Juraj Onderik | onderik@sccg.sk

Lesson 05

Mid Phase

Collision Detection

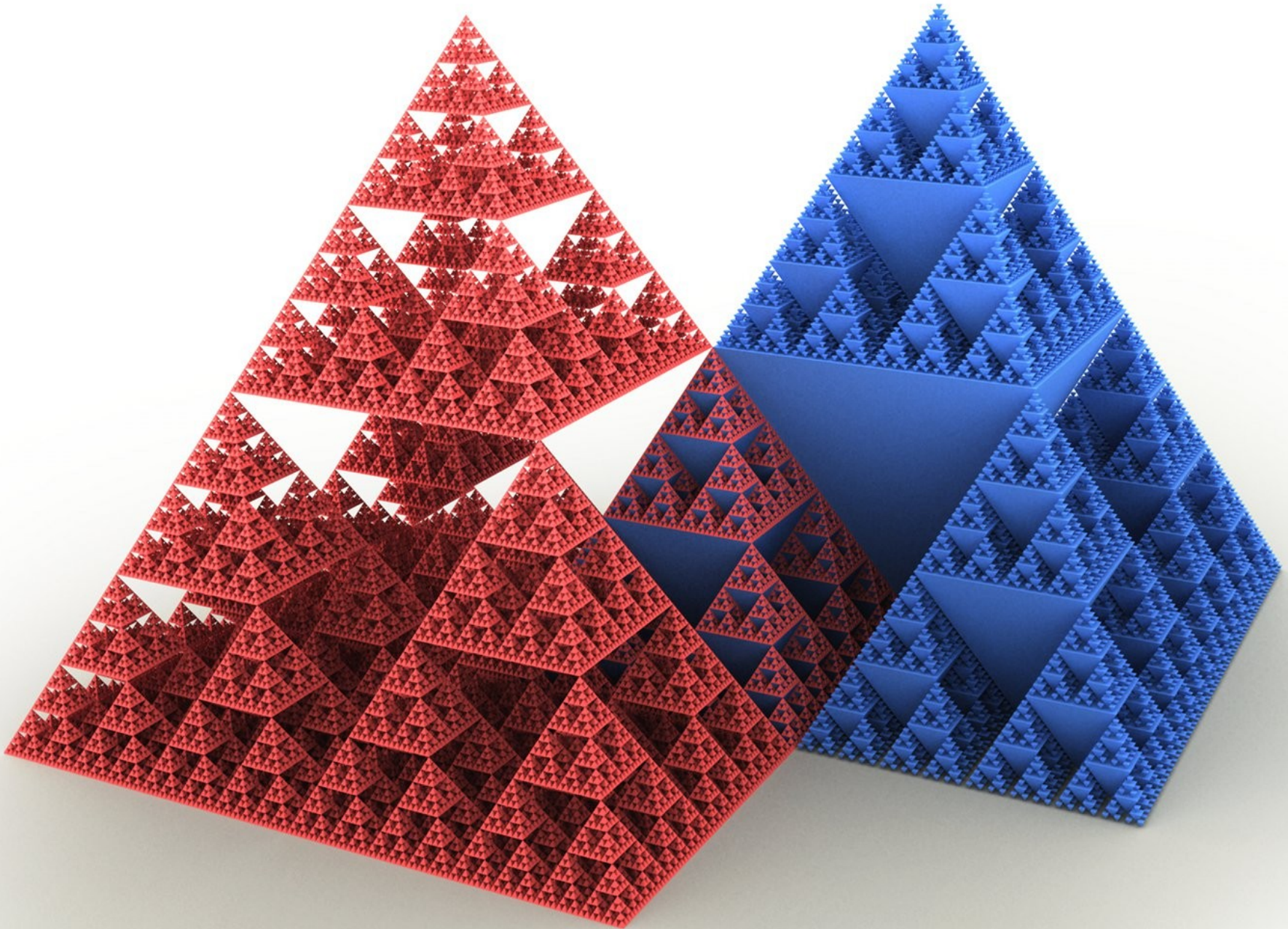
Lecture 05 Outline

- Problem definition and motivations
- Generic Bounding Volume Hierarchy (BVH)
  - BVH construction, fitting, overlapping
  - Metrics and Tandem traversal
- Several bounding volume strategies
  - OBBs, kDOPs, SSVs
- Proximity evaluation of primitive geometries
  - Sphere x Capsule collisions
- Approximate convex decomposition

Mid-Phase Collision Detection

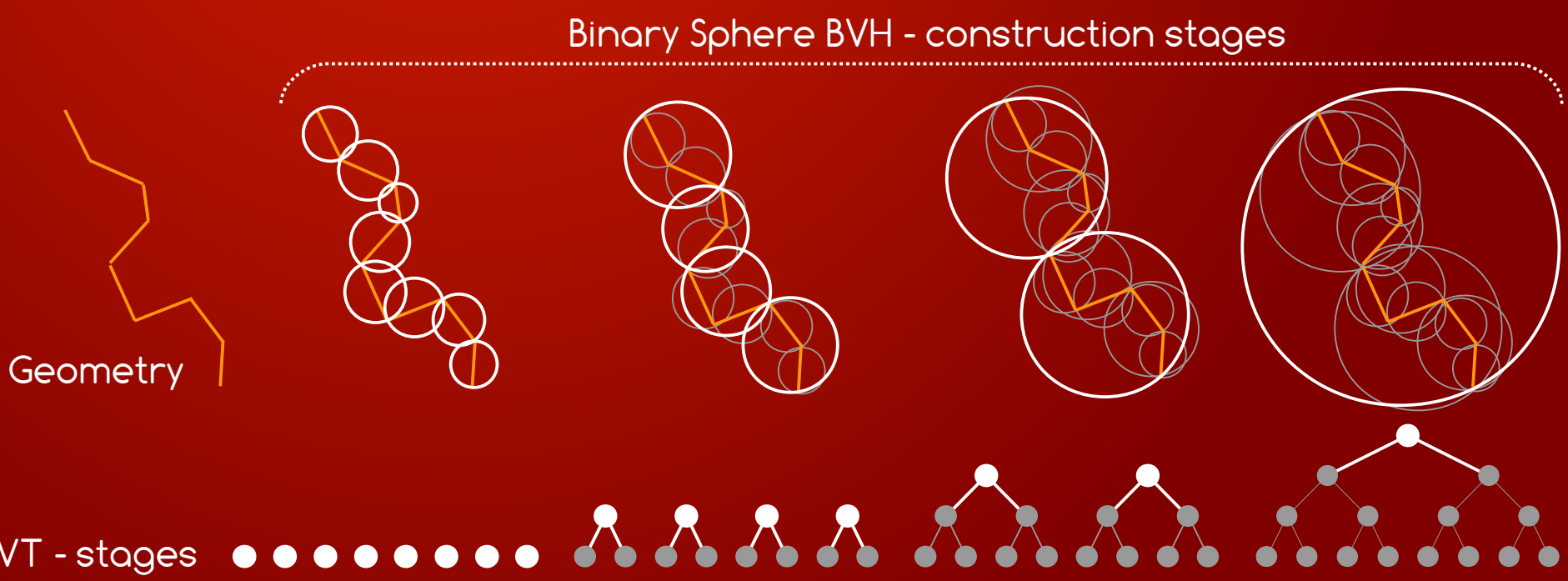
- **Input:** List of pairs of potentially colliding objects.
- **Problem:** Refine this list based on more accurate geometrical properties of objects – prune out pairs of objects surely no colliding.
- **Output:** Refined (smaller) list of pairs of potentially colliding objects.
- **Solutions:**
  - Simplify complex geometry with simpler convex bounding volumes arranged into inclusive hierarchy
  - Decompose complex geometry into convex sub-parts. Calculate narrow phase using this sub-parts only.

Bounding Volume Hierarchy



Bounding Volume Hierarchy

- **Definition:** A **Bounding Volume Hierarchy** (BHV) also known as **Bounding Volume Tree** (BVT) is generally an m-ary tree $T = \{T_1, \cdots, T_m, BV, G\}$, whose nodes ($T_i$) contain a specific bounding volume (BV) which must cover some part of object's geometry (G).

