

Narrow Phase



Collision Detection

Lesson

06

Lecture 06 Outline

- * Problem definition and motivations
- * Proximity queries for convex objects
 - Minkowski space, CSO, Support function
- * GJK based algorithms (GJK, EPA, ISA-GJK)
- * Voronoi Clipping Algorithm (V-Clip)
- * Signed Distance Maps for collision detection
- * Demos / tools / libs

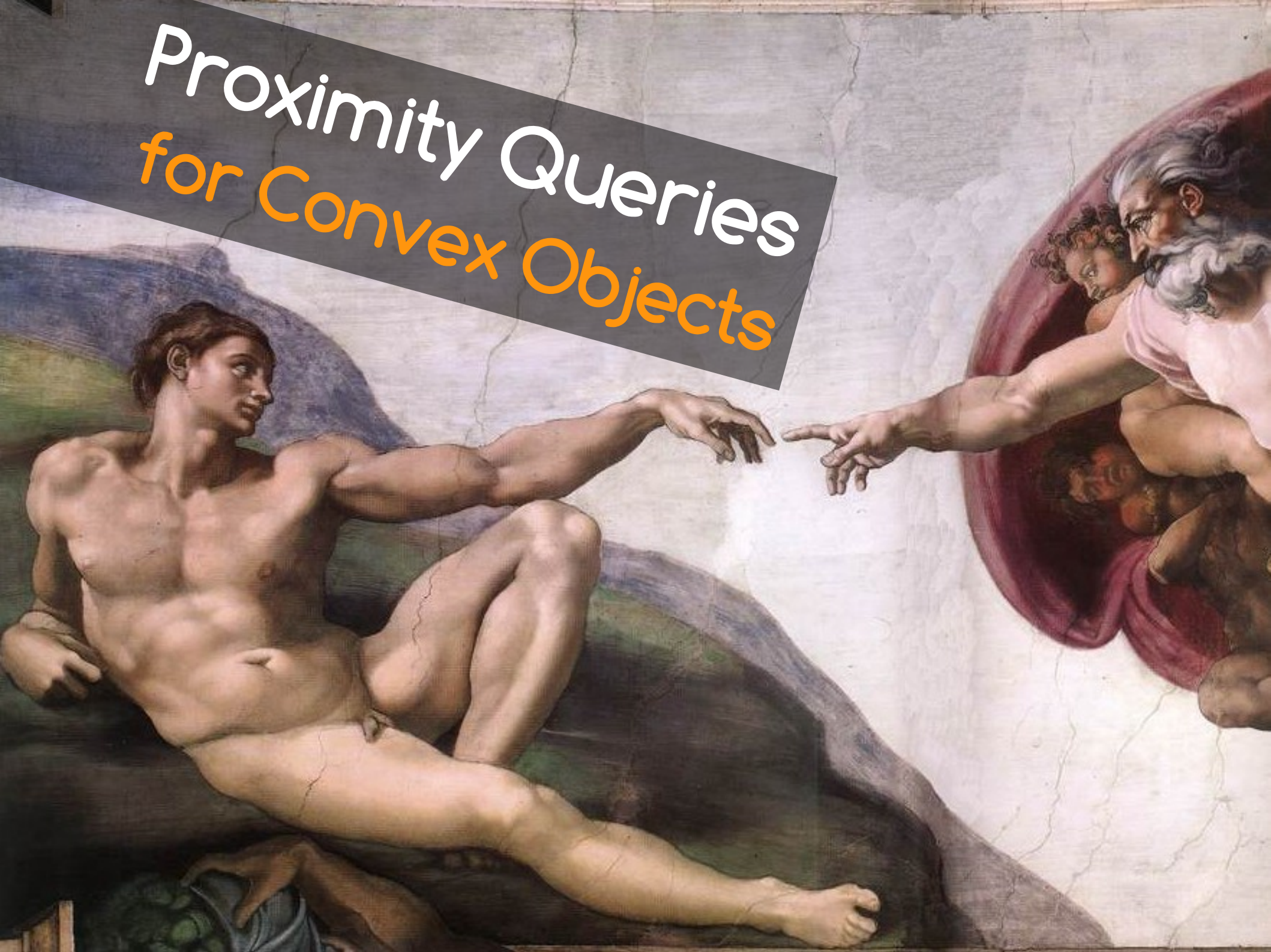
Narrow-Phase Collision Detection

- ★ **Input:** List of pairs of potentially colliding objects.
- ★ **Problem1:** Find which sub-objects are really intersecting and remove all non-colliding pairs.
- ★ **Problem2:** Determine the proximity/contact information, i.e. exact points where objects are touching (interpenetrating), surface normal at that contact point and separating / penetrating distance of objects.
- ★ **Problem3:** Recognize persistent contacts, i.e. topologically equivalent contacts from previous time steps

Narrow-Phase Collision Detection

- * **Output:** List of contact regions with necessary proximity information between colliding objects
- * **Strategies:**
 - Simplex based traversal of CSO – GJK based algorithms
 - Feature tracking base algorithms as Lin-Canny or V-Clip
 - Signed Distance Maps for collision detection
 - Persistent clustering for contact generation and reduction

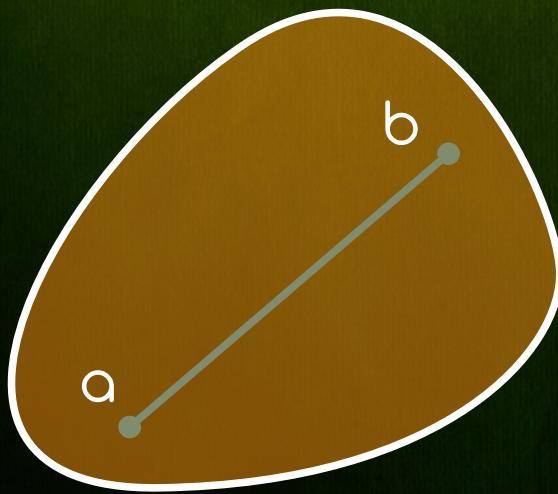
Proximity Queries for Convex Objects



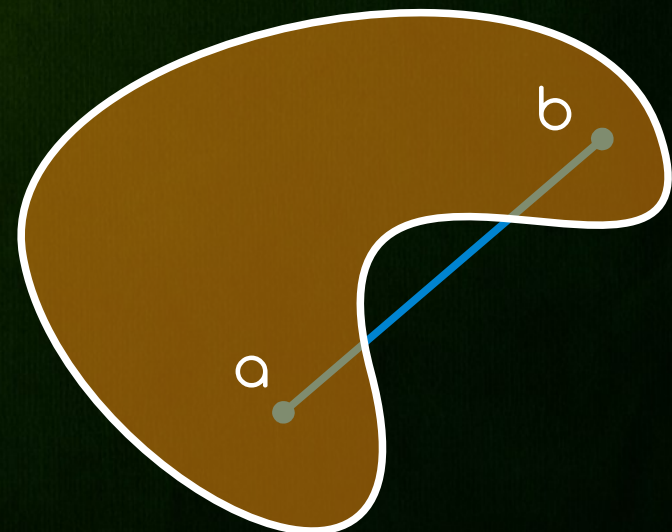
Minkowski Space

* Convex Bounded Point Set

- A set S of points $p \in \mathbb{R}^n$ is called convex and bounded if for any two points a and b the line segment ab lies entirely in S and the distance $|a - b|$ is finite (at most β)
- $a \in S \wedge b \in S \wedge t \in (0, 1) \Rightarrow (1 - t)a + tb \in S \wedge |a - b| \leq \beta$
- S must be continuous, but needs not to be smooth



Convex set

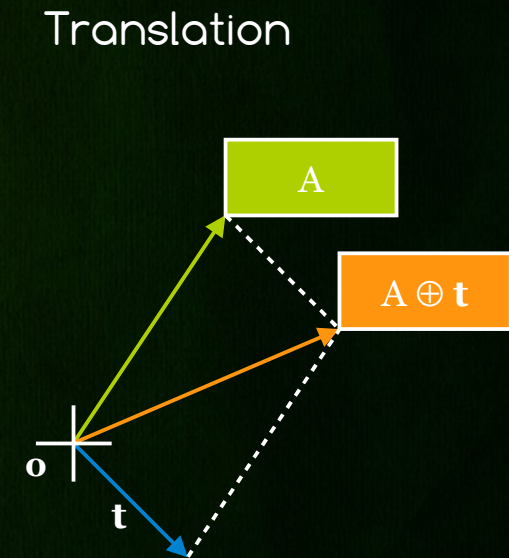
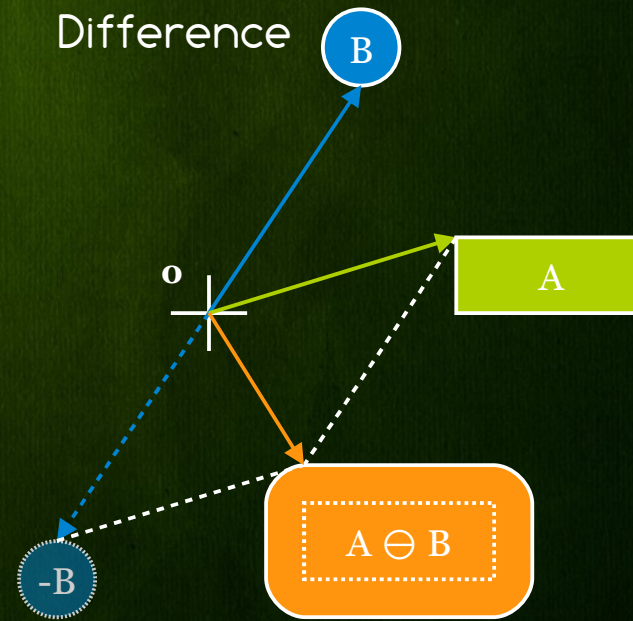
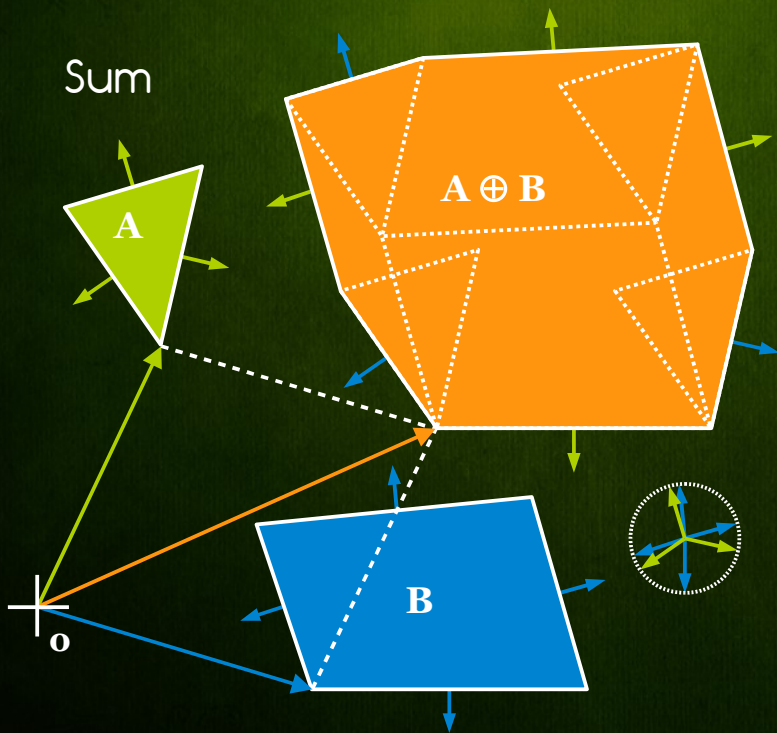


Non-convex set

Minkowski Space

- ★ Given any two convex objects A and B we define Minkowski Sum, Difference and Translation as
- ★ Minkowski Sum $A \oplus B$
 - $A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$
- ★ Minkowski Difference $A \ominus B$ (known as CSO)
 - $A \ominus B = A \oplus (-B) = \{a - b \mid a \in A \wedge b \in B\}$
- ★ Minkowski Translation $A \oplus t$
 - $A \oplus t = A \oplus \{t\} = \{a + t \mid a \in A\}$

Minkowski Space



Touching Vectors

* Touching Contact

- Two convex objects A and B are in touching contact, iff their intersection (as a point set) is a subset of some (contact) plane β . Formally: $A \cap B \subset \beta$

* Touching Vector

- The touching vector \mathbf{t}_{AB} between two convex objects A and B is any shortest translational vector t moving objects into the touching contact.

