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

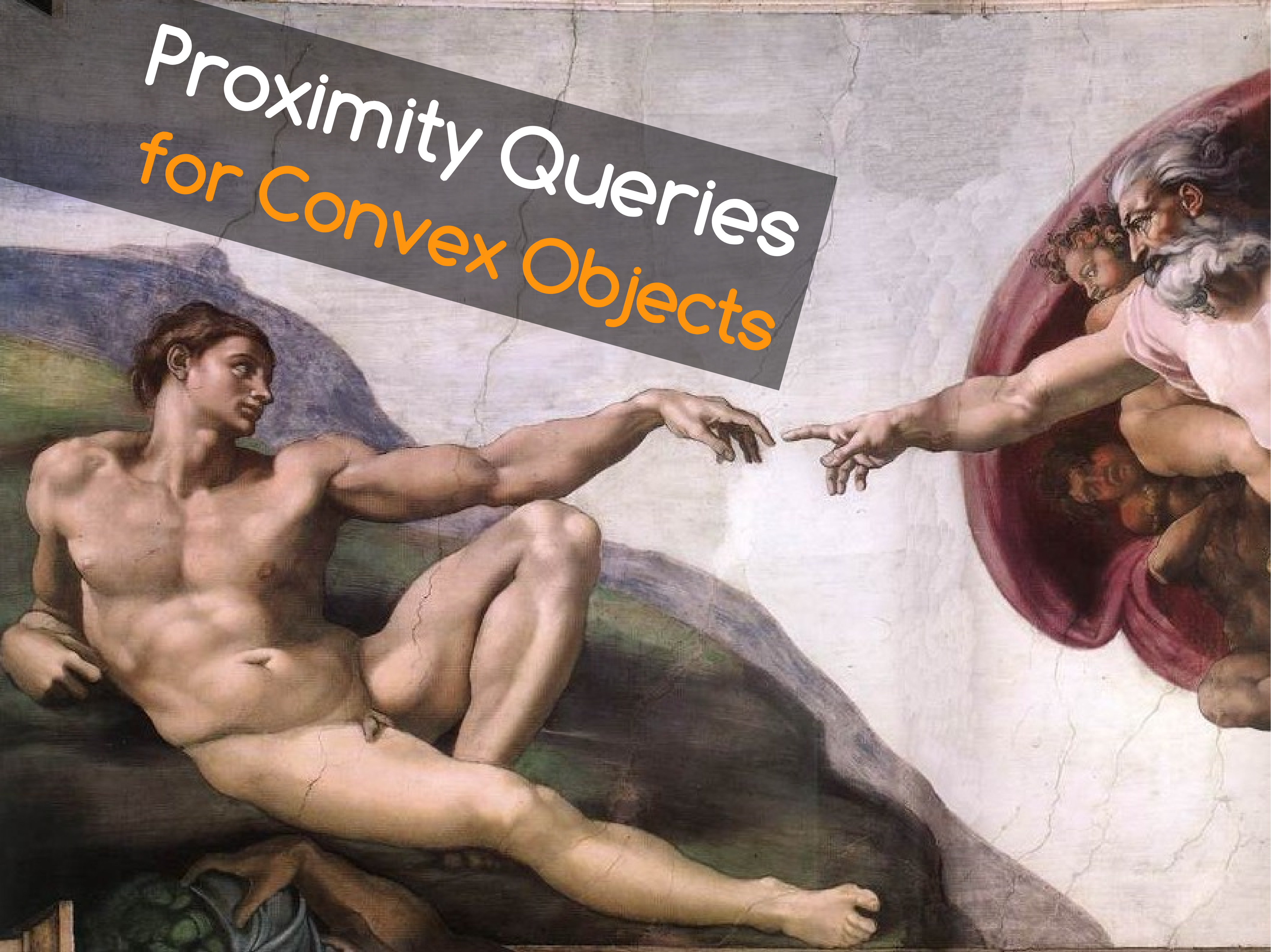
Narrow-Phase Collision Detection

- **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

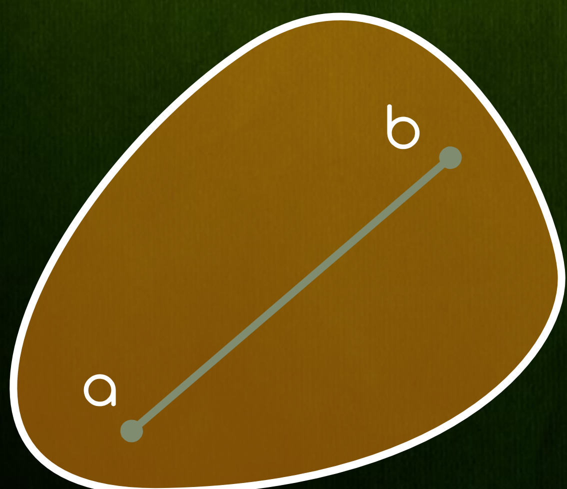
- **Output:** List of contact regions with necessary proximity information between colliding objects
- **Strategies:**
  - Simplex based traversal of CSO – GJK based algorithms
  - Feature tracking base algorithms as Lin-Canny or V-Clip
  - Signed Distance Maps for collision detection
  - Persistent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

Proximity Queries for Convex Objects

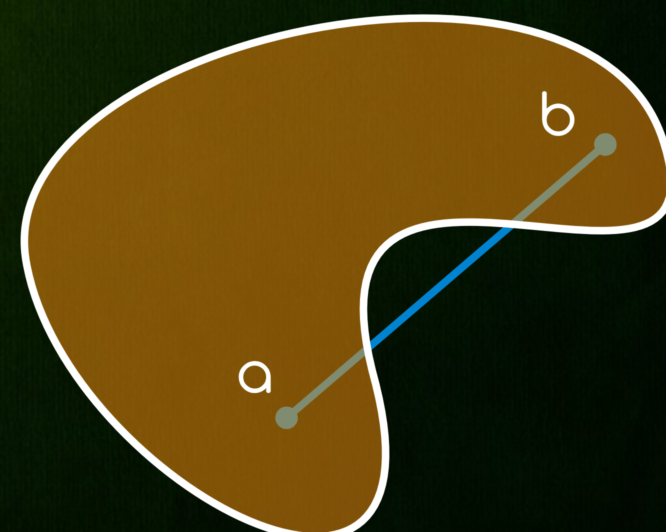


Minkowski Space

- Convex Bounded Point Set
  - A set S of points $\mathbf{p} \in \mathbf{R}^n$ is called convex and bounded if for any two points **a** and **b** the line segment **ab** lies entirely in S and the distance $|\mathbf{a} - \mathbf{b}|$ 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

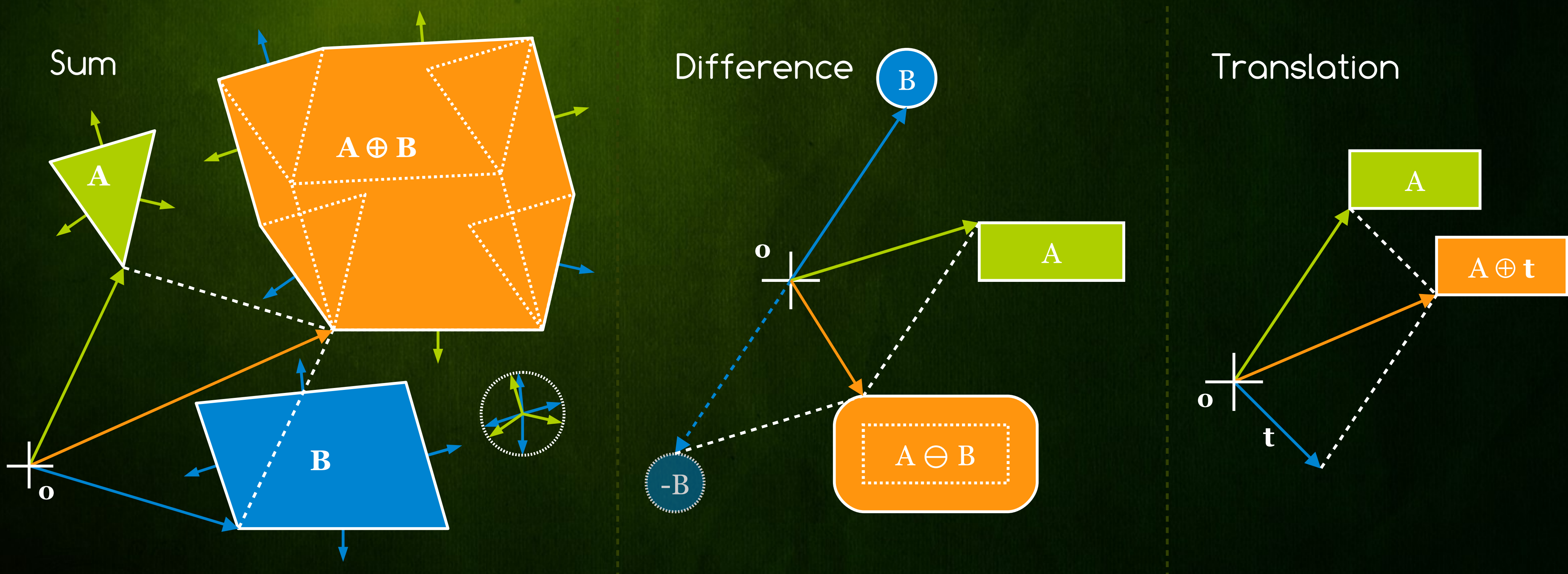
- Given any two convex objects A and B we define Minkowski Sum, Difference and Translation as

- 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 $A \oplus t$
  - $A \oplus t = A \oplus \{t\} = \{a + t \mid a \in A\}$