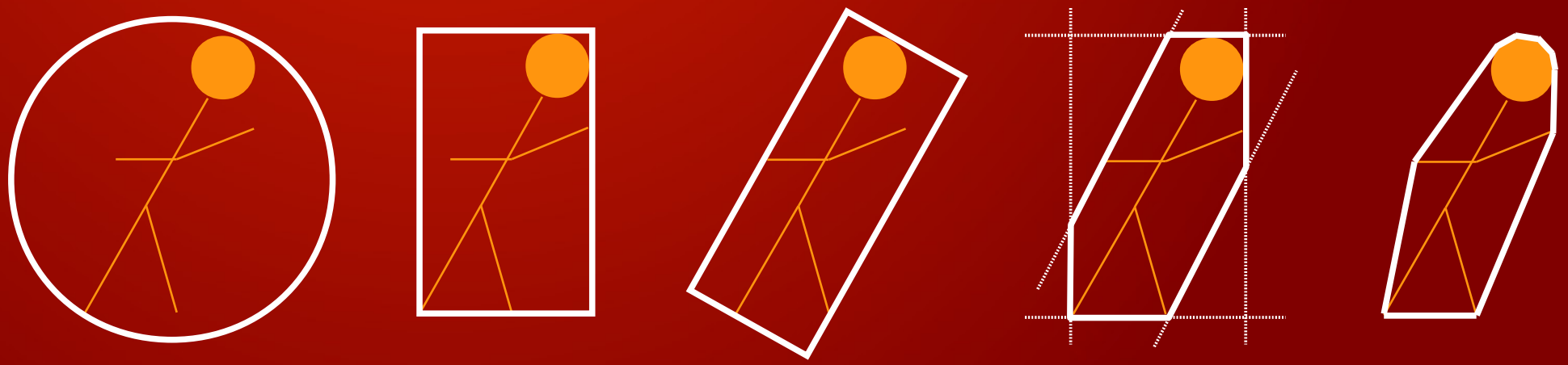


BVH - Properties

- Each lower level of the hierarchy should represent better approximation of the geometry.
- Child nodes should cover together the same part of geometry as their parent node.
- The BVH construction should be automatic, with only a few user defined parameters.
- To speed up the update process BVs should be invariant to rigid motion
- BVs should tightly fit object's geometry and minimize their volume, surface or other measure.

BVH – Choice of Bounding Volume

- Bounding Volume should
  - Be simple (usually) convex well defined geometry
  - Fit the non-spherical geometry as good as possible
  - Have fast and efficient overlap test
  - Rotate and translate with the geometry
  - …



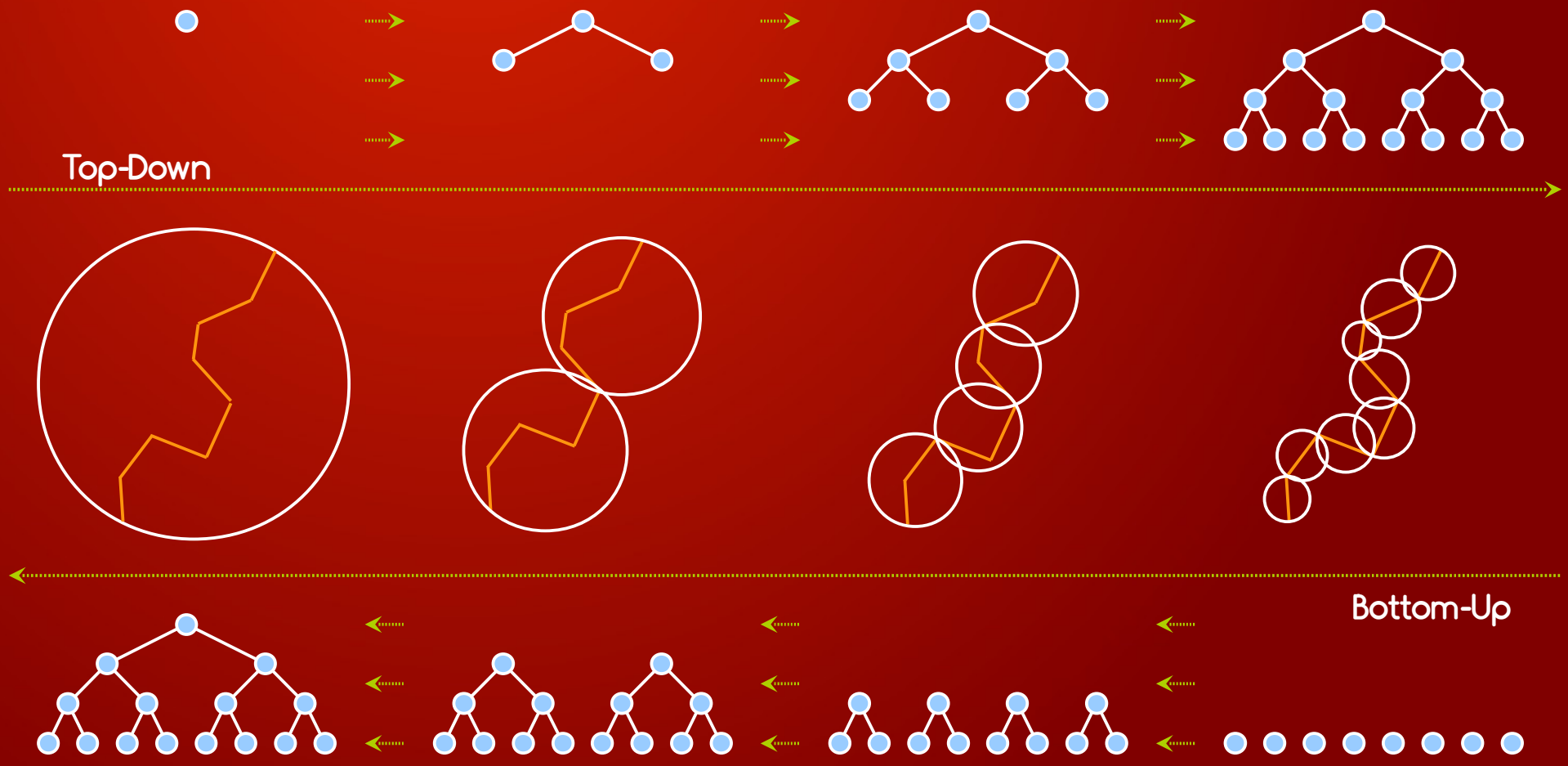
Sphere AABB OBB kDop Convex hull

BVH - Hierarchy Construction

- Problem
  - Given a complex (rigid) geometry define a strategy how to create appropriate fitting BVT
- Properties
  - Hierarchy is usually created before simulation
  - Construction should be as automatic as possible
  - Transformation update must be fast
- Strategies
  - Top-down BVT construction strategies
  - Bottom-up BVT construction strategies

BVH - Hierarchy Construction

- Top-Down vs. Bottom-up construction strategies



BVH - Construction: Bottom-Up

- Define the clustering factor “m” (+ other params)
- Cover smallest geometry sub-parts with Bvs
- Find “m” closest BVs
  - Compute distance of BV centroids for clustering
  - Compute BV surface distances for clustering
- Merge them into parent BV
  - Fit vertices of child BVs or original geometry
- Repeat this process until one root is found
- Pros/Cons:
  - Spatial locality provides usually optimally balanced BVT
  - Clustering can be very time consuming