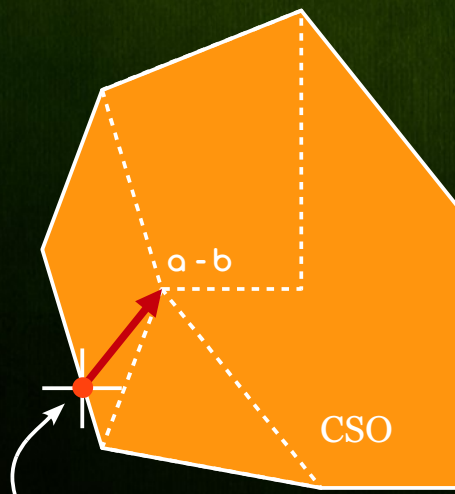
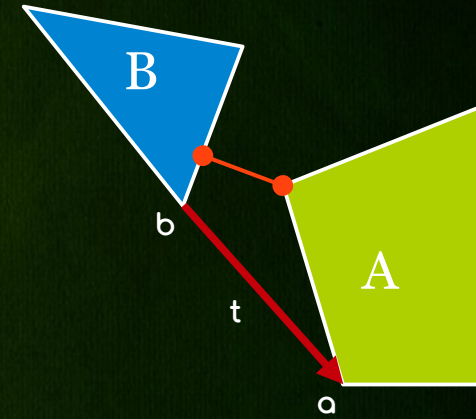
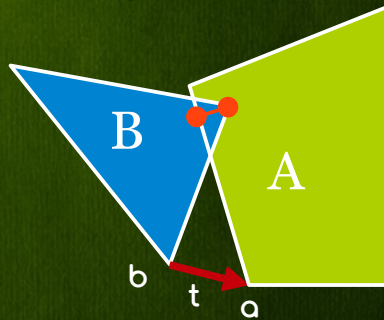
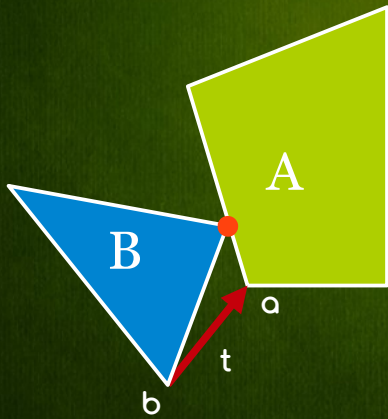
- $\mathbf{t}_{AB} \in \{t \mid A \cap (B \oplus t) \subset \beta \wedge t \in \mathbb{R}^3 \wedge |t| = d_{AB}\}$

* Touching Distance

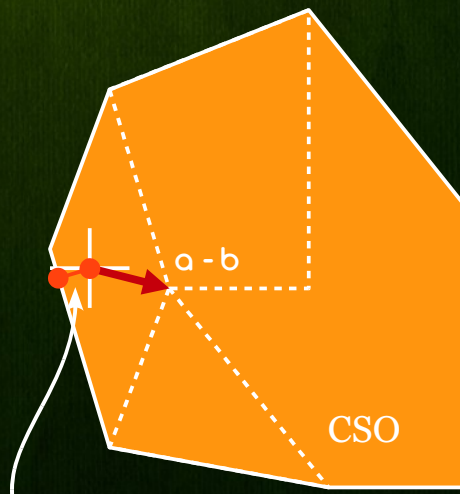
- Touching distance d_{AB} is the length of touching vector \mathbf{t}_{AB} .

- $d_{AB} = \min \{|t| \mid A \cap (B \oplus t) \subset \beta \wedge t \in \mathbb{R}^3\}$

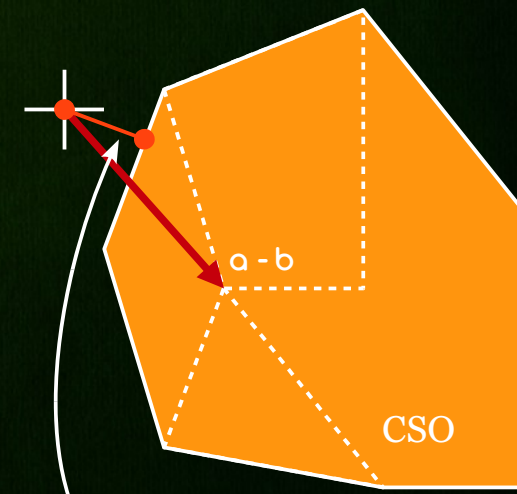
Touching Vectors and CSO



Touching vector



Penetration vector



Separation vector

Touching Vectors

- * Objects are in close proximity if their touching distance is smaller than a defined threshold
- * If objects are disjoint touching vector (distance) is usually called as **separation vector (distance)**
- * If objects are intersecting touching vector (distance) is usually called as **penetration vector (depth)**
- * Separation vector is unique. Penetration vector is usually not unique (co-centric circles)

Support Set and Boundary

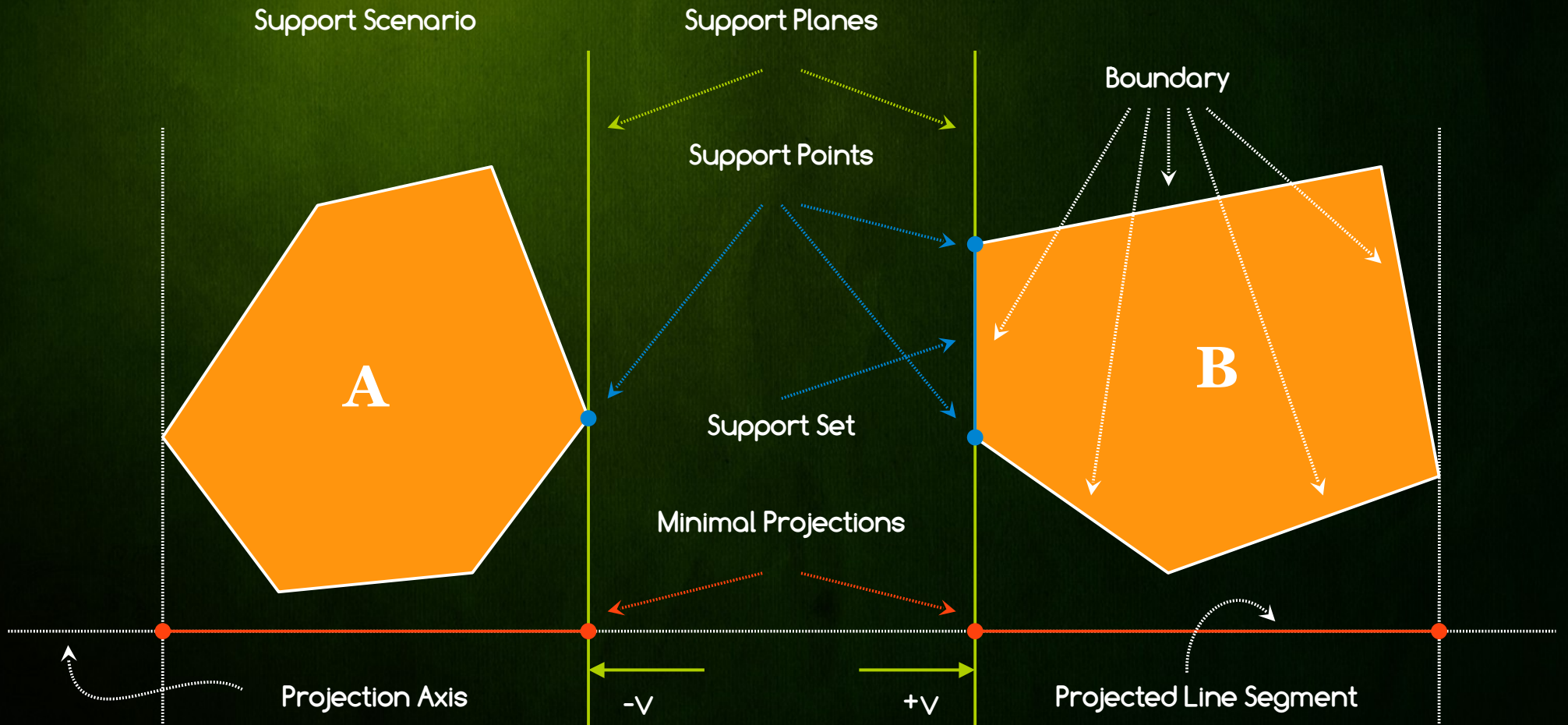
* Support Set

- The set of points from a convex object C which have a minimal projection onto a direction axis d is the support set of C
- $S^d_C = \{ p \mid p \in C \wedge d^T p = \min\{ d^T c \mid c \in C \} \}$

* Support Boundary

- The set of all support points from a convex object C with respect to any direction d is the boundary of C
- $\partial(C) = \{ p \mid p \in S^d_C \wedge d \in R^3 \}$

Support Set and Boundary



Touching Vectors and Boundary

- ★ Touching Vector Theorem

- Any translational vector t moves two convex objects A and B into touching contact, iff it lies on the boundary of their CSO