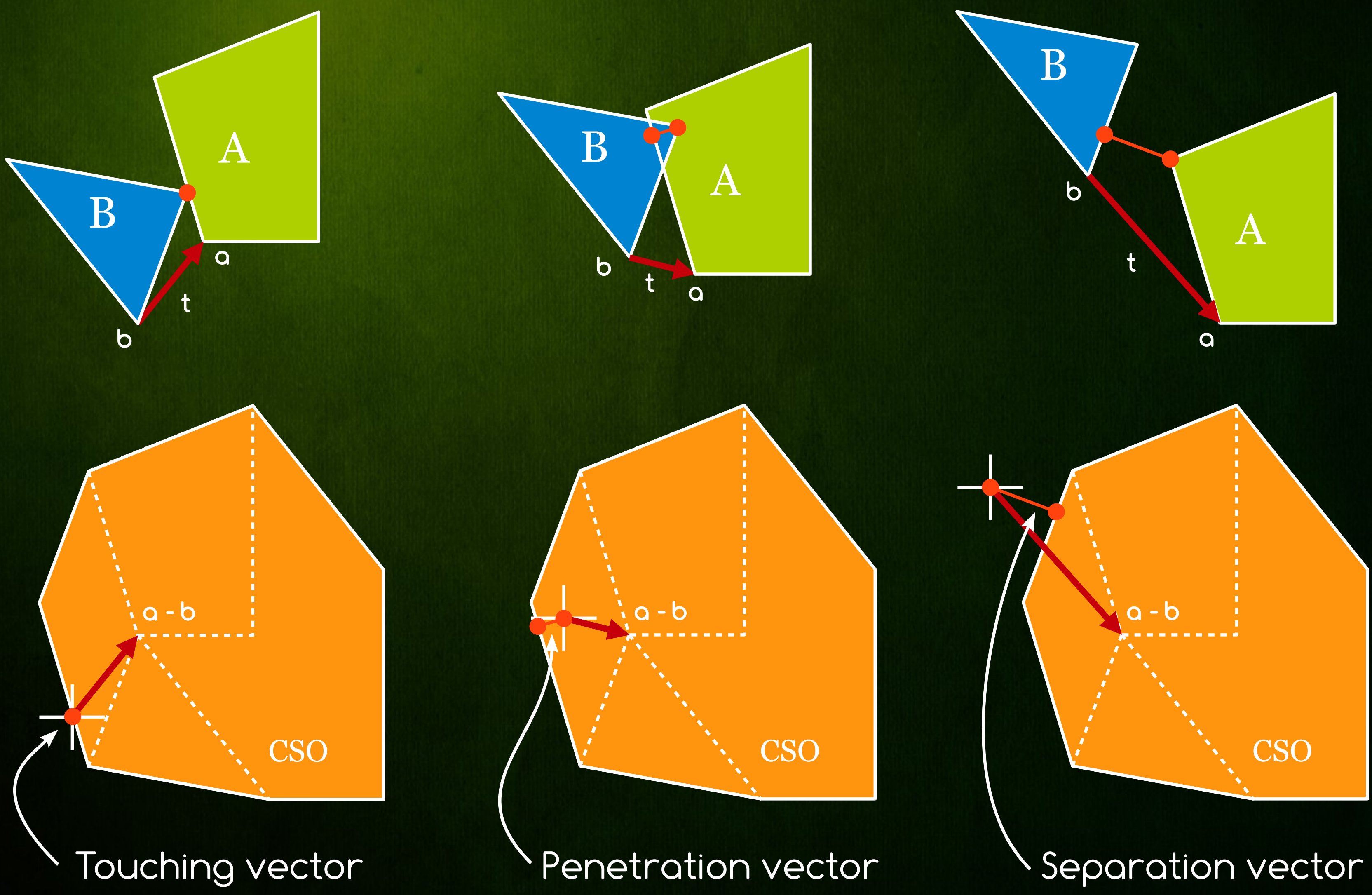
Minkowski Space



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.
  - $\mathbf{t}_{AB} \in \{t \mid A \cap (B \oplus t) \subset \beta \wedge t \in R^3 \wedge |t| = d_{AB}\}$
- Touching Distance
  - Touching distance d_{AB} is the length of touching vector $\mathbf{t}_{AB}$.
  - $d_{AB} = \min \{|t| \mid A \cap (B \oplus t) \subset \beta \wedge t \in R^3 \}$

Touching Vectors and CSO



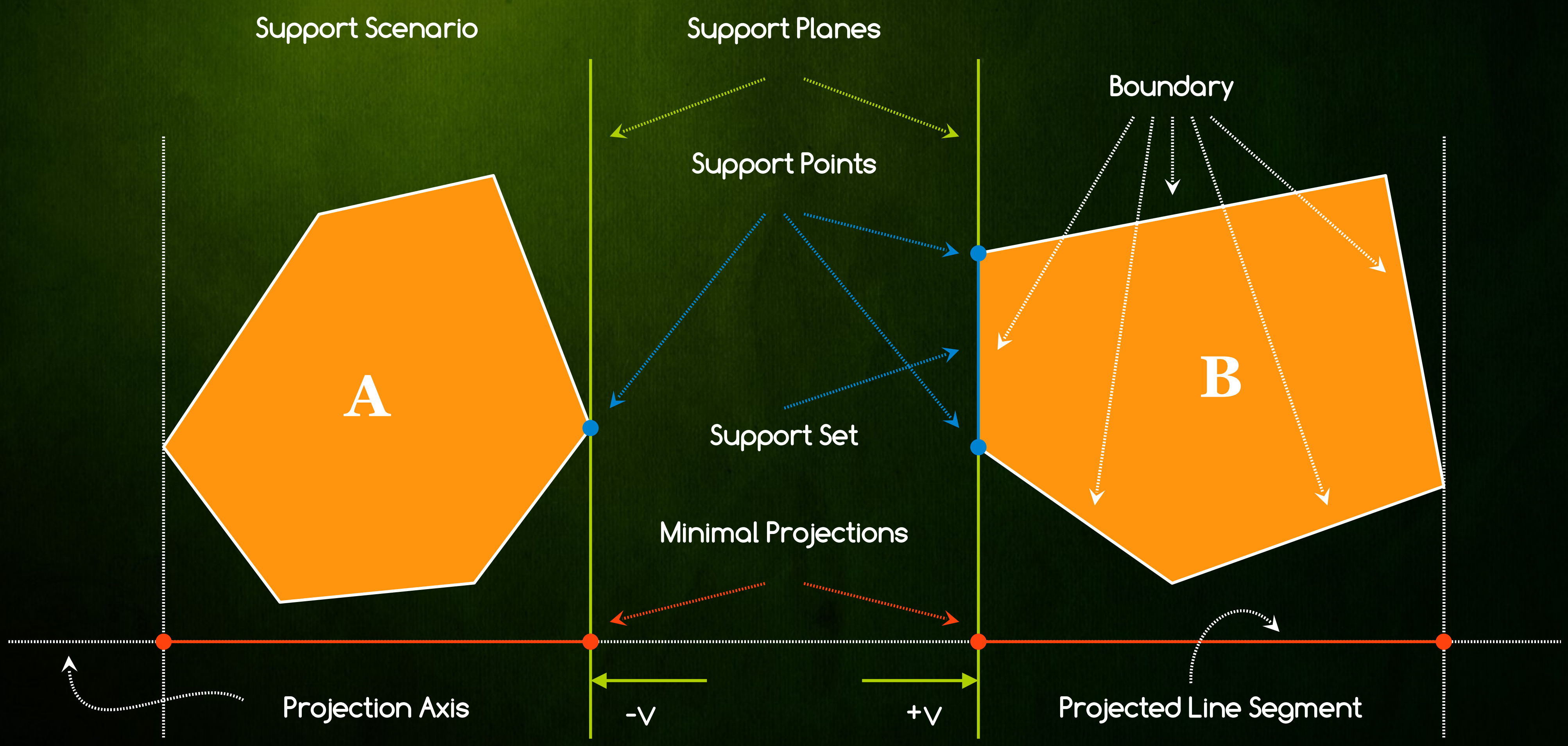
Touching Vectors

- 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
  - $\partial(C) = \{ p \mid p \in S^d_C \wedge d \in R^3 \}$

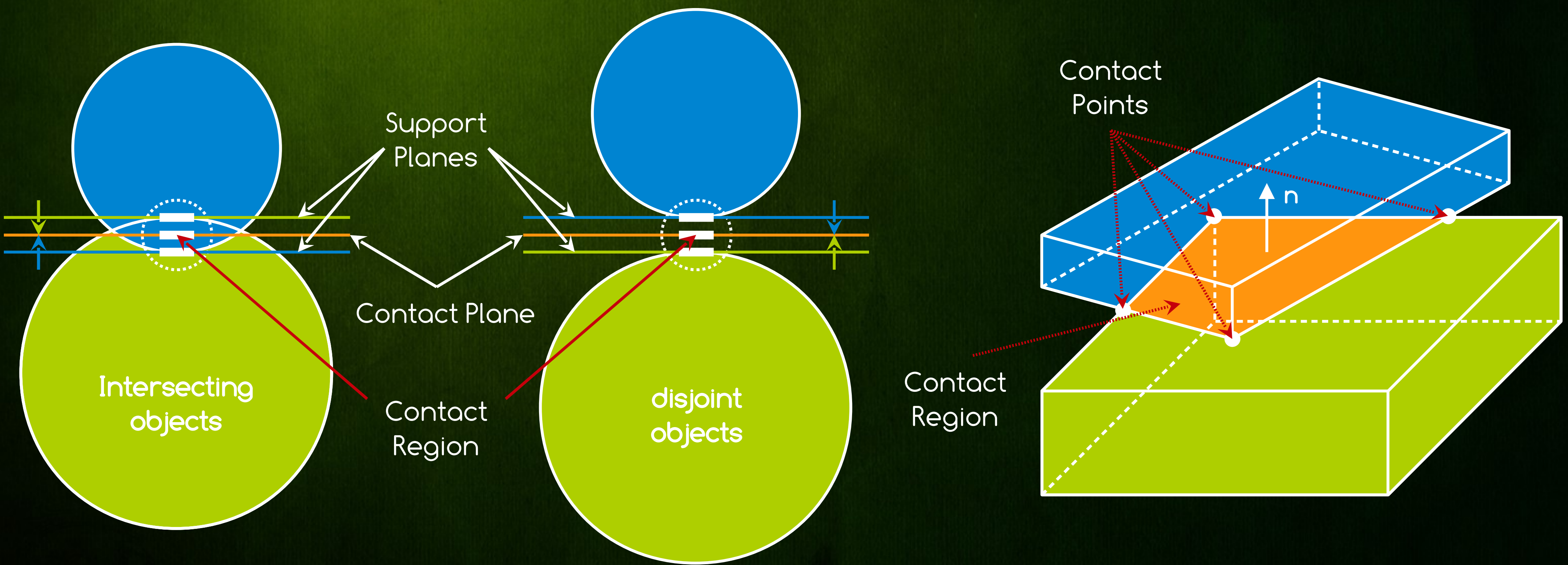
Support Set and Boundary



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^{+tAB} and S_B^{-tAB} for A and B w.r.t $\mathbf{t}_{AB}$
  - Project both sets onto touching vector $\mathbf{t}_{AB}$ and take median
  - Form contact plane with median as origin and normal as $\mathbf{t}_{AB}$
  - Project both support sets onto contact plane and take their (ideally) intersection as contact region

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.

