly convex ellipse (left) and non-strictly convex regular N-gon with rounded corners.

Methods for defining shapes in two dimensions

We define shapes using parametric equations. It allows to express a set of points (a perimeter of the shape) as a set of values of univariate functions.

An ellipse with ½ horizontal axis length a and ½ vertical axis b is given by following equations:

$$\begin{aligned} x &= a \cos\varphi, \\ y &= b \sin\varphi, \quad -\pi \leqslant \varphi \leqslant \pi\,. \end{aligned} \tag{1}$$

In most cases, parameter φ is not equal to an angle between axis X positive direction (X+) and a ray from origin through a point at the perimeter.

Also it requires outward unit normal vector at the point:

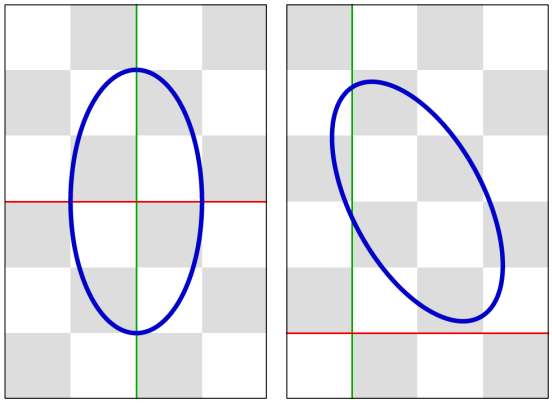
$$\vec{N} = \left(\frac{\cos\varphi}{a}, \frac{\sin\varphi}{b} \right) \Big/ \sqrt{\frac{\cos^2\varphi}{a^2} + \frac{\sin^2\varphi}{b^2}}\,. \tag{2}$$

We also use piecewise functions. An example is given in Appendix A. Example Of A Shape Defined By Piecewise Function.

Methods for positioning objects in two dimensions

We define a center of object C (a point), and rotation angle `angle`.

It exists a way to define rotation using 2x2 matrix. It may reduce amount of computation as it computes cosine and sine only once.



Left: an ellipse a=1, b=2.
Right: same ellipse translated by C=(1, 2) and rotated by `angle`=0.5.

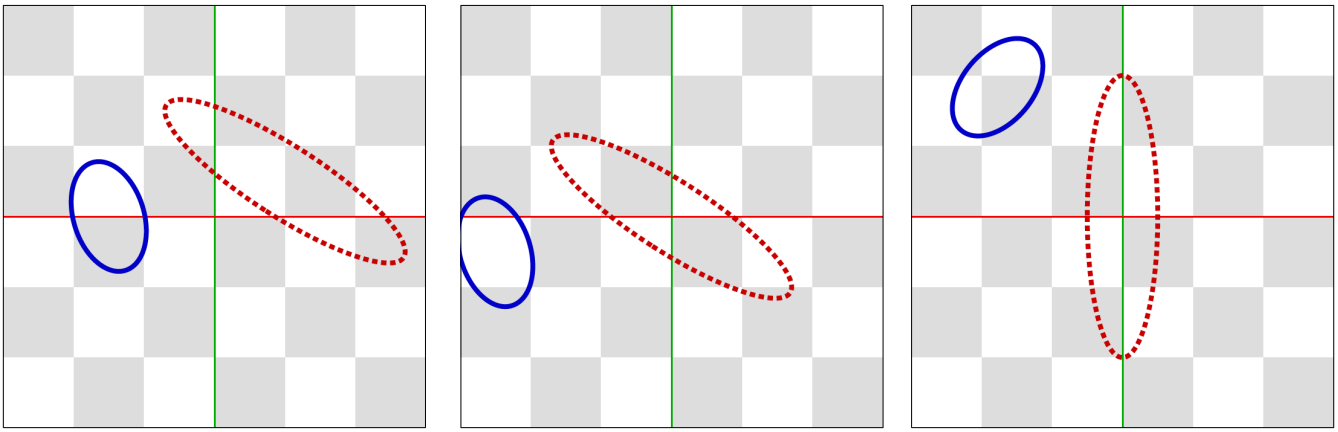
Rotation of a point.

To rotate a point (x, y) by angle θ about the origin the following formula is used:

$$\begin{aligned} \hat{x} &= x\cos\theta - y\sin\theta \\ \hat{y} &= y\cos\theta + x\sin\theta \end{aligned} \qquad (3)$$

We refer to the 1st object as to `object1`, 2nd object as to `object2`.

Coordinate transformation such that `object2` is positioned at the origin and is not rotated (`object2` local coordinate space)

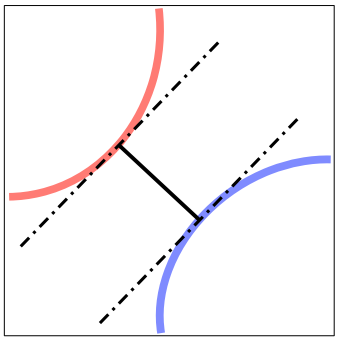


Left: `object1`, solid line, is an ellipse a_1=0.5, b_1=0.8, C_1=(-1.5, 0), `angle`$_1$=0.3 and `object2`, dotted line (an ellipse a_2=0.5, b_2=2), C_2=(1, 0.5), `angle`$_2$=1.

Center: coordinate translation by C_1-C_2, so `object2` is at the origin.

Right: rotation by -1 radian, so `object2` is not rotated.

Distance Between Two Stationary Objects



First we consider distance between two stationary objects. Same function arises between two moving objects in any fixed time *t*.

The least distance between two curves is the distance between 2 points on curves such that a line segment between 2 points is perpendicular to a curve at the point. At that points, distance derivative is zero.

We implemented following approach. Given parameter φ get point at the perimeter of `object1` and unit normal vector at the point. Then using negated unit normal vector find a point at the perimeter of `object2`. Then compute distance between two points.

Find point at the perimeter given negated unit normal vector.

It was derived a formula for an ellipse:

$$(x,\ y)\ =\ (-a^2 N_x,\ -b^2 N_y)\ /\ \sqrt{a^2 N_x^2 + b^2 N_y^2}\ . \qquad (4)$$

For non-strictly convex shapes there exist many points where a unit normal at the point equals to given unit vector. Because of that, our approach is valid only when `object2` is strictly convex.

Compute distances given φ

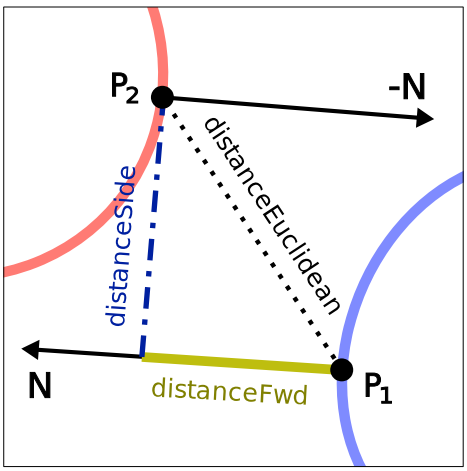
1. Given φ find point P_1 at the perimeter of `object1` and unit normal N at the point;
2. Create coordinate system where `object2` is at the origin and is not rotated (`object2` local space);
3. Transform P_1 and N into `object2` local space;
4. Using -N find point P_2 at the perimeter of `object2`;
5. Compute distance between two points P_1 and P_2.

When computing distances it also computes 1st and 2nd derivatives with respect to φ.