BVH - Construction: Bottom-Up

In: Objects geometry G
Out: A corresponding Bounding Volume Tree $\mathcal{T}$

function CREATEBOTTOMUP(G) : $\mathcal{T}$

1: $P \leftarrow$ DECOMPOSEPRIMITIVES(G)
2: $\{n,\ k\} \leftarrow \{|P|,\ 1\}$
3: **for** $i \leftarrow 1$ **to** n **do** $\mathcal{T}_i^0 \leftarrow$ FITBV(P_i)
4: **while** $n > 1$ **do**
5: $\mathcal{L} \leftarrow \mathcal{T}^k$ /* save current hierarchy level */
6: $\{i,\ n,\ k\} \leftarrow \{1,\ n/m,\ k+1\}$
7: **while** $i \leq n$ **do**
8: $C \leftarrow$ FINDCLOSESTBVS($\mathcal{L}, m$)
9: $\mathcal{T}_i^k \leftarrow$ MERGEBVS(C)
10: $\mathcal{L} \leftarrow \mathcal{L} \setminus C$ /* remove merged BV from level */
11: $i \leftarrow i+1$ /* goto next parent */
12: **end**
13: **end**
14: **return** $\mathcal{T}$
end

Algorithm 1: Bottom-Up construction of the BVT

BVH - Construction: Top-Down

- Define the branching factor “m” (+ other params)
- Cover the whole geometry with root BV
- Split the geometry into “m” child parts
  - Split along largest vertex variance
  - Sub-parts should have similar volume
- Proceed recursively until stop criterion (volume of part is small …
- Pros/cons
  - Very simple idea (implementation of the overall algorithm)
  - Sensitive to branching factor and stop condition

BVH - Construction: Top-Down

In: Objects geometry G
Out: A corresponding Bounding Volume Tree $\mathcal{T}$

function CreateTopDown(G) : $\mathcal{T}$
1: $\mathcal{T} \leftarrow$ FitBV(G)
2: **if** IsPrimitive(G) **then return** $\mathcal{T}$
3: $G \leftarrow$ SplitGeom(G, m)
4: **for** $i \leftarrow 1$ **to** m **do**
5: $\mathcal{T}_i \leftarrow$ CreateTopDown(G_i)
6: **end**
7: **return** $\mathcal{T}$
end

Algorithm 2: Top-Down construction of the BVT

BVH - Tandem Traversal

- Given nodes T_A and T_B from geometries A and B
  - Test T_a and T_B for overlap – report false if no overlap
  - T_a and T_B overlap we have to solve 3 cases
- T_A and T_B are leaf nodes - Report A and B overlap
- Only T_A or T_B is a leaf node
  - Take all child nodes of the non-leaf node and do recursively tandem traversal between leaf node and child nodes.
- Both T_A and T_B are not leaf nodes
  - Choose which node (T_A or T_B) has larger geometry
  - Do tandem traversal of all child nodes of the larger node with the smaller node.

In: The BVT $\mathcal{T}_A$ and $\mathcal{T}_B$ for both objects
Out: List of primitive pairs in close proximity $\mathcal{L}$

function TandemTraversal($\mathcal{T}_A, \mathcal{T}_B$) : $\mathcal{L}$

1: **if not** Overlap($\mathcal{T}_A, \mathcal{T}_B$) **then return** $\emptyset$
2: $\mathcal{L} \leftarrow \emptyset$
3: **if** IsLeaf($\mathcal{T}_A$) **then**
4: **if** IsLeaf($\mathcal{T}_B$) **then**
5: $\mathcal{L} \leftarrow (\mathcal{T}_A, \mathcal{T}_B)$ `/* primitive pair in close proximity */`
6: **else**
7: **foreach** $\mathcal{T}_B^i$ **in** Children($\mathcal{T}_B$) **do** $\mathcal{L} \leftarrow \mathcal{L} \cup$ TandemTraversal($\mathcal{T}_A, \mathcal{T}_B^i$)
8: **end**
9: **else**
10: **if** IsLeaf($\mathcal{T}_B$) **then**
11: **foreach** $\mathcal{T}_A^i$ **in** Children($\mathcal{T}_A$) **do** $\mathcal{L} \leftarrow \mathcal{L} \cup$ TandemTraversal($\mathcal{T}_A^i, \mathcal{T}_B$)
12: **else**
13: **if** Larger($\mathcal{T}_A, \mathcal{T}_B$) **then**
14: **foreach** $\mathcal{T}_A^i$ **in** Children($\mathcal{T}_A$) **do** $\mathcal{L} \leftarrow \mathcal{L} \cup$ TandemTraversal($\mathcal{T}_A^i, \mathcal{T}_B$)
15: **else**
16: **foreach** $\mathcal{T}_B^i$ **in** Children($\mathcal{T}_B$) **do** $\mathcal{L} \leftarrow \mathcal{L} \cup$ TandemTraversal($\mathcal{T}_A, \mathcal{T}_B^i$)
17: **end**
18: **end**
19: **end**
20: **return** $\mathcal{L}$
end

Algorithm 3: Tandem Traversal algorithm

BVH - Cost Function

- Cost Function: $T_{AB} = N_b \times T_b + N_u \times T_u + N_p \times T_p$
  - T_{AB}: is total time spent for interference detection between two objects A and B.
  - $N_b \times T_b$: is the time spent on the overlap tests between all N_b BV pairs.
  - $N_u \times T_u$: is the time spent on the update of all N_u BVs.
  - $N_p \times T_p$: is the time spent on the exact collision tests between all N_p primitive pairs.

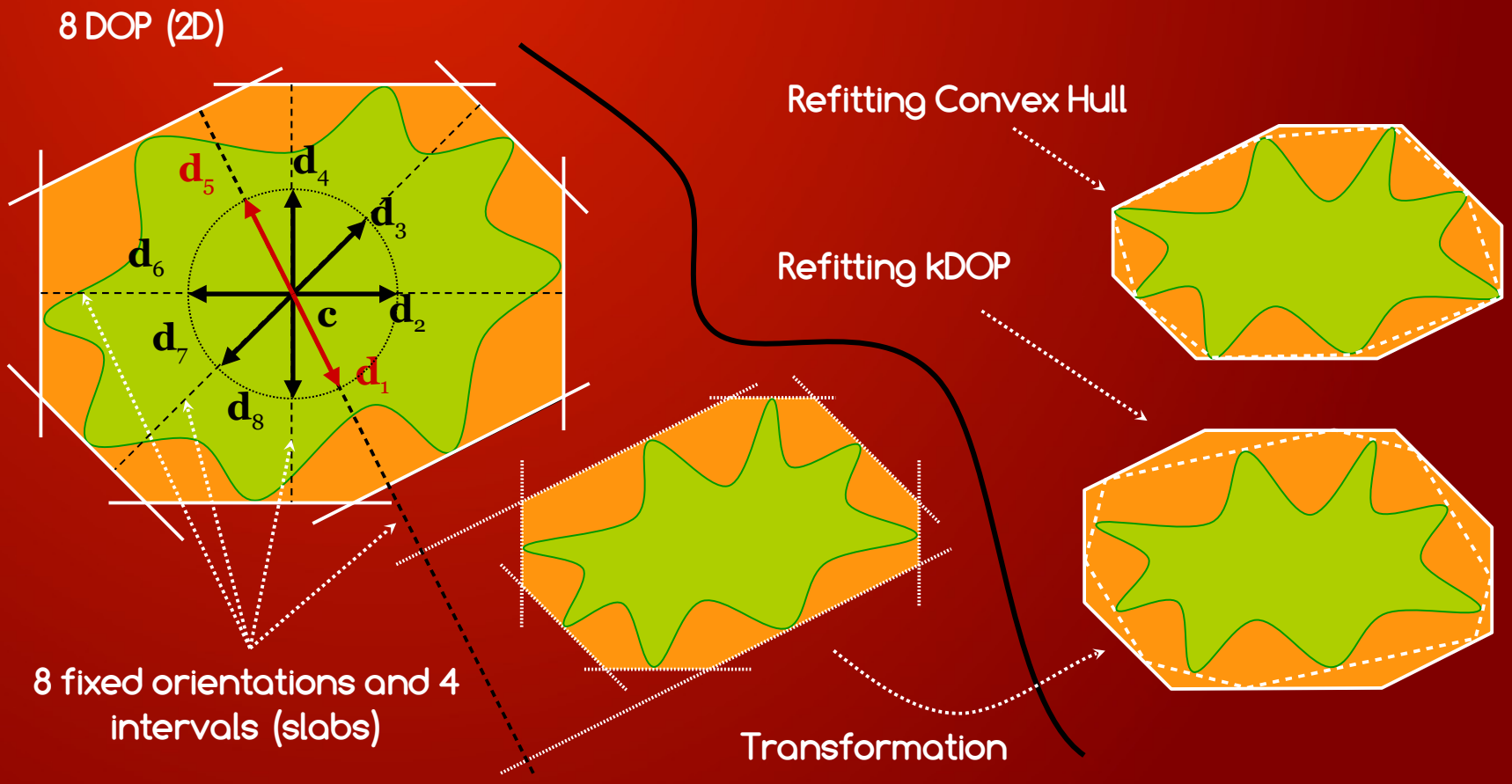
  - N_b, N_u and N_p : Number of operations
  - T_b, T_u and T_p : Time spent on one operation