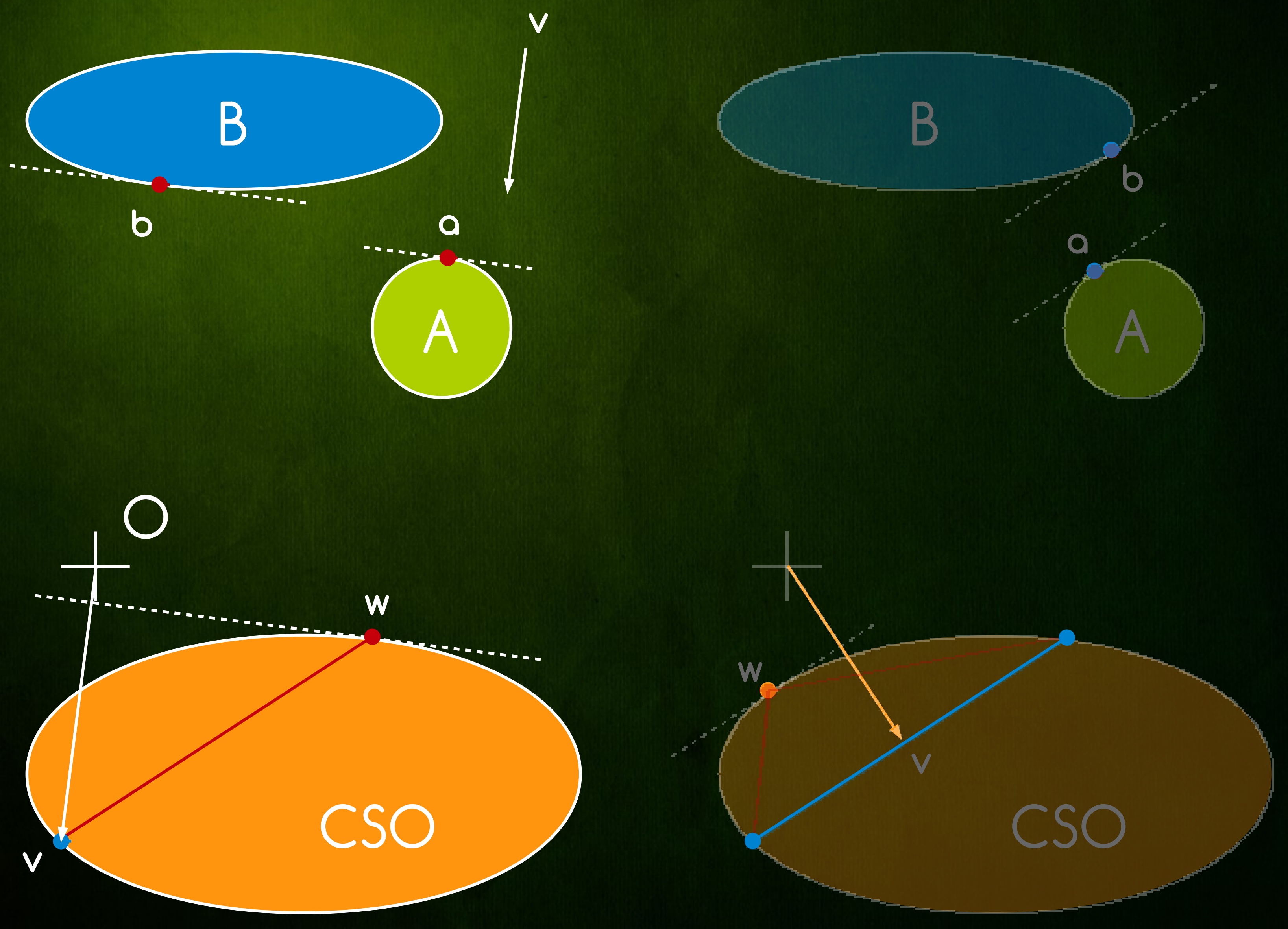
Support Function

- Support function **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 **support**$(A, d) \in S^d{}_A$ and **support**$(B, d) \in S^d{}_B$, we define the support functions as follows
  - **support**$(-B, d)$ = -**support**$(B,-d) \in S^d{}_{-B}$
  - **support**$(A \oplus B, d)$ = **support**(A, d) + **support**$(B, d) \in S^d{}_{A\oplus B}$
  - **support**$(A \ominus B, d)$ = **support**$(A \oplus (-B), d)$
    = **support**(A, d) + **support**$(-B, d)$
    = **support**$(A,+d)$ − **support**$(B,-d)$

