# Gilbert-Johnson-Keerthi (GJK) Algorhytmus

Contents

[Gilbert-Johnson-Keerthi (GJK) Algorhytmus 1](#_Toc210242169)

[1. Goals 1](#_Toc210242170)

[2. Basic Principle 1](#_Toc210242171)

[3. Algorithm sequence 1](#_Toc210242172)

[4. Implementation in Python 2](#_Toc210242173)

[5. Properties 3](#_Toc210242174)

[6. Visualization (2D) 3](#_Toc210242175)

[7. Sources 4](#_Toc210242176)

# Goals

* Detection of collisions between convex bodies (fast)
* With Extensions: calculate distances between objects
* Application: Physics engines in games, Robotics, CAD systems

# Basic Principle

**Minkowski difference**

Two bodies collide if the origin (0,0,0) lies inside the Minkowski difference.

**Support-Function**Instead of computing the full Minkowski set, GJK uses a support function:For direction 𝑑:

returns the outermost point of the difference in direction 𝑑.

# Algorithm sequence

1. Select an **initial search direction** (e.g., to the right).
2. **Calculate support point**
3. If the dot product → **no collision**.
4. Add the point to the **simplex** (set of up to 3 points in 2D).
5. Update direction towards the origin and repeat.
6. Check simplex cases:
   1. 1 point → Change direction
   2. 2 points → Check edge
   3. 3 points → Check triangle (2D)
   4. 4 points → Check tetrahedron (3D)
7. Termination criteria:
   1. Origin inside simplex → **collision detected**
   2. No further improvement after several iterations → **no collision**

# Implementation in Python

* 1. **Support Function**

Finds the farthest vertex of a convex shape in a given direction using the dot product.

def support(shape, direction):   
dots = np.dot(shape, direction)  
return shape[np.argmax(dots)]

* 1. **Minkowski Support Function**

Computes the **support point of the Minkowski difference**.  
This means: the farthest point of shape A in direction *d*, minus the farthest point of shape B in the opposite direction.

def support\_minkowski(A, B, direction):  
return support(A, direction) - support(B, -direction)

* 1. **GJK Main Loop**

def gjk\_collision\_detection(A, B, max\_iterations=20):  
simplex = []  
direction = np.array([1.0, 0.0]) # initial direction  
for \_ in range(max\_iterations):  
 point = support\_minkowski(A, B, direction)  
 if np.dot(point, direction) <= 0:  
 return False  
 simplex.append(point)  
 collision, direction = handle\_simplex(simplex)  
 if collision:  
 return True  
return False

The main GJK algorithm:

* Starts with an initial search direction.
* Iteratively adds support points to the simplex.
* Updates the search direction towards the origin.
* If the simplex eventually contains the origin → **collision detected**.

# Properties

**Advantages**:

* Very efficient, even for real-time physics.
* Any dimensions possible.
* Only support function required → no complete Minkowski object.

**Limitations**:

* Only works for convex bodies.
* For non-convex objects → decomposition into convex parts.
* Only returns yes/no. Extensions are required for exact distance (e.g., EPA – Expanding Polytope Algorithm).