Bounding Volumes

k-Discrete Orientation Polytopes (kDOP)

- Definition: k-Discrete Orientation Polytope (kDOP) is a convex polyhedron formed by the intersection of negative half-spaces of planes whose normals come from a small set of k fixed orientations di and have distances λi to the center c of kDOP.
- kDOP=$\{ p \text{ in } R^3 \mid d_i^T(p - c) \leq \lambda_i \text{ and } 1 \leq i \leq k \}$
- Axis Aligned Bounding Box (AABB) is 6DOP with 6 directions
  - (+1,0,0); (-1,0,0); (0,+1,0); (0,-1,0); (0,0,+1); (0,0,-1);

kDOP – Overlap Test and Fitting



kDOP - Hierarchy Construction

- Create binary BVT using Top-Down approach
- Split G into G1 and G2 by a cutting plane with normal being one of k directions
- Assuming geometry is a mesh – choose planes origin as one of triangles centroid
- Splitting strategies (choice of origin and normal)
  - **Min Sum:** Minimize the sum of volumes of G1 and G2
  - **Min Max:** Minimize the larger volume of G1 and G2
  - **Splatter or Longes Side:** We choose the direction that yields the largest variance and the reference point being the mean (or median) centroid along such direction.

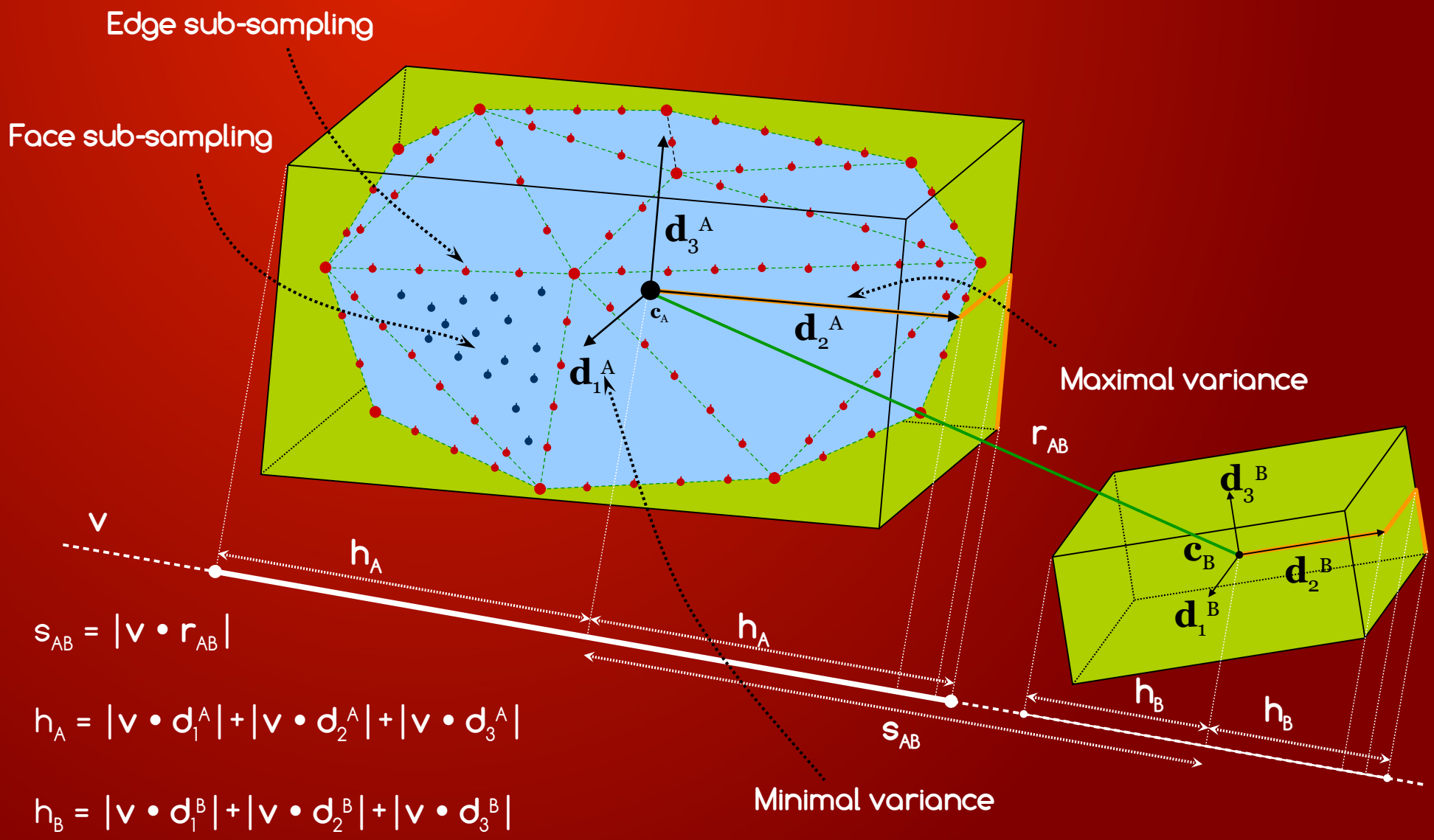
kDOP – Overlap Test and Update

- Overlap Test
  - Since kDOPs are convex polytopes we can use SAT for overlap test
  - Since normals of all faces comes from k orientations we can use conservative SAT → just test all 1D interval overlaps
- Hierarchy Update
  - kDOPs are not transformation invariant → we must refit geometry when the transformation changes
  - Full fitting is expensive → We must use approximate refitting
  - **Hill Climbing:** Precompute convex hulls, during simulation use local search to find new interval limits
  - **Approximate Refitting:** Similar to hill Climbing just precompute kDOP vertices instead of convex hull

Oriented Bounding Boxes (OBB)

- Definition: **Oriented Bounding Box** (OBB) is a set of points $p \in R^3$ inside a box defined with a center point c and 3 mutually orthogonal unit direction vectors d_1, d_2, d_3 and their extents λ_1, λ_2, λ_3
- $OBB = \{ c + s_1 d_1 + s_2 d_2 + s_3 d_3 \mid |s_i| \leq |\lambda_i| \}$
- Similar to AABB but can be freely rotated with the geometry
- Suitable for fast overlap test using SAT