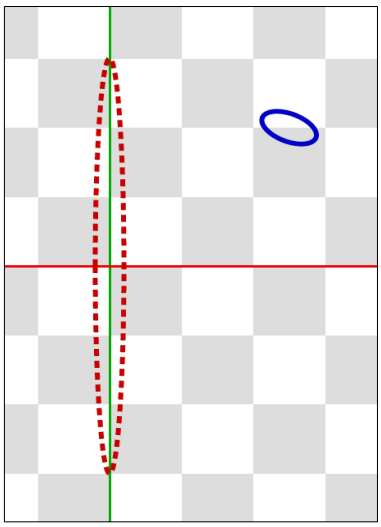
Three different distances are shown in the image to the right:

- "Usual" distance between two points along straight line on Euclidean plane – function `distanceEuclidean(φ)`
- "Forward" distance from point P_1 along vector N – function `distanceFwd(φ)`
- "Side" distance between the line formed by P_1 and N, and point P_2 – function `distanceSide(φ)`

Almost all amount of computations when computing these distances is the same.

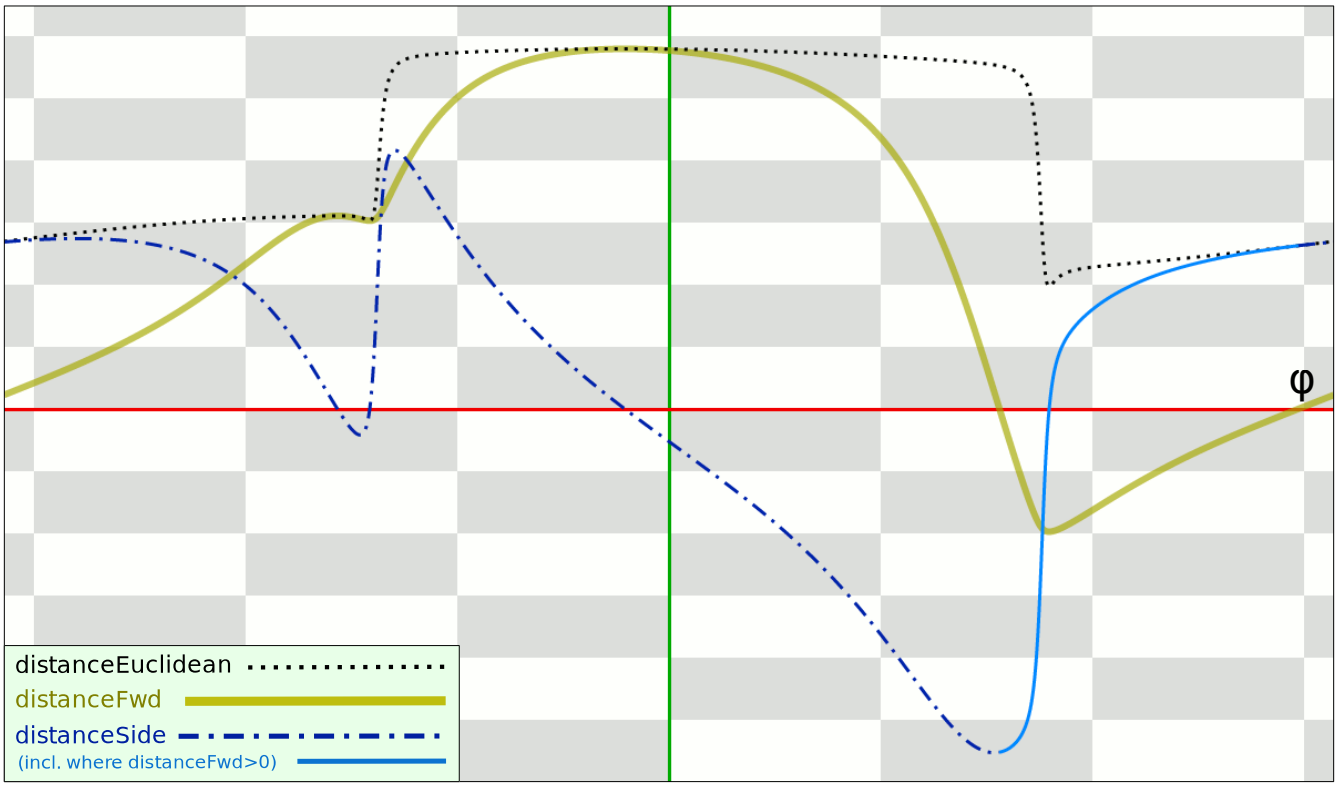


Distance functions analysis



Two objects are shown in the image to the right. Their distance functions depending on $\varphi \in [-\pi, \pi]$ are shown below. Values of `distanceFwd(`φ`)` are plotted negated for an ease of comparison. It requires to find either a minimum of `distanceEuclidean(`φ`)`, or a maximum of `distanceFwd(`φ`)`, or zero of `distanceSide(`φ`)`.

- Functions have several local extrema. Global extremum is distinctive by `distanceFwd(`φ`)` > 0.
- Function `distanceFwd(`φ`)` has substantially wider convergency intervals if compared with `distanceEuclidean(`φ`)`, meaning intervals such that if numerical method starts at any point within an interval then local extrema will be found.
- It requires to address issues about non-strictly convex objects. Please see Appendix B. Example where one object is non-strictly convex.



We implemented finding global minimum of `distanceEuclidean(`φ`)`. The algorithm is substantially based on Newton's method in optimization. If it finds local minimum where `distanceFwd(`φ`)` < 0, (that is, not possessing a property of global minimum discussed above), it continues to find other minima using bisection of intervals which have width more than defined and where the search was not performed before.

To find initial value of φ, it creates coordinates where `object1` is at the origin and is not rotated, creates vector $\vec{N}$ from the origin to the center of `object2` (equally, $\vec{N}$ is a center of `object2` in `object1` local space). Then, in case of an ellipse:

$$\varphi = \texttt{atan2}(a\ N_y,\ b\ N_x). \tag{5}$$

Also see Appendix C. Ellipses having axes ratio 1:100 and above.

Distance Between Two Moving Objects

In the algorithm we use functions which express distance dependent on φ and t: `distanceFwd(`φ, t`)` and `distanceSide(`φ, t`)`. To compute function values, it executes the following:

1. Compute positions `object1` and `object2` at time t;
2. Perform computations listed in "Compute distances given φ";

Along function values it computes order 1 and 2 derivatives with respect to φ and t:

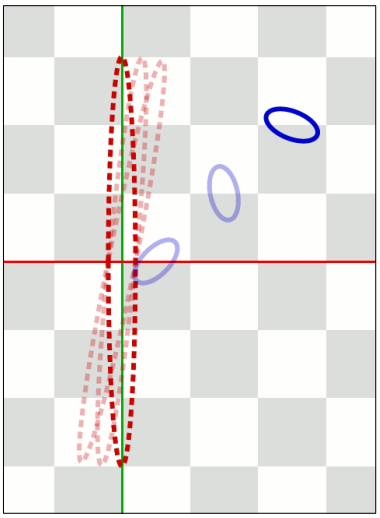
$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial \varphi} \\ \frac{\partial f}{\partial t} \end{bmatrix}; \qquad \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial \varphi^2} & \frac{\partial^2 f}{\partial \varphi \partial t} \\ \frac{\partial^2 f}{\partial t \partial \varphi} & \frac{\partial^2 f}{\partial t^2} \end{bmatrix}. \tag{6}$$

We implemented the following algorithm:

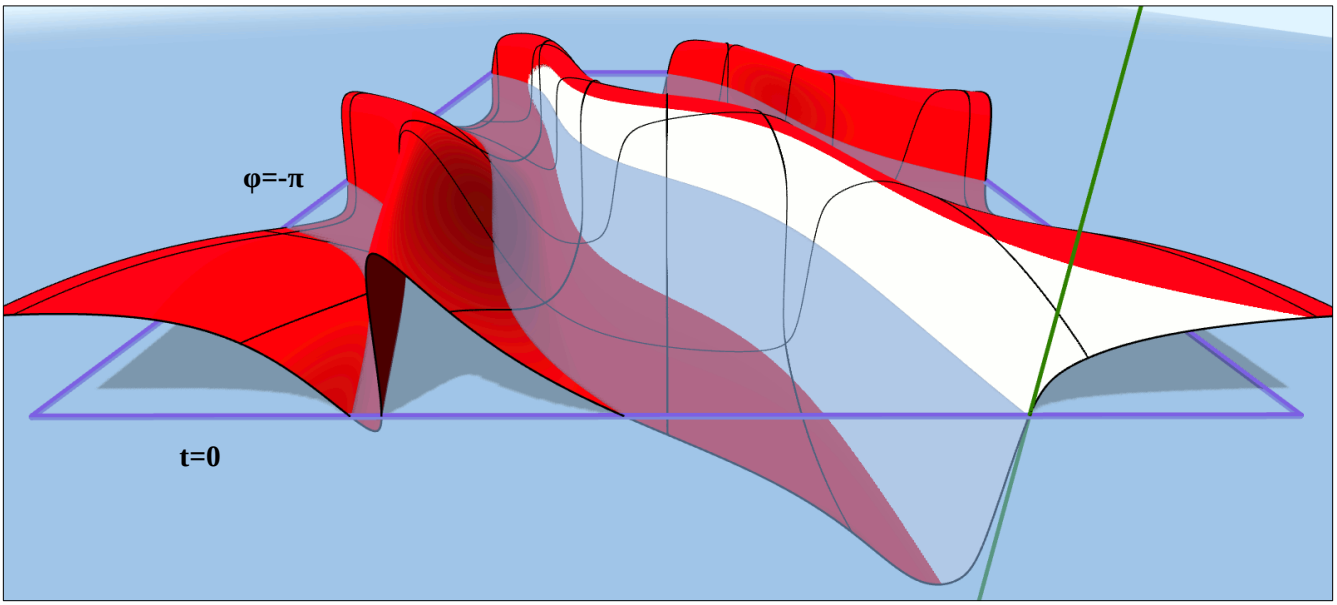
1. Determine φ and distance between two objects when t=0;
2. Perform algorithm steps with increase of t, with correction of φ so the parameter matches minimum distance.

Termination criteria:

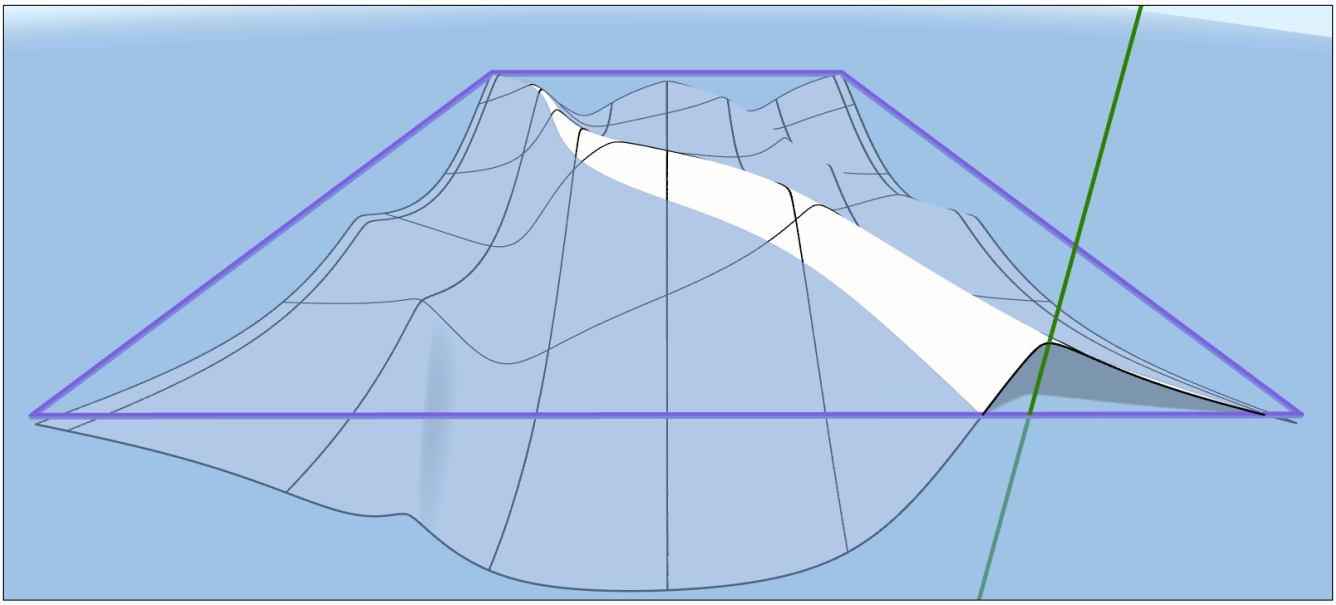
1. Objects collide. Objects are determined as being in collision when distance between them reaches some value (e.g. $(1..4)\cdot 10^{-6}$). It does not allow object intersection.
2. Distance increases enough to reach computationally inexpensive criterion which excludes possibility for a collision. It could be bounding circles no longer overlap.



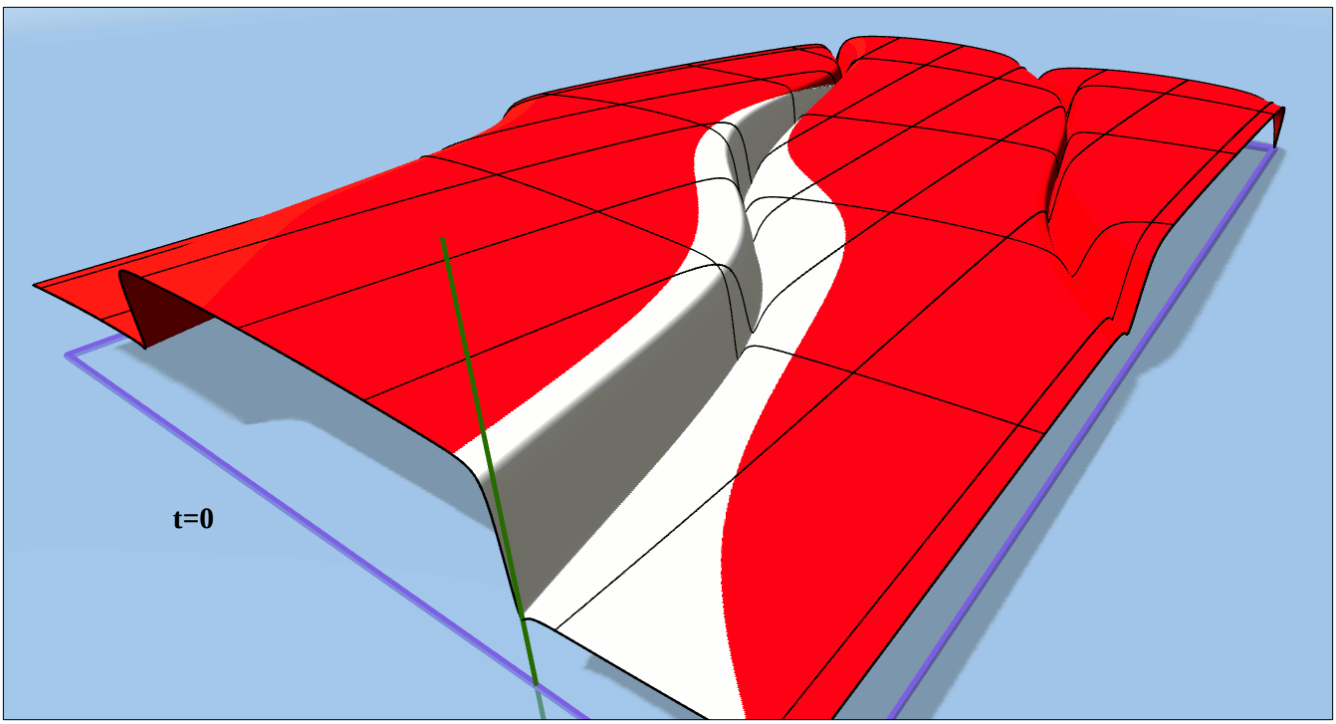
Objects at t=0, t=1, t=2.