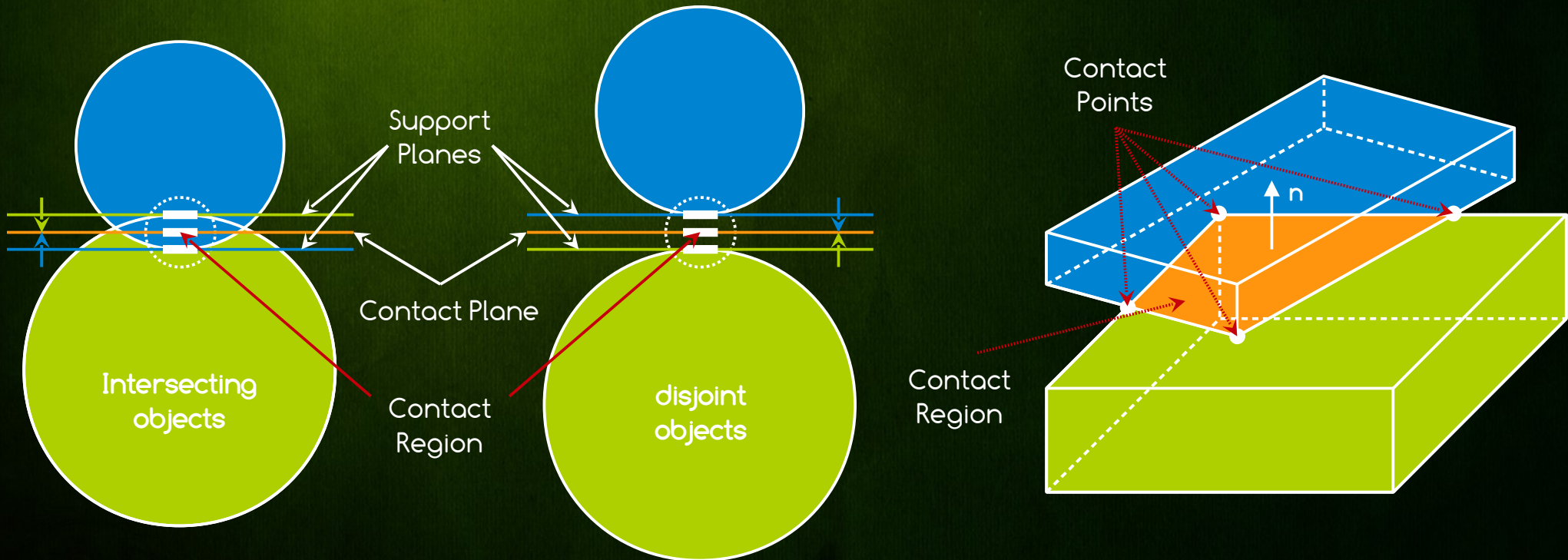
- $A \cap (B \oplus t) \subset \beta \Leftrightarrow t \in \partial(A \ominus B)$

- ★ This theorem can simplify the definition of touching contact, vector and distance, by replacing $(A \cap (B \oplus t) \subset \beta)$ with the $t \in \partial(A \ominus B)$

- $d_{AB} = \min \{ |t| \mid t \in \partial(A \ominus B) \}$

- $t_{AB} \in \{ t \mid t \in \partial(A \ominus B) \wedge |t| = d_{AB} \}$

Contact Region



Contact Region

- ★ If objects are in touching contact (t_{AB} is zero), their intersection simply forms the contact region
- ★ If objects penetrate or are disjoint (t_{AB} is non-zero) contact region is constructed as follows
 - Compute two support sets $S_A^{+t_{AB}}$ and $S_B^{-t_{AB}}$ for A and B w.r.t t_{AB}
 - Project both sets onto touching vector t_{AB} and take median
 - Form contact plane with median as origin and normal as t_{AB}
 - Project both support sets onto contact plane and take their (ideally) intersection as contact region

GJK



Gilbert - Johnson - Keerthi Algorithm

Gilbert - Johnson - Keerthi Algorithm

- ★ **Key idea** of all GJK based algorithms:
iterative search for the touching vector in CSO
- ★ **Strategy:** Perform a descent traversal of the CSO surface to find the closest point to the origin
- ★ **Problem:** Naive construction and traversal of CSO is expensive and slow
- ★ **Solution:** Simple support function can select proper support points on CSO and thus speed up the traversal to an almost constant time assuming coherent simulation.

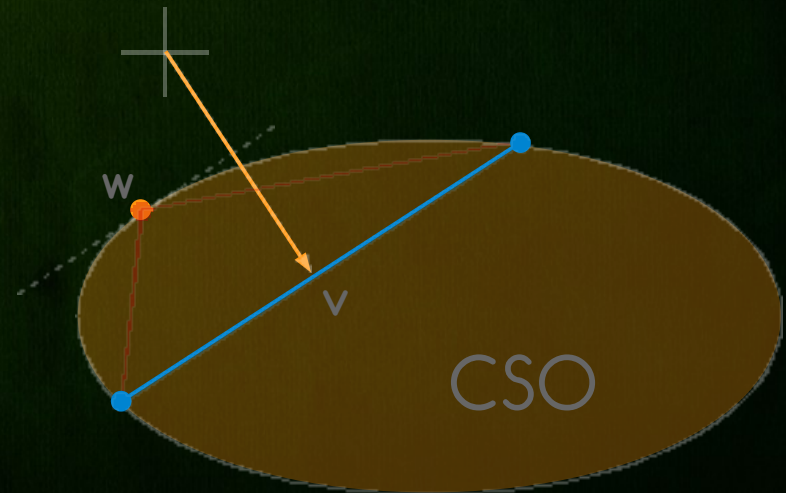
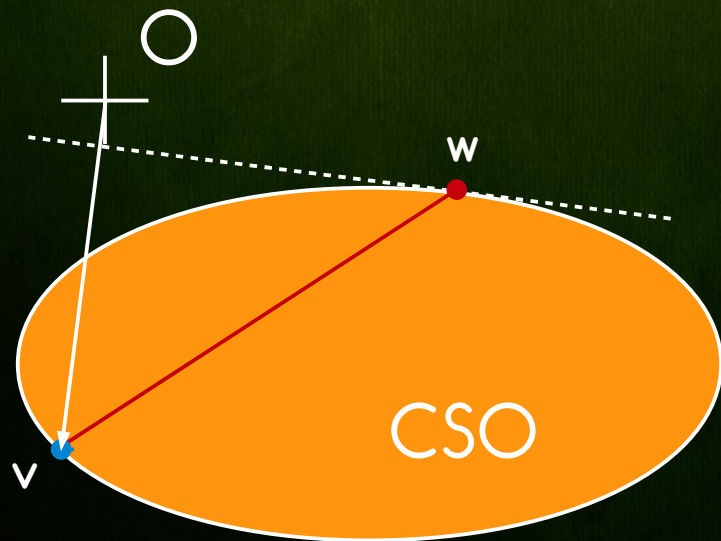
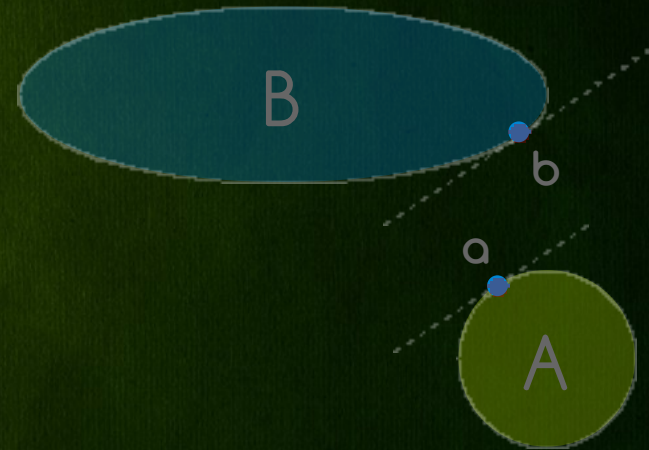
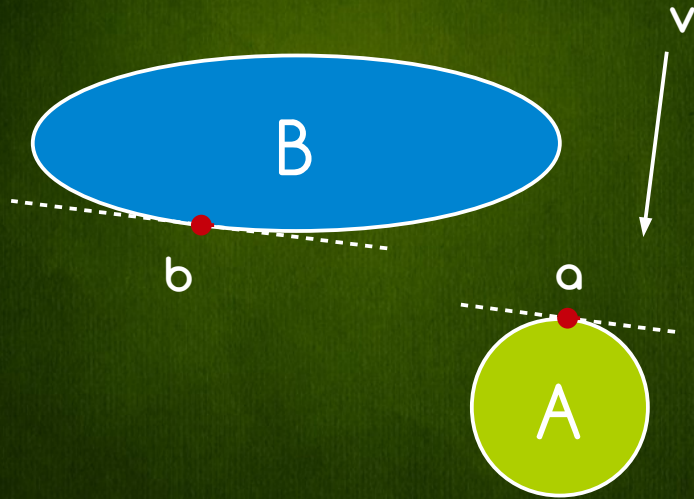
Support Function

- * Support function $\text{support}(C, d) \in S^d_C$ of a convex object C w.r.t. direction d simply returns any support point from the respective support set S^d_C
- * Support Function Operations
 - Assuming $\text{support}(A, d) \in S^d_A$ and $\text{support}(B, d) \in S^d_B$, we define the support functions as follows
 - $\text{support}(-B, d) = -\text{support}(B, -d) \in S^d_{-B}$
 - $\text{support}(A \oplus B, d) = \text{support}(A, d) + \text{support}(B, d) \in S^d_{A \oplus B}$
 - $\text{support}(A \ominus B, d) = \text{support}(A \oplus (-B), d)$
 $= \text{support}(A, d) + \text{support}(-B, d)$
 $= \text{support}(A, +d) - \text{support}(B, -d)$