OBB - Overlap test



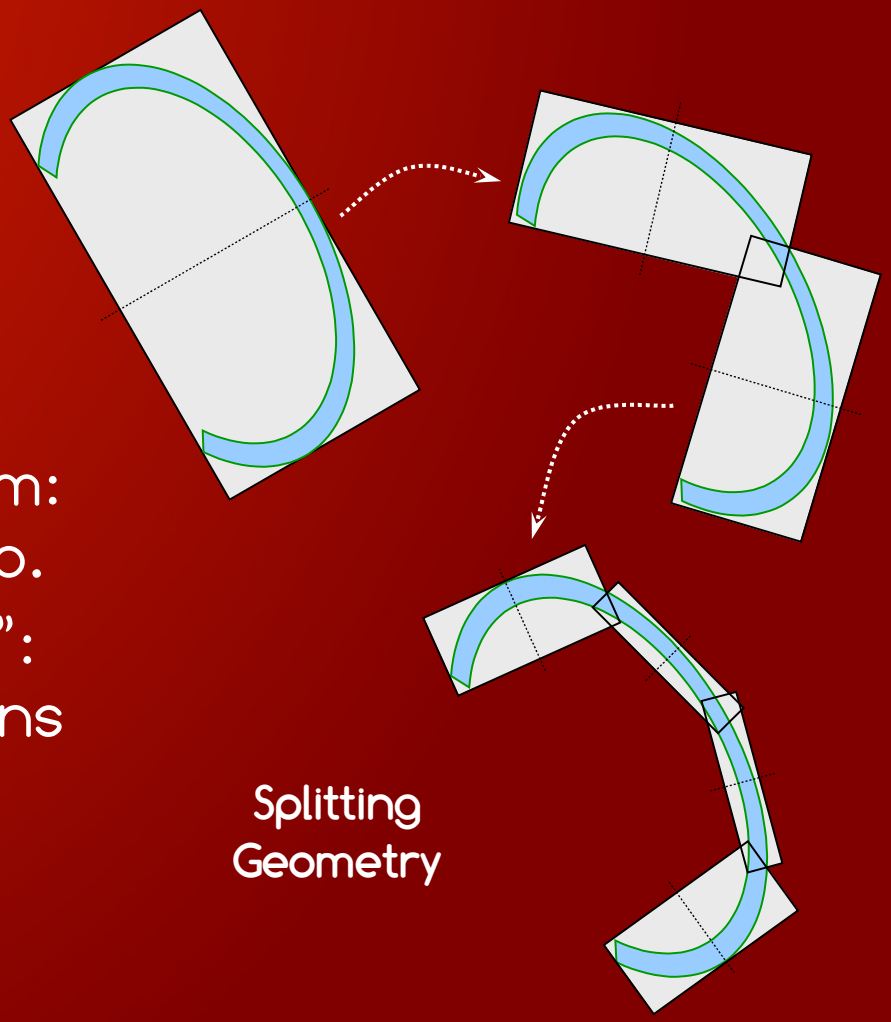
$$s_{AB} = |v \bullet r_{AB}|$$

$$h_A = |v \bullet d_1^A| + |v \bullet d_2^A| + |v \bullet d_3^A|$$

$$h_B = |v \bullet d_1^B| + |v \bullet d_2^B| + |v \bullet d_3^B|$$

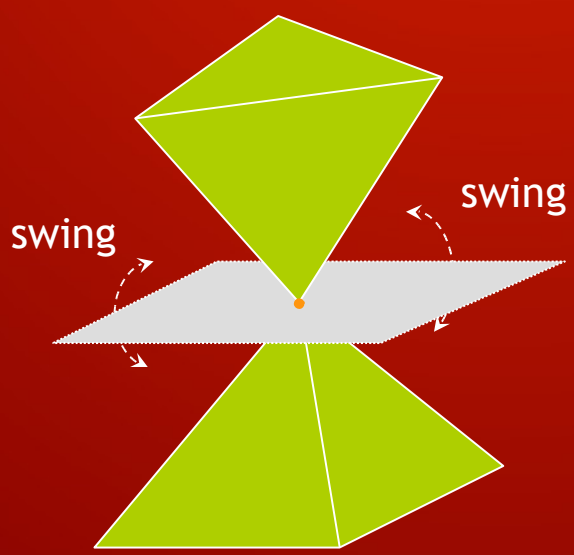
OBB - Hierarchy Construction

- Again Top-Down splitting
  - Fit geometry with optimal OBB
  - Split in halves along longest axis
  - Use similar rules as for kDOPs
- Fitting OBB
  - Optimal fit $O(n^3)$ – slow
  - Approximate "Core set" algorithm: reduce vertices, use optimal algo.
  - Approximate "Principal direction": Use PCA to find principal directions and variance along them.
- Update OBB: just rotate directions and move center

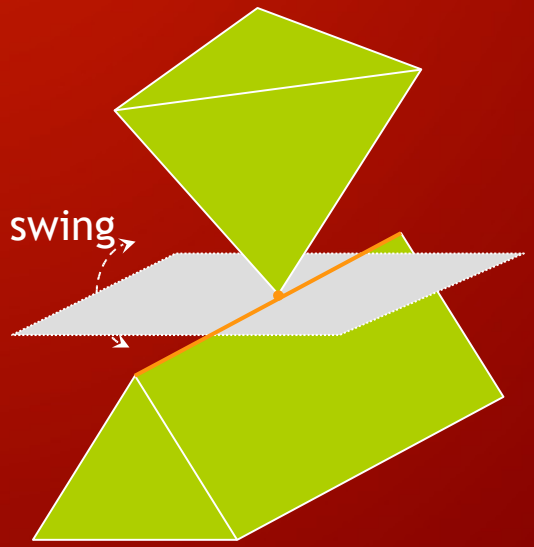


Separating Axis Theorem for Polytopes

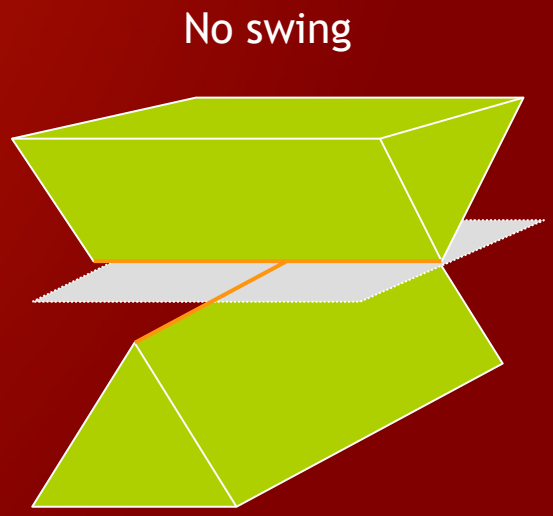
- Polytope SAT: Any two convex polytopes are disjoint iff there exists a separating axis which is either perpendicular to some face of the polytopes or to any two edges each taken from one polytope