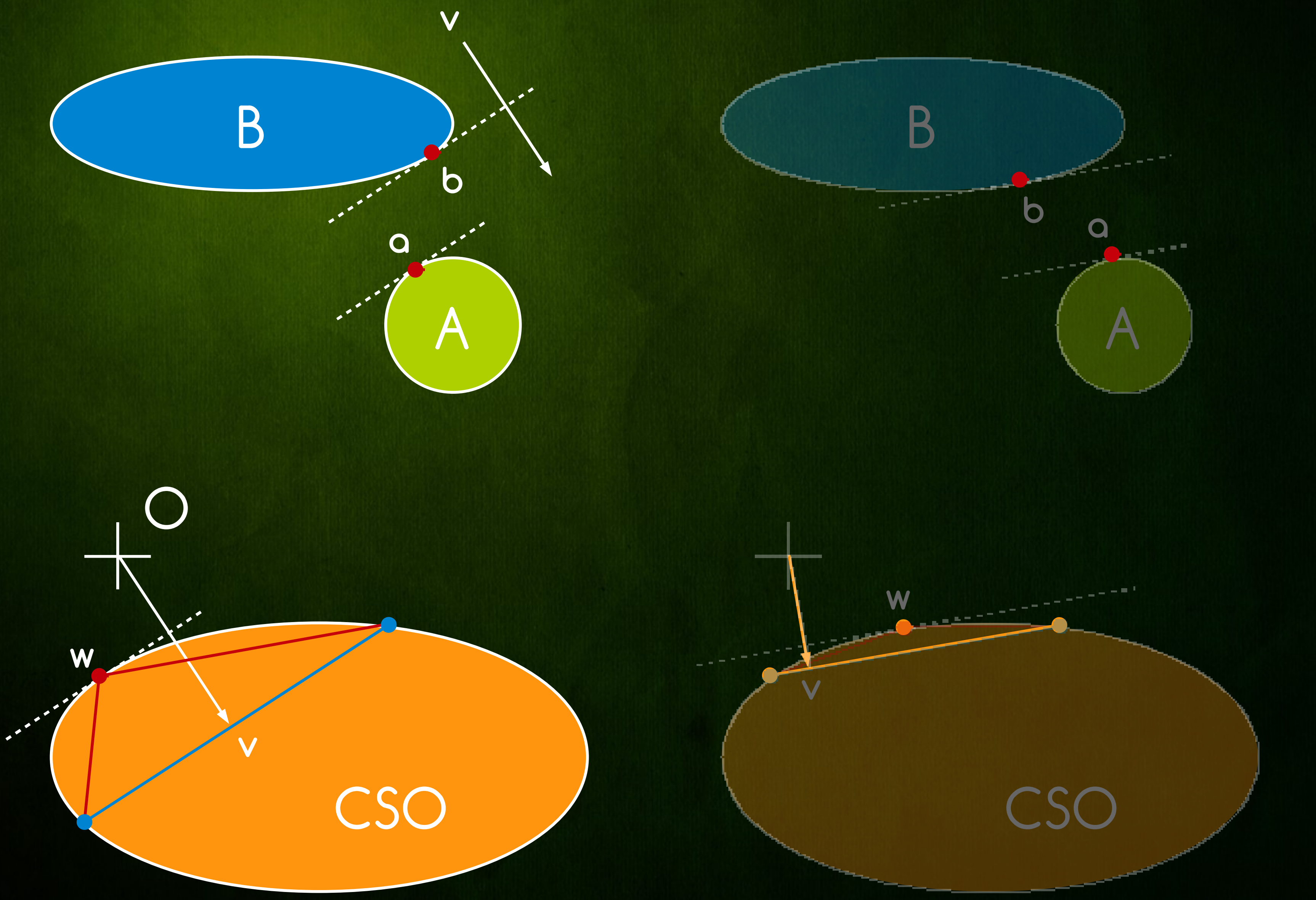
Proximity GJK Algorithm

- 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

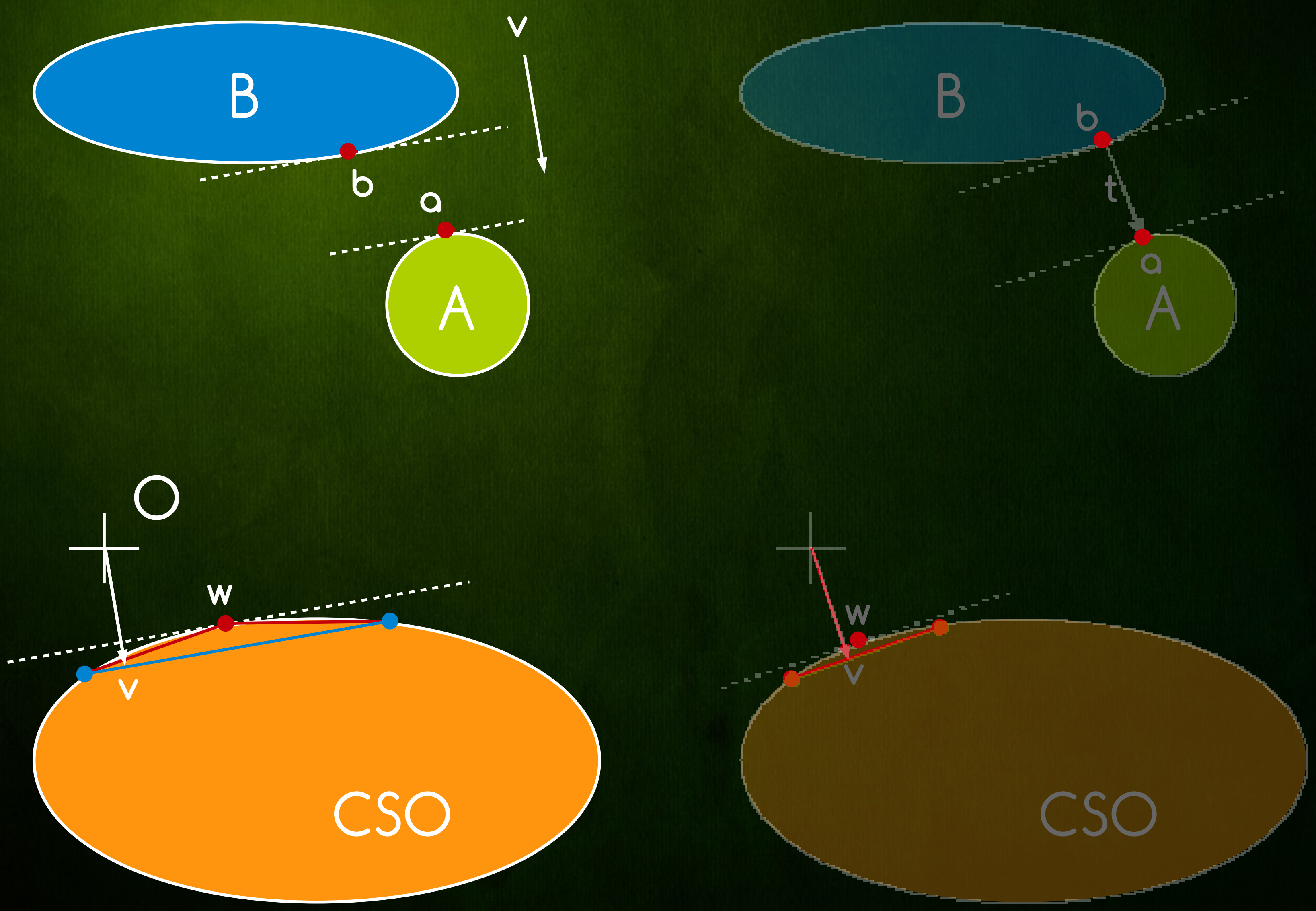
Proximity GJK Algorithm



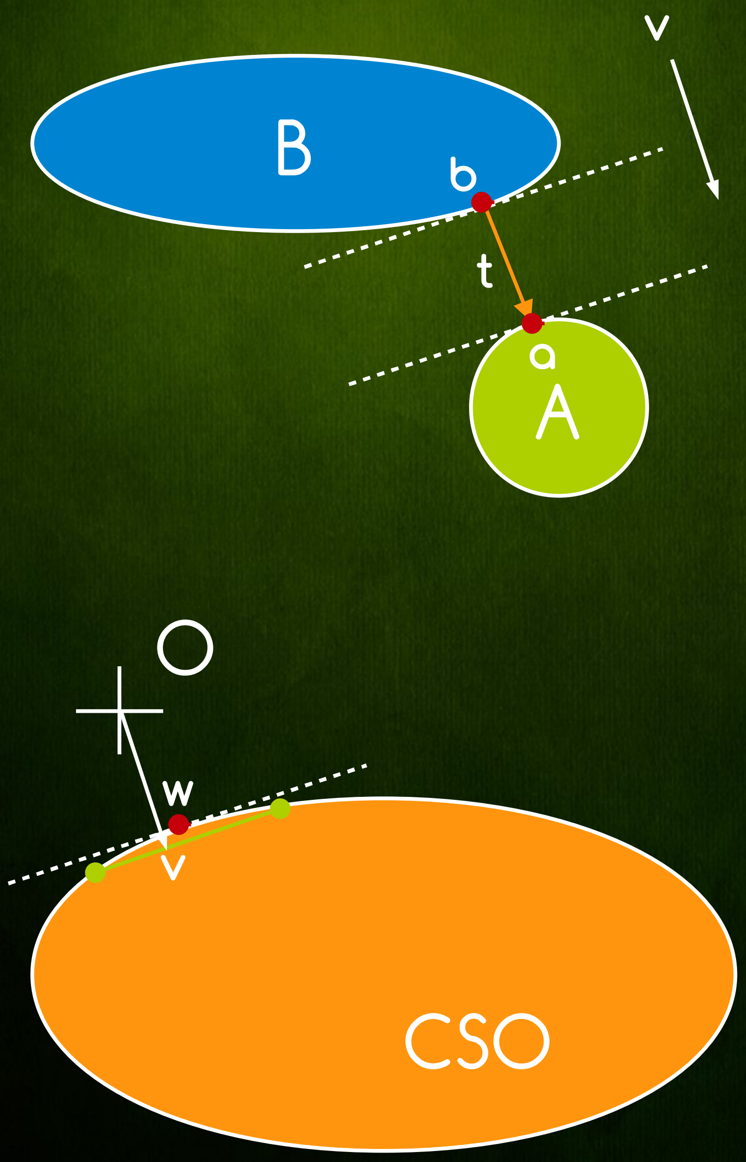
Proximity GJK Algorithm



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 $\textsc{ProximityGJK}(A, B, W) : \mathbf{w}$

  $\{\mathbf{v},\ \delta\} \leftarrow \{\mathbf{1},\ 0\}$

  while $(\|\mathbf{v}\|^2 - \delta^2 > \varepsilon)$ **do**

    $\mathbf{v} \leftarrow \textsc{ClosestPoint}(W)$

    $\mathbf{w} \leftarrow \textsc{Support}(A \ominus B, \mathbf{v}) = \textsc{Support}(A, +\mathbf{v}) - \textsc{Support}(B, -\mathbf{v})$

    $W \leftarrow \textsc{BestSimplex}(W, \mathbf{w})$

    if $(|W| = 4)$ **then** **return** $\textsc{ProximityEPA}(A, B, W)$;

    if $(\mathbf{v}^{\mathrm{T}}\mathbf{w} > 0)$ **then** $\delta^2 \leftarrow \max\left\{\delta^2,\ \frac{(\mathbf{v}^{\mathrm{T}}\mathbf{w})^2}{\|\mathbf{v}\|^2}\right\}$

  end

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

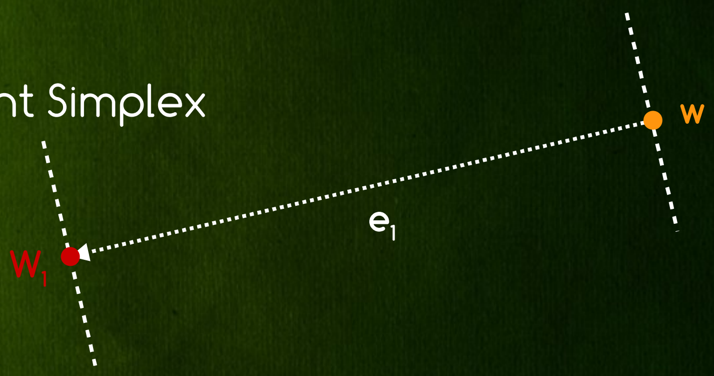
$\{\mu,\ \mathsf{Found}\} \leftarrow \{\mathbf{d}^{\mathrm{T}}\mathbf{w},\ \mathbf{false}\}$
while not Found **do**
  Found $\leftarrow$ **true**
  foreach $\mathbf{w}'$ **in** NEIGHBOURS$(\mathbf{w})$ **do**
    if $(\mathbf{d}^{\mathrm{T}}\mathbf{w}' < \mu)$ **then** $\{\mu,\ \mathbf{w},\ \mathsf{Found}\} \leftarrow \{\mathbf{d}^{\mathrm{T}}\mathbf{w}',\ \mathbf{w}',\ \mathbf{false}\}$; **break**
  end
end
return $\mathbf{w}$
end