Function `distanceSide(`φ, t`)`.



Function `distanceFwd(`φ, t`)`.



Function `distanceEuclidean(`φ, t`)`. Using this function is found to be less beneficial: on average its intervals of convergence are more thin, numerical method takes more iterations.

One step of the algorithm

It is still in the stage of the development however key ingredients are already understood.

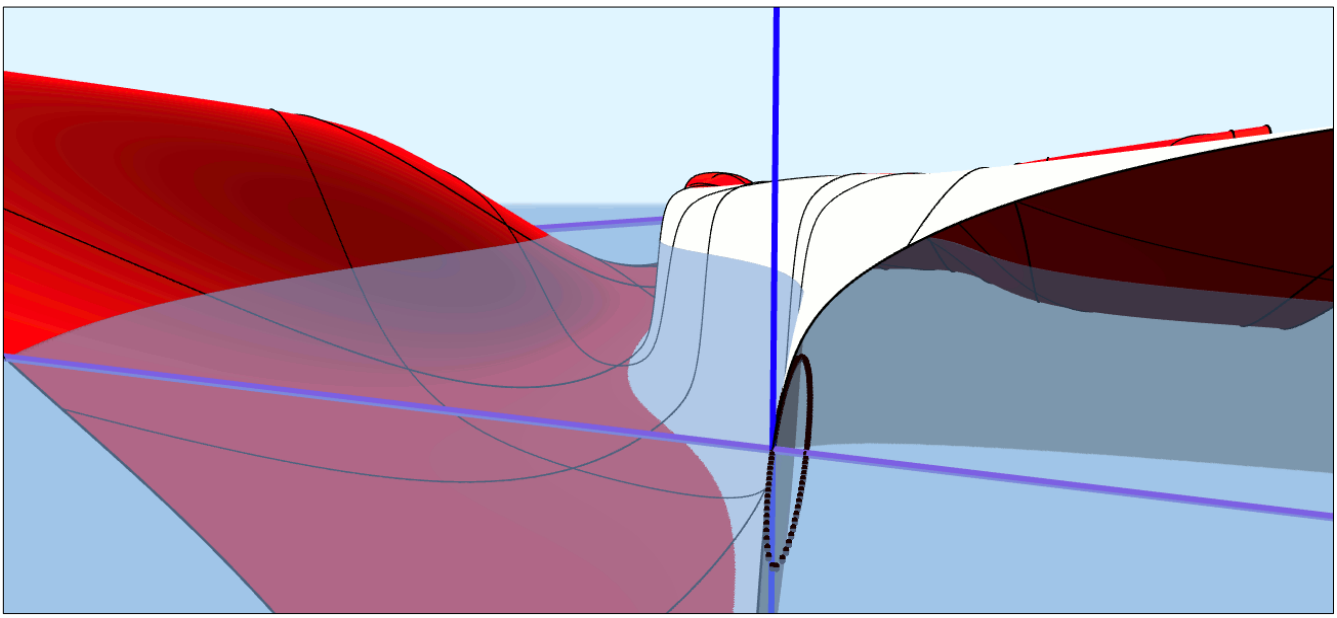
Algorithm proceeds along zero level of `distanceSide(`φ, t`)`. In contrast with movement along extrema it allows for 1 additional order of approximation.

In the neighborghood of a point (φ_0, t_0) the function is expressed as order 2 Taylor polynomial:

$$P = \begin{bmatrix} \varphi - \varphi_0 \\ t - t_0 \end{bmatrix}; \qquad f(\varphi, t) = f(\varphi_0, t_0) + P^T \nabla f(\varphi_0, t_0) + \frac{1}{2} P^T \nabla^2 f(\varphi_0, t_0) P \ . \tag{7}$$

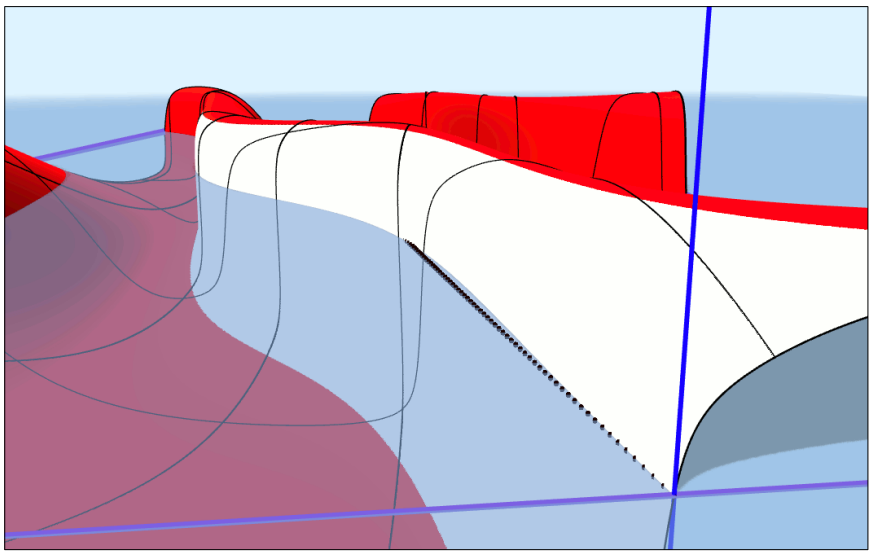
An equivalent notation without matrices or operators:

$$\begin{aligned} f(\varphi, t) = f(\varphi_0, t_0) + \frac{\partial f}{\partial \varphi}(\varphi - \varphi_0) + \frac{\partial f}{\partial t}(t - t_0) + \\ \frac{1}{2}\frac{\partial^2 f}{\partial \varphi^2}(\varphi - \varphi_0)^2 + \frac{\partial^2 f}{\partial \varphi \partial t}(\varphi - \varphi_0)(t - t_0) + \frac{1}{2}\frac{\partial^2 f}{\partial t^2}(t - t_0)^2. \end{aligned} \tag{8}$$