Vertex – Vertex Case



Vertex – Edge Case



Edge – Edge Case

OBB – Overlap Test

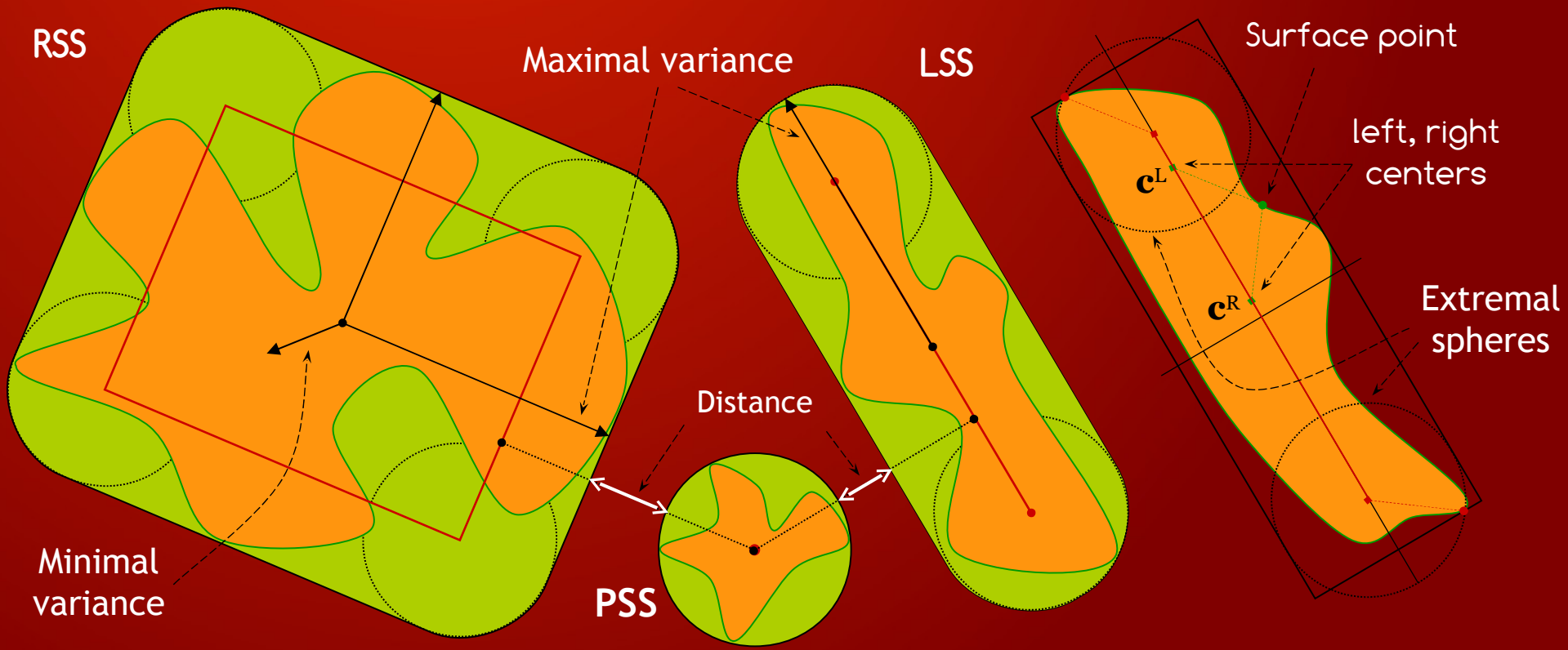
- General SAT for polytopes needs C axis checks
- $C = |F_A| + |F_B| + |E_A| . |E_B|$
  - $|F_A|$ and $|F_B|$ = number of faces of A and B
  - $|E_A|$ and $|E_B|$ = number of edges of A and B
- For OBB all faces and edges have only 3 principal directions: C = 3+3 + 3 x 3 = 15 checks
  - $s_{AB} = |\mathbf{v} \bullet \mathbf{r}_{AB}|$ (projected distance of centers onto v)
  - $h_A = |\mathbf{v} \bullet \mathbf{d}_1^A| + |\mathbf{v} \bullet \mathbf{d}_2^A| + |\mathbf{v} \bullet \mathbf{d}_3^A|$ (projection of A onto v)
  - $h_B = |\mathbf{v} \bullet \mathbf{d}_1^B| + |\mathbf{v} \bullet \mathbf{d}_2^B| + |\mathbf{v} \bullet \mathbf{d}_3^B|$ (projection of B onto v)
- 15 Directions v are: $\mathbf{d}_1^A \times \mathbf{d}_1^B$, $\mathbf{d}_1^A \times \mathbf{d}_2^B$, …
  - If all of them are separating OBBs do not overlap

Swept Sphere Volumes (SSV)

- Definition: The Swept Sphere Volume SSVV is a region of points $p \in R3$ whose distance to some primitive volume V is at most the radius r. Alternatively (SSV) is defined as the Minkowski sum of a primitive volume V and a sphere $S = \{ p \mid |p - 0| k \leq r \}$ with a radius r located at the origin.
- $SSVV = \{ p \mid |p - q| \leq r \wedge q \in V \} = V \oplus S$
- Point Swept Sphere (PSS): V is a point
- Line swept Sphere (LSS): V is a line segment
- Rectangle swept Sphere (RSS): V is a rectangle
- …

SSV - Overlap Test and Update

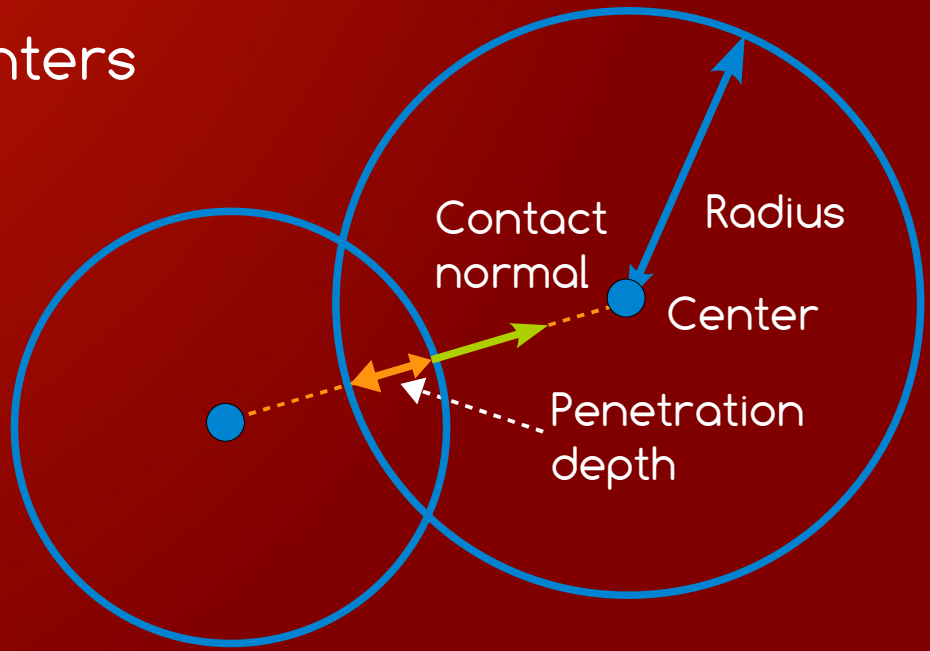
- Overlap test: True if distance between primitive volumes is less than the sum of radius
- Update: Transform primitive geometries




Proximity
of Primitive Geometries

Sphere x Sphere

- Contact point/normal:
  - Take the direction from one center to other
  - Calculate points on both spheres along direction vector.
  - Take their average as contact point
  - Contact normal is just normalized distance vector
- Penetration depth:
  - Take the distance between centers minus radius of both spheres