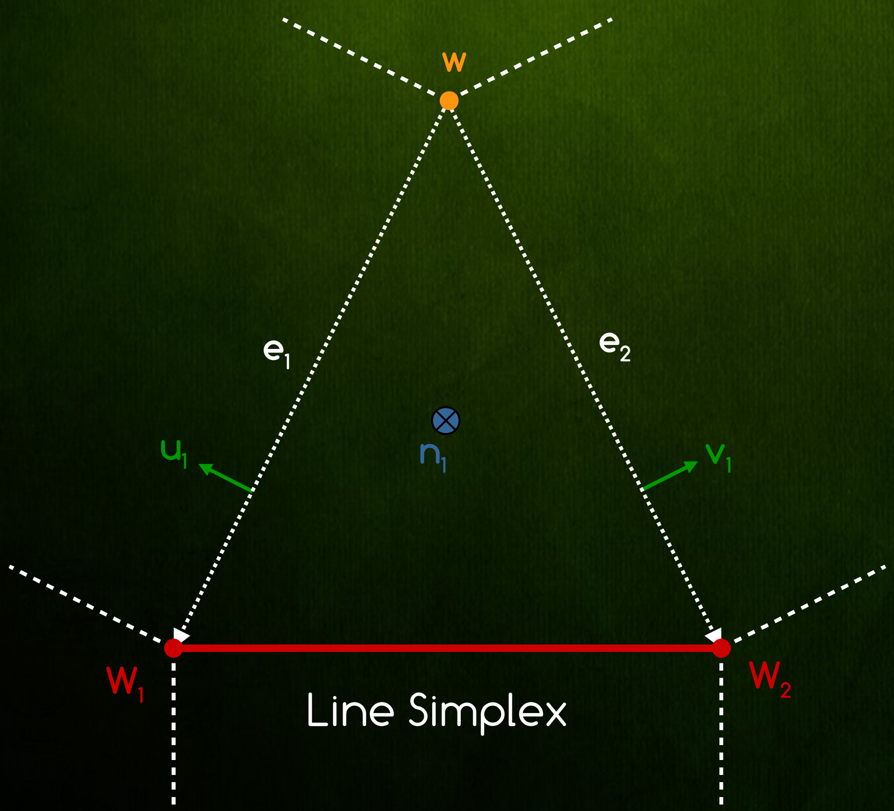
Simplex Refinement

- **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

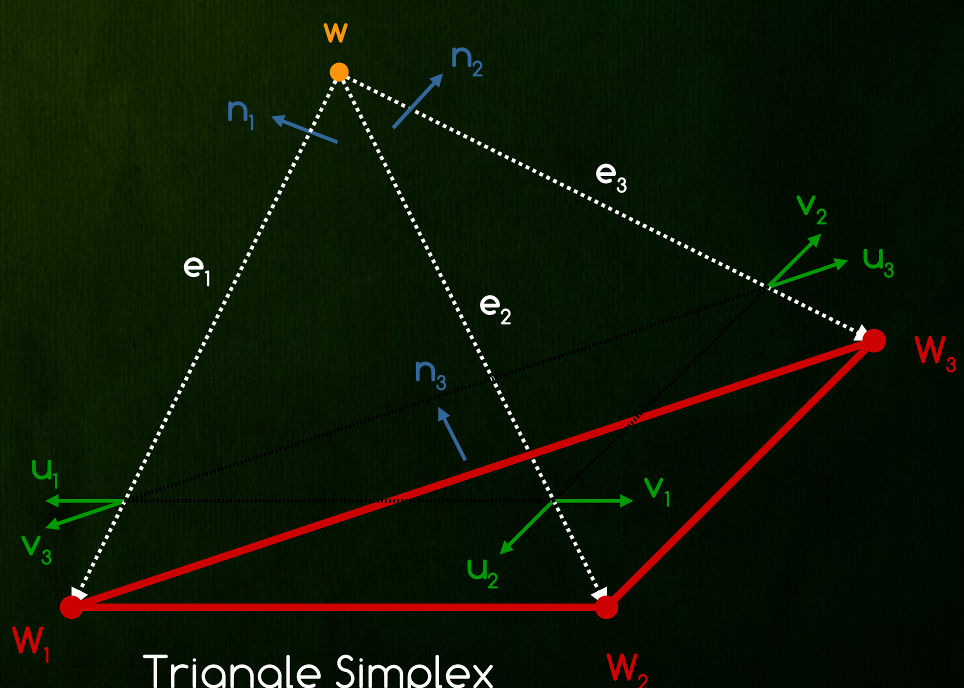
Voronoi Simplex Refinement

Point Simplex





Line Simplex



Triangle Simplex

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 $\mathbf{w}$

Out: New smallest simplex W containing $\mathbf{w}$ and the closest point to the origin

```
function BestSimplex(W, w) : W
    d ← 0 − w
    e1 ← W1 − w;        e2 ← W2 − w;        e3 ← W3 − w;
    n1 ← e1 × e2;       n2 ← e2 × e3;       n3 ← e3 × e1;
    u1 ← e1 × n1;       u2 ← e2 × n2;       u3 ← e3 × n3;
    v1 ← n1 × e2;       v2 ← n2 × e3;       v3 ← n3 × e1;
    switch |W| do
        case 0                                                    /* empty simplex */
            return {w}
        end
        case 1                                                    /* vertex simplex */
            if (d^T e1 > 0) then return {w}
            if (d^T e1 < 0) then return {W1, w}
        end
        case 2                                                    /* edge simplex */
            if (d^T e1 < 0) ∧ (d^T e2 < 0) then return {w}
            if (d^T e1 > 0) ∧ (d^T u1 > 0) then return {W1, w}
            if (d^T e2 > 0) ∧ (d^T v1 > 0) then return {W2, w}
            if (d^T u1 < 0) ∧ (d^T v1 < 0) then return {W1, W2, w}
        end
        case 3                                                    /* face simplex */
            if (d^T e1 < 0) ∧ (d^T e2 < 0) ∧ (d^T e3 < 0) then return {w}
            if (d^T e1 > 0) ∧ (d^T u1 > 0) ∧ (d^T v3 > 0) then return {W1, w}
            if (d^T e2 > 0) ∧ (d^T u2 > 0) ∧ (d^T v1 > 0) then return {W2, w}
            if (d^T e3 > 0) ∧ (d^T u3 > 0) ∧ (d^T v2 > 0) then return {W3, w}
            if (d^T n1 > 0) ∧ (d^T u1 < 0) ∧ (d^T v1 < 0) then return {W1, W2, w}
            if (d^T n2 > 0) ∧ (d^T u2 < 0) ∧ (d^T v2 < 0) then return {W2, W3, w}
            if (d^T n3 > 0) ∧ (d^T u3 < 0) ∧ (d^T v3 < 0) then return {W3, W1, w}
            if (d^T n1 < 0) ∧ (d^T n2 < 0) ∧ (d^T n3 < 0) then return {W1, W2, W3, w}
        end
    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) : $\mathbf{v}$

$\mathbf{d} \leftarrow \mathbf{W}_2 - \mathbf{W}_1$

$\mathbf{n} \leftarrow (\mathbf{W}_2 - \mathbf{W}_1) \times (\mathbf{W}_3 - \mathbf{W}_1)$

switch $|W|$ **do**

case *0* **return** $\mathbf{0}$; `/* empty simplex */`

case *1* **return** $\mathbf{W}_1$; `/* vertex simplex */`

case *2* **return** $\mathbf{W}_1 - \frac{\mathbf{d}^T\mathbf{W}_1}{\mathbf{d}^T\mathbf{d}}\mathbf{d}$; `/* edge simplex */`