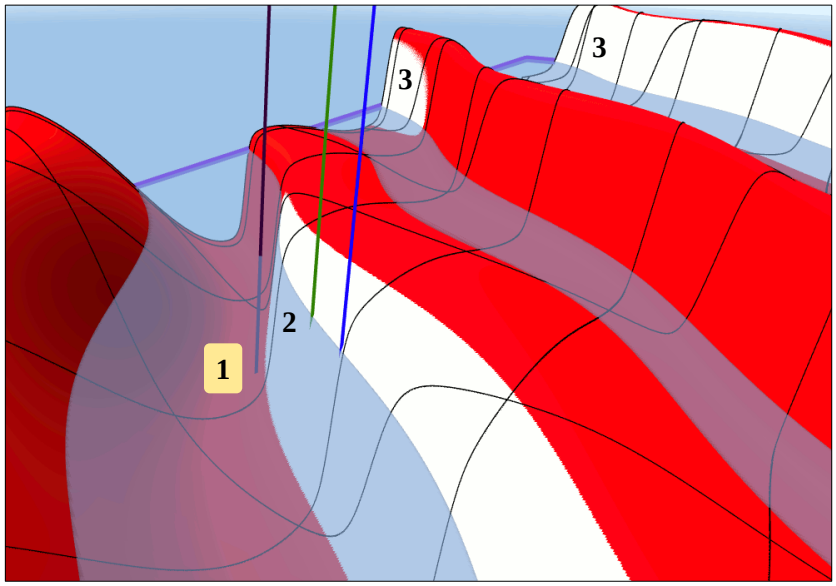
It results in order 2 algebraic curve. As the curve bends it may bend enough for a movement along the curve to head in the direction opposite to the increase of *t*.

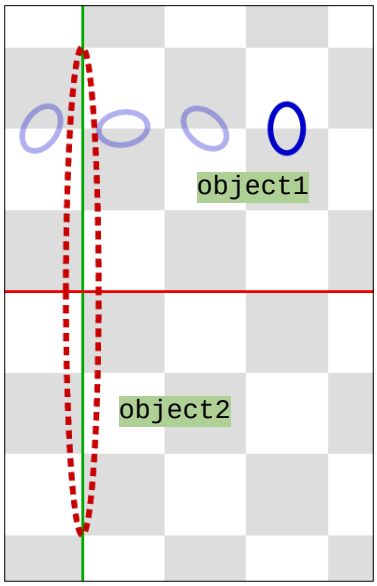
This is one of factors limiting algorithm step length.



There is an approximation with a straight line, for a comparison.

Backtracking

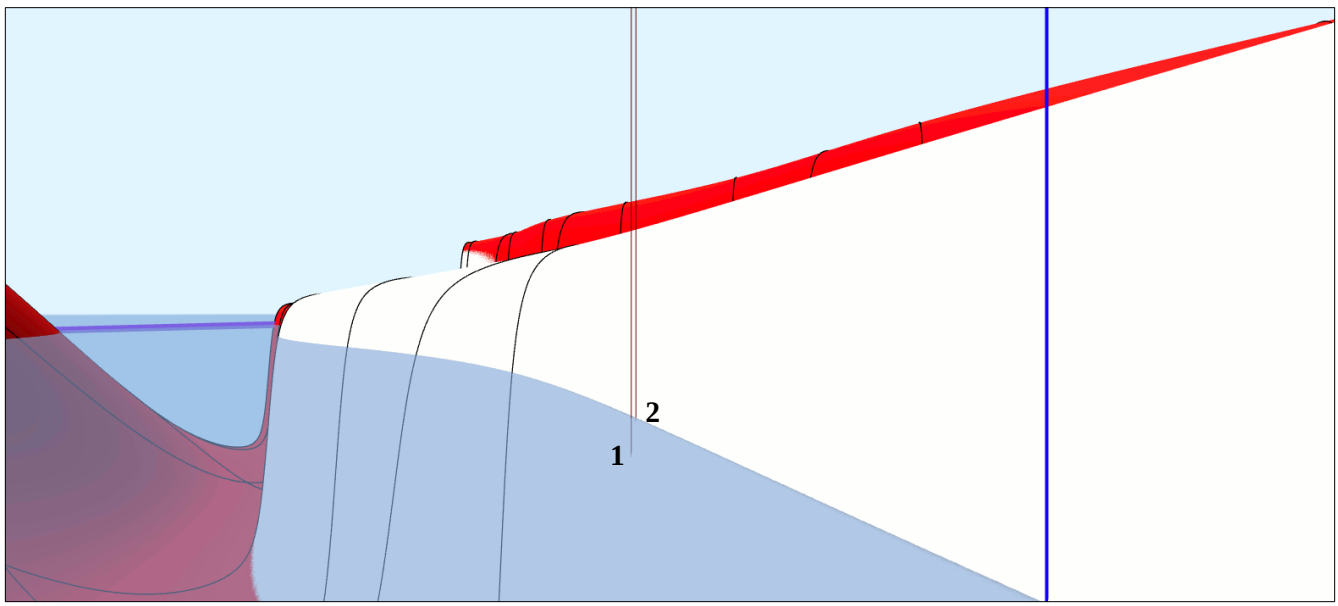




1. Algorithm step appears in the point which does not satisfy criteria.
2. Backtracking performed.
3. An area where formally algorithm can move, it is not connected to the area where algorithm steps are going. It corresponds to the movement of `object1` (see image to the right) in case after it passes through `object2`.

Correction φ to let it zero

After algorithm step is performed it is often the case where function value is not zero.

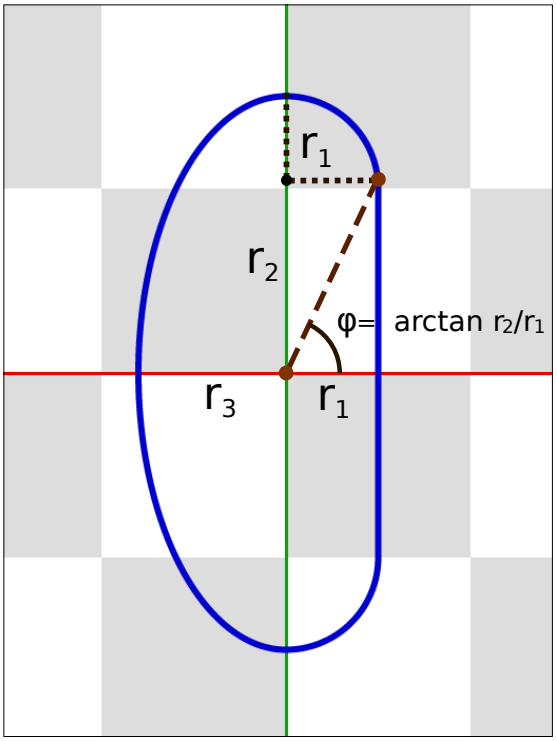


1. Algorithm step is performed. `distanceSide(`φ, t`)` $\approx$ -0.2
2. A correction step using same t is performed. `distanceSide(`φ, t`)` $\approx$ -0.0007
3. It requires more correction. After one more correction step (not shown), `distanceSide(`φ, t`)` $\approx -3 \cdot 10^{-11}$, algorithm step is considered as completed.

For correction steps it reuses object positions. Also it does not require derivatives with respect to t. Halley's method is applied.

Appendix A. Example Of A Shape Defined By Piecewise Function

Shape `shape1` (see image to the left) is determined by three parameters r_1, r_2, r_3. It has continuous perimeter and normal vector. The domain of definition $-\pi \leqslant \varphi \leqslant \pi$ consists of several intervals:



Shape `shape1`, its parameters are r_1=0.5, r_2=1, r_3=0.8.