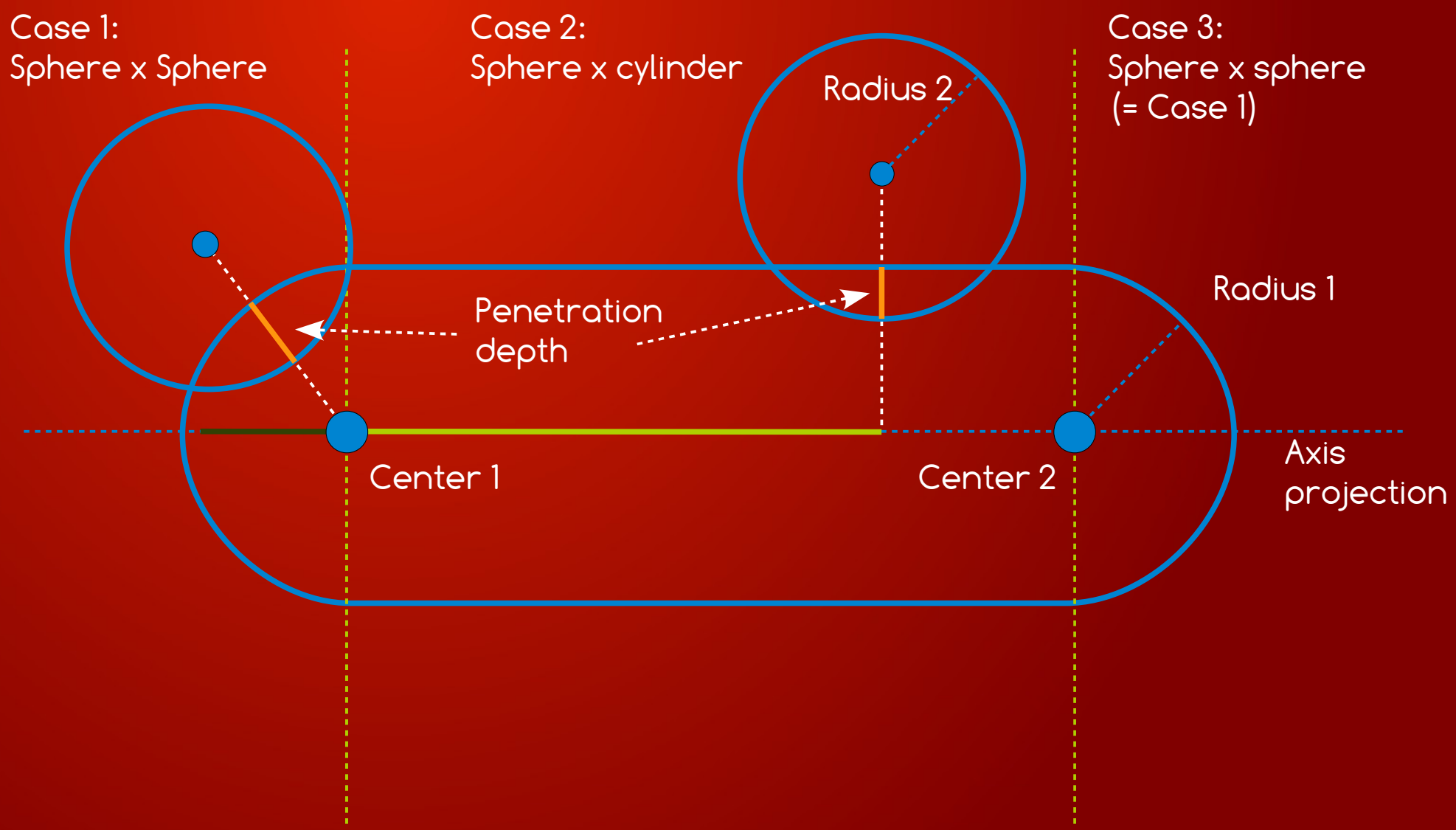


Capsule x Sphere

- Contact point/normal:
  - Project center "C" of sphere onto capsule direction axis "a"
    $a = C_2 - C_1$; $u = \text{norm}(a)$; $v = C - C_1$; $q = u \cdot v$
  - Solve Case 1 ($q<0$) and Case 3 ($q>|a|$) as Sphere x Sphere contact
  - Case 2 is sphere to infinite cylinder contact:

    $Q = c_1 + qu$; $m = C - Q$; $n = \text{norm}(m)$; $P1 = Q + r_1 n$; $P2 = C - r_2 n$
  - Contact point/normal: $p = 0.5(P_1 + P_2)$; $n = \text{norm}(P_2 - P_1)$
- Penetration depth:
  - Take $d = -|P_2 - P_1|$

Capsule x Sphere



Capsule x Capsule

- Principle: use external Voronoi Regions to classify centers of capsules
- Project centers of capsule A onto axis of B
- Project centers of capsule B onto axis of A
- Classify centers on axes as

	CenterB1	CenterB2
RegionA1	A1B1	A1B2
RegionA	AB1	AB2
RegionA2	A2B1	A2B2

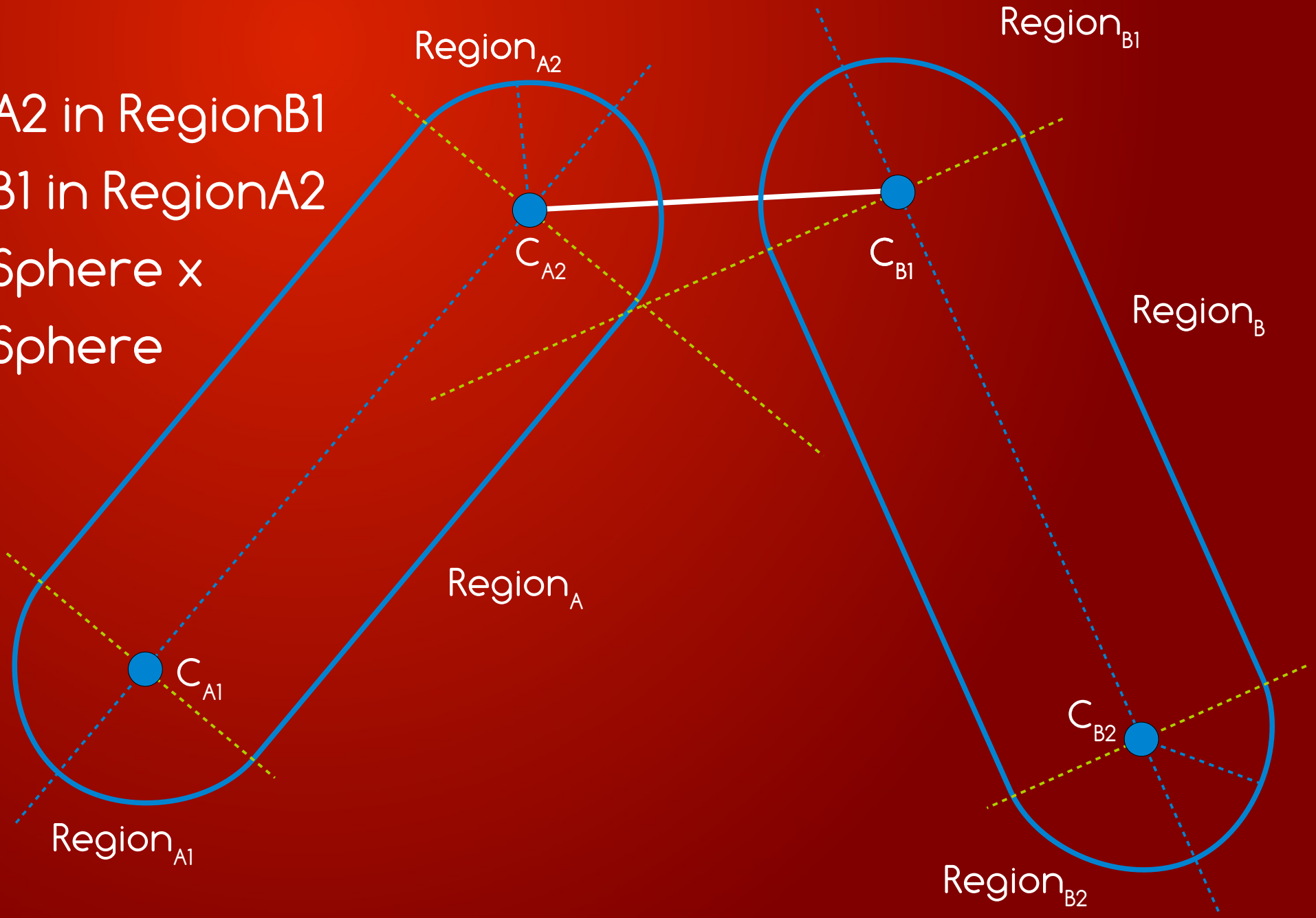
	CenterA1	CenterA2
RegionB1	B1A1	B1A2
RegionB	BA1	BA2
RegionB2	B2A1	B2A2

Capsule x Capsule

- Sphere x Sphere cases:
  - (A1B1 ; B1A1) , (A1B2 ; B2A1) , (A2B1 ; B1A2) , (A2B2 ; B2A2)
- Project projected centers PA1, PA2, (PB1, PB2) back onto its original axes A (B) = PPA1, PPA2, (PPB1, PPB2)
- Sphere x Cylinder cases:
  - e.g. PA1 is projected A1 onto B
  - Now project PA1 back onto A (=PPA1) and see where it lies
  - If PPA1 is in RegionA1 or RegionA2 we have sphere x cylinder
  - PPA2 in RegionA1/A2; PPB2 in RegionB1/B2; PPB2 in RegionB1/B2
- Cylinder x Cylinder cases:
  - Otherwise (e.g. PPA1 lies in RegionA)
  - or PPA2 is in RegionA or PPB1 is in RegionB or PPB2 is in RegionB