case *3* **return** $\frac{\mathbf{n}^T\mathbf{W}_1}{\mathbf{n}^T\mathbf{n}}\mathbf{n}$; `/* face simplex */`

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 $\textsc{OverlapGJK}(A, B, W)$: **bool**

  $\{\mathbf{v}, \mathbf{w}\} \leftarrow \{\mathbf{1},\ \mathbf{1}\}$

  while $(\mathbf{v}^{\mathrm{T}}\mathbf{w} \leq 0)$ **do**

    $\mathbf{v} \leftarrow \textsc{ClosestPoint}(W)$

    $\mathbf{w} \leftarrow \textsc{Support}(A \ominus B, \mathbf{v}) = \textsc{Support}(A, +\mathbf{v}) - \textsc{Support}(B, -\mathbf{v})$

    $W \leftarrow \textsc{BestSimplex}(W, \mathbf{w})$

    if $(|W| = 4)$ **then** **return true** ; `/* intersection */`

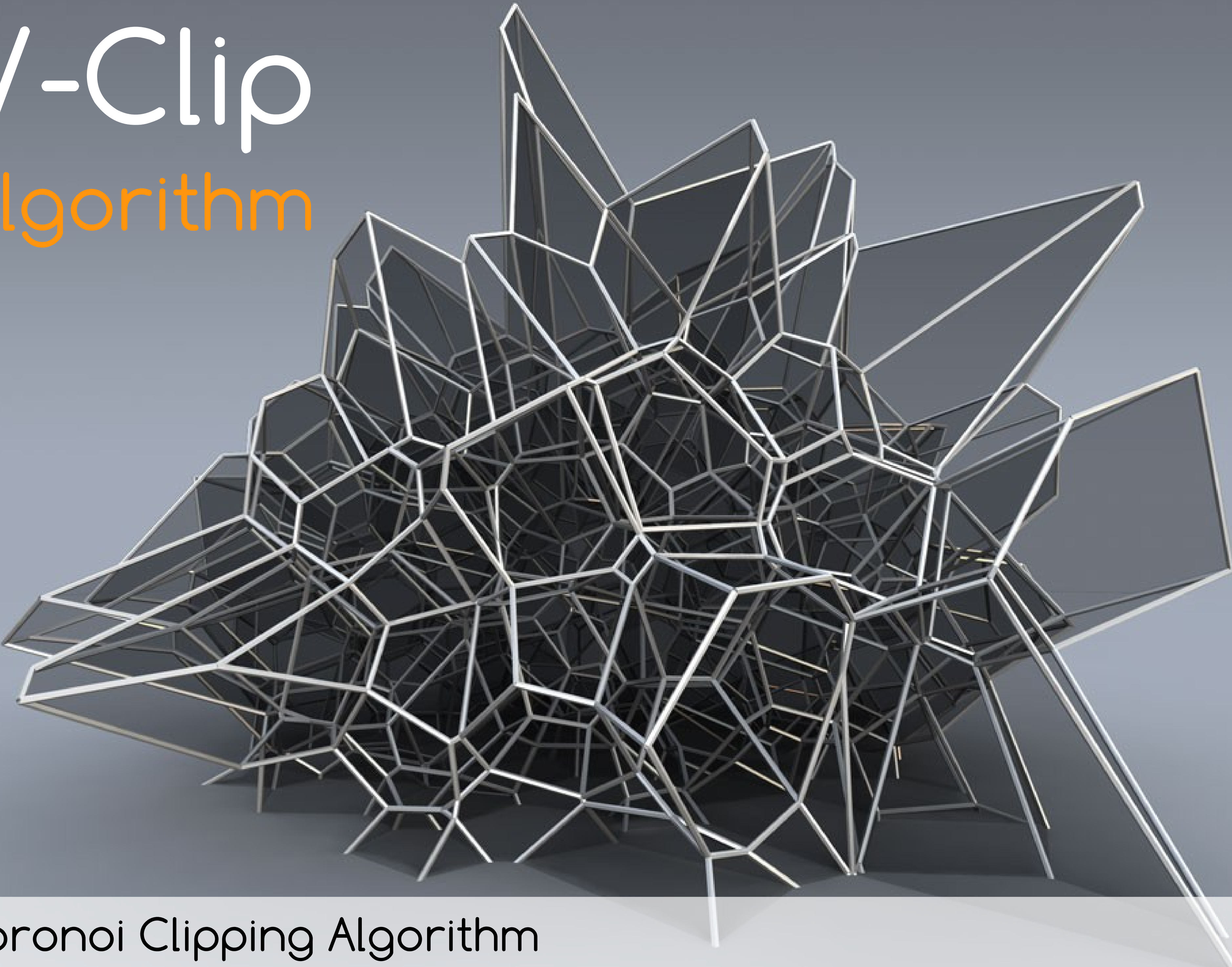
  end

  return false

end

V-Clip

Algorithm



Voronoi Clipping Algorithm

External Voronoi Regions

- Interior Set:
  - The set of all interior points 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)
- $int(C) = \{ c \in R3 \mid ds(c, F) < 0 \wedge F \in C \}$

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

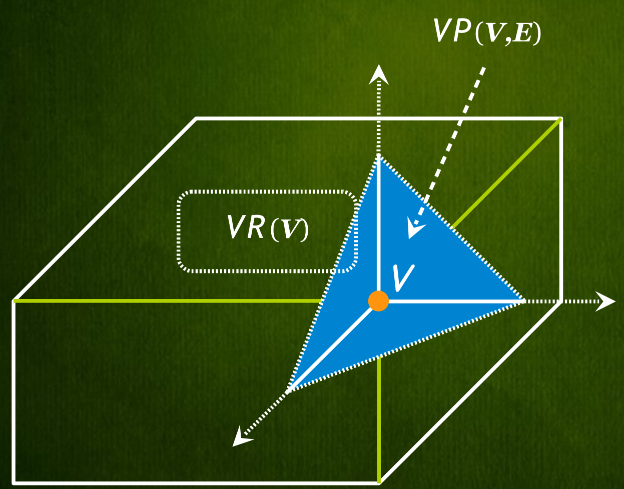
External Voronoi Regions

- 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
- $ds(c, F) = n^T_F (c - o_F)$
- 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
- $X \cap Y \wedge \dim(X) < \dim(Y) \Rightarrow X \subset Y \Rightarrow d(c,X) \leq d(c,Y)$

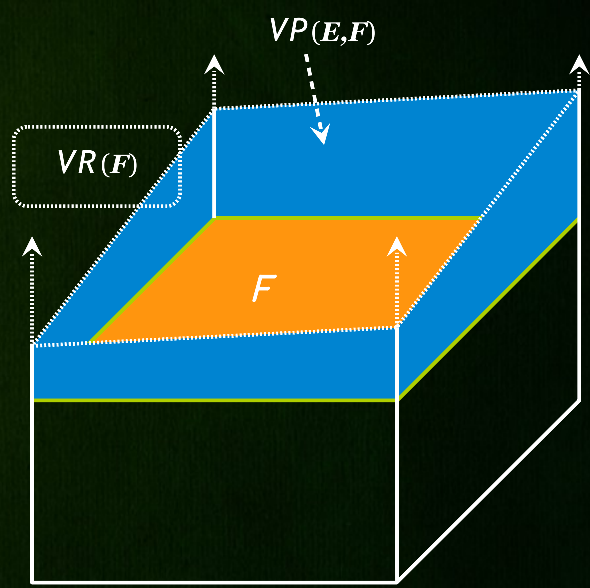
External Voronoi Regions

- 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
  - $VR(X) = \{ c \notin int(C) \mid d(c,X) \leq d(c, Y) \wedge Y \in C \}$
- 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 $y \in Y$
  - $d(X,Y) = \min \{ |x - y| \mid x \in X \wedge y \in Y \}$

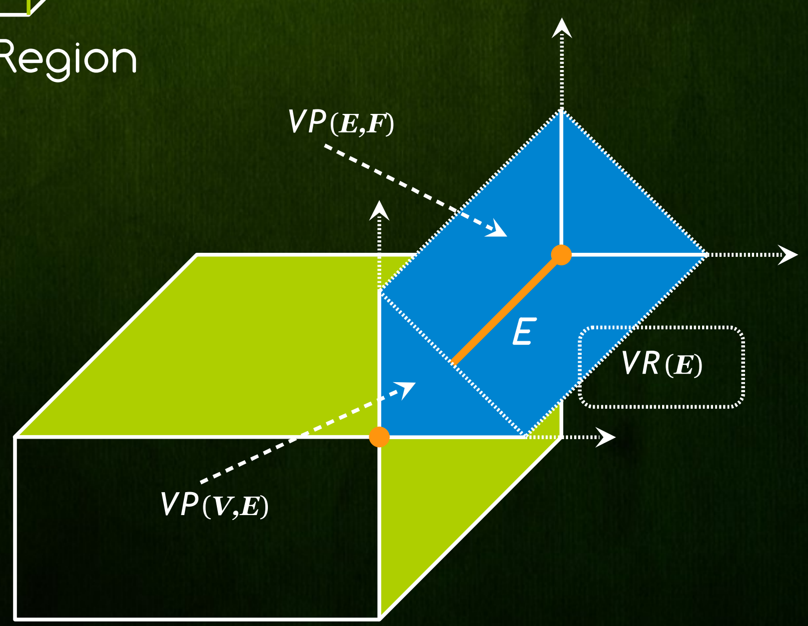
External Voronoi Regions



Vertex Voronoi Region



Face Voronoi Region

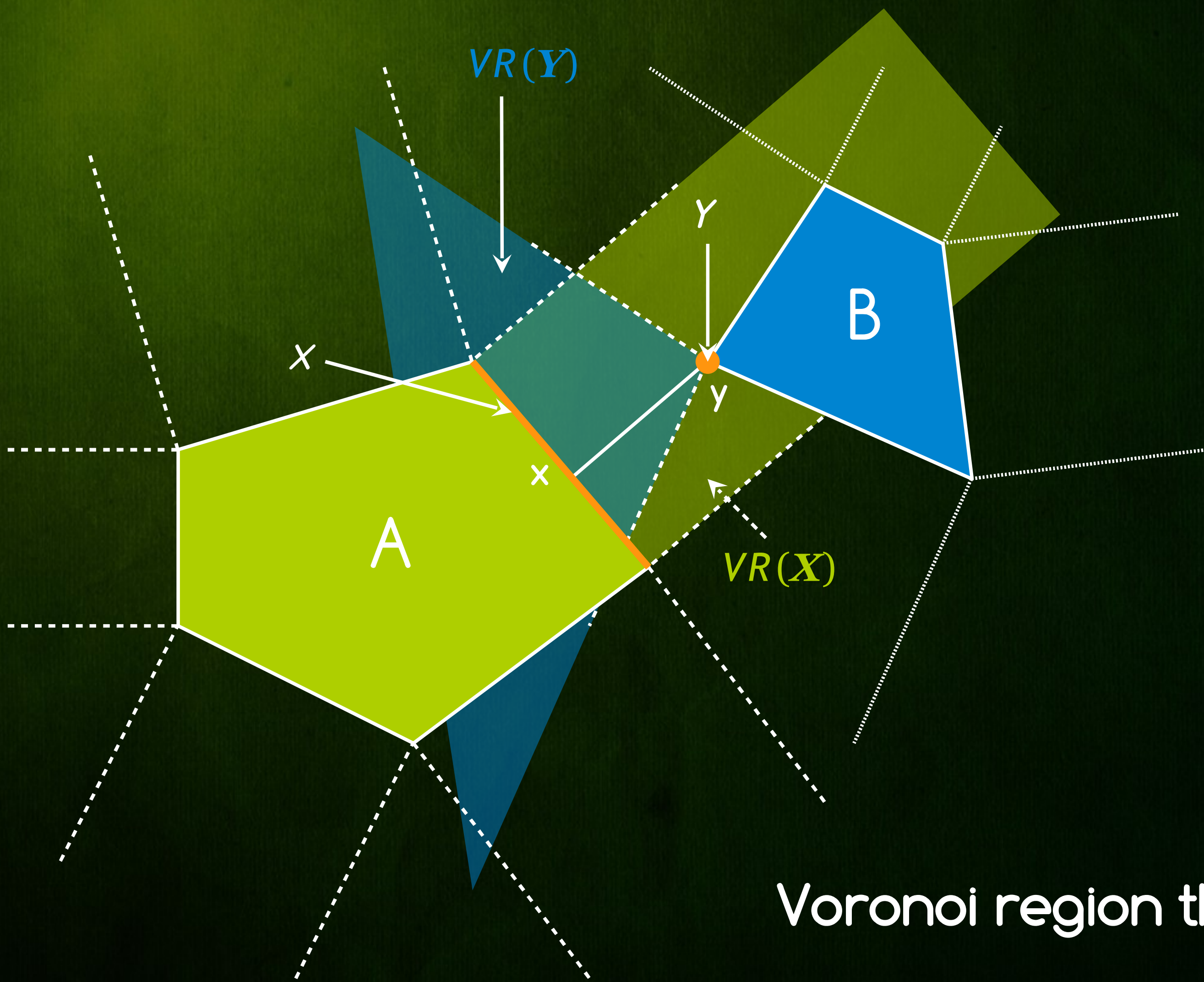


Edge Voronoi Region

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 as 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 $\mathbf{w}$, or $\emptyset$ if penetration occurred

function V-CLIP(A, B, X, Y) : $\mathbf{w}$

  while (**true**) **do**

    switch PAIRTYPE(X, Y) **do**

      case *VV type* : `/* Vertex-Vertex */`

        if CLIPVERTEX$(X, Y, \{\ YE \ \mid\ E \in \text{EDGES}(Y)\ \})$ **then continue**

        if CLIPVERTEX$(Y, X, \{\ XE \ \mid\ E \in \text{EDGES}(X)\ \})$ **then continue**

        return $X - Y$

      end

      case *VE type* : `/* Vertex-Edge */`

        if CLIPVERTEX$(X, Y, \{\ V_1^Y Y,\ V_2^Y Y,\ YF_1^Y,\ YF_2^Y\ \})$ **then continue**

        if CLIPEDGE$(Y, X, \{\ XE \ \mid\ E \in \text{EDGES}(X)\ \})$ **then continue**

        $\mathbf{u} \leftarrow V_2^Y - V_1^Y$

