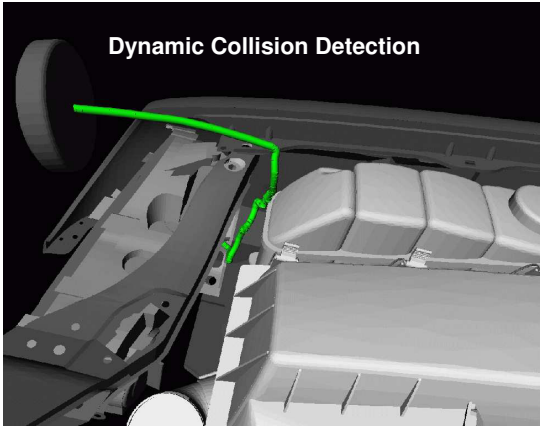
# Distance Computation Between Non-Convex Polyhedra

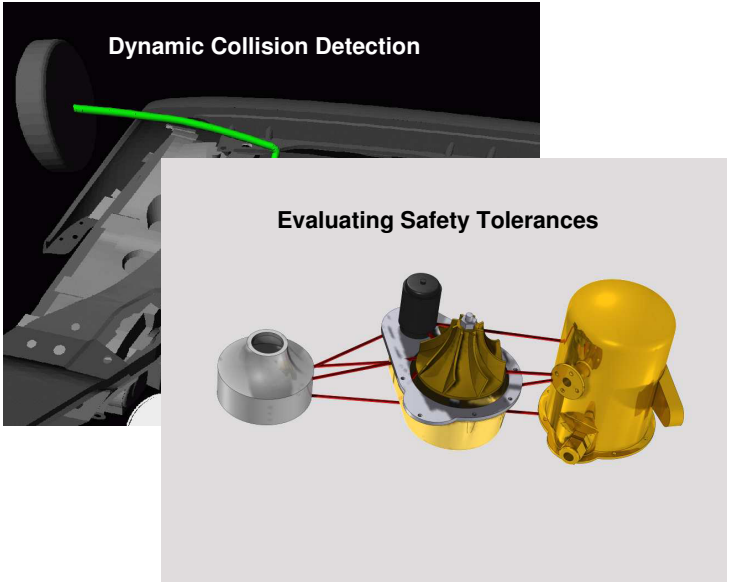
Christian Lennerz

June 17, 2002

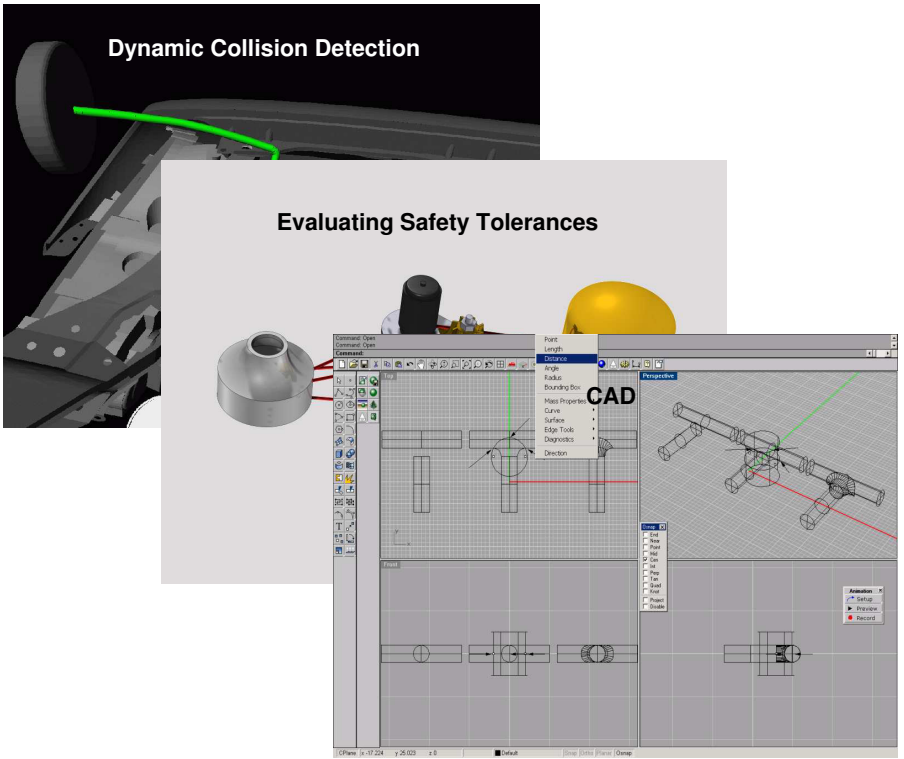
# Applications



# Applications



# Applications



# The Distance Computation Problem

**Definition 1 (Distance Computation Problem)**
*Given two polyhedra $\mathbf{P}_1$, $\mathbf{P}_2$. The distance computation problem is to determine the global minimum of the distance function $\delta$ between the respective point sets, together with a pair of witness points i.e.*

*(i) the value $\delta^* := \delta(\mathbf{P}_1, \mathbf{P}_2)$,*

*(ii) a pair of points $(\mathbf{p}, \mathbf{q})$, s.t. $\delta^* = \delta(\mathbf{p}, \mathbf{q})$,*

*where $\delta$ denotes the (quadratic)* EUCLIDEAN *distance function between two points or set of points, respectively.*

# Reduction to Basic Proximity Tests

**Observation 1**
*Given two polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ in boundary representation. Let $\mathcal{F}_i$, $\mathcal{E}_i$ and $\mathcal{V}_i$, $i = 1, 2$, denote the respective sets of faces, edges and vertices.*
*For the distance between $\mathcal{P}_1$ and $\mathcal{P}_2$ we have:*

$$\delta(\mathcal{P}_1, \mathcal{P}_2) \;=$$

# Reduction to Basic Proximity Tests

**Observation 1**
*Given two polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ in boundary representation. Let $\mathcal{F}_i$, $\mathcal{E}_i$ and $\mathcal{V}_i$, $i = 1, 2$, denote the respective sets of faces, edges and vertices.*
*For the distance between $\mathcal{P}_1$ and $\mathcal{P}_2$ we have:*

$$\delta(\mathcal{P}_1, \mathcal{P}_2) \;=\; \delta(\partial\mathcal{P}_1, \partial\mathcal{P}_2)$$

# Reduction to Basic Proximity Tests

**Observation 1**
*Given two polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ in boundary representation. Let $\mathcal{F}_i$, $\mathcal{E}_i$ and $\mathcal{V}_i$, $i = 1, 2$, denote the respective sets of faces, edges and vertices.*
*For the distance between $\mathcal{P}_1$ and $\mathcal{P}_2$ we have:*

$$\begin{aligned}\delta(\mathcal{P}_1, \mathcal{P}_2) &= \delta(\partial\mathcal{P}_1, \partial\mathcal{P}_2) \\ &= \min\left\{\delta(f_1, f_2) \,\middle|\, f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_1\right\}\end{aligned}$$

# Reduction to Basic Proximity Tests

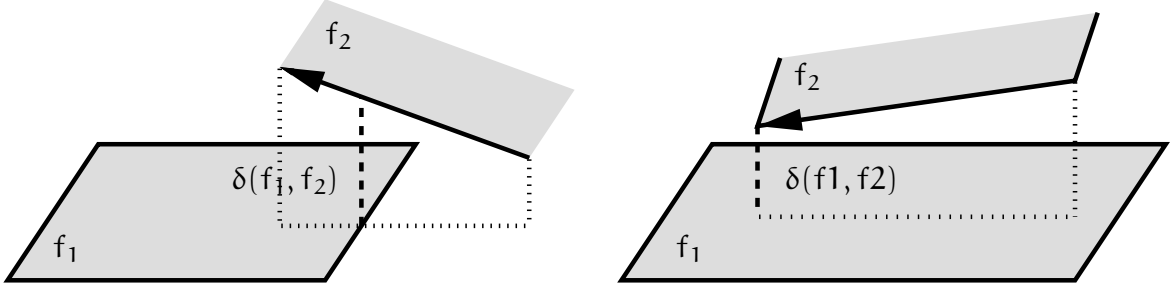
**Observation 1**
*Given two polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ in boundary representation. Let $\mathcal{F}_i$, $\mathcal{E}_i$ and $\mathcal{V}_i$, $i = 1, 2$, denote the respective sets of faces, edges and vertices.*
*For the distance between $\mathcal{P}_1$ and $\mathcal{P}_2$ we have:*

$$\begin{aligned}
\delta(\mathcal{P}_1, \mathcal{P}_2) &= \delta(\partial\mathcal{P}_1, \partial\mathcal{P}_2) \\
&= \min\left\{\delta(f_1, f_2) \,\middle|\, f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_1\right\} \\
&= \begin{cases} \min\left\{\delta(\mathcal{F}_1, \mathcal{V}_2)\,, \delta(\mathcal{V}_1, \mathcal{F}_2),\ \delta(\mathcal{E}_1, \mathcal{E}_2)\right\} & \text{if } \mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset\,; \\ 0 & \text{else}\,. \end{cases}
\end{aligned}$$

# Reduction to Basic Proximity Tests

**Observation 1**
*Given two polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ in boundary representation. Let $\mathcal{F}_i$, $\mathcal{E}_i$ and $\mathcal{V}_i$, $i = 1, 2$, denote the respective sets of faces, edges and vertices.*
*For the distance between $\mathcal{P}_1$ and $\mathcal{P}_2$ we have:*

$$\begin{aligned}
\delta(\mathcal{P}_1, \mathcal{P}_2) &= \delta(\partial\mathcal{P}_1, \partial\mathcal{P}_2) \\
&= \min\left\{\delta(f_1, f_2) \,\middle|\, f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_1\right\} \\
&= \begin{cases} \min\left\{\delta(\mathcal{F}_1, \mathcal{V}_2)\,, \delta(\mathcal{V}_1, \mathcal{F}_2),\ \delta(\mathcal{E}_1, \mathcal{E}_2)\right\} & \text{if } \mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset\,; \\ 0 & \text{else}\,. \end{cases}
\end{aligned}$$



# A Simple Algorithm

DISTANCE($\mathcal{P}_1, \mathcal{P}_2$)
(1) $d^* \leftarrow \infty$
(2) **foreach** $f_1 \in \mathcal{F}_1$
(3) **foreach** $f_2 \in \mathcal{F}_2$

# A Simple Algorithm

DISTANCE($\mathcal{P}_1, \mathcal{P}_2$)
(1) $d^* \leftarrow \infty$
(2) **foreach** $f_1 \in \mathcal{F}_1$
(3) **foreach** $f_2 \in \mathcal{F}_2$
(4) $\left[\text{isDisjoint}, (p_1, p_2)\right] \leftarrow$ INTERSECT($f_1, f_2$)
(5) **if** $\text{isDisjoint} =$ **false then return** $\left[0, (p_1, p_2)\right]$

# A Simple Algorithm

DISTANCE($\mathcal{P}_1, \mathcal{P}_2$)
(1) $d^* \leftarrow \infty$
(2) **foreach** $f_1 \in \mathcal{F}_1$
(3) **foreach** $f_2 \in \mathcal{F}_2$
(4) $\left[\text{isDisjoint}, (p_1, p_2)\right] \leftarrow$ INTERSECT($f_1, f_2$)
(5) **if** isDisjoint $=$ **false then return** $\left[0, (p_1, p_2)\right]$
(6) **foreach** $v_1$ of $f_1$
(7) $\left[d, (p_1, p_2)\right] \leftarrow$ VERTEXFACEDISTANCE($v_1, f_2$)
(8) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$