# Visualization (2D)

* 1. **No Collision**

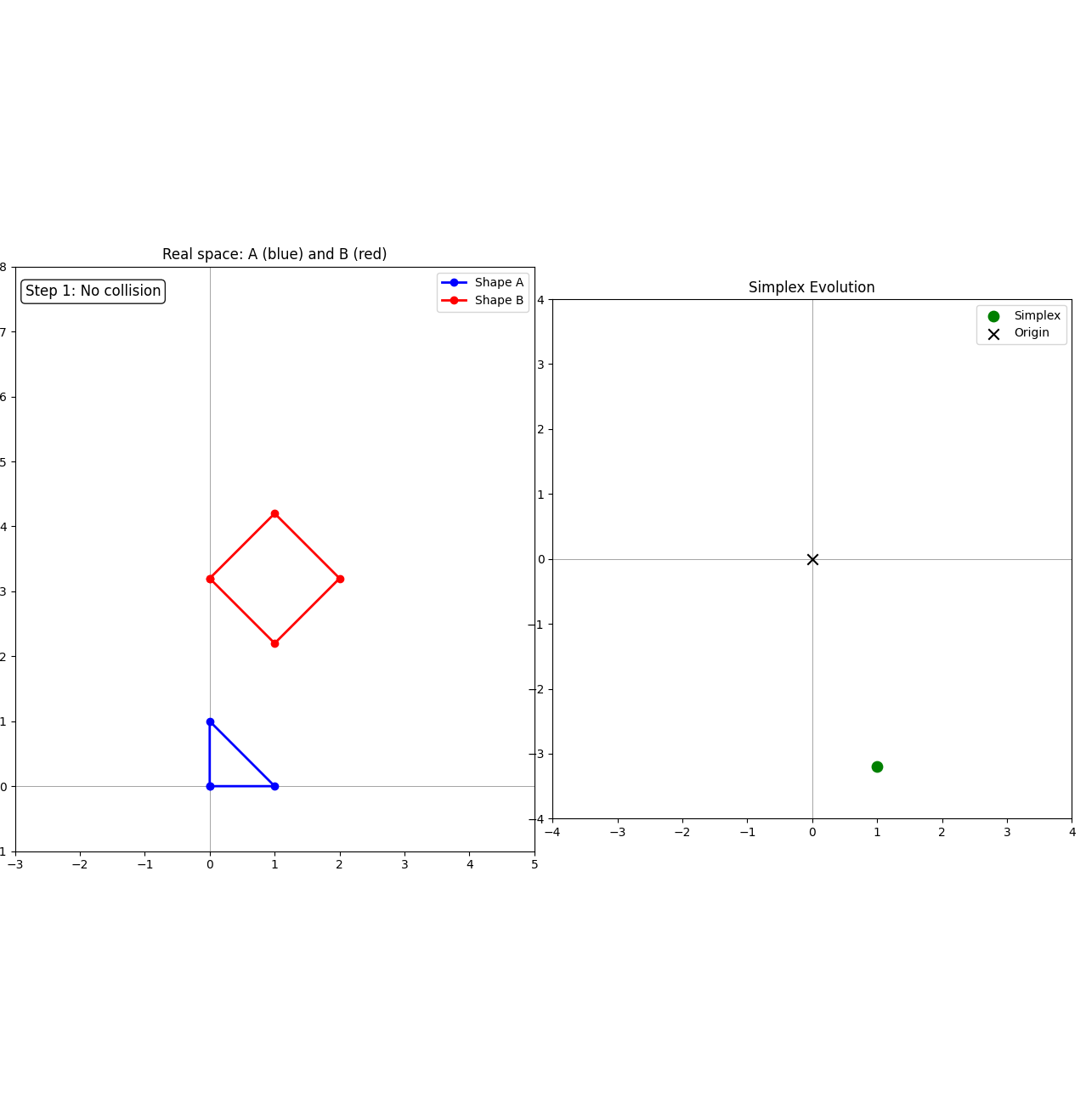


Figure 1 - no collision - the simplex does not enclose the origin

The left panel shows two convex shapes in real space:

* Shape A (blue), a triangle, and
* Shape B (red), a diamond.

They do not overlap, so there is no collision.

The right panel illustrates the simplex evolution in Minkowski space.