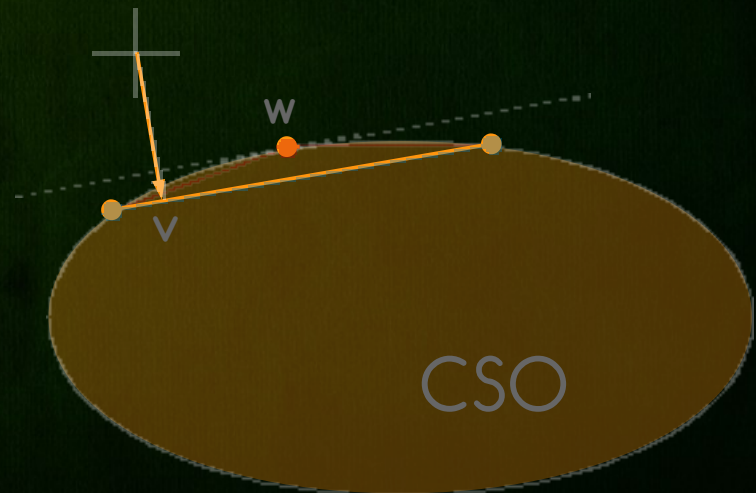
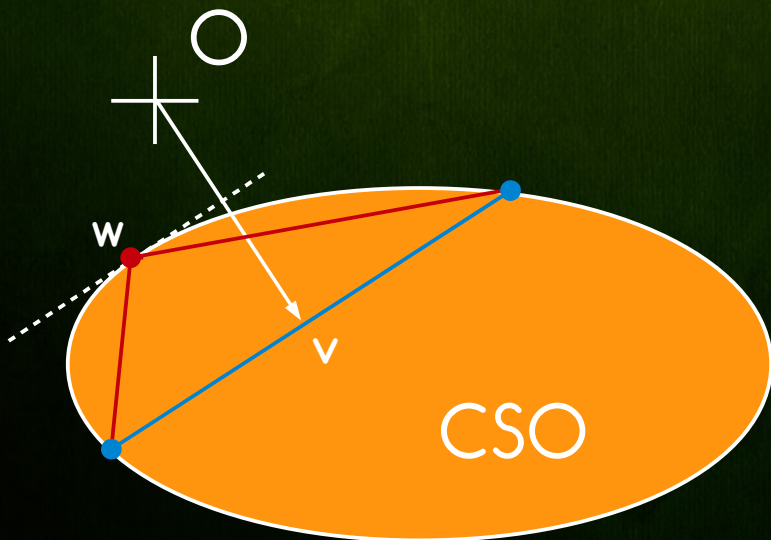
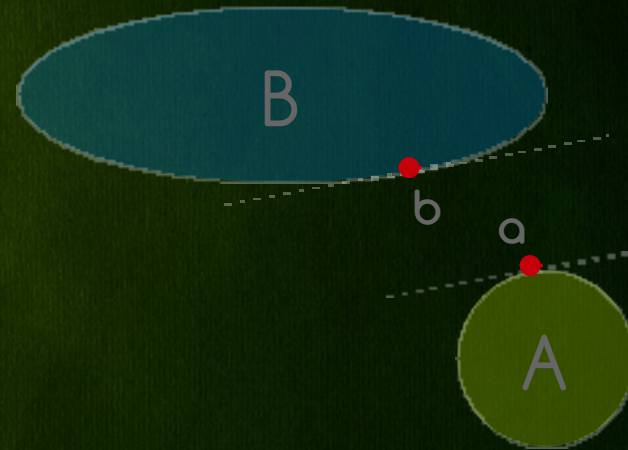
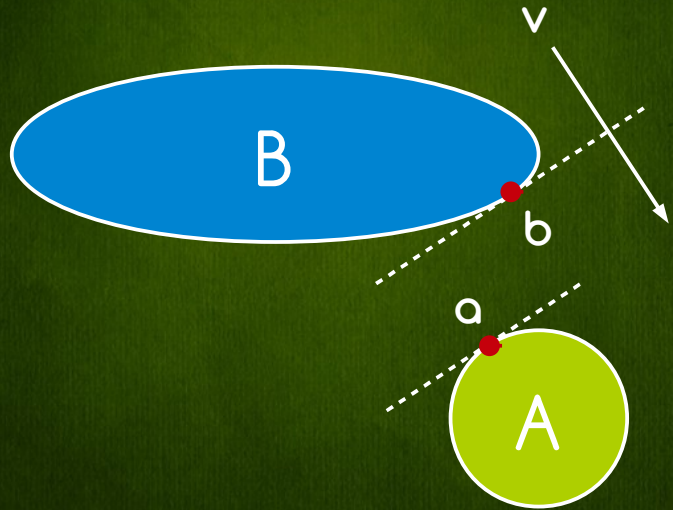
Proximity GJK Algorithm

- ★ The traversal is done by iteratively constructing a sequence of simplices in 3D
 - point or line or triangle or tetrahedron
- ★ In each iteration newly created simplex is closer to the origin as the one in previous iteration
- ★ New simplex is created by
 - ★ 1) Adding a support point to the former simplex
 - ★ 2) Taking the smallest sub-simplex which contains the closest point to the origin

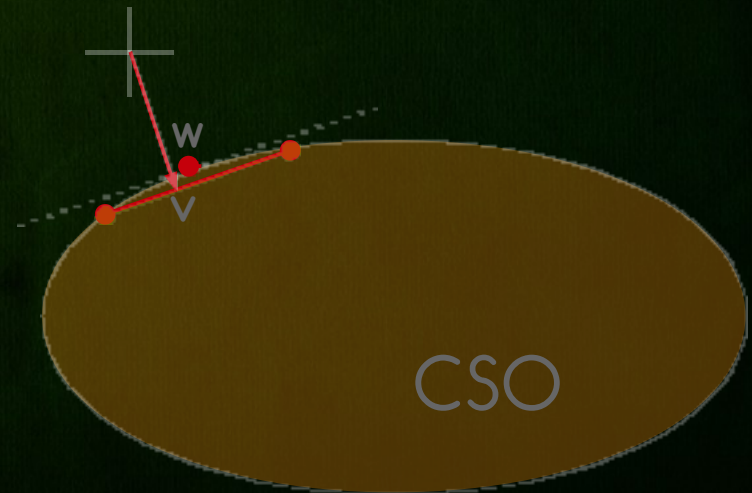
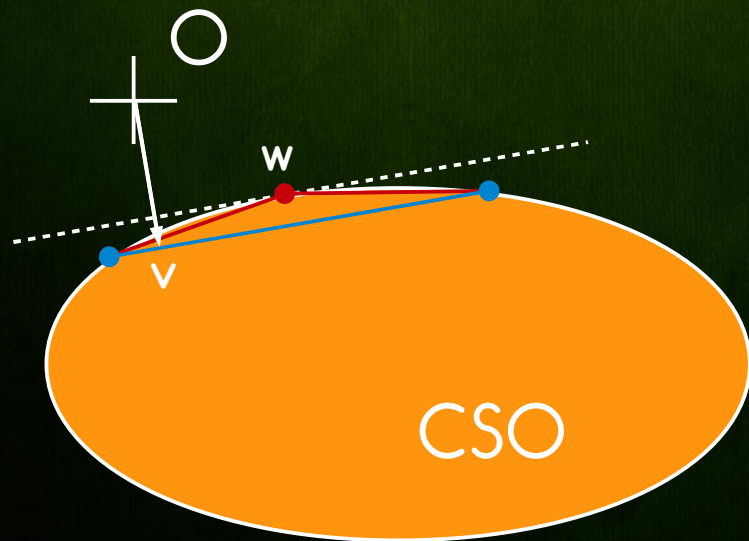
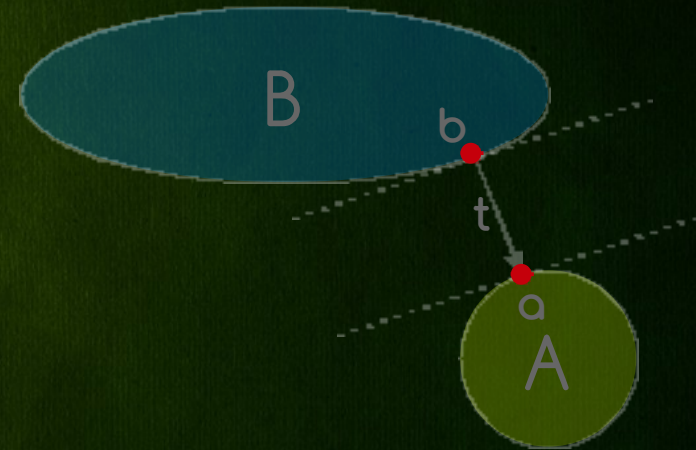
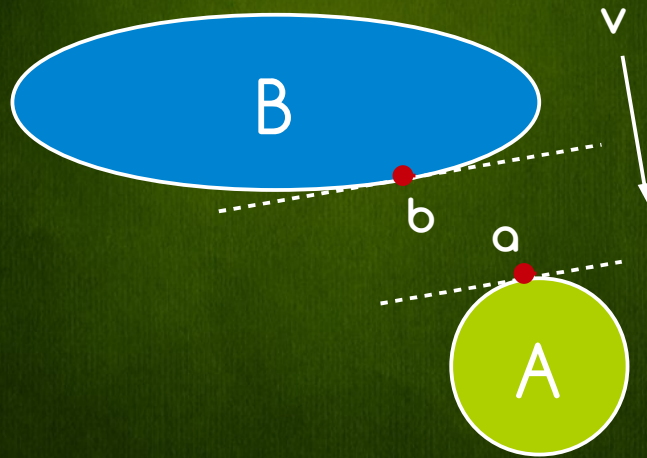
Proximity GJK Algorithm



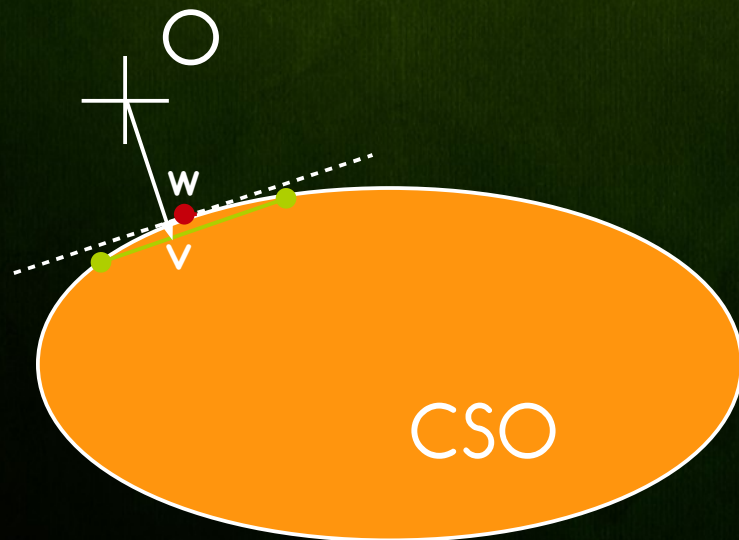
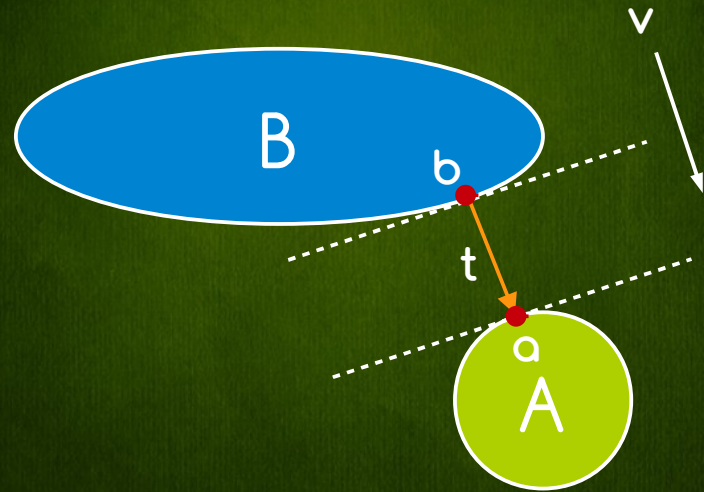
Proximity GJK Algorithm



Proximity GJK Algorithm



Proximity GJK Algorithm Algorithm



Proximity GJK Algorithm

In: Convex objects A, B and initial simplex W

Out: Touching vector \mathbf{w}

function PROXIMITYGJK(A, B, W) : \mathbf{w}

```
1:   $\{\mathbf{v}, \delta\} \leftarrow \{\mathbf{1}, 0\}$ 
2:  while ( $\|\mathbf{v}\|^2 - \delta^2 > \varepsilon$ ) do
3:       $\mathbf{v} \leftarrow \text{CLOSESTPOINT}(W)$ 
4:       $\mathbf{w} \leftarrow \text{SUPPORT}(A \ominus B, \mathbf{v}) = \text{SUPPORT}(A, +\mathbf{v}) - \text{SUPPORT}(B, -\mathbf{v})$ 
5:       $W \leftarrow \text{BESTSIMPLEX}(W, \mathbf{w})$ 
6:      if ( $|W| = 4$ ) then return PROXIMITYEPA( $A, B, W$ ) ;
7:      if ( $\mathbf{v}^T \mathbf{w} > 0$ ) then  $\delta^2 \leftarrow \max \left\{ \delta^2, \frac{(\mathbf{v}^T \mathbf{w})^2}{\|\mathbf{v}\|^2} \right\}$ 
8:  end
9:  return  $\mathbf{w}$ 
end
```