# A Simple Algorithm

DISTANCE($\mathcal{P}_1, \mathcal{P}_2$)
(1) $d^* \leftarrow \infty$
(2) **foreach** $f_1 \in \mathcal{F}_1$
(3) **foreach** $f_2 \in \mathcal{F}_2$
(4) $\left[\text{isDisjoint}, (p_1, p_2)\right] \leftarrow$ INTERSECT($f_1, f_2$)
(5) **if** isDisjoint $=$ **false then return** $\left[0, (p_1, p_2)\right]$
(6) **foreach** $v_1$ of $f_1$
(7) $\left[d, (p_1, p_2)\right] \leftarrow$ VERTEXFACEDISTANCE($v_1, f_2$)
(8) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$
(9) **foreach** $v_2$ of $f_2$
(10) $\left[d, (p_1, p_2)\right] \leftarrow$ VERTEXFACEDISTANCE($v_2, f_1$)
(11) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$

# A Simple Algorithm

DISTANCE($\mathcal{P}_1, \mathcal{P}_2$)
(1) $d^* \leftarrow \infty$
(2) **foreach** $f_1 \in \mathcal{F}_1$
(3) **foreach** $f_2 \in \mathcal{F}_2$
(4) $\left[\text{isDisjoint}, (p_1, p_2)\right] \leftarrow$ INTERSECT($f_1, f_2$)
(5) **if** isDisjoint $=$ **false then return** $\left[0, (p_1, p_2)\right]$
(6) **foreach** $v_1$ of $f_1$
(7) $\left[d, (p_1, p_2)\right] \leftarrow$ VERTEXFACEDISTANCE($v_1, f_2$)
(8) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$
(9) **foreach** $v_2$ of $f_2$
(10) $\left[d, (p_1, p_2)\right] \leftarrow$ VERTEXFACEDISTANCE($v_2, f_1$)
(11) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$
(12) **foreach** $e_1$ of $f_1$
(13) **foreach** $e_2$ of $f_2$
(14) $\left[d, (p_1, p_2)\right] \leftarrow$ EDGEEDGEDISTANCE($e_1, e_2$)
(15) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$

# A Simple Algorithm

DISTANCE($\mathcal{P}_1, \mathcal{P}_2$)
(1) $d^* \leftarrow \infty$
(2) **foreach** $f_1 \in \mathcal{F}_1$
(3) **foreach** $f_2 \in \mathcal{F}_2$
(4) $\left[\text{isDisjoint}, (p_1, p_2)\right] \leftarrow$ INTERSECT($f_1, f_2$)
(5) **if** isDisjoint $=$ **false then return** $\left[0, (p_1, p_2)\right]$
(6) **foreach** $v_1$ of $f_1$
(7) $\left[d, (p_1, p_2)\right] \leftarrow$ VERTEXFACEDISTANCE($v_1, f_2$)
(8) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$
(9) **foreach** $v_2$ of $f_2$
(10) $\left[d, (p_1, p_2)\right] \leftarrow$ VERTEXFACEDISTANCE($v_2, f_1$)
(11) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$
(12) **foreach** $e_1$ of $f_1$
(13) **foreach** $e_2$ of $f_2$
(14) $\left[d, (p_1, p_2)\right] \leftarrow$ EDGEEDGEDISTANCE($e_1, e_2$)
(15) **if** $d < d^*$ **then** $d^* \leftarrow d, \quad p_i^* \leftarrow p_i \quad i = 1, 2$
(16) **return** $\left[d^*, (p_1^*, p_2^*)\right]$

# Edge-Edge-Distance (I)

Let $g_1$ and $g_2$ denote the straight lines, on which the edges are embedded:

$$g_i = \left\{ \mathbf{a}_i + \lambda_i \boldsymbol{\nu}_i \,|\, \lambda_i \in \mathbb{R} \right\} \qquad i = 1, 2 \,.$$

# Edge-Edge-Distance (I)

Let $g_1$ and $g_2$ denote the straight lines, on which the edges are embedded:

$$g_i = \left\{ \mathbf{a}_i + \lambda_i \boldsymbol{\nu}_i \,|\, \lambda_i \in \mathbb{R} \right\} \qquad i = 1, 2 \,.$$

The quadratic distance between $g_1$ and $g_2$ is given by

$$\delta(g_1, g_2) = \left( \lambda_1 \boldsymbol{\nu}_1 - \lambda_2 \boldsymbol{\nu}_2 - \boldsymbol{\nu}_{12} \right)^2 \,,$$

with $\boldsymbol{\nu}_{12} := \mathbf{a}_2 - \mathbf{a}_1$.

# Edge-Edge-Distance (I)

Let $g_1$ and $g_2$ denote the straight lines, on which the edges are embedded:

$$g_i = \left\{ \mathbf{a}_i + \lambda_i \boldsymbol{v}_i \,|\, \lambda_i \in \mathbb{R} \right\} \qquad i = 1, 2 \,.$$

The quadratic distance between $g_1$ and $g_2$ is given by

$$\delta(g_1, g_2) = \left(\lambda_1 \boldsymbol{v}_1 - \lambda_2 \boldsymbol{v}_2 - \boldsymbol{v}_{12}\right)^2 \,,$$

with $\boldsymbol{v}_{12} := \mathbf{a}_2 - \mathbf{a}_1$.

To minimize the distance function, we consider its partial derivatives with respect to $\lambda_1$ and $\lambda_2$:

$$\frac{\partial \delta(\lambda_1, \lambda_2)}{\partial \lambda_1} = 2 \left(\lambda_1 \boldsymbol{v}_1 - \lambda_2 \boldsymbol{v}_2 - \boldsymbol{v}_{12}\right)^T \boldsymbol{v}_1 = 0 \;\Leftrightarrow\; \lambda_1 \boldsymbol{v}_1^2 - \lambda_2 \boldsymbol{v}_1^T \boldsymbol{v}_2 = \boldsymbol{v}_1^T \boldsymbol{v}_{12} \,,$$

$$\frac{\partial \delta(\lambda_1, \lambda_2)}{\partial \lambda_2} = -2 \left(\lambda_1 \boldsymbol{v}_1 - \lambda_2 \boldsymbol{v}_2 - \boldsymbol{v}_{12}\right)^T \boldsymbol{v}_2 = 0 \;\Leftrightarrow\; \lambda_1 \boldsymbol{v}_1^T \boldsymbol{v}_2 - \lambda_2 \boldsymbol{v}_2^2 = \boldsymbol{v}_2^T \boldsymbol{v}_{12} \,.$$

# Edge-Edge-Distance (II)

So far, the parameter $\lambda_i^*$ that minimizes the distance depends on the other parameter:

# Edge-Edge-Distance (II)

So far, the parameter $\lambda_i^*$ that minimizes the distance depends on the other parameter:

$$\lambda_1^*(\lambda_2) = \frac{\lambda_2 \boldsymbol{v}_1^T \boldsymbol{v}_2 + \boldsymbol{v}_1^T \boldsymbol{v}_{12}}{\boldsymbol{v}_1^2} = \frac{\lambda_2 V_{12} + W_1}{V_1}\,, \tag{1}$$

$$\lambda_2^*(\lambda_1) = \frac{\lambda_1 \boldsymbol{v}_1^T \boldsymbol{v}_2 - \boldsymbol{v}_2^T \boldsymbol{v}_{12}}{\boldsymbol{v}_2^2} = \frac{\lambda_1 V_{12} - W_2}{V_2}\,, \tag{2}$$

with $V_{12} := \boldsymbol{v}_1^T \boldsymbol{v}_2,\ V_i := \boldsymbol{v}_i^2,\ W_i := \boldsymbol{v}_i^T \boldsymbol{v}_{12},\ i = 1, 2\,.$

# Edge-Edge-Distance (II)

So far, the parameter $\lambda_i^*$ that minimizes the distance depends on the other parameter:

$$\lambda_1^*(\lambda_2) = \frac{\lambda_2 \boldsymbol{v}_1^T \boldsymbol{v}_2 + \boldsymbol{v}_1^T \boldsymbol{v}_{12}}{\boldsymbol{v}_1^2} = \frac{\lambda_2 V_{12} + W_1}{V_1}\,, \tag{1}$$

$$\lambda_2^*(\lambda_1) = \frac{\lambda_1 \boldsymbol{v}_1^T \boldsymbol{v}_2 - \boldsymbol{v}_2^T \boldsymbol{v}_{12}}{\boldsymbol{v}_2^2} = \frac{\lambda_1 V_{12} - W_2}{V_2}\,, \tag{2}$$

with $V_{12} := \boldsymbol{v}_1^T \boldsymbol{v}_2,\ V_i := \boldsymbol{v}_i^2,\ W_i := \boldsymbol{v}_i^T \boldsymbol{v}_{12},\ i = 1, 2\,.$

Eliminating one variable yields $\lambda_i^*$, $i = 1, 2$, explicitly:

$$\lambda_1^* = \frac{W_1 V_2 - W_2 V_{12}}{V_1 V_2 - V_{12}^2} \qquad \lambda_2^* = \frac{W_1 V_{12} - W_2 V_1}{V_1 V_2 - V_{12}^2}\,, \tag{3}$$

if $V_1 V_2 - V_{12}^2 = \boldsymbol{v}_1^2 \boldsymbol{v}_2^2 - (\boldsymbol{v}_1^T \boldsymbol{v}_2)^2 = (\boldsymbol{v}_1 \times \boldsymbol{v}_2)^2 \neq 0$.

# Edge-Edge-Distance (III)

**Problems:**

- $\lambda_1^*$ or $\lambda_2^*$ are outside the parameter interval $[0, 1]$:
  - Explanation: The edges represent line segments, not straight lines.
  - In this case we know that at least one of the closest points is an end point of an edge.
  - Solution: Consider the endpoints and use equation 1 and 2 to compute the closest points on the other line.

# Edge-Edge-Distance (III)

**Problems:**

- $\lambda_1^*$ or $\lambda_2^*$ are outside the parameter interval $[0, 1]$:
  - Explanation: The edges represent line segments, not straight lines.
  - In this case we know that at least one of the closest points is an end point of an edge.
  - Solution: Consider the endpoints and use equation 1 and 2 to compute the closest points on the other line.
- The denominator in equation 3 vanishes:
  - Explanation: The edges are parallel.
  - Also in this case we have that at least one of the closest points is an end point of an edge.
  - Solution: same as above.

# Vertex-Face-Distance

Let $\Sigma(f)$ denote the plane on which the face $f$ is embedded, i.e.

$$\Sigma(f) = \left\{ x \in \mathbb{R}^3 \,\middle|\, \mathbf{n}^T \mathbf{x} = n_0,\ \|\mathbf{n}\| = 1 \right\}.$$

Thereby the value $n_0$ corresponds to the distance from the plane to the origin.

# Vertex-Face-Distance

Let $\Sigma(f)$ denote the plane on which the face $f$ is embedded, i.e.

$$\Sigma(f) \;=\; \left\{ x \in \mathbb{R}^3 \;\middle|\; \mathbf{n}^T\mathbf{x} \;=\; n_0,\; \|\mathbf{n}\| = 1 \right\}.$$

Thereby the value $n_0$ corresponds to the distance from the plane to the origin.

The distance between the query point $\mathbf{p}$ an the plane $\Sigma(f)$ is given by

$$\delta\big(\mathbf{p}, \Sigma(f)\big) = |\, \mathbf{n}^T\mathbf{p} - n_0 \,|^2 \,.$$

# Vertex-Face-Distance

Let $\Sigma(f)$ denote the plane on which the face $f$ is embedded, i.e.

$$\Sigma(f) \;=\; \left\{ x \in \mathbb{R}^3 \;\middle|\; \mathbf{n}^T\mathbf{x} \;=\; n_0,\; \|\mathbf{n}\| = 1 \right\}.$$

Thereby the value $n_0$ corresponds to the distance from the plane to the origin.