* The green point represents the current simplex point.
* The black cross marks the origin.
* Since the simplex does not enclose the origin, the algorithm concludes that no collision is detected.
  1. **Collision**

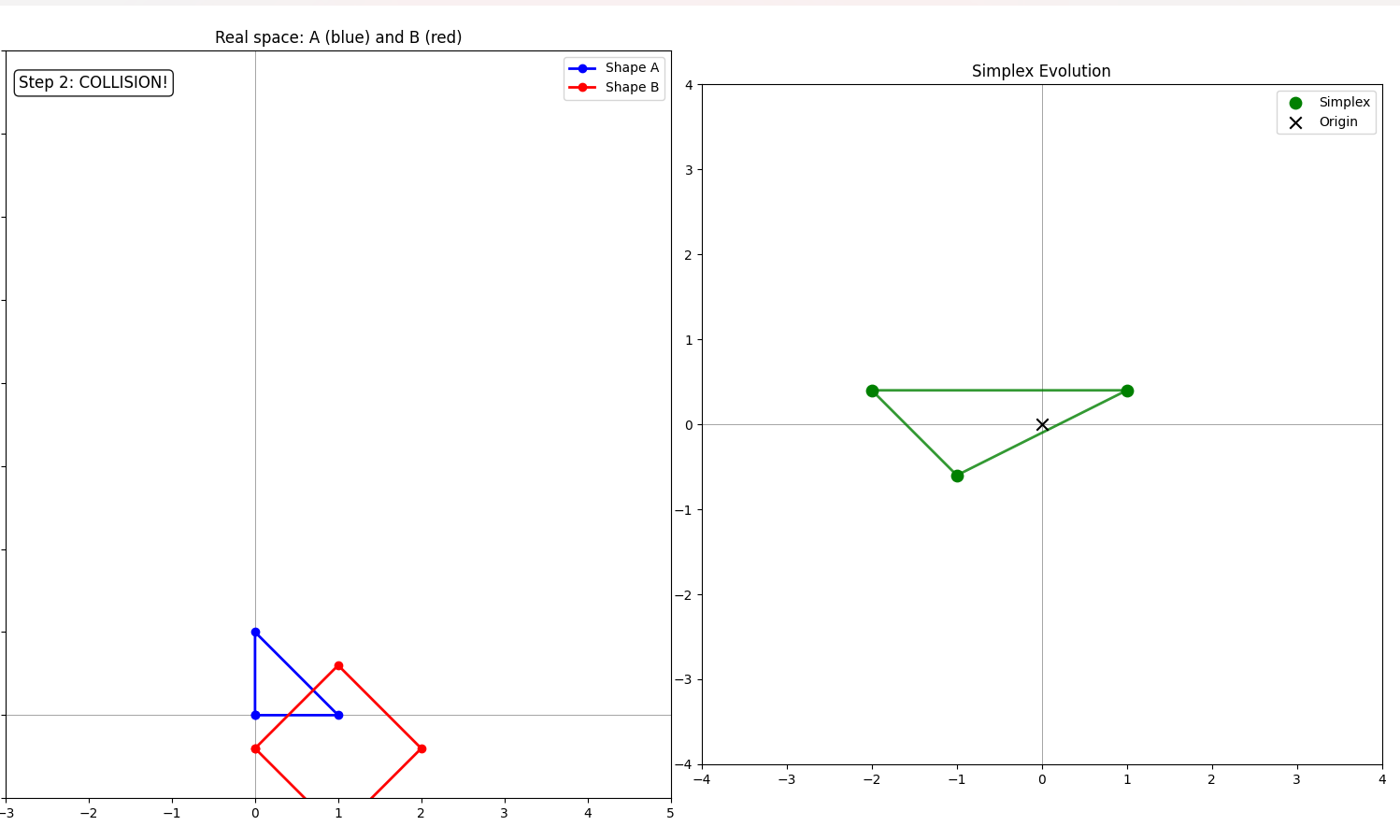


Figure 2 - collision – the simplex contains the origin.

The left panel shows the similar convex shapes as in 6.1. This time the shapes overlap, which indicates a collision.

The right panel visualizes the simplex evolution in Minkowski space.

* The green points represent the simplex vertices.
* The black cross marks the origin.
* Since the simplex encloses the origin, the algorithm correctly concludes that a collision has occurred.

# Sources

Gilbert, E. G., Johnson, D. W., & Keerthi, S. S. (1988). *A fast procedure for computing the distance between complex objects in three-dimensional space.* IEEE Journal of Robotics and Automation.