Computing Support Function

- * Searching for the support vertex w heavily depends on the representation of the convex objects A and B
- * For a simple primitives it can be computed directly
- * For convex polytopes
 - Naive approach is to project all vertices onto the direction axis and take any one with the minimal projection
 - if we consider a coherent simulation we can use a local search sometimes called as “hill climbing” and find the support vertex in almost constant time

Hill Climbing Support Function

- ★ For convex polytopes do a local search to “refine” the support point from previous simulation state

In: Convex polytope A , initial support vertex \mathbf{w} and the direction vector \mathbf{d}

Out: New support vertex with minimal projection \mathbf{w}

function SUPPORTHC($A, \mathbf{d}, \mathbf{w}$) : \mathbf{w}

1: $\{\mu, \text{Found}\} \leftarrow \{\mathbf{d}^T \mathbf{w}, \text{false}\}$

2: **while not Found do**

3: $\text{Found} \leftarrow \text{true}$

4: **foreach** \mathbf{w}' **in** NEIGHBOURS(\mathbf{w}) **do**

5: **if** $(\mathbf{d}^T \mathbf{w}' < \mu)$ **then** $\{\mu, \mathbf{w}, \text{Found}\} \leftarrow \{\mathbf{d}^T \mathbf{w}', \mathbf{w}', \text{false}\}$; **break**

6: **end**

7: **end**

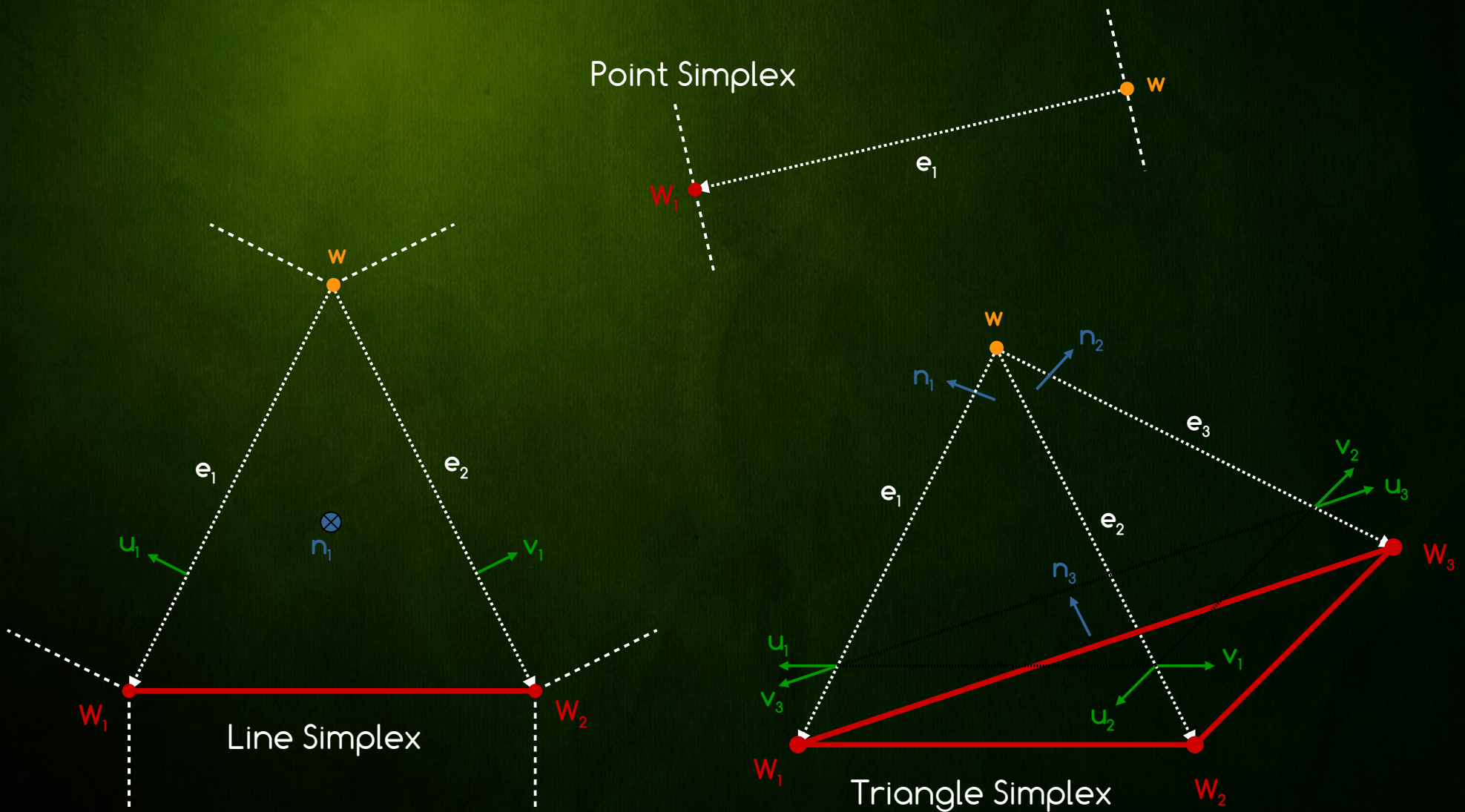
8: **return** \mathbf{w}

end

Simplex Refinement

- ★ **Problem:** Given a simplex and new vertex form new simple by adding the vertex and select sub-simplex closest to the origin
- ★ **Bad solution:** The simplex can be done by solving a system of linear equations (slow, numeric issues)
- ★ **Good solution:** Form new simplex and test in which external Voronoi region the origin lies.
- ★ The selected Voronoi region directly shows us which sub-simplex is the desired (closest) one

Voronoi Simplex Refinement



Voronoi Simplex Refinement

- * Empty Simplex: A vertex simplex $\{w\}$ is formed
 - The smallest simplex, which contains the closest point to the origin is $\{w\}$ (case 0)
- * Vertex Simplex: An edge simplex $\{W1, w\}$ is formed
 - It has 2 vertex regions $\{W1, w\}$ and one edge region $\{e1\}$
 - Since $W1$ lies on support plane which is perpendicular to the support axis (vector w) origin can not be in the region of $W1$
 - Thus we check only regions of w and $e1$ by projecting $-w$ onto the edge $e1$ (case 1)

Voronoi Simplex Refinement

- ★ Edge Simplex: A face simplex $\{W1, W2, w\}$ is formed
 - It has 3 vertex regions, 3 edge regions and 2 face regions
 - The origin can be only in $\{w, e1, e2, n1\}$ regions
 - Construct Voronoi planes with normals $\{e1, e2, u1, v1\}$ and test whether the origin is above or below these planes, i.e. compare signs of $-w$ projections onto these normals
- ★ Face Simplex: A tetrahedron simplex $\{W1, W2, W3, w\}$ is formed
 - A tetrahedron has 4 vertex regions, 4 face regions, 6 edge regions and 1 interior region (T)
 - Origin can lie only only in regions $\{w, e1, e2, e3, n1, n2, n3, T\}$
 - Construct Voronoi planes with normals $\{e1, e2, e3, n1, n2, n3, u1, u2, u3, v1, v2, v3\}$ and test sign $-w$ projection onto normals

In: Simplex W and new point on CSO surface w

Out: New smallest simplex W containing w and the closest point to the origin

function BESTSIMPLEX(W, w) : W

```
1:   $d \leftarrow 0 - w$ 
2:   $e_1 \leftarrow W_1 - w$ ;            $e_2 \leftarrow W_2 - w$ ;            $e_3 \leftarrow W_3 - w$ ;
3:   $n_1 \leftarrow e_1 \times e_2$ ;      $n_2 \leftarrow e_2 \times e_3$ ;      $n_3 \leftarrow e_3 \times e_1$ ;
4:   $u_1 \leftarrow e_1 \times n_1$ ;      $u_2 \leftarrow e_2 \times n_2$ ;      $u_3 \leftarrow e_3 \times n_3$ ;
5:   $v_1 \leftarrow n_1 \times e_2$ ;      $v_2 \leftarrow n_2 \times e_3$ ;      $v_3 \leftarrow n_3 \times e_1$ ;
6:  switch  $|W|$  do
7:    case 0                                     /* empty simplex */
8:      return { $w$ }
9:    end
10:   case 1                                     /* vertex simplex */
11:     if  $(d^T e_1 > 0)$  then return { $w$ }
12:     if  $(d^T e_1 < 0)$  then return  $\{W_1, w\}$ 
13:   end
14:   case 2                                     /* edge simplex */
15:     if  $(d^T e_1 < 0) \wedge (d^T e_2 < 0)$  then return { $w$ }
16:     if  $(d^T e_1 > 0) \wedge (d^T u_1 > 0)$  then return  $\{W_1, w\}$ 
17:     if  $(d^T e_2 > 0) \wedge (d^T v_1 > 0)$  then return  $\{W_2, w\}$ 
18:     if  $(d^T u_1 < 0) \wedge (d^T v_1 < 0)$  then return  $\{W_1, W_2, w\}$ 
19:   end
20:   case 3                                     /* face simplex */
21:     if  $(d^T e_1 < 0) \wedge (d^T e_2 < 0) \wedge (d^T e_3 < 0)$  then return { $w$ }
22:     if  $(d^T e_1 > 0) \wedge (d^T u_1 > 0) \wedge (d^T v_3 > 0)$  then return  $\{W_1, w\}$ 
23:     if  $(d^T e_2 > 0) \wedge (d^T u_2 > 0) \wedge (d^T v_1 > 0)$  then return  $\{W_2, w\}$ 
24:     if  $(d^T e_3 > 0) \wedge (d^T u_3 > 0) \wedge (d^T v_2 > 0)$  then return  $\{W_3, w\}$ 
25:     if  $(d^T n_1 > 0) \wedge (d^T u_1 < 0) \wedge (d^T v_1 < 0)$  then return  $\{W_1, W_2, w\}$ 
26:     if  $(d^T n_2 > 0) \wedge (d^T u_2 < 0) \wedge (d^T v_2 < 0)$  then return  $\{W_2, W_3, w\}$ 
27:     if  $(d^T n_3 > 0) \wedge (d^T u_3 < 0) \wedge (d^T v_3 < 0)$  then return  $\{W_3, W_1, w\}$ 
28:     if  $(d^T n_1 < 0) \wedge (d^T n_2 < 0) \wedge (d^T n_3 < 0)$  then return  $\{W_1, W_2, W_3, w\}$ 
29:   end
30: end
end
```


Closest Point on Simplex

- * Problem: Given (0 or 1 or 2 or 3) simplex $\{W1, W2, W3\}$ find the closest point to the origin
- * Empty Simplex: Return 0
- * Vertex Simplex: Return $W1$
- * Edge Simplex: Return the closest point on line $\{W1, W2\}$ to the origin.
 - No need to check other regions (eg. vertex $W1$ region etc.)
- * Face Simplex: Return the closest point on plane $\{W1, W2, W3\}$ to the origin.
 - No need to check other regions (eg. vertex $W1$ region etc.)