- If $|\varphi| > \pi/2$ then it is a part of an ellipse and is given by the same equations as an ellipse plugging $a=r_3$, $b=r_1+r_2$.
- If $|\varphi| \leqslant \pi/2 \ \vee \ |\varphi| \geqslant \arctan(r_2/r_1)$ then it is a part of a circumference with its center at the point $(0, \ \pm r_2)$ and radius r_1. A point at the perimeter is given by equations:

$$\begin{aligned} d &= r_2|\sin\varphi| + \sqrt{r_1^2 - r_2^2\cos^2\varphi}\,, \\ x &= d\,\cos\varphi\,, \\ y &= d\,\sin\varphi\,. \end{aligned} \qquad (9)$$

Normal vector at the point:

$$(N_x, N_y) = \left(\frac{x}{r_1}, \ \mathrm{sgn}(y)\,\frac{|y| - r_2}{r_1}\right) \qquad (10)$$

It requires unit vector (of length 1). It computes vector length as $\texttt{length} = \sqrt{(N_x^2 + N_y^2)}$ then divides vector components by its length.

- If $|\varphi| < \arctan(r_2/r_1)$ then the part of the perimeter is vertical line segment. Point coordinates are given by:

$$(x, y) = (r_1, \ r_1 \tan\varphi)\,. \qquad (11)$$

Unit normal vector on this interval is independent of φ. It equals to:

$$\vec{N} = (1, 0)\,. \qquad (12)$$

Because $\vec{N}$ is not dependent on φ, derivatives of vector componets $\frac{d}{d\varphi}N_x(\varphi)$, $\frac{d}{d\varphi}N_y(\varphi)$ are zero.

To summarize it, for subsequent computations in collision detection it requires:

- Point coordinates by φ;
- Unit normal vector at the point by φ;
- 1st and 2nd derivatives of the above quantities (totaling 4 scalars) with respect to φ.

Derivatives are computed alongside function values using automatic differentiation.

On some intervals some derivatives are zero. Also at interval endpoints $\varphi = \pm\arctan(r_2/r_1)$, $\varphi = \pm\pi/2$ some derivatives are discontinuous, that is some of 4 functions are not continuously differentiable. Function values are all continuous on entire domain of definition.

Check Using Computer Algebra System (CAS) “Wolfram Alpha”

Let $r_1 = 0.5$, $r_2 = 1$, $r_3 = 0.8$. Then $\arctan(r_2/r_1) \approx 1.10715$. It seems like this CAS has no ability to plot piecewise parametric function or to plot several parametric functions at once.

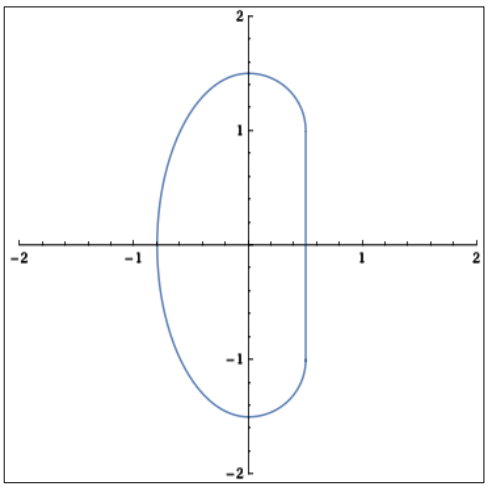
We performed several plots providing different intervals for *t*, while maintaining same ranges for x,y and same aspect ratio x/y:

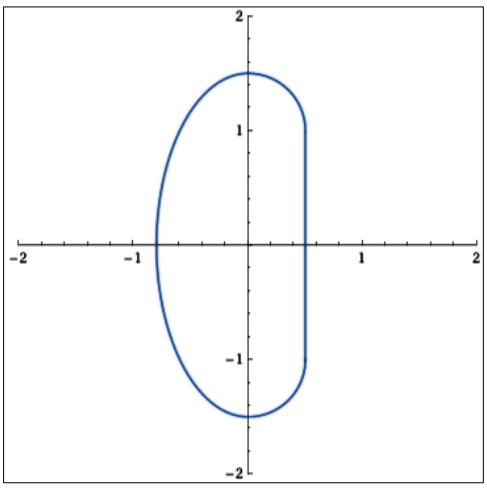
```
parametric plot x=0.8 cos t, y=1.5 sin t, {t,-3/2 pi,-pi/2}, plotrange ((-2,2),(-2,2)), AspectRatio 1

parametric plot x=cos t(|sin t|+sqrt(0.25-cos^2 t)), y=sin t(|sin t|+sqrt(0.25-cos^2 t)), {t,1.10715,pi/2}, plotrange ((-2,2),(-2,2)), AspectRatio 1

parametric plot x=cos t(|sin t|+sqrt(0.25-cos^2 t)), y=sin t(|sin t|+sqrt(0.25-cos^2 t)), {t,-pi/2,-1.10715}, plotrange ((-2,2),(-2,2)), AspectRatio 1

parametric plot x=0.5, y=0.5 tan(t), {t,-1.10715,1.10715}, plotrange ((-2,2),(-2,2)), AspectRatio 1
```



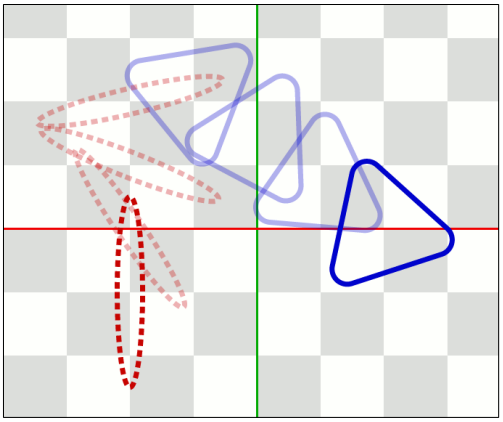


Screenshots from several plots were combined using gimp: first arranged in several layers then removed background from upper layers using Layer → Transparency → Color to Alpha.

The result (left image) is what was expected.