The distance between the query point $\mathbf{p}$ an the plane $\Sigma(f)$ is given by

$$\delta\big(\mathbf{p}, \Sigma(f)\big) = |\,\mathbf{n}^T\mathbf{p} - n_0\,|^2\,.$$

The closest point to $\mathbf{p}$ on $\Sigma(f)$ is equal to the projection of $\mathbf{p}$ onto the plane:

$$\overline{\mathbf{p}} \;:=\; \mathbf{p} - (\mathbf{n}^T\mathbf{p} - n_0)\mathbf{n}\,.$$

# Vertex-Face-Distance

Let $\Sigma(f)$ denote the plane on which the face $f$ is embedded, i.e.

$$\Sigma(f) \;=\; \left\{ x \in \mathbb{R}^3 \;\middle|\; \mathbf{n}^T\mathbf{x} \;=\; n_0,\; \|\mathbf{n}\| = 1 \right\}.$$

Thereby the value $n_0$ corresponds to the distance from the plane to the origin.

The distance between the query point $\mathbf{p}$ an the plane $\Sigma(f)$ is given by

$$\delta\big(\mathbf{p}, \Sigma(f)\big) = |\,\mathbf{n}^T\mathbf{p} - n_0\,|^2\,.$$

The closest point to $\mathbf{p}$ on $\Sigma(f)$ is equal to the projection of $\mathbf{p}$ onto the plane:

$$\overline{\mathbf{p}} \;:=\; \mathbf{p} - (\mathbf{n}^T\mathbf{p} - n_0)\mathbf{n}\,.$$

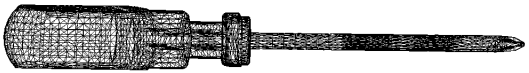
- $\overline{\mathbf{p}}$ is also the closest point of $f$ to $\mathbf{p}$, iff it is lying inside the polygon.
- Otherwise, a closest point of $f$ to $\mathbf{p}$ is located on one of the boundary edges of the polygon.

# Bounding Volumes

**Definition 2 (Bounding Volume)**
*A bounding volume is a geometric primitive enclosing an arbitrary point set, i.e. an outer approximation.*

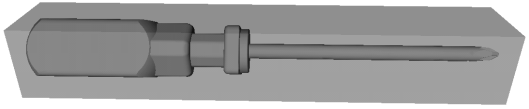
# Bounding Volumes

**Definition 2 (Bounding Volume)**
*A bounding volume is a geometric primitive enclosing an arbitrary point set, i.e. an outer approximation.*



# Bounding Volumes

**Definition 2 (Bounding Volume)**
*A bounding volume is a geometric primitive enclosing an arbitrary point set, i.e. an outer approximation.*



# Bounding Volumes

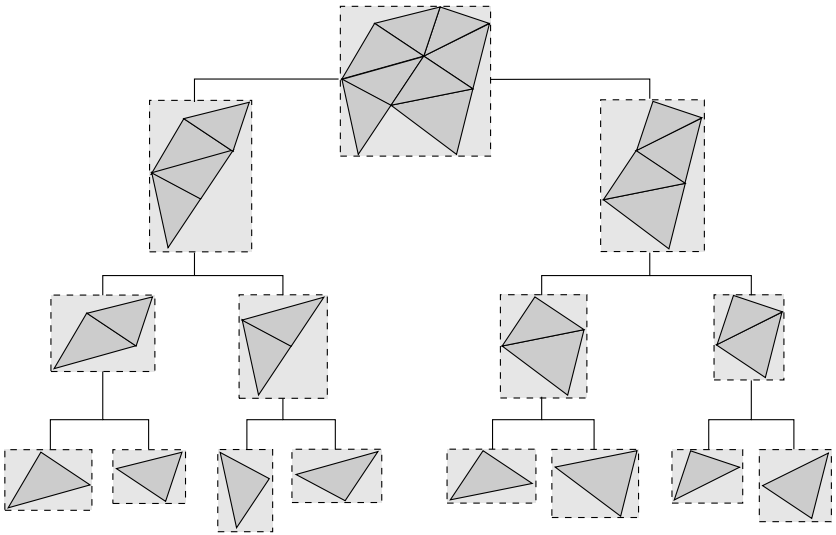
**Definition 2 (Bounding Volume)**
*A bounding volume is a geometric primitive enclosing an arbitrary point set, i.e. an outer approximation.*

**Types of Bounding Volumes:**

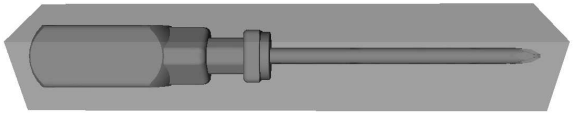
- Spheres [*Hubbard95*],[*Palmer,Grimsdale95*]
- Oriented Bounded Boxes (OBB) [*Gottschalk,Lin,Manocha96*]
- Fixed-Directions Hulls ($FDH_k$ or k-DOPS) [*Held,Klosowski,Mitchell96*]
  Special case: Axis-Aligned Bounding Boxes (AABB) [*Zachmann,Felger95*]
- Swept Sphere Volumes [*Larsen,Gottschalk,Lin,Manocha99*]

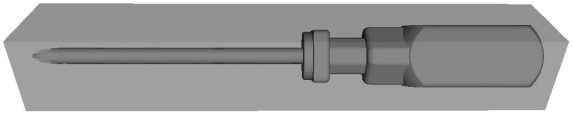
# Bounding Volume Hierarchy

**Definition 3 (Bounding Volume Hierarchy)**
*A bounding volume hierarchy is a tree structure successively refining the boundary of a polyhedron (or even a polygon soup) and covering each refinement by a set of bounding volumes.*



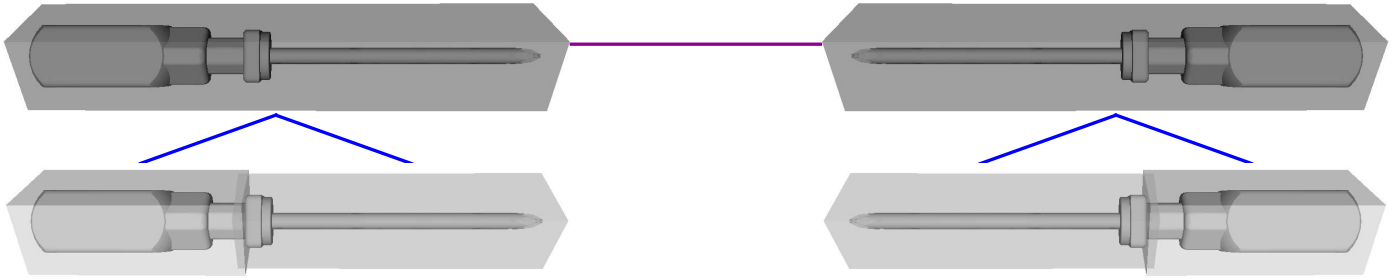
# Distance Computation Using BV-Hierarchies



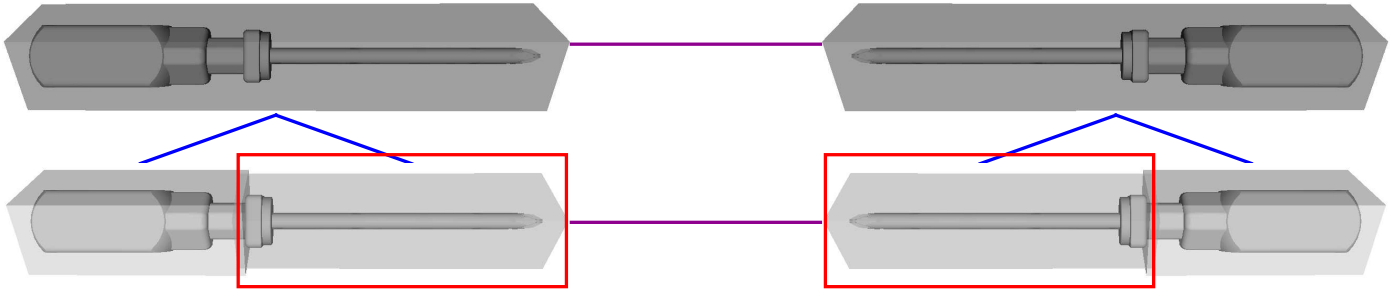


# Distance Computation Using BV-Hierarchies

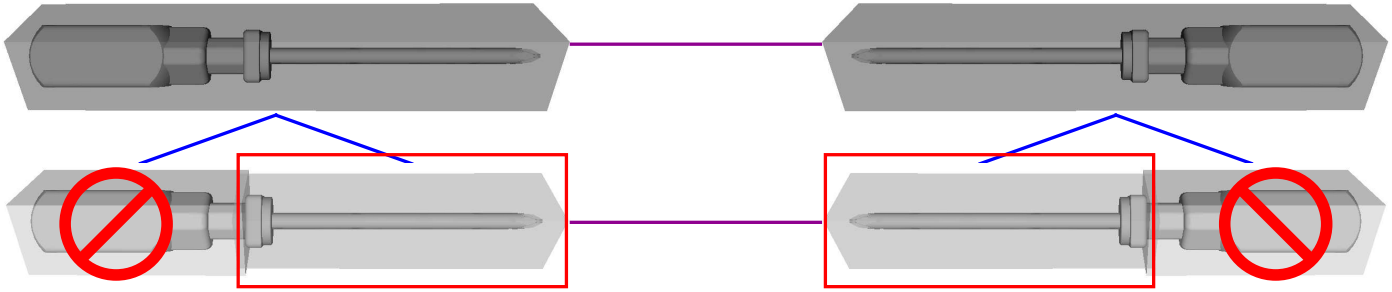
# Distance Computation Using BV-Hierarchies



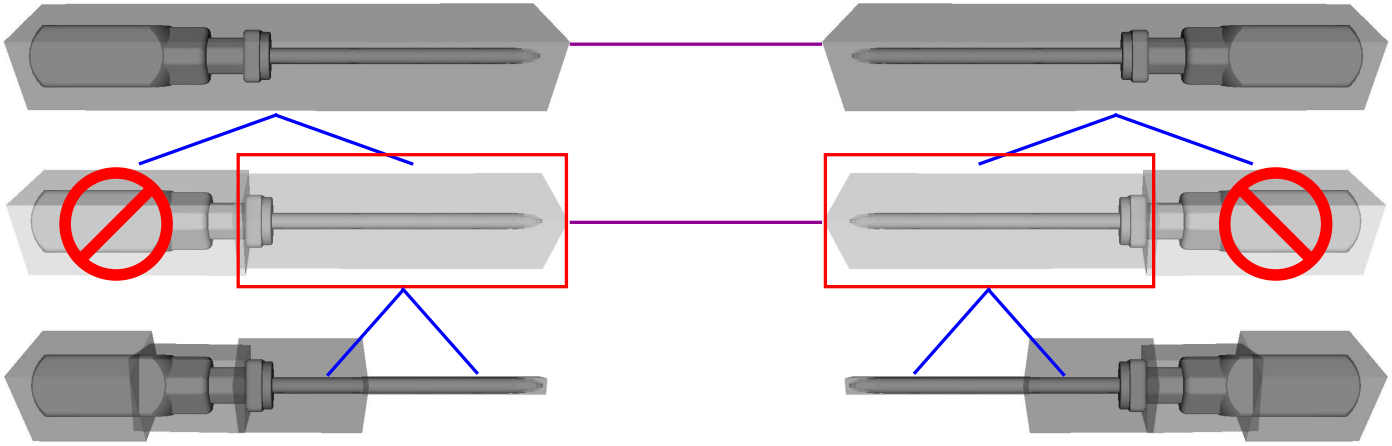
# Distance Computation Using BV-Hierarchies