Closest Point Algorithm

In: Simplex W

Out: Closest point on simplex to the origin \mathbf{v}

function CLOSESTPOINT(W) : \mathbf{v}

```
1:  $\mathbf{d} \leftarrow \mathbf{W}_2 - \mathbf{W}_1$ 
2:  $\mathbf{n} \leftarrow (\mathbf{W}_2 - \mathbf{W}_1) \times (\mathbf{W}_3 - \mathbf{W}_1)$ 
3: switch  $|W|$  do
4:   case 0 return 0 ;           /* empty simplex */
5:   case 1 return  $\mathbf{W}_1$  ;     /* vertex simplex */
6:   case 2 return  $\mathbf{W}_1 - \frac{\mathbf{d}^T \mathbf{W}_1}{\mathbf{d}^T \mathbf{d}} \mathbf{d}$  ; /* edge simplex */
7:   case 3 return  $\frac{\mathbf{n}^T \mathbf{W}_1}{\mathbf{n}^T \mathbf{n}} \mathbf{n}$  ; /* face simplex */
8: end
end
```

GJK Overlap Test

- ★ Incremental Separating-Axis GJK (ISA-GJK)
 - A subtle modification to the proximity GJK
 - Descent overlap test for convex objects
 - Iteratively searches for some separating axis
 - Average constant time complexity in coherent simulation
 - ★ Principle: Similar traversal to Proximity GJK
 - Reports overlap: When the best simplex is tetrahedron
 - Reports no-overlap: When the signed distance of the support plane to the origin is positive
- $v^T w = v^T \text{support}(A \ominus B, v) = v^T \text{support}(A, +v) - v^T \text{support}(B, -v) > 0$

ISA-GJK Algorithm

In: Convex objects A, B and initial Simplex W

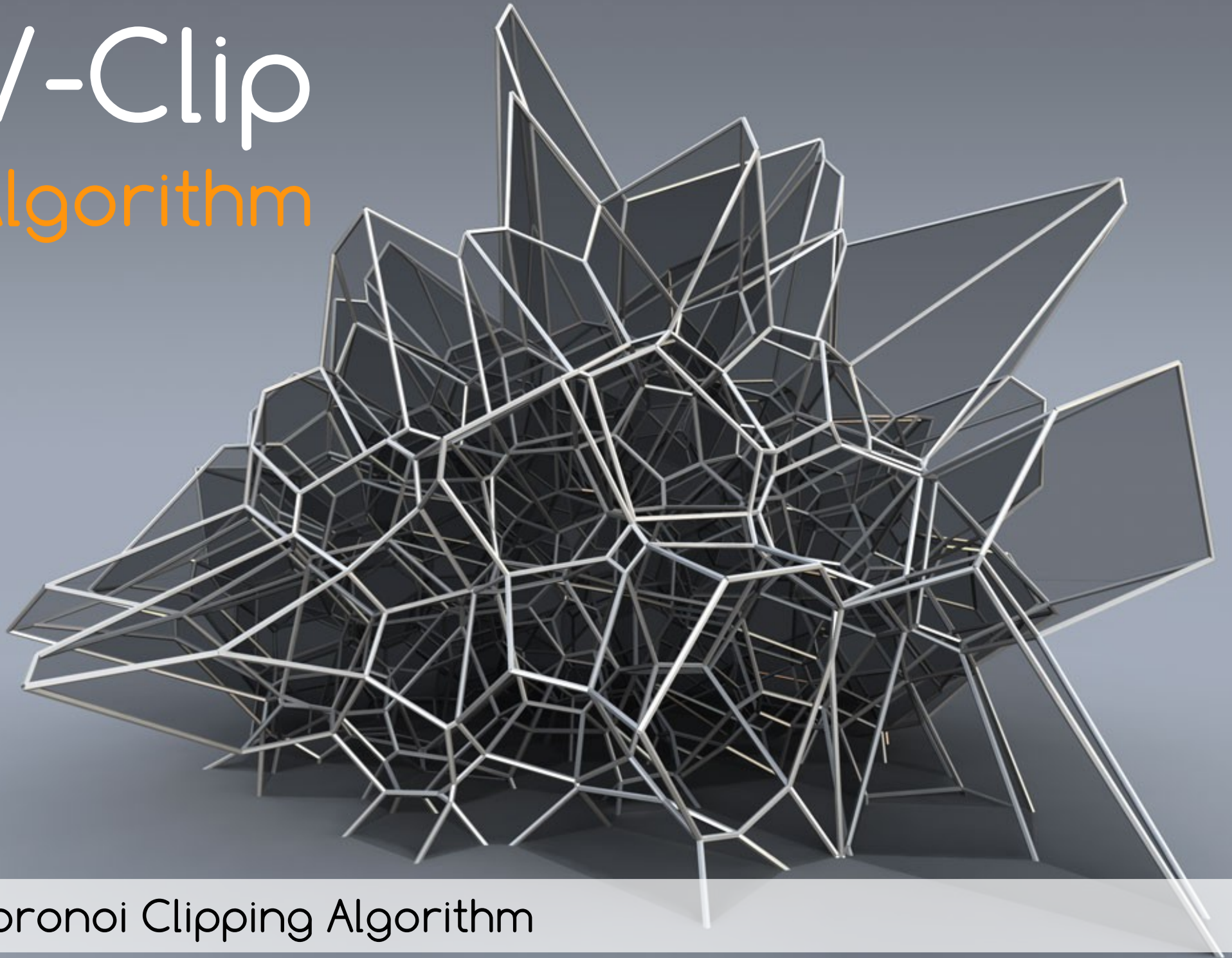
Out: Overlap check: (**true/false**)

function OVERLAPGJK(A, B, W) : **bool**

```
1:   {v, w} ← {1, 1}
2:   while ( $\mathbf{v}^T \mathbf{w} \leq 0$ ) do
3:     v ← CLOSESTPOINT( $W$ )
4:     w ← SUPPORT( $A \ominus B, \mathbf{v}$ ) = SUPPORT( $A, +\mathbf{v}$ ) - SUPPORT( $B, -\mathbf{v}$ )
5:      $W$  ← BESTSIMPLEX( $W, \mathbf{w}$ )
6:     if ( $|W| = 4$ ) then return true ;           /* intersection */
7:   end
8:   return false
end
```

V-Clip

Algorithm



Voronoi Clipping Algorithm

External Voronoi Regions

- * Interior Set:

- The set of all interior points $\text{int}(C)$ of a convex polytope C is the intersection of negative half-spaces formed by all faces of C (surface points are not included)

- * $\text{int}(C) = \{ c \in \mathbb{R}^3 \mid ds(c, F) < 0 \wedge F \in C \}$

- * Distance:

- The distance $d(c, X)$ between a feature X and some point c is the minimum distance between c and any point of X

- * $d(c, X) = \min \{ |x - c| \mid x \in X \}$

External Voronoi Regions

- ★ Signed Distance

- The signed distance $d_s(c, F)$ between a point c and a plane F , defined by a unit normal n_F and a reference point o_F is the projection of the reference vector $(c - o_F)$ onto planes normal

- ★ $d_s(c, F) = n_F^T (c - o_F)$

- ★ Having two incident features X, Y : if X has a lower dimension than Y , then X must be a subset of Y and therefore the distance of any point c to X is less than or equal to Y

- ★ $X \cap Y \wedge \dim(X) < \dim(Y) \Rightarrow X \subset Y \Rightarrow d(c, X) \leq d(c, Y)$

External Voronoi Regions

* External Voronoi Region

- The Voronoi region $VR(X)$ of a feature X on some convex polytope C is a set of external points which are closer (\leq) to X than to any other feature Y in C
- $VR(X) = \{ c \notin \text{int}(C) \mid d(c,X) \leq d(c,Y) \wedge Y \in C \}$

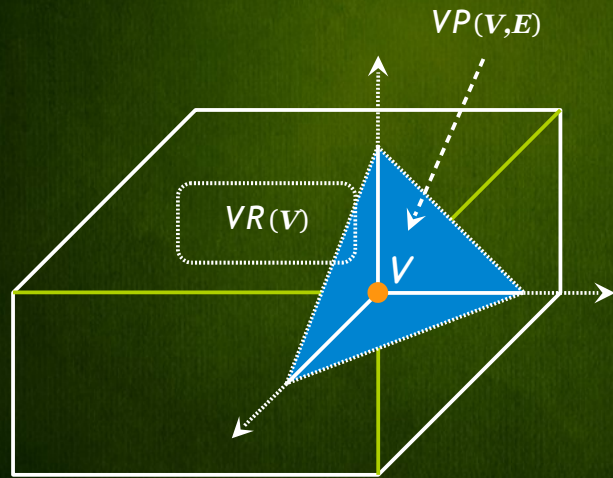
* External Voronoi Plane

- The Voronoi plane $VP(X,Y)$ of two incident features X and Y is the plane containing the intersection of their Voronoi regions.
- $VP(X,Y) = \beta \wedge VR(X) \cap VR(Y) \subset \beta$

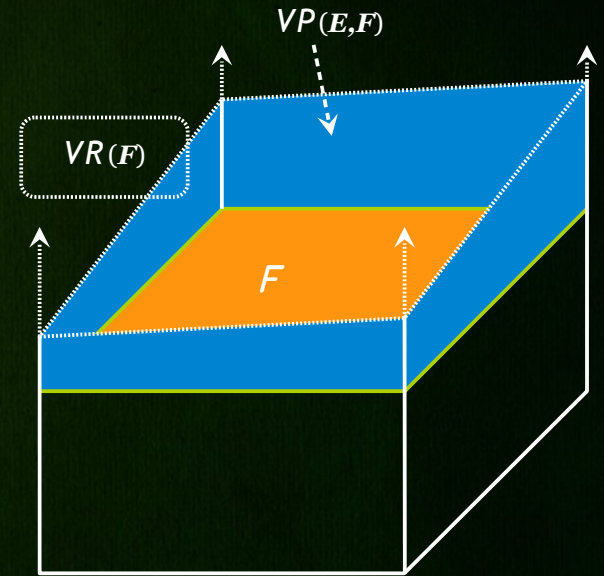
* Inter-feature Distance

- The inter-feature distance $d(X,Y)$ between features X and Y is the minimum distance between any points $x \in X$ and $y \in Y$
- $d(X,Y) = \min \{ |x - y| \mid x \in X \wedge y \in Y \}$