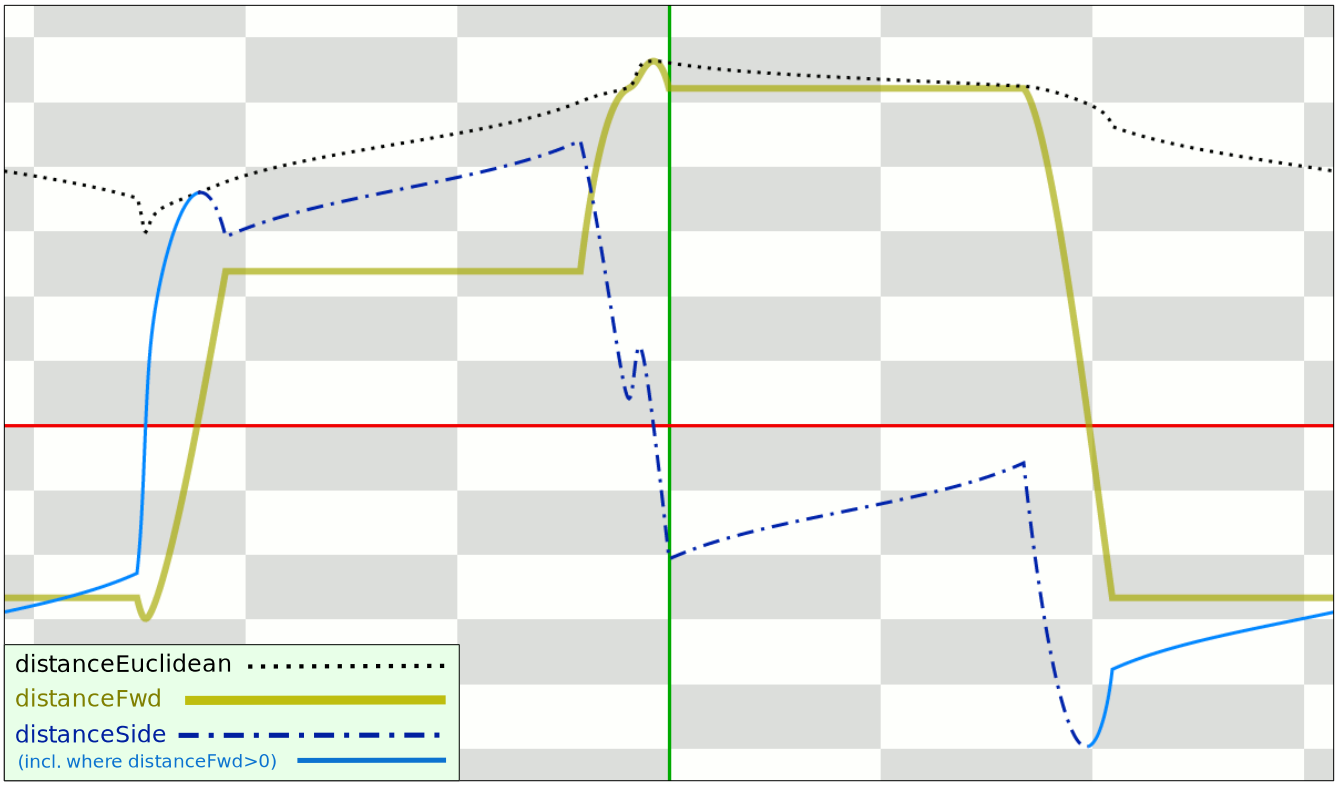
Right image: additional treatment with gimp.

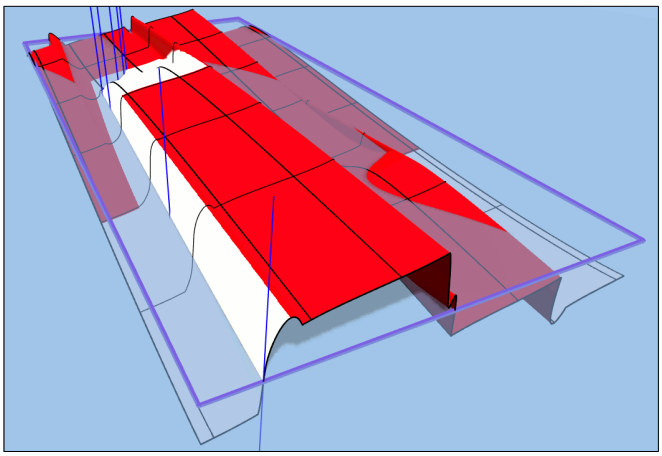
Appendix B. Example where one object is non-strictly convex



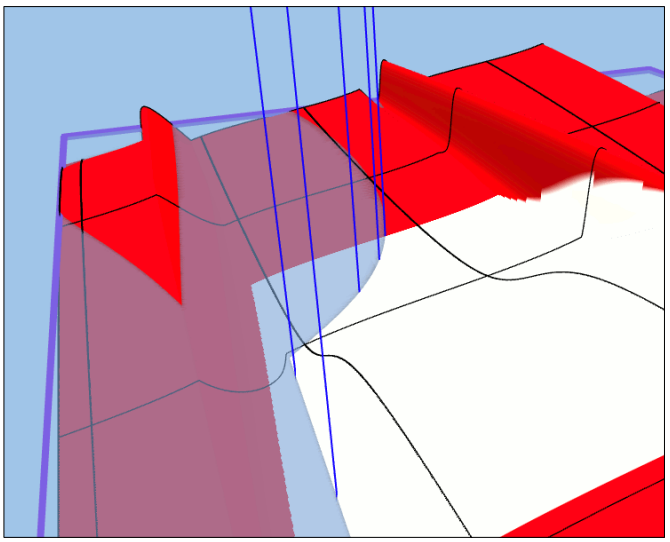
In the plot at t=0 (below) there are intervals where the 1st and the 2nd derivatives are zero. Newton's method in optimization does not work on such intervals.

Also there are points of discontinuity of the 1st derivative.





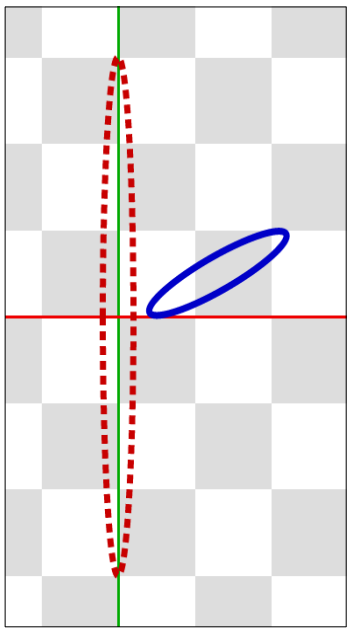
`distanceSide(`φ, t`)`.



Right: fragment of 3D plot in the neighborhood of the collision point. The point belongs to non-strictly convex interval of the perimeter of `object2`.

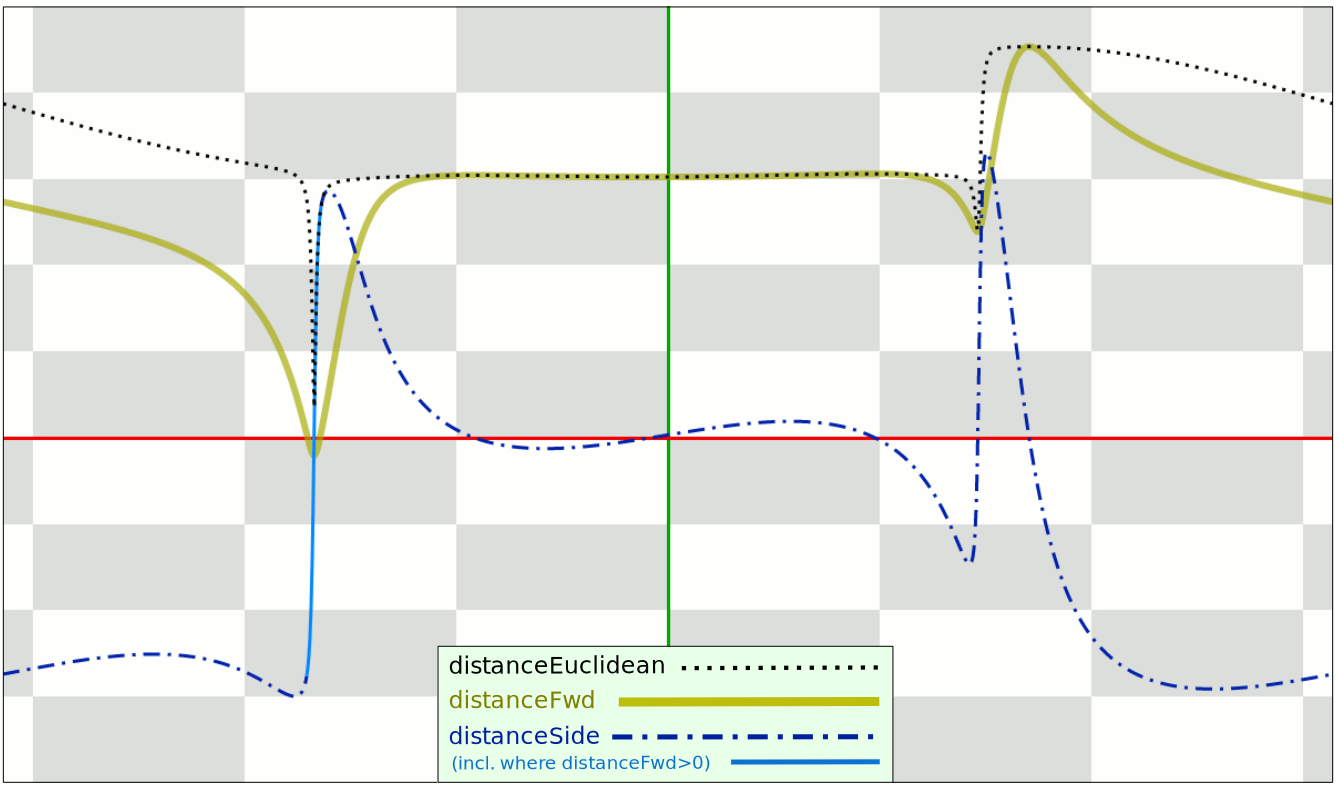
Appendix C. Ellipses having axes ratio 1:100 and above