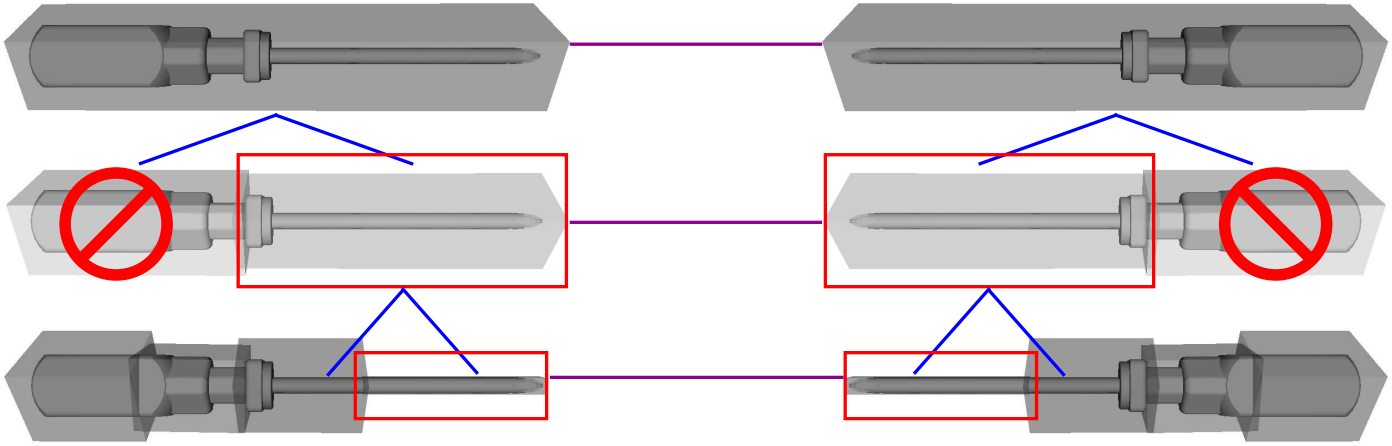
# Distance Computation Using BV-Hierarchies



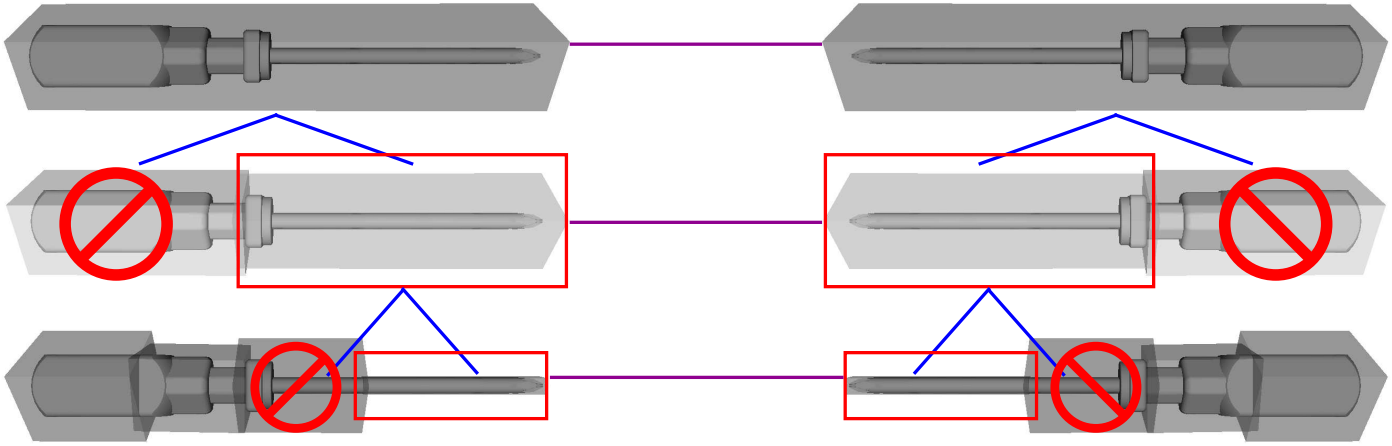
# Distance Computation Using BV-Hierarchies



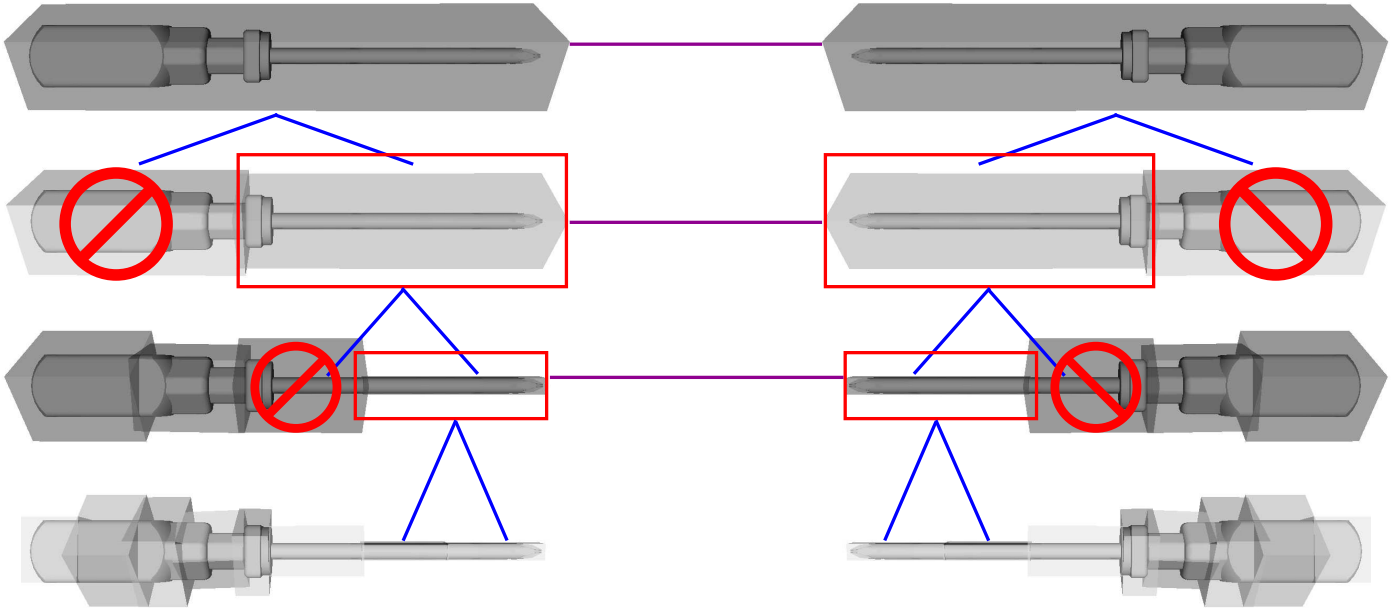
# Distance Computation Using BV-Hierarchies



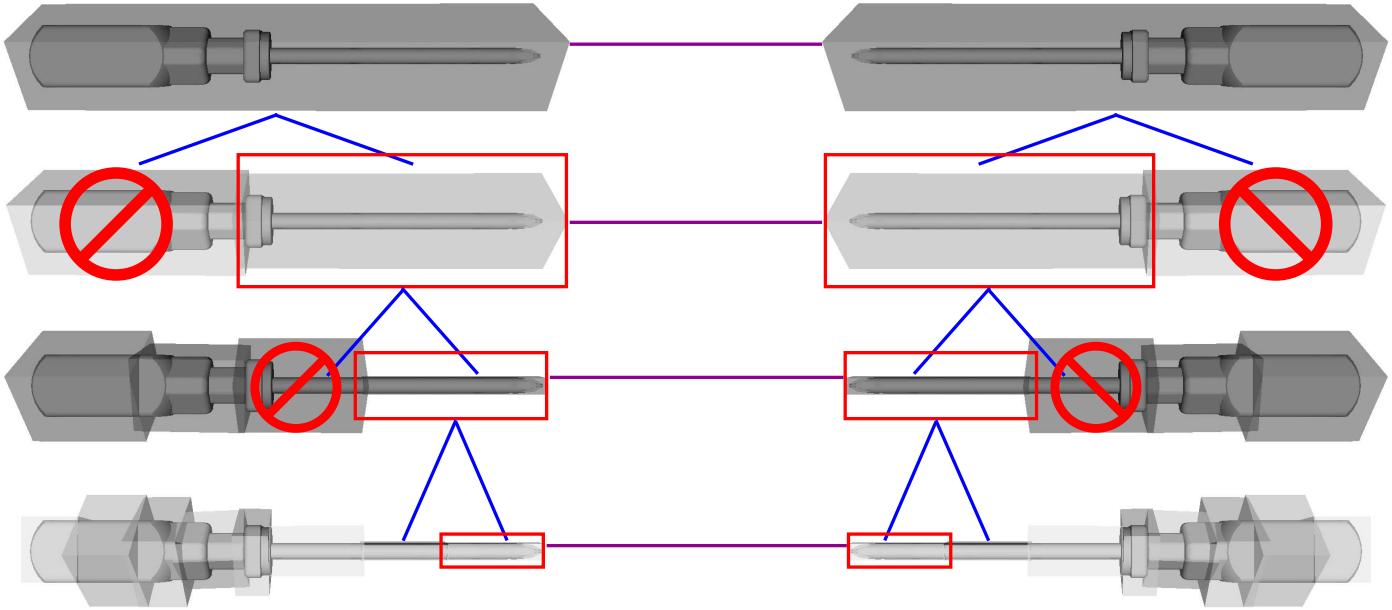
# Distance Computation Using BV-Hierarchies



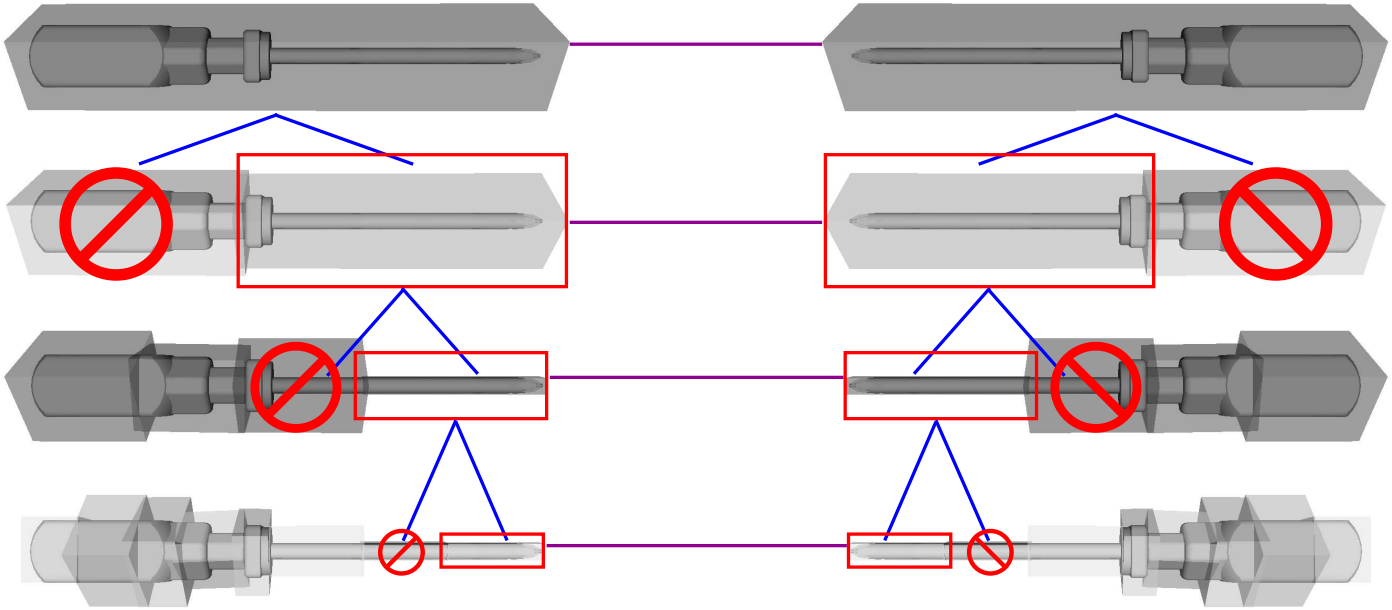
# Distance Computation Using BV-Hierarchies