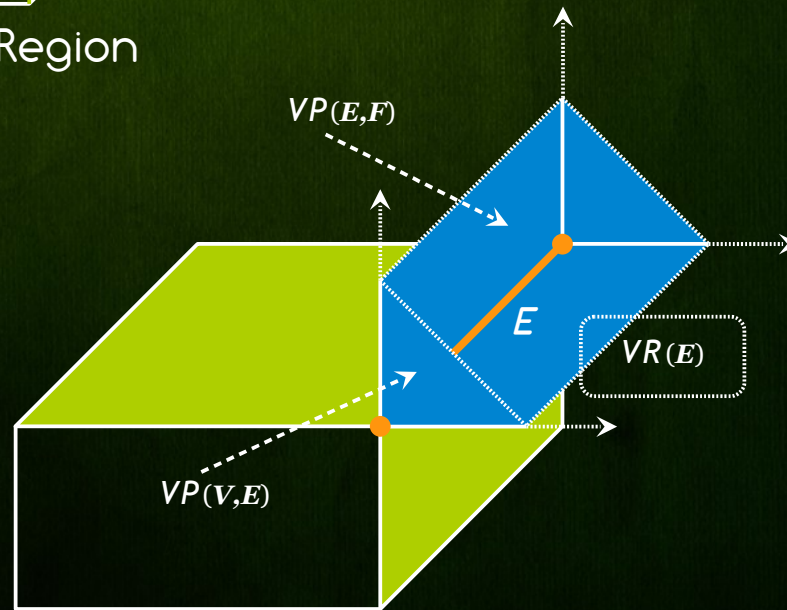
External Voronoi Regions



Vertex Voronoi Region



Face Voronoi Region

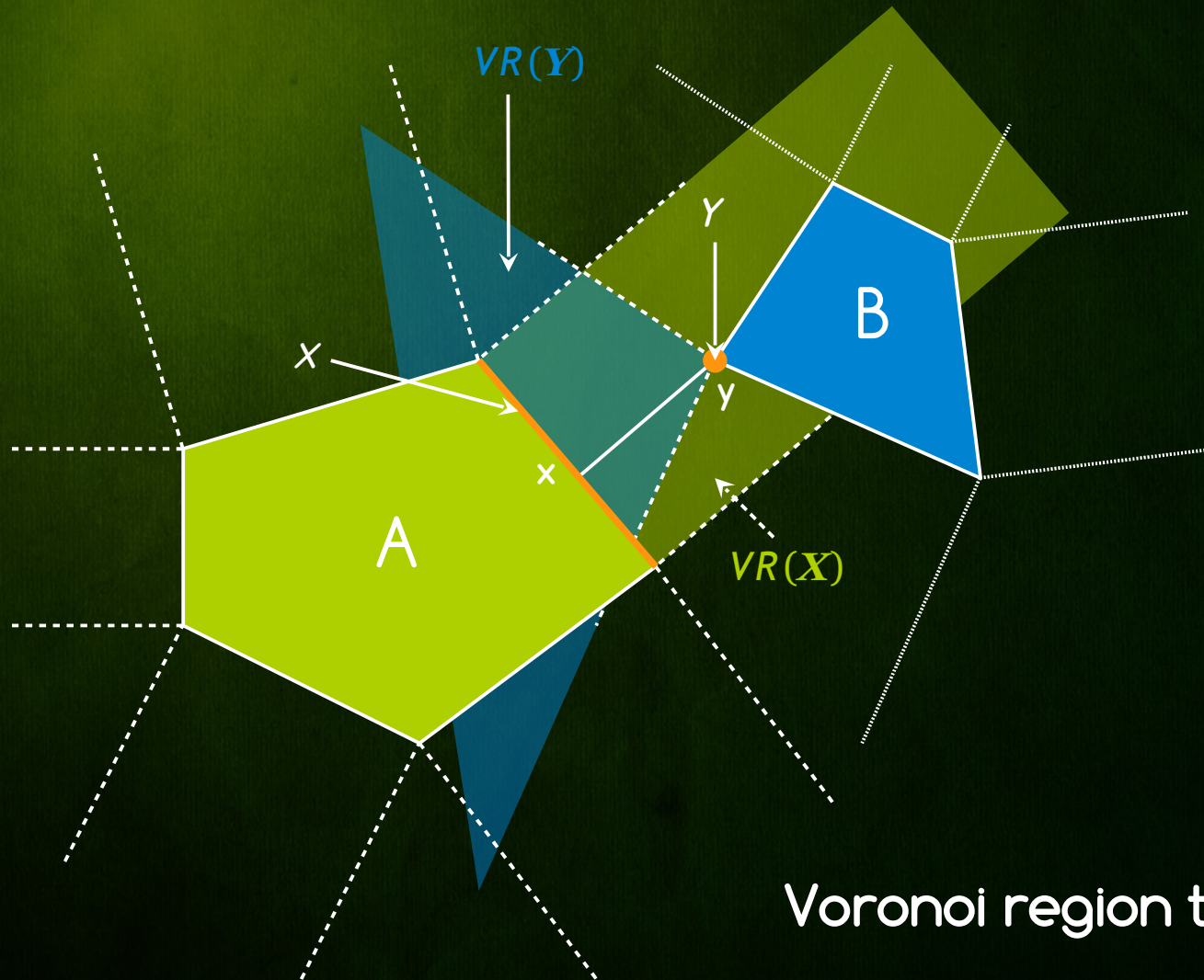


Edge Voronoi Region

Voronoi Region Theorem

- ★ Let $X \in A$ and $Y \in B$ be a pair of features from disjoint convex polytopes A and B .
- ★ Let $x \in X$ and $y \in Y$ be the closest points between X and Y
- ★ Points x and y are the (globally) closest points between A and B iff $x \in VR(Y) \wedge y \in VR(X)$

Voronoi Region Theorem



Voronoi region theorem

V-Clip Algorithm

- ★ Key idea of the V-Clip algorithm is an efficient search for two closest features.
- ★ Obviously an exhaustive search is a very expensive solution
- ★ Fortunately the following Voronoi Region Theorem allows us to find the global minimum of the inter-feature distance, by performing usually only a few iterations of a local search

V-Clip Algorithm

- ★ Given two convex polytopes A, B and any two features $X \in A, Y \in B$
- ★ In each iteration V-Clip checks if they satisfy the Voronoi Region Theorem.
 - If they don't, it changes X and Y to some (usually incident) features X' and Y' , so that either the sum their dimensions or the inter-feature distance strictly decreases.
 - Assuming a finite number of features the algorithm can never cycle
 - If we initialize X and Y with the closest features from the previous time-step and the simulation is coherent, then we probably need only a few iterations to find new closest features.

In: A pair of convex polytopes A, B and respective initial features X, Y

Out: A Separation vector w , or \emptyset if penetration occurred

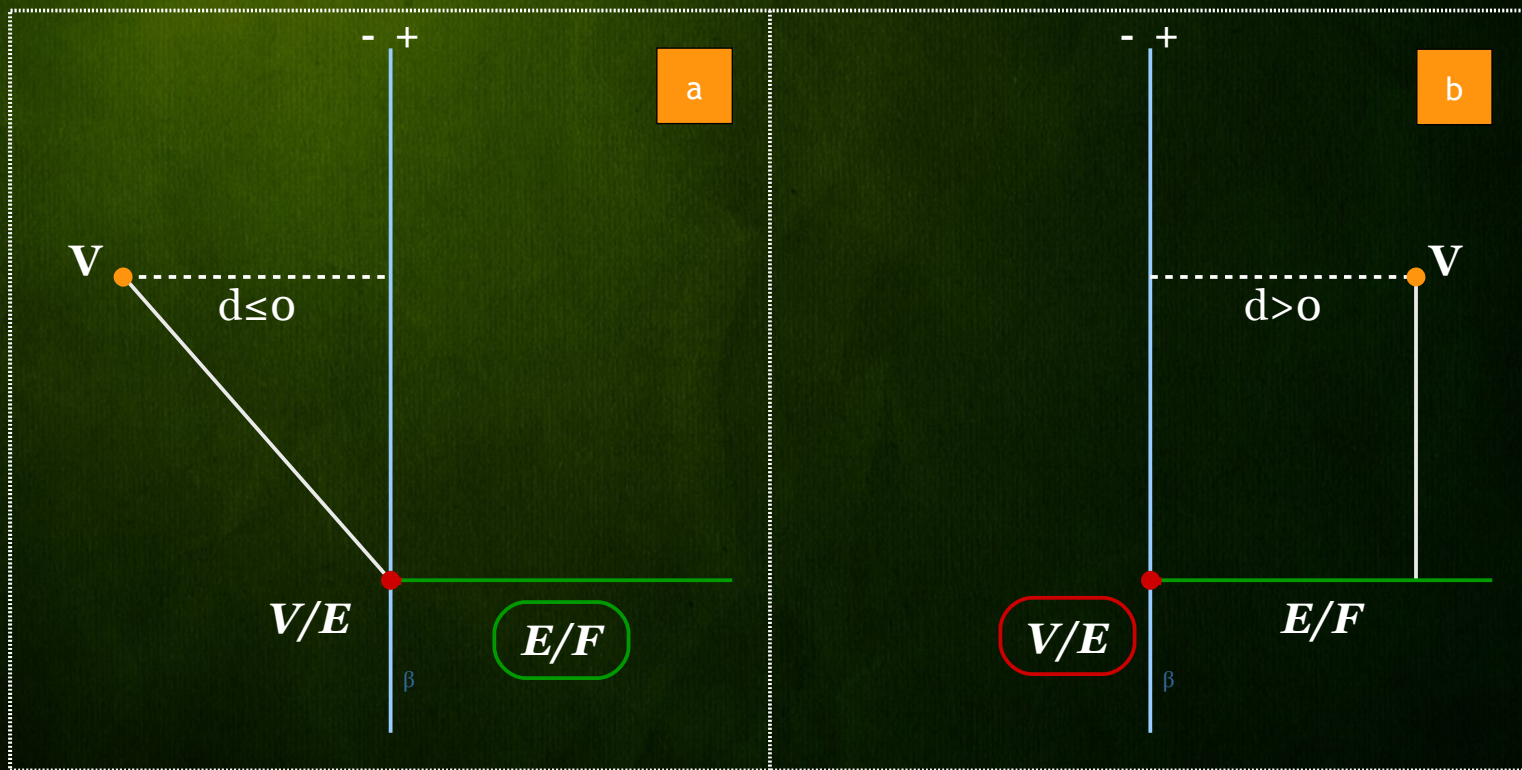
function V-CLIP(A, B, X, Y) : w

```
1:  while (true) do
2:      switch PAIRTYPE( $X, Y$ ) do
3:          case VV type :                                     /* Vertex-Vertex */
4:              if CLIPVERTEX( $X, Y, \{ YE \mid E \in \text{EDGES}(Y) \}$ ) then continue
5:              if CLIPVERTEX( $Y, X, \{ XE \mid E \in \text{EDGES}(X) \}$ ) then continue
6:              return  $X - Y$ 
7:          end
8:          case VE type :                                     /* Vertex-Edge */
9:              if CLIPVERTEX( $X, Y, \{ V_1^Y Y, V_2^Y Y, YF_1^Y, YF_2^Y \}$ ) then continue
10:             if CLIPEDGE( $Y, X, \{ XE \mid E \in \text{EDGES}(X) \}$ ) then continue
11:              $u \leftarrow V_2^Y - V_1^Y$ 
12:             return  $X - \left( V_1^Y + \frac{u^T(X - V_1^Y)}{u^T u} u \right)$ 
13:          end
14:          case VF type :                                     /* Vertex-Face */
15:              if CLIPVERTEX( $X, Y, \{ EY, V_1^E E, V_2^E E \mid E \in \text{EDGES}(Y) \}$ ) then continue
16:              if CLIPFACE( $Y, X, A$ ) then continue
17:              return  $X - \left( X + \frac{n^T(V_1^Y - X)}{n^T n} n \right)$ 
18:          end
19:          case EE type :                                     /* Edge-Edge */
20:              if CLIPEDGE( $X, Y, \{ V_1^Y Y, V_2^Y Y, YF_1^Y, YF_2^Y \}$ ) then continue
21:              if CLIPEDGE( $Y, X, \{ V_1^X X, V_2^X X, XF_1^X, XF_2^X \}$ ) then continue
22:               $\{u^X, u^Y\} \leftarrow \{V_2^X - V_1^X, V_2^Y - V_1^Y\}$ 
23:               $\{n^X, n^Y\} \leftarrow \{(u^X \times u^Y) \times u^Y, (u^Y \times u^X) \times u^X\}$ 
24:              return  $\left( V_1^X + \frac{(n^Y)^T(V_1^Y - V_1^X)}{(n^Y)^T u^X} u^X \right) - \left( V_1^Y + \frac{(n^X)^T(V_1^X - V_1^Y)}{(n^X)^T u^Y} u^Y \right)$ 
25:          end
26:          case EF type :                                     /* Edge-Face */
27:              if CLIPEDGE( $X, Y, \{ EY, V_1^E E, V_2^E E \mid E \in \text{EDGES}(Y) \}$ ) then continue
28:               $\{d_1, d_2\} \leftarrow \{d_s(V_1^X, Y), d_s(V_2^X, Y)\}$ 
29:              if  $(\text{sgn}(d_1 d_2) < 0)$  then  $Y \leftarrow \emptyset$ ; continue
30:              if  $(|d_1| < |d_2|)$  then  $X \leftarrow V_1^X$  else  $X \leftarrow V_2^X$ 
31:              continue
32:          end
33:          case EV, FV, FE type : SWAP( $X, Y$ ); SWAP( $A, B$ ); continue; /* Swap Cases */
34:      end
35:      if  $(Y = \emptyset)$  then return  $\emptyset$ 
36:  end
end
```