To illustrate difficulties arising when finding closest distance between stationary objects at t=0 we consider two ellipses having axes ratio 1:5 and 1:15 (image to the right).



In the plot below, `distanceEuclidean(`φ`)` has 3 local minima. The minimum at $\varphi \approx 0$ has large radius of convergence.

Global minimum has radius of convergence approximately 0.2 – 0.3. If axes ratio increases to 1:100 then radius of convergence reduces to approximately 0.01 and the corresponding peak in the plot has width less than 1px.

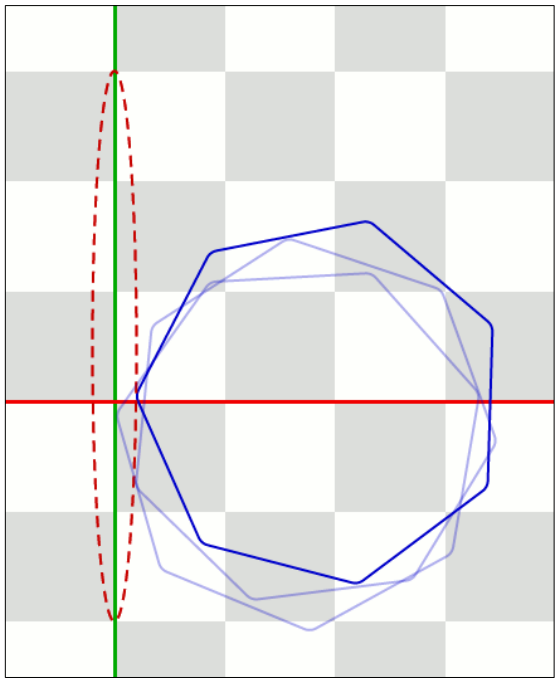


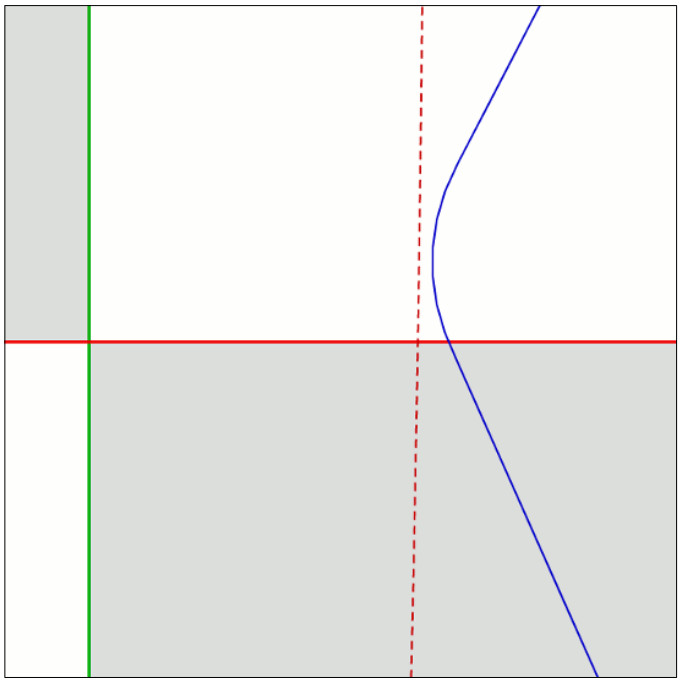
Appendix D. Object Rotates at 30 radian/s

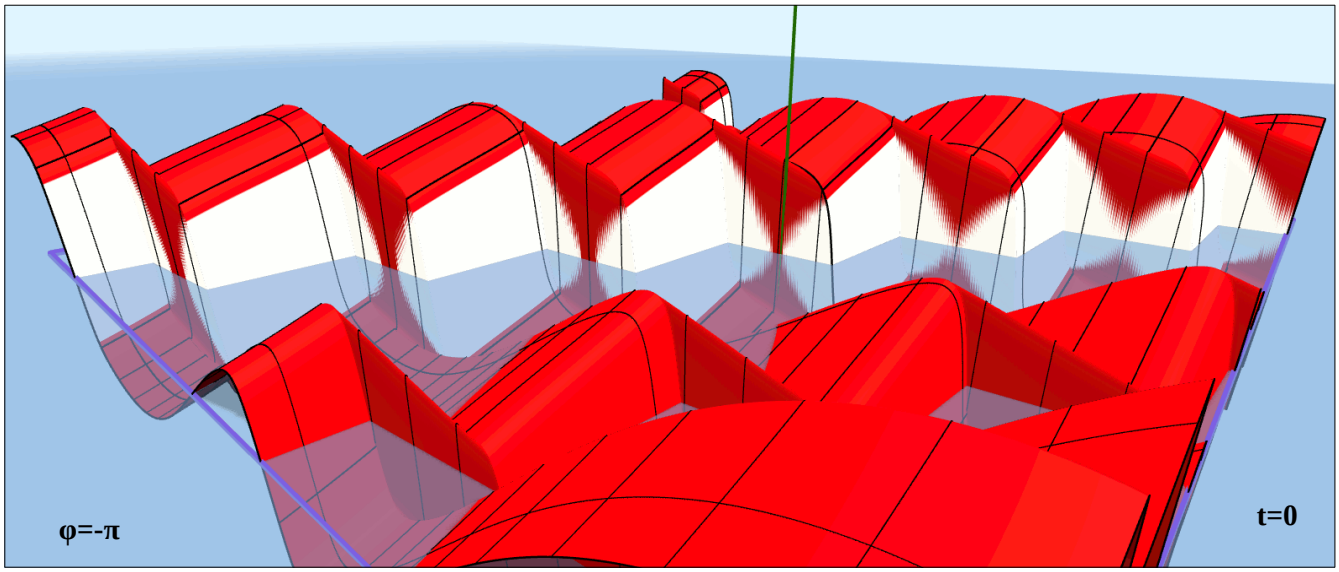
In applications there may be a requirement to handle substantially much faster rotation.

`object1` is a regular heptagon with rounded corners. It rotates and also has slow linear motion towards `object2`. It takes approximately half-round until the collision with `object2` occurs.

Right: same plot near the origin scaled 20:1.







`distanceSide(`φ, t`).`