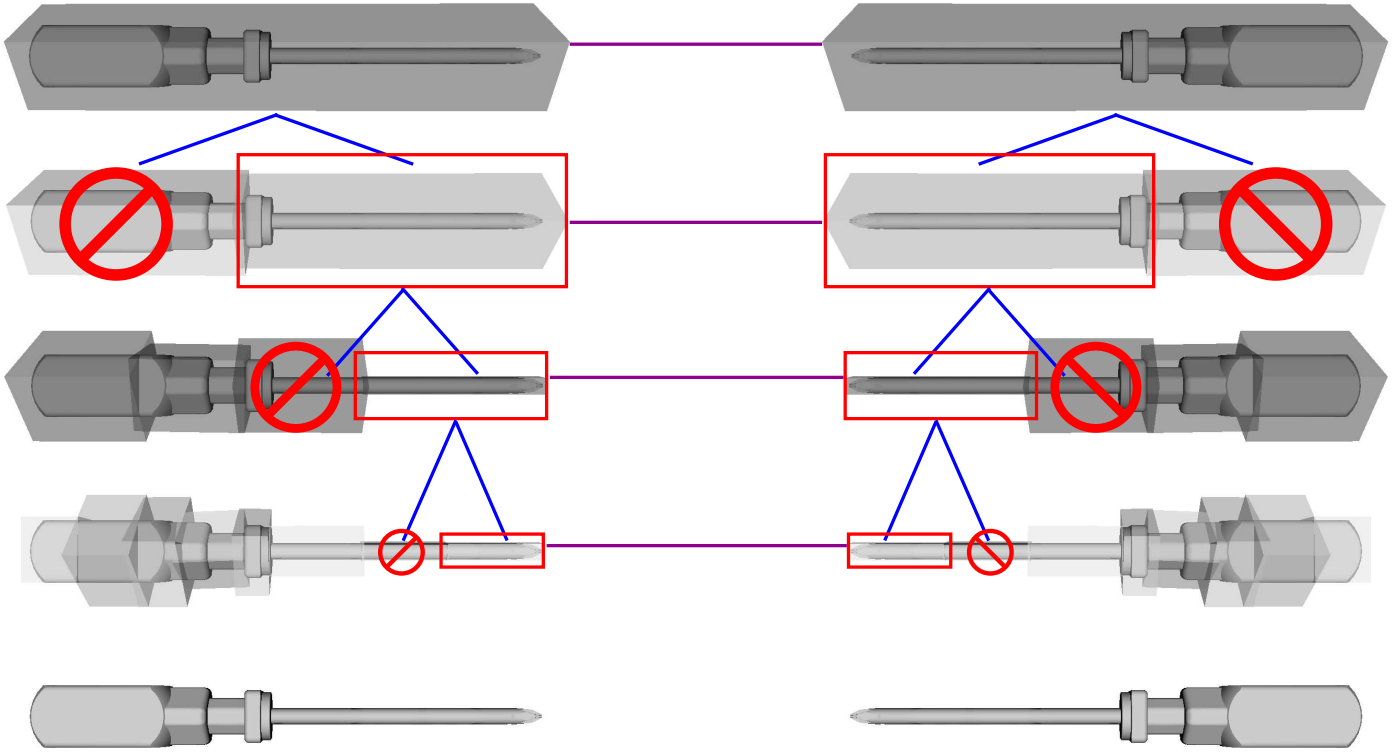
# Distance Computation Using BV-Hierarchies



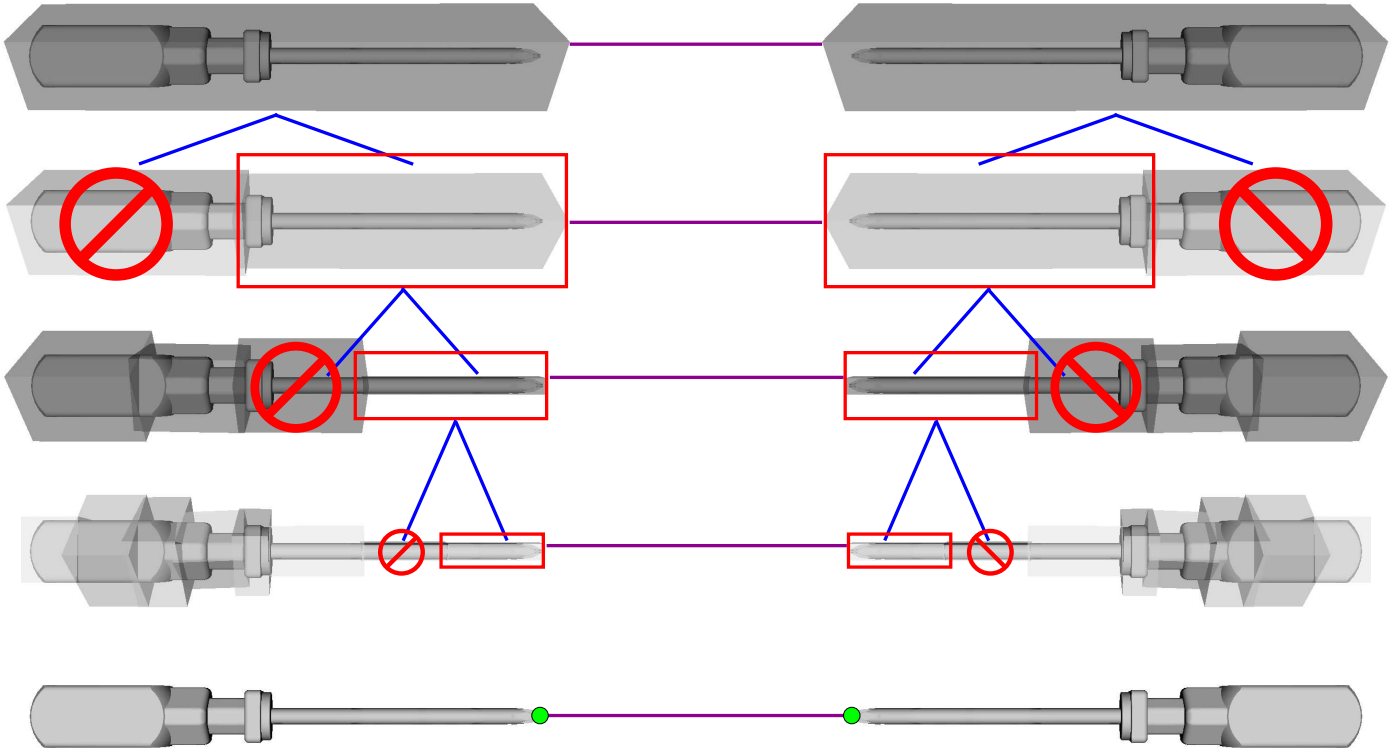
# Distance Computation Using BV-Hierarchies



# Distance Computation Using BV-Hierarchies



# Distance Computation Using BV-Hierarchies



# Implementing the Algorithm

**Topics to discuss:**

- To build the hierarchy we need
  - a strategy for partitioning a given face set,
  - algorithms to compute "tight" bounding volumes of a given face set.

# Implementing the Algorithm

**Topics to discuss:**

- To build the hierarchy we need
  - a strategy for partitioning a given face set,
  - algorithms to compute "tight" bounding volumes of a given face set.
- To prune subtrees in the hierarchy we need
  - efficient algorithms to update bounding volumes after object movement,
  - efficient algorithms to compute the distance between bounding volumes.

# Minimal Bounding Volumes

**Task:**

*Compute the bounding volume for a set of faces such that the volume is optimal with respect to a given measure of approximation quality.*

# Minimal Bounding Volumes

**Task:**

*Compute the bounding volume for a set of faces such that the volume is optimal with respect to a given measure of approximation quality.*

**Measures of Approximation Quality:**

- Volume
- Diameter
- Directed HAUSDORFF-Distance [*Eckstein98*]

# Minimal Bounding Volumes

**Task:**

*Compute the bounding volume for a set of faces such that the volume is optimal with respect to a given measure of approximation quality.*

**Measures of Approximation Quality:**

- Volume
- Diameter
- Directed HAUSDORFF-Distance [*Eckstein98*]

  **Definition 4 (Directed HAUSDORFF Distance)**
  *Given two point sets* $A$ *and* $B$*, then the value*

$$\delta_H(A, B) := \max_{a \in A} \min_{b \in B} \delta(\mathbf{a}, \mathbf{b})$$

  *is called directed* HAUSDORFF*-distance from* $A$ *to* $B$*.*

# Bounding Sphere of Minimal Volume

Let $S(\mathbf{c}, r)$ be a sphere with center $\mathbf{c}$ and radius $r$. The volume is given as

$$V(S) = \frac{4}{3}\pi r^3 \, .$$

# Bounding Sphere of Minimal Volume

Let $S(\mathbf{c}, r)$ be a sphere with center $\mathbf{c}$ and radius $r$. The volume is given as

$$V(S) = \frac{4}{3}\pi r^3 .$$

To minimize the volume we have to find the minimal radius $r^* := r(\mathbf{c}^*)$,

$$\begin{aligned} r(\mathbf{c}) &= \max_{v \in \mathcal{V}(\mathcal{F})} \|\boldsymbol{v} - \mathbf{c}\| \\ &= \max_{v \in CH(\mathcal{V})} \|\boldsymbol{v} - \mathbf{c}\|. \end{aligned}$$

# Bounding Sphere of Minimal Volume

Let $S(\mathbf{c}, r)$ be a sphere with center $\mathbf{c}$ and radius $r$. The volume is given as

$$V(S) = \frac{4}{3}\pi r^3 .$$

To minimize the volume we have to find the minimal radius $r^* := r(\mathbf{c}^*)$,

$$\begin{aligned} r(\mathbf{c}) &= \max_{v \in \mathcal{V}(\mathcal{F})} \|\boldsymbol{v} - \mathbf{c}\| \\ &= \max_{v \in CH(\mathcal{V})} \|\boldsymbol{v} - \mathbf{c}\|. \end{aligned}$$

- The function $r(\mathbf{c}) : \mathbb{R}^3 \to \mathbb{R}$ is convex but not differentiable.

# Bounding Sphere of Minimal Volume

Let $S(\mathbf{c}, r)$ be a sphere with center $\mathbf{c}$ and radius $r$. The volume is given as

$$V(S) = \frac{4}{3}\pi r^3 .$$

To minimize the volume we have to find the minimal radius $r^* := r(\mathbf{c}^*)$,

$$\begin{aligned} r(\mathbf{c}) &= \max_{v \in \mathcal{V}(\mathcal{F})} \|\boldsymbol{v} - \mathbf{c}\| \\ &= \max_{v \in CH(\mathcal{V})} \|\boldsymbol{v} - \mathbf{c}\|. \end{aligned}$$

- The function $r(\mathbf{c}) : \mathbb{R}^3 \to \mathbb{R}$ is convex but not differentiable.
- The minimization problem can be solved using methods of non-linear convex optimization.