Vertex Clipping

- * Given a vertex V from one object, some "old" feature N from another object and a set of feature pairs S_n
- * The vertex clipping simply marks X (Y) if the vertex V lies above (below) the $VP(X,Y)$ for each feature pair $XY \in S_N$
 - First it clears all features among SN ($ClearAll(S_N)$)
 - Next it tests the side (w.r.t. Voronoi plane) of V and mark "further" features.
 - Finally it updates N with some unmarked feature ($UpdateClear(N, SN)$) and returns true if N was changed.

Vertex Clipping Cases



Edge Clipping

- ★ Take an edge E , the "old" feature N , a set of respective feature pairs S_N and perform a sequence of local tests to properly mark "further" features
- ★ Let d_1, d_2 represent signed distances of the endpoint vertices V_1^E, V_2^E to the Voronoi plane $\beta = VP(X, Y)$ of a particular feature pair $XY \in S_N$
- ★ If both vertices lie on the same side of the clipping plane ($\text{sgn}(d_1 d_2) > 0$), we simply mark the feature of the opposite side as in vertex clipping

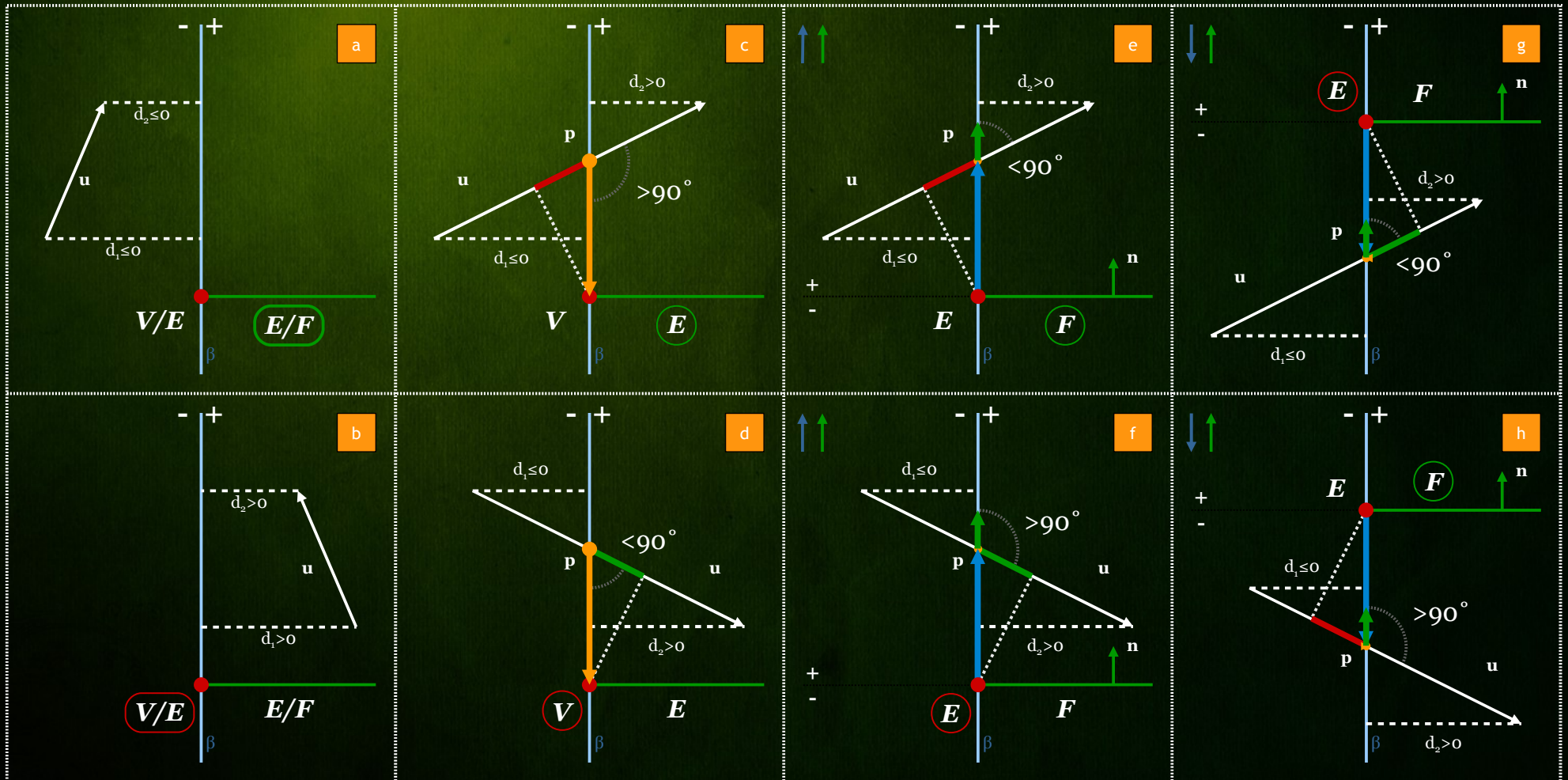
Edge Clipping

- ★ If vertices lie on different sides ($\text{sgn}(d_1 d_2) < 0$), edge E intersects the clipping plane in some point $\rho = (1 - \lambda)V_1^E + \lambda V_2^E$, where $\lambda = d_2 / (d_1 - d_2)$ and we must consider two sub-cases depending on the type of the feature pair
- ★ Let vector $u = \text{sgn}(d_2)(V_2^E - V_1^E)$ represent the edge E pointing out of the negative half-space to the positive half-space of β
- ★ If XY is a "VE" pair, the local test depends on the sign of the $(X - \rho)$ projection onto the edge vector u , i.e. $+\text{sgn}(u^T(X - \rho))$

Edge Clipping

- ★ If XY is a "EF" pair, there are another two sub-cases.
- ★ If p lies above the face Y , the local test depends on the angle between edge vector u and the face normal vector n
- ★ If p lies below the face Y we use the similar local test, but mark opposite features
- ★ Therefore the final local test (handling both sub-cases) can be written as: $-\text{sgn}(n^T u) \text{sgn}(d_s(p, Y))$

Edge Clipping Cases



ClipEdge Algorithm

In: An edge E , a feature N to be updated and a set of clipping feature pairs \mathcal{S}_N

Out: Test if the feature N was updated (**true/false**)

function CLIPEDGE(E, N, \mathcal{S}_N) : **bool**

```
1:  CLEARALL( $\mathcal{S}_N$ )
2:  foreach  $XY$  in  $\mathcal{S}_N$  do
3:       $\beta \leftarrow \mathcal{VP}(X, Y)$ 
4:       $\{d_1, d_2\} \leftarrow \{d_s(V_1^E, \beta), d_s(V_2^E, \beta)\}$            /* signed distances to  $\beta$  */
5:       $\{\mathbf{p}, \mathbf{u}\} \leftarrow \{E(d_2/(d_1 - d_2)), \text{sgn}(d_2)(V_2^E - V_1^E)\}$ 
6:      if ( $\text{sgn}(d_1 d_2) > 0$ ) then  $\text{Test} \leftarrow \text{sgn}(d_1)$ 
7:      if ( $\text{sgn}(d_1 d_2) < 0 \wedge XY$  is "VE") then  $\text{Test} \leftarrow +\text{sgn}(\mathbf{u}^T(X - \mathbf{p}))$ 
8:      if ( $\text{sgn}(d_1 d_2) < 0 \wedge XY$  is "EF") then  $\text{Test} \leftarrow -\text{sgn}(\mathbf{n}^T \mathbf{u}) \text{sgn}(d_s(\mathbf{p}, Y))$ 
9:      if ( $\text{Test} > 0$ ) then MARK( $X$ ) else MARK( $Y$ )
10: end
11: return UPDATECLEAR( $N, \mathcal{S}_N$ )
end
```

Signed Distance Maps



for collision detection

Signed Distance Map

- ★ **Signed distance map:** $\text{SDM}_N(V)$ is $N \times N \times N$ regular grid, where each unit cell with a center point ρ stores the signed distance to the closest point on the surface of some volume V .
- ★ This signed distance is a combination of a sign function $\text{sgn}_V(\rho)$ and the unsigned distance function $d(\rho, V)$ w.r.t. V .
 - $\text{SDM}_N(V) = \{ \text{sgn}_V(\rho) d(\rho, V) \mid \rho = (i + 0.5, j + 0.5, k + 0.5) \wedge 1 \leq i, j, k \leq N \}$

Signed Distance Maps

- ★ Signed distance maps (SDM) become recently a popular technique for approximate collision detection and distance computation.
- ★ Pros: Efficient overlap test, fast contact generation and penetration depth computation for arbitrary shaped, non-convex objects with complex and highly tessellated geometry
- ★ Suitable even for real-time applications as games
- ★ Cons: Huge amount of memory necessary for massive scenarios and a large number of redundant (unnecessary) contacts generated during the collision detection

Distance Map Construction

* Brute force construction

- For each grid cell we need to compute the distance of its center to each surface triangle and store the shortest distance
- Assuming N is the grid size and M is the number of triangles, we have to call the primitive point-to-triangle distance function $N \times N \times N \times M$ times

* Other Efficient Methods

- Lower-Upper Bound Tree (LUB-Tree)
- Characteristic/Scan Conversion (CSC)
- Chamfer and Vector Distance Transform (CDT, VDT)
- Fast Marching Method (FMM)

Proximity Queries with SDM

- ★ Performing proximity queries using SDM involves simple point location tests.
- ★ The key idea is to sample several points on the surface and store it together with the SDM.
- ★ During the collision detection sample points of one object are transformed into the local space of the other object and are "looked-up" in the SDM of the other object and vice versa.
- ★ Surface points located inside other object (lie under the zero level ($SDM_A(\rho_B) \leq 0$)) are used to create necessary contact information (contact point, contact normal, penetration depth, etc.)



The End