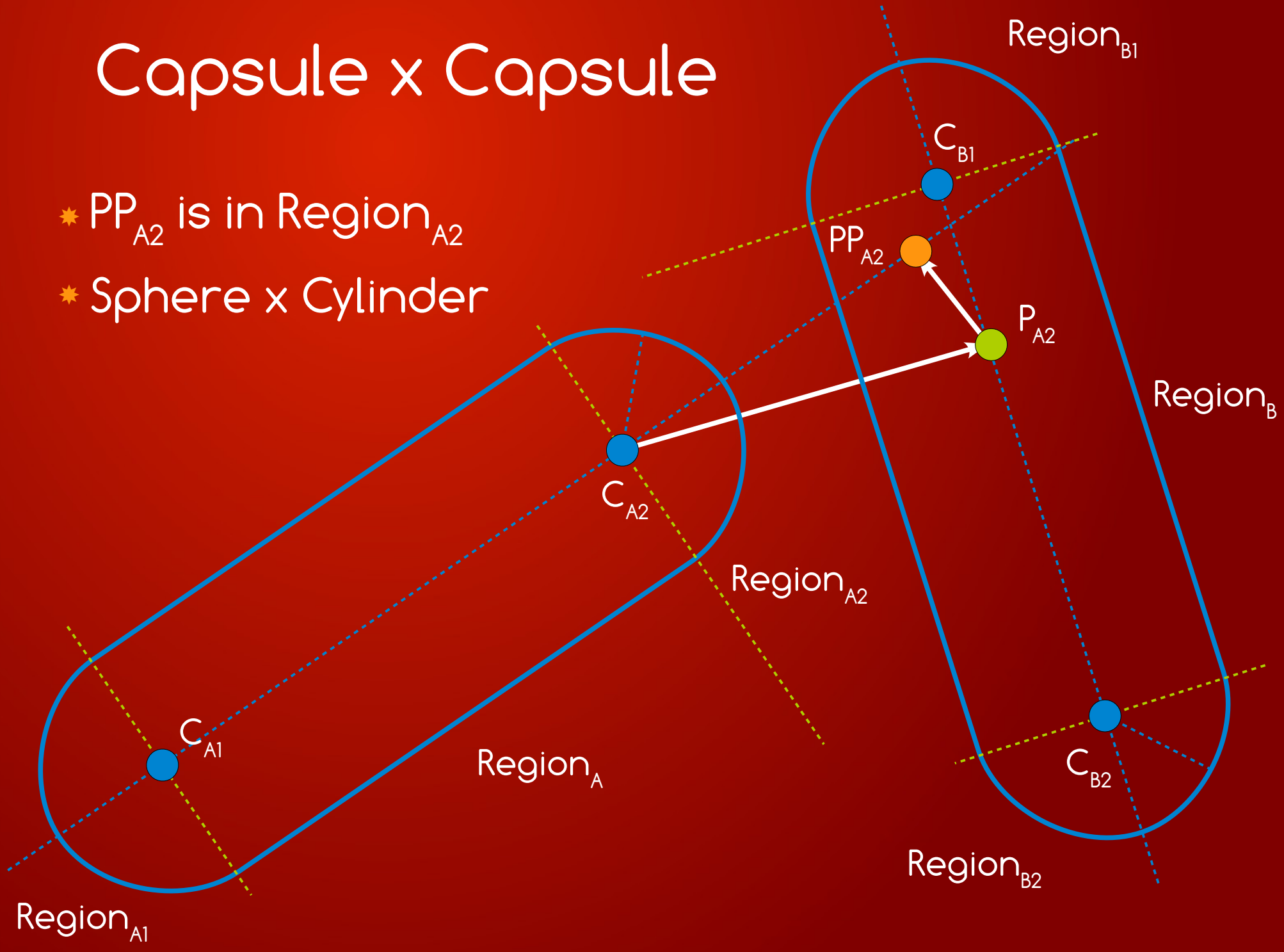
Capsule x Capsule

- A2 in RegionB1
- B1 in RegionA2
- Sphere x Sphere



Capsule x Capsule

- PP_{A2} is in $Region_{A2}$
- Sphere x Cylinder



Approximate Convex Decomposition

Approximate Convex Decomposition

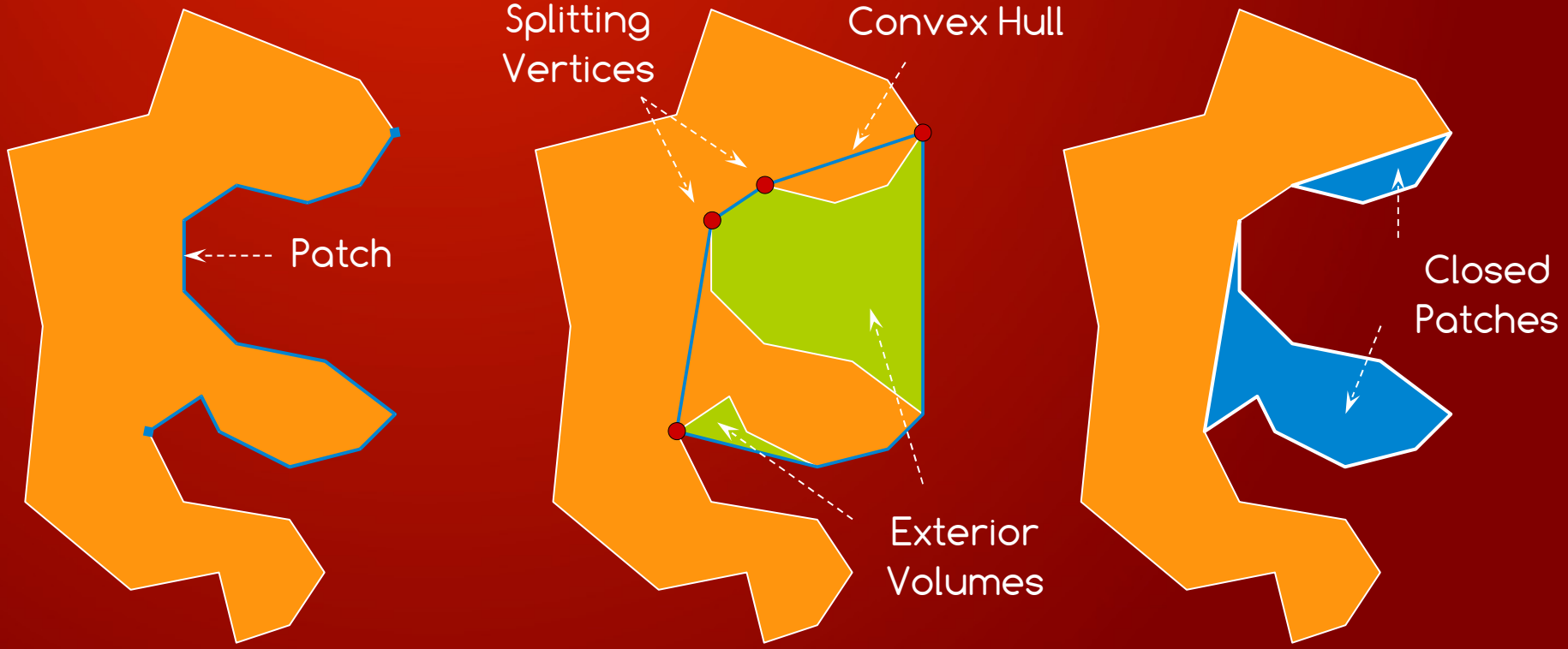
- Problem: For a given non-convex geometry find a small set of sub-parts which are almost convex
- A geometry is **almost convex** if the difference between its volume and the volume of its convex hull is under given threshold
- It is usually done only once before simulation
- A number of complex algorithms exists
  - Measuring concavity, fuzzy clustering, …
- We provide here simple relaxation strategy

ACD - Relaxation strategy

- Choose a Top-Down splitting strategy (e.g. OBB)
- Split recursively geometry until small leaf nodes
- Use volume threshold and stop criterion
- We have now a (large) set of small (almost) convex sub-parts.
- Put them into priority queue based on their volume (sort upon volume)
- Pop first part and try to merge it with some other small part. Merge only when the ratio between merged volume and appropriate convex hull is under given threshold

Approximate Convex Decomposition

- Choose patch, create convex hull, mark splitting vertices → Create sub-parts. Exterior volume → 0



The End

let me go !