# Bounding Sphere of Minimal Volume

Let $S(\mathbf{c}, r)$ be a sphere with center $\mathbf{c}$ and radius $r$. The volume is given as

$$V(S) = \frac{4}{3}\pi r^3 .$$

To minimize the volume we have to find the minimal radius $r^* := r(\mathbf{c}^*)$,

$$\begin{aligned} r(\mathbf{c}) &= \max_{v \in \mathcal{V}(\mathcal{F})} \|\boldsymbol{v} - \mathbf{c}\| \\ &= \max_{v \in \mathsf{CH}(\mathcal{V})} \|\boldsymbol{v} - \mathbf{c}\|. \end{aligned}$$

- The function $r(\mathbf{c}) : \mathbb{R}^3 \to \mathbb{R}$ is convex but not differentiable.
- The minimization problem can be solved using methods of non-linear convex optimization.
- In practice one uses a combinatorial algorithm [*Welzl91*] that incrementally computes the minimal enclosing sphere in expected time $O(|\mathcal{V}|)$.

# OBB of Minimal Volume

The OBB is represented by a matrix $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]$ of normalized box directions, a center $\mathbf{c}$ and a vector of extends $\overline{\mathbf{d}}$. The volume is given by

$$V(B) = \prod_{i=1}^{3} \overline{d}_i \,,$$

$$\text{with} \quad \overline{d}_i = \Big( \max_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i^T \boldsymbol{v} - \min_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i^T \boldsymbol{v} \Big), \quad i = 1, 2, 3 \,.$$

# OBB of Minimal Volume

The OBB is represented by a matrix $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]$ of normalized box directions, a center $\mathbf{c}$ and a vector of extends $\overline{\mathbf{d}}$. The volume is given by

$$V(B) = \prod_{i=1}^{3} \overline{d}_i \,,$$

$$\text{with} \quad \overline{d}_i = \Big( \max_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i^T \boldsymbol{v} - \min_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i^T \boldsymbol{v} \Big), \quad i = 1, 2, 3 \,.$$

Since $\mathbf{D}$ is a rotation matrix, there are EULER-angles $\alpha, \beta, \gamma$, s.t.

$$\mathbf{D} \;=\; \mathbf{R}(\alpha, \beta, \gamma).$$

# OBB of Minimal Volume

The OBB is represented by a matrix $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]$ of normalized box directions, a center $\mathbf{c}$ and a vector of extends $\overline{\mathbf{d}}$. The volume is given by

$$V(B) = \prod_{i=1}^{3} \overline{d}_i \,,$$

$$\text{with} \quad \overline{d}_i = \Big( \max_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i^T \boldsymbol{v} - \min_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i^T \boldsymbol{v} \Big), \quad i = 1, 2, 3 \,.$$

Since $\mathbf{D}$ is a rotation matrix, there are EULER-angles $\alpha, \beta, \gamma$, s.t.

$$\mathbf{D} \;=\; \mathbf{R}(\alpha, \beta, \gamma).$$

Now we have to solve a non-linear and non-convex optimization problem:

$$\begin{aligned} \min \quad & \prod_{i=1}^{3} \Big( \max_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i(\alpha, \beta, \gamma)^T \boldsymbol{v} - \min_{v \in \mathcal{V}(\mathcal{F})} \mathbf{d}_i(\alpha, \beta, \gamma)^T \boldsymbol{v} \Big) \\ \text{s.t.} \quad & \alpha, \beta, \gamma \in [0, 2\pi] \end{aligned}$$

# Combinatorial Algorithms

- The algorithm of O'ROURKE exploits the following necessary condition to find the enclosing box of minimal volume:

  *A box of minimal volume circumscribing a convex polyhedron must have at least two adjacent faces flush with the edges of the polyhedron.* [*O'Rourke85*]

# Combinatorial Algorithms

- The algorithm of O'ROURKE exploits the following necessary condition to find the enclosing box of minimal volume:

  *A box of minimal volume circumscribing a convex polyhedron must have at least two adjacent faces flush with the edges of the polyhedron.* [*O'Rourke85*]

  – It enumerates all pairs of edges of the polyhedron and determines for each pair the OBB of minimal volume in time $O(|\mathcal{E}|)$.

# Combinatorial Algorithms

- The algorithm of O'ROURKE exploits the following necessary condition to find the enclosing box of minimal volume:

  *A box of minimal volume circumscribing a convex polyhedron must have at least two adjacent faces flush with the edges of the polyhedron.* [*O'Rourke85*]

  - It enumerates all pairs of edges of the polyhedron and determines for each pair the OBB of minimal volume in time $O(|\mathcal{E}|)$.
  - Running Time: $O(|\mathcal{E}|^3)$

# Combinatorial Algorithms

- The algorithm of O'ROURKE exploits the following necessary condition to find the enclosing box of minimal volume:

  *A box of minimal volume circumscribing a convex polyhedron must have at least two adjacent faces flush with the edges of the polyhedron.* [*O'Rourke85*]

  - It enumerates all pairs of edges of the polyhedron and determines for each pair the OBB of minimal volume in time $O(|\mathcal{E}|)$.
  - Running Time: $O(|\mathcal{E}|^3)$

- The heuristic of GOTTSCHALK ET AL. is based on the following idea:

  *The bounding box should be oriented along the principal axes of the enclosed face set.* [*Gottschalk,Lin,Manocha96*]

# Combinatorial Algorithms

- The algorithm of O'ROURKE exploits the following necessary condition to find the enclosing box of minimal volume:

  *A box of minimal volume circumscribing a convex polyhedron must have at least two adjacent faces flush with the edges of the polyhedron.* [*O'Rourke85*]

  - It enumerates all pairs of edges of the polyhedron and determines for each pair the OBB of minimal volume in time $O(|\mathcal{E}|)$.
  - Running Time: $O(|\mathcal{E}|^3)$

- The heuristic of GOTTSCHALK ET AL. is based on the following idea:

  *The bounding box should be oriented along the principal axes of the enclosed face set.* [*Gottschalk,Lin,Manocha96*]

  - The algorithm first computes the covariance matrix by sampling over the vertices of $CH(\mathcal{V}(\mathcal{F}))$ or (better) all surface points of $\mathcal{F}$.

# Combinatorial Algorithms

- The algorithm of O'ROURKE exploits the following necessary condition to find the enclosing box of minimal volume:

  *A box of minimal volume circumscribing a convex polyhedron must have at least two adjacent faces flush with the edges of the polyhedron.* [*O'Rourke85*]

  - It enumerates all pairs of edges of the polyhedron and determines for each pair the OBB of minimal volume in time $O(|\mathcal{E}|)$.
  - Running Time: $O(|\mathcal{E}|^3)$

- The heuristic of GOTTSCHALK ET AL. is based on the following idea:

  *The bounding box should be oriented along the principal axes of the enclosed face set.* [*Gottschalk,Lin,Manocha96*]

  - The algorithm first computes the covariance matrix by sampling over the vertices of $CH(\mathcal{V}(\mathcal{F}))$ or (better) all surface points of $\mathcal{F}$.
  - Determining the eigenvectors of the covariance matrix gives the principal axes.

# Partitioning of Face Sets

**Task:**

*Let $\alpha \geq 2$ denote the degree of the hierarchy and $\mathcal{F}$ the given face set. Determine a partition $\{\mathcal{F}_1, \ldots, \mathcal{F}_\alpha\}$ of $\mathcal{F}$ and bounding volumes $H_1, \ldots H_\alpha$, s.t. a given measure of quality is maximized.*

# Partitioning of Face Sets

**Task:**

*Let $\alpha \geq 2$ denote the degree of the hierarchy and $\mathcal{F}$ the given face set. Determine a partition $\{\mathcal{F}_1, \ldots, \mathcal{F}_\alpha\}$ of $\mathcal{F}$ and bounding volumes $H_1, \ldots H_\alpha$, s.t. a given measure of quality is maximized.*

- Combinatorics: for $\alpha = 2$ there are already $\frac{1}{2}(2^n - 2)$ partitions.

# Partitioning of Face Sets

**Task:**

*Let $\alpha \geq 2$ denote the degree of the hierarchy and $\mathcal{F}$ the given face set. Determine a partition $\{\mathcal{F}_1, \ldots, \mathcal{F}_\alpha\}$ of $\mathcal{F}$ and bounding volumes $H_1, \ldots H_\alpha$, s.t. a given measure of quality is maximized.*

- Combinatorics: for $\alpha = 2$ there are already $\frac{1}{2}(2^n - 2)$ partitions.
- Some related problems are NP-complete: e.g. *Euclidean-k-center*.

# Partitioning of Face Sets

**Task:**

*Let $\alpha \geq 2$ denote the degree of the hierarchy and $\mathcal{F}$ the given face set. Determine a partition $\{\mathcal{F}_1, \ldots, \mathcal{F}_\alpha\}$ of $\mathcal{F}$ and bounding volumes $H_1, \ldots H_\alpha$, s.t. a given measure of quality is maximized.*

- Combinatorics: for $\alpha = 2$ there are already $\frac{1}{2}(2^n - 2)$ partitions.
- Some related problems are NP-complete: e.g. *Euclidean-k-center*.
- In practice [*Held,Klosowski,Mitchell95*],[*Gottschalk,Lin,Manocha96*] one uses a simple heuristic with geometric intuition:

# Partitioning of Face Sets

**Task:**

*Let $\alpha \geq 2$ denote the degree of the hierarchy and $\mathcal{F}$ the given face set. Determine a partition $\{\mathcal{F}_1, \ldots, \mathcal{F}_\alpha\}$ of $\mathcal{F}$ and bounding volumes $H_1, \ldots H_\alpha$, s.t. a given measure of quality is maximized.*

- Combinatorics: for $\alpha = 2$ there are already $\frac{1}{2}(2^n - 2)$ partitions.
- Some related problems are NP-complete: e.g. *Euclidean-k-center*.
- In practice [*Held,Klosowski,Mitchell95*],[*Gottschalk,Lin,Manocha96*] one uses a simple heuristic with geometric intuition:
  - Associate each polygon with a single reference point (e.g. the centroid).

# Partitioning of Face Sets

**Task:**

*Let $\alpha \geq 2$ denote the degree of the hierarchy and $\mathcal{F}$ the given face set. Determine a partition $\{\mathcal{F}_1, \ldots, \mathcal{F}_\alpha\}$ of $\mathcal{F}$ and bounding volumes $H_1, \ldots H_\alpha$, s.t. a given measure of quality is maximized.*

- Combinatorics: for $\alpha = 2$ there are already $\frac{1}{2}(2^n - 2)$ partitions.
- Some related problems are NP-complete: e.g. *Euclidean-k-center*.
- In practice [*Held,Klosowski,Mitchell95*],[*Gottschalk,Lin,Manocha96*] one uses a simple heuristic with geometric intuition:
  - Associate each polygon with a single reference point (e.g. the centroid).
  - Choose a splitting plane (direction and position).