        return $X - \left(V_1^Y + \dfrac{\mathbf{u}^{\mathrm{T}}(X - V_1^Y)}{\mathbf{u}^{\mathrm{T}}\mathbf{u}}\mathbf{u}\right)$

      end

      case *VF type* : `/* Vertex-Face */`

        if CLIPVERTEX$(X, Y, \{\ EY,\ V_1^E E,\ V_2^E E \ \mid\ E \in \text{EDGES}(Y)\ \})$ **then continue**

        if CLIPFACE(Y, X, A) **then continue**

        return $X - \left(X + \dfrac{\mathbf{n}^{\mathrm{T}}(V_1^Y - X)}{\mathbf{n}^{\mathrm{T}}\mathbf{n}}\mathbf{n}\right)$

      end

      case *EE type* : `/* Edge-Edge */`

        if CLIPEDGE$(X, Y, \{\ V_1^Y Y,\ V_2^Y Y,\ YF_1^Y,\ YF_2^Y\ \})$ **then continue**

        if CLIPEDGE$(Y, X, \{\ V_1^X X,\ V_2^X X,\ XF_1^X,\ XF_2^X\ \})$ **then continue**

        $\{\mathbf{u}^X,\ \mathbf{u}^Y\} \leftarrow \{V_2^X - V_1^X,\ \ V_2^Y - V_1^Y\}$

        $\{\mathbf{n}^X,\ \mathbf{n}^Y\} \leftarrow \{(\mathbf{u}^X \times \mathbf{u}^Y) \times \mathbf{u}^Y,\ \ (\mathbf{u}^Y \times \mathbf{u}^X) \times \mathbf{u}^X\}$

        return $\left(V_1^X + \dfrac{(\mathbf{n}^Y)^{\mathrm{T}}(V_1^Y - V_1^X)}{(\mathbf{n}^Y)^{\mathrm{T}}\mathbf{u}^X}\mathbf{u}^X\right) - \left(V_1^Y + \dfrac{(\mathbf{n}^X)^{\mathrm{T}}(V_1^X - V_1^Y)}{(\mathbf{n}^X)^{\mathrm{T}}\mathbf{u}^Y}\mathbf{u}^Y\right)$

      end

      case *EF type* : `/* Edge-Face */`

        if CLIPEDGE$(X, Y, \{\ EY,\ V_1^E E,\ V_2^E E \ \mid\ E \in \text{EDGES}(Y)\ \})$ **then continue**

        $\{d_1,\ d_2\} \leftarrow \{\mathrm{d_s}(V_1^X, Y),\ \ \mathrm{d_s}(V_2^X, Y)\}$

        if $(\mathrm{sgn}(d_1 d_2) < 0)$ **then** $Y \leftarrow \emptyset$; **continue**

        if $(|d_1| < |d_2|)$ **then** $X \leftarrow V_1^X$ **else** $X \leftarrow V_2^X$

        continue

      end

      case *EV, FV, FE type* : SWAP(X, Y); SWAP(A, B); **continue** ; `/* Swap Cases */`

    end

    if $(Y = \emptyset)$ **then return** $\emptyset$

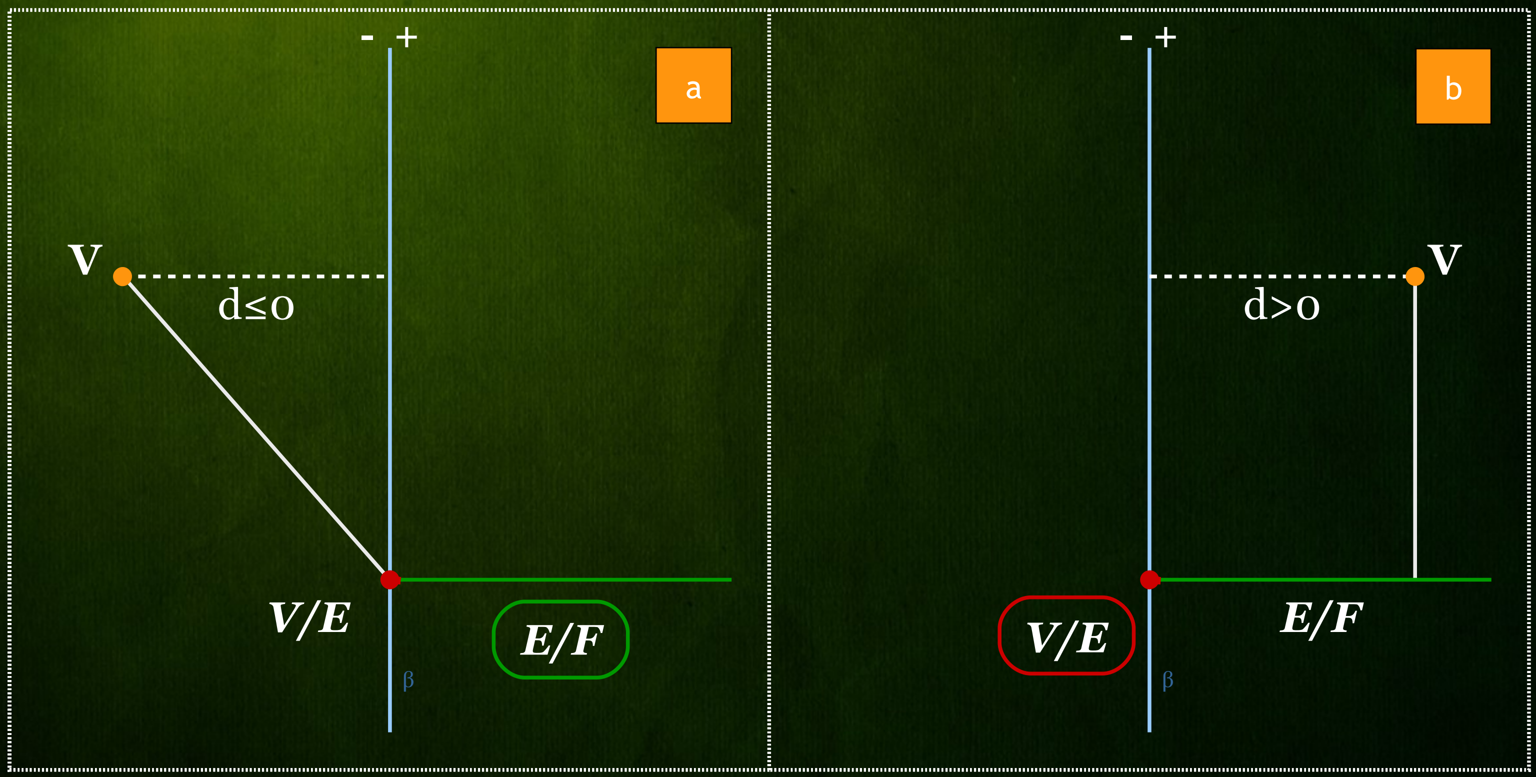
  end

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



ClipVertex and UpdateClear

In: A vertex V, 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 ClipVertex($V, N, \mathcal{S}_N$) : **bool**

ClearAll($\mathcal{S}_N$)
foreach XY **in** $\mathcal{S}_N$ **do**
  Test $\leftarrow \text{sgn}(\text{d}_s(V, \mathcal{VP}(X, Y)))$
  if (Test > 0) **then** Mark(X) **else** Mark(Y)
end
return UpdateClear($N, \mathcal{S}_N$)
end

In: A feature N to be updated and a set of clipping feature pairs S_N
Out: Test if the feature N was updated (**true**/**false**)

function UpdateClear(N, S_N) : **bool**

M $\leftarrow$ N; `/* store old feature */`
foreach XY **in** S_N **do**
  if (X **is** *"clear"*) **then** $N \leftarrow X$; **break;** `/* update old to closest feature */`