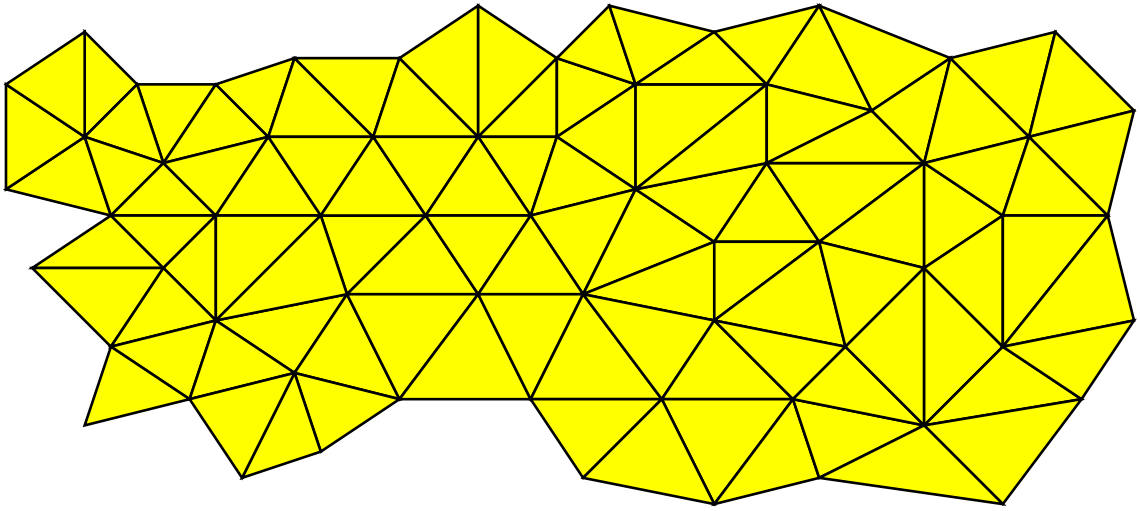
# Partitioning of Face Sets

**Task:**

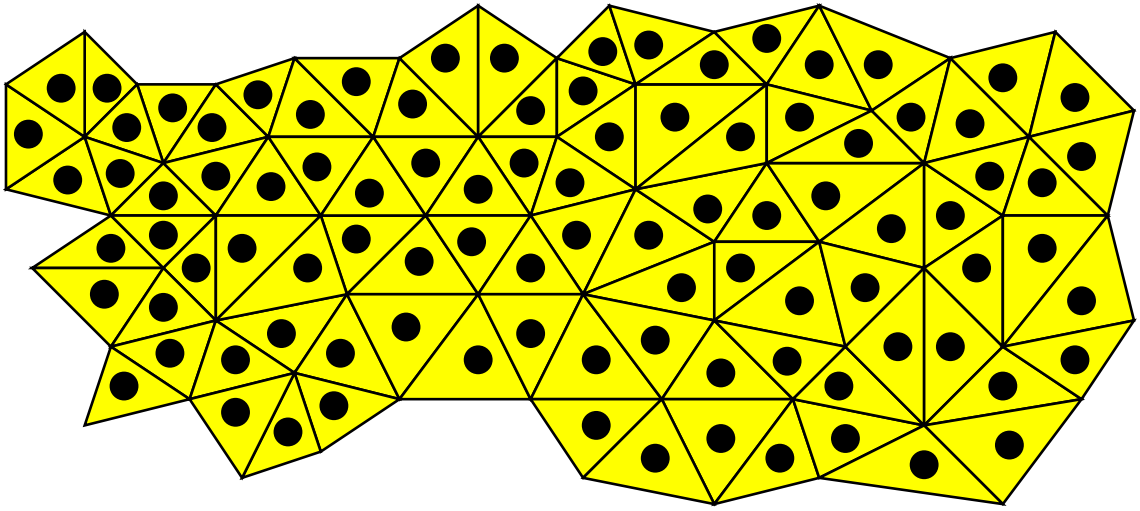
*Let $\alpha \geq 2$ denote the degree of the hierarchy and $\mathcal{F}$ the given face set. Determine a partition $\{\mathcal{F}_1, \ldots, \mathcal{F}_\alpha\}$ of $\mathcal{F}$ and bounding volumes $H_1, \ldots H_\alpha$, s.t. a given measure of quality is maximized.*

- Combinatorics: for $\alpha = 2$ there are already $\frac{1}{2}(2^n - 2)$ partitions.
- Some related problems are NP-complete: e.g. *Euclidean-k-center*.
- In practice [*Held,Klosowski,Mitchell95*],[*Gottschalk,Lin,Manocha96*] one uses a simple heuristic with geometric intuition:
  - Associate each polygon with a single reference point (e.g. the centroid).
  - Choose a splitting plane (direction and position).
  - Assign each polygon to one side of the plane by locating its reference point.

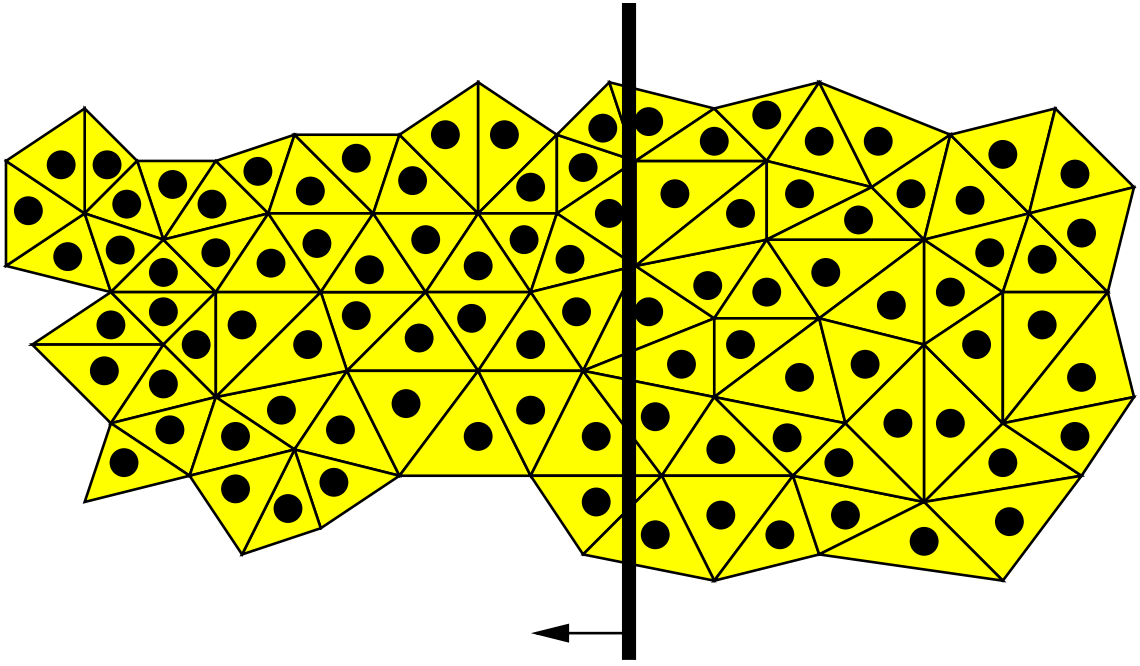
# Example of the Partitioning Heuristic



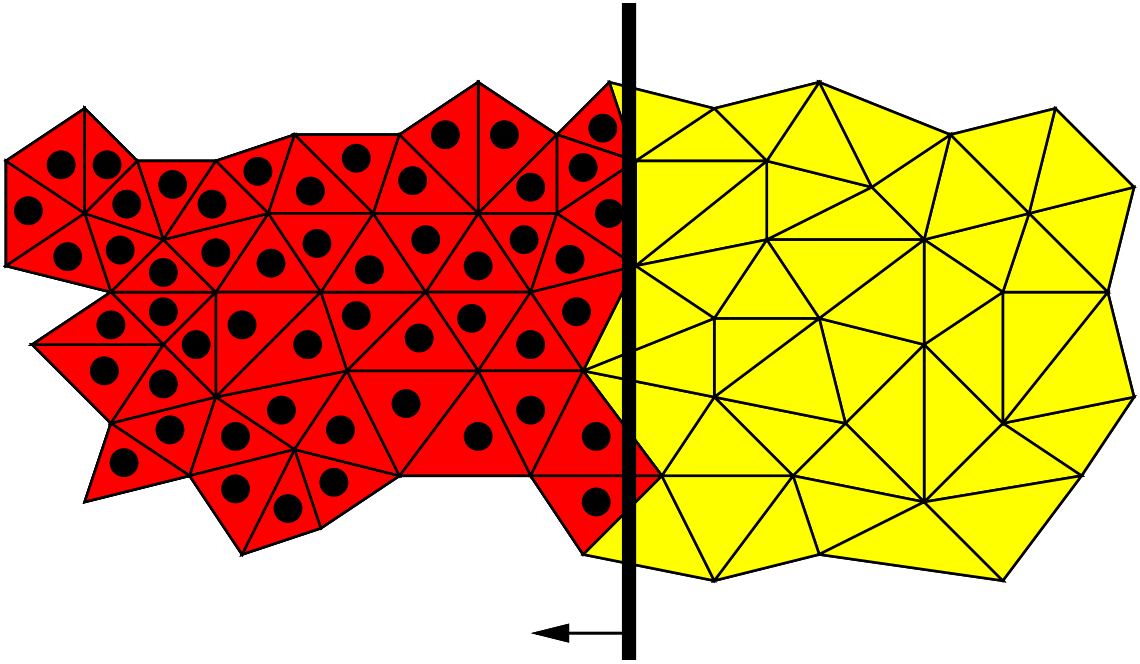
# Example of the Partitioning Heuristic



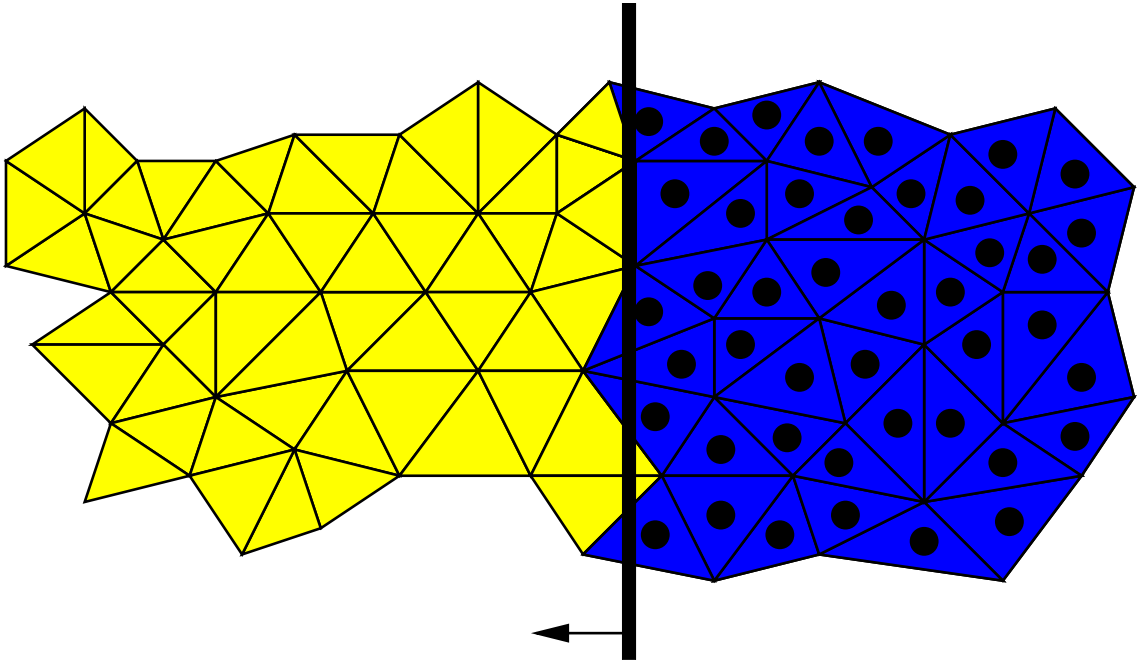
# Example of the Partitioning Heuristic



# Example of the Partitioning Heuristic



# Example of the Partitioning Heuristic



# Choosing the Direction of the Splitting Plane

The normal of the splitting plane is typically chosen as

- one of the principal axes of the given face set [*Gottschalk,Lin,Manocha96*],
- one of the coordinate axes [*Held,Klosowski,Mitchell95*].

# Choosing the Direction of the Splitting Plane

The normal of the splitting plane is typically chosen as

- one of the principal axes of the given face set [*Gottschalk,Lin,Manocha96*],
- one of the coordinate axes [*Held,Klosowski,Mitchell95*].

Thereby one can consider the following objective functions:

- Choose the axis, that minimizes the sum of the volumes or the maximal volume of the resulting child BVs:

$$\min_{d \in D} \sum_{j=1}^{\alpha} V\left(H_d\right) \quad \text{or} \quad \min_{d \in D} \max_{1 \le j \le \alpha} V\left(H_d\right) .$$

# Choosing the Direction of the Splitting Plane

The normal of the splitting plane is typically chosen as

- one of the principal axes of the given face set [*Gottschalk,Lin,Manocha96*],
- one of the coordinate axes [*Held,Klosowski,Mitchell95*].

Thereby one can consider the following objective functions:

- Choose the axis, that minimizes the sum of the volumes or the maximal volume of the resulting child BVs:

$$\min_{d \in D} \sum_{j=1}^{\alpha} V\big(H_d\big) \quad \text{or} \quad \min_{d \in D} \max_{1 \le j \le \alpha} V\big(H_d\big) \, .$$

- Choose the axis that yields the largest variance when projecting the reference points onto:

$$\max_{d \in D} \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} (\mathbf{d}^T \mathbf{p}_f - \mu)^2 \, , \qquad \mu := \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \mathbf{d}^T \mathbf{p}_f \, .$$

# Choosing the Position of the Splitting Plane

To choose the position of splitting plane one can consider the following objective functions:

- Choose the point $\mathbf{p}_f$ for which the projection is closest to the mean of all projected reference points:

$$\min_{f \in \mathcal{F}} \|\mathbf{d}^T \mathbf{p}_f - \mu\| \, .$$

# Choosing the Position of the Splitting Plane

To choose the position of splitting plane one can consider the following objective functions:

- Choose the point $\mathbf{p}_f$ for which the projection is closest to the mean of all projected reference points:

$$\min_{f \in \mathcal{F}} \|\mathbf{d}^T \mathbf{p}_f - \mu\| .$$

- Choose the point $\mathbf{p}_f$ that yields the median of the projected reference points.

# Distance Computation Between Bounding Volumes

**Tasks:**

- Compute the Euclidean distance between the geometric primitives: spheres, OBBs, AABBs, $FDH_k$...

- Minimize the effort spent on updating the bounding volumes during object movement.

# Updating Bounding Volumes

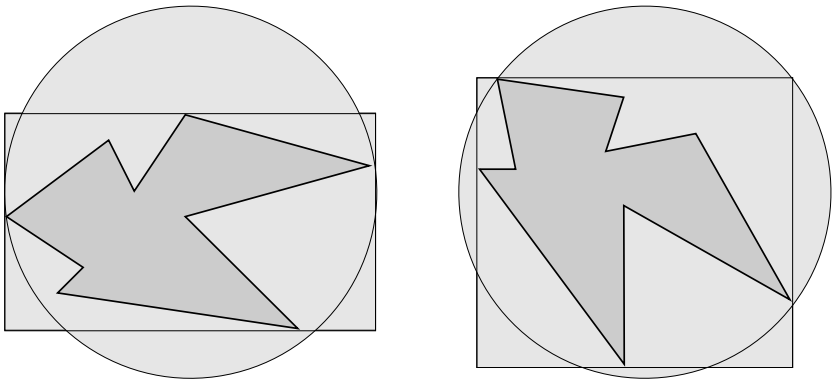
**Problem:**
The movement of objects in time implies that:

- the geometry has to be updated according to the new position and orientation of the object,
- the bounding volume hierarchy has to be updated, since, in general, the BVs are no longer optimal with respect the transformed face set.

# Updating Bounding Volumes

**Problem:**
The movement of objects in time implies that:

- the geometry has to be updated according to the new position and orientation of the object,
- the bounding volume hierarchy has to be updated, since, in general, the BVs are no longer optimal with respect the transformed face set.



# Cost of BV-Update

The cost of updating bounding volumes depends on their closure properties under translation and rotation:

- Minimal Bounding Spheres and OBBs preserve their optimality properties under translation and rotation.
- $\mathrm{FDH}_k$s and AABBs must be recomputed to become optimal with respect to the rotated face set.

# Cost of BV-Update

The cost of updating bounding volumes depends on their closure properties under translation and rotation:

- Minimal Bounding Spheres and OBBs preserve their optimality properties under translation and rotation.
- $\mathrm{FDH}_k$s and AABBs must be recomputed to become optimal with respect to the rotated face set.

**Reducing the update costs:**

- By transforming one object into the local coordinate system of the other, we only have to update the BV-hierarchy of one object.
- For $\mathrm{FDH}_k$s and AABBs, we can apply the object transformation to the BV of the previous time frame and compute the optimal BV of this transformed BV.

# Distance Computation Between OBBs

**The Edge Classification Algorithm [*Meyer86*]**

**Observation 2 (Box-Box-Distance)**
*If the boxes are disjoint, then the minimal distance is determined by*

*(i) an edge of* $B_1$ *and the box* $B_2$ **or**

*(ii) a vertex of* $B_2$ *and a face of* $B_1$.

# Distance Computation Between OBBs

**The Edge Classification Algorithm [*Meyer86*]**

**Observation 2 (Box-Box-Distance)**
*If the boxes are disjoint, then the minimal distance is determined by*

*(i) an edge of* $B_1$ *and the box* $B_2$ **or**

*(ii) a vertex of* $B_2$ *and a face of* $B_1$.

**Idea:**

- Case (ii): In the local coordinate system of $B_1$ one only needs coordinate comparisons.

# Distance Computation Between OBBs

**The Edge Classification Algorithm [*Meyer86*]**

**Observation 2 (Box-Box-Distance)**
*If the boxes are disjoint, then the minimal distance is determined by*

*(i) an edge of* $B_1$ *and the box* $B_2$ **or**

*(ii) a vertex of* $B_2$ *and a face of* $B_1$.

**Idea:**

- Case (ii): In the local coordinate system of $B_1$ one only needs coordinate comparisons.
- Case (i): Exploit box geometry to reduce the number of edge-edge tests:

# Distance Computation Between OBBs

**The Edge Classification Algorithm [*Meyer86*]**

**Observation 2 (Box-Box-Distance)**
*If the boxes are disjoint, then the minimal distance is determined by*

*(i) an edge of* $B_1$ *and the box* $B_2$ **or**

*(ii) a vertex of* $B_2$ *and a face of* $B_1$.

**Idea:**

- Case (ii): In the local coordinate system of $B_1$ one only needs coordinate comparisons.
- Case (i): Exploit box geometry to reduce the number of edge-edge tests:
  - Assign the edges of one box to the VORONOI regions of the other box.

# Distance Computation Between OBBs

**The Edge Classification Algorithm [*Meyer86*]**

**Observation 2 (Box-Box-Distance)**
*If the boxes are disjoint, then the minimal distance is determined by*

*(i) an edge of* $B_1$ *and the box* $B_2$ **or**

*(ii) a vertex of* $B_2$ *and a face of* $B_1$.

**Idea:**

- Case (ii): In the local coordinate system of $B_1$ one only needs coordinate comparisons.
- Case (i): Exploit box geometry to reduce the number of edge-edge tests:
  - Assign the edges of one box to the VORONOI regions of the other box.
  - Use region specific proximity tests to compute the distance.

# Distance Computation Between OBBs

**The Edge Classification Algorithm [*Meyer86*]**

**Observation 2 (Box-Box-Distance)**
*If the boxes are disjoint, then the minimal distance is determined by*

*(i) an edge of* $B_1$ *and the box* $B_2$ **or**

*(ii) a vertex of* $B_2$ *and a face of* $B_1$.

**Idea:**

- Case (ii): In the local coordinate system of $B_1$ one only needs coordinate comparisons.
- Case (i): Exploit box geometry to reduce the number of edge-edge tests:
  - Assign the edges of one box to the VORONOI regions of the other box.
  - Use region specific proximity tests to compute the distance.
  - Break the edge into pieces if it passes through more than one cell.

# Distance Computation Between OBBs

**The Edge Classification Algorithm [*Meyer86*]**

**Observation 2 (Box-Box-Distance)**
*If the boxes are disjoint, then the minimal distance is determined by*

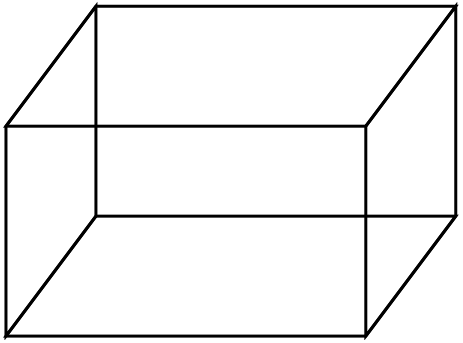
*(i) an edge of* $B_1$ *and the box* $B_2$ **or**

*(ii) a vertex of* $B_2$ *and a face of* $B_1$.

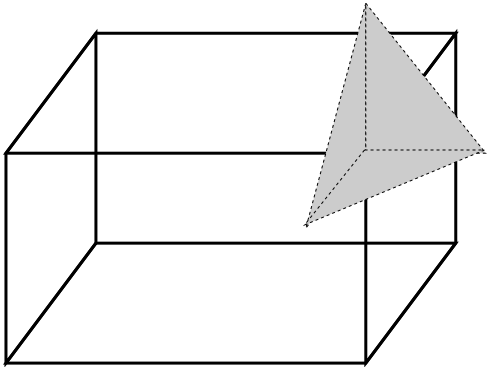
**Idea:**

- Case (ii): In the local coordinate system of $B_1$ one only needs coordinate comparisons.
- Case (i): Exploit box geometry to reduce the number of edge-edge tests:
  - Assign the edges of one box to the VORONOI regions of the other box.
  - Use region specific proximity tests to compute the distance.
  - Break the edge into pieces if it passes through more than one cell.

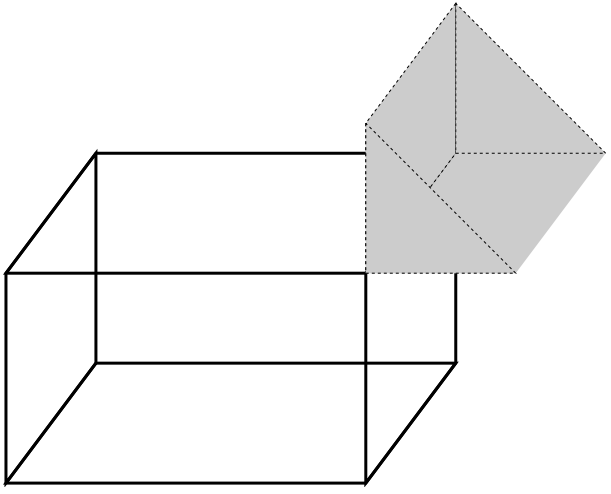
# Classification of Edges



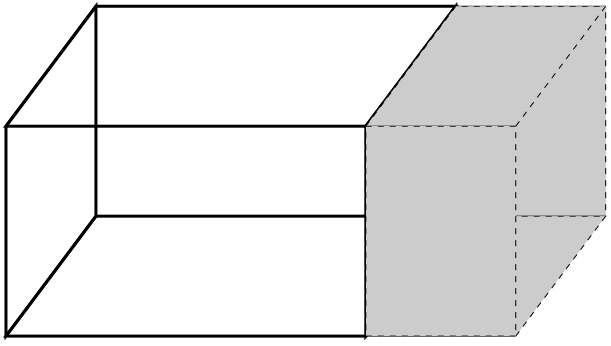
# Classification of Edges



# Classification of Edges



# Classification of Edges



# Classification of Edges