  if (Y **is** *"clear"*) **then** $N \leftarrow Y$; **break;** `/* update old to closest feature */`
end
return N **!=** M; `/* true if feature changed */`
end

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 $(\mathrm{sgn}(d_1 d_2) > 0)$, we simply mark the feature of the opposite side as in vertex clipping

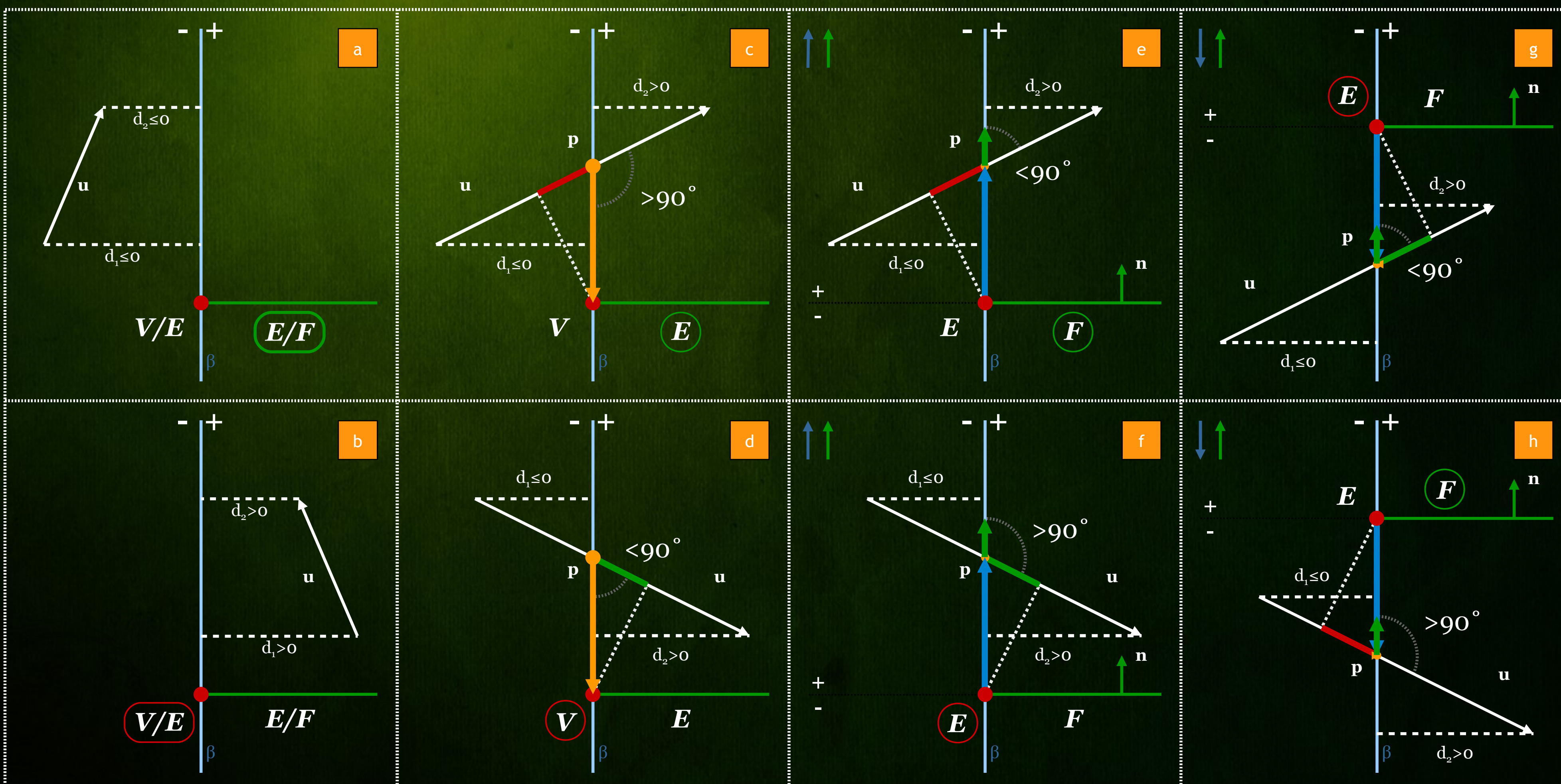
Edge Clipping

- If vertices lie on different sides ($\mathrm{sgn}(d_1 d_2) < 0$), 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 $u = \mathrm{sgn}(d_2)(V_2^E - V_1^E)$ 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. $+\mathrm{sgn}(u^T(X - \rho))$

Edge Clipping

- If XY is a "EF" pair, there are another two sub-cases.
- If ρ lies above the face Y, the local test depends on the angle between edge vector u and the face normal vector n
- If ρ 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: $-\operatorname{sgn}(n^T u)\operatorname{sgn}(d_s(\rho, Y))$

Edge Clipping Cases



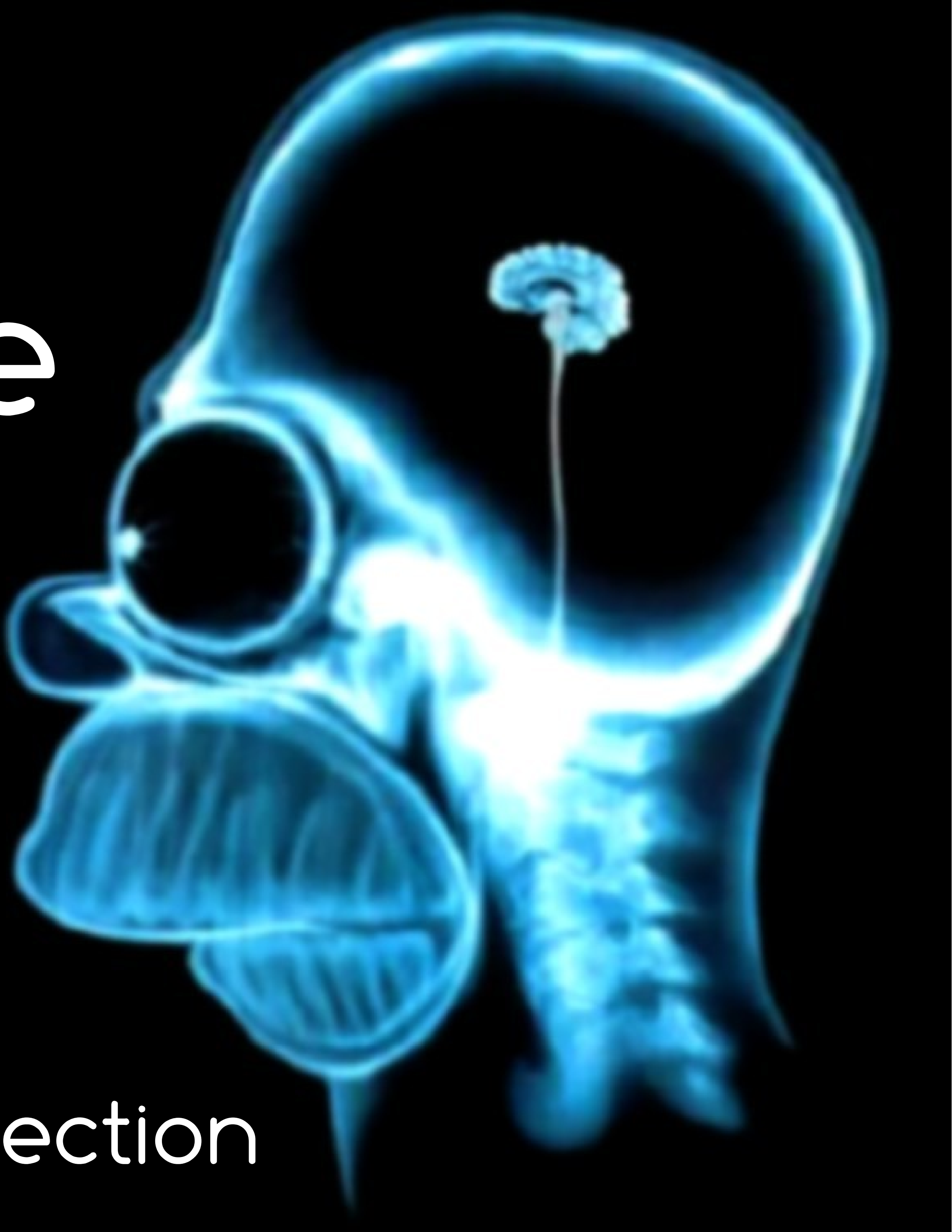
ClipEdge Algorithm

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 $\textsc{ClipEdge}(E, N, \mathcal{S}_N)$: **bool**

$\textsc{ClearAll}(\mathcal{S}_N)$
foreach XY **in** $\mathcal{S}_N$ **do**
  $\beta \leftarrow \mathcal{VP}(X, Y)$
  $\{d_1,\ d_2\} \leftarrow \{\ \mathrm{d_s}(V_1^E, \beta),\ \mathrm{d_s}(V_2^E, \beta)\ \}$ `/* signed distances to` β `*/`
  $\{\mathbf{p},\ \mathbf{u}\} \leftarrow \{\ E(d_2/(d_1 - d_2)),\ \mathrm{sgn}(d_2)(V_2^E - V_1^E)\ \}$
  if $(\mathrm{sgn}(d_1 d_2) > 0)$ **then** $\mathsf{Test} \leftarrow \mathrm{sgn}(d_1)$
  if $(\mathrm{sgn}(d_1 d_2) < 0 \ \wedge\ XY$ **is** "VE") **then** $\mathsf{Test} \leftarrow +\mathrm{sgn}(\mathbf{u}^{\mathrm{T}}(X - \mathbf{p}))$
  if $(\mathrm{sgn}(d_1 d_2) < 0 \ \wedge\ XY$ **is** "EF") **then** $\mathsf{Test} \leftarrow -\mathrm{sgn}(\mathbf{n}^{\mathrm{T}}\mathbf{u})\,\mathrm{sgn}(\mathrm{d_s}(\mathbf{p}, Y))$
  if $(\mathsf{Test} > 0)$ **then** $\textsc{Mark}(X)$ **else** $\textsc{Mark}(Y)$
end
return $\textsc{UpdateClear}(N, \mathcal{S}_N)$
end

Signed Distance Maps



for collision detection

Signed Distance Map

- **Signed distance map:** $SDM_N(V)$ 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 $sgn_V(\rho)$ and the unsigned distance function $d(\rho, V)$ w.r.t. V.
  - $SDM_N(V) = \{ sgn_V(\rho) d(\rho, V) \mid \rho = (i + 0.5, j + 0.5, k + 0.5) \wedge 1 \le i, j, k \le N \}$

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 ($SDM_A(p_B) \leq 0$)) are used to create necessary contact information (contact point, contact normal, penetration depth, etc.)


The End