Computational Geometry

Binary Space Partitions: The Painter’s Algorithm
Lecture #13

[Comp. Geom A&A : Chapter 12]
The Painter’s Algorithm

Idea:
The Painter’s Algorithm

Idea:
The Painter’s Algorithm

Idea:

scan-convert
The Painter’s Algorithm

Idea: scan-convert
The Painter’s Algorithm

Idea:

scan-convert
The Painter's Algorithm

Idea:

scan-convert

Does this always work?
The Painter’s Algorithm

Idea:

Problem:

Does this always work?

The Painter’s Algorithm

Idea:

1. scan-convert

2. Does this always work?

3. Cut objects into smaller pieces!

Problem:

BSP and BSP Tree
BSP and BSP Tree
BSP and BSP Tree

\[ \ell_1 \]

\[ \ell_2 \]

\[ \ell_3 \]

\[ \ell_4 \]

\[ \ell_5 \]

\[ \ell_6 \]

\[ o_1 \]

\[ o_2 \]

\[ o_3 \]

\[ o_4 \]

\[ o_5 \]

\[ l_1 \]

\[ l_2 \]

\[ l_3 \]
BSP and BSP Tree

Diagram showing a BSP (Binary Space Partitioning) and a BSP Tree. The diagram includes lines $\ell_1$, $\ell_2$, $\ell_3$, $\ell_4$, $\ell_5$, and $\ell_6$, and objects $O_1$, $O_2$, $O_3$, $O_4$, and $O_5$. The BSP Tree is structured with nodes $\ell_1$, $\ell_2$, $\ell_3$, $\ell_4$, $\ell_5$, and $\ell_6$, and leaves $O_2$, $O_1$, $O_1$, and $O_3$.
BSP and BSP Tree
BSP and BSP Tree
Observation.
Nodes of a BSP tree correspond to line segments, and to regions of the plane.
Autopartitions

ℓ₁
Autopartitions

\[ \ell_1 \]

New rule (*autopartition*):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).
**Autopartitions**

New rule (*autopartition*):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).
Any separating hyperplane (here: lines) must contain an object (or the face of an object).

New rule (autopartition):
New rule (autopartition):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).
Autopartitions

Idea:
Any separating hyperplane (here: lines) must contain an object (or the face of an object).

New rule (autopartition):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).
Autopartitions

New rule (autopartition):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).

Idea:
– First, draw objects in $\ell_1^-$;
Autopartitions

New rule (autopartition):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).

Idea:
– First, draw objects in \( \ell_1^- \);
– then, draw objects in \( \ell_1^+ \);
Autopartitions

New rule (*autopartition*):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).

Idea:
- First, draw objects in $\ell_1^-$;
- then, draw objects in $\ell_1$;
- finally, draw objects in $\ell_1^+$.  

Any separating hyperplane (here: lines) must contain an object (or the face of an object).
Autopartitions

Idea:
– First, draw objects in $\ell_1^-$;
– then, draw objects in $\ell_1$;
– finally, draw objects in $\ell_1^+$.

This corresponds to an in-order tree walk!

New rule (autopartition):
Any separating hyperplane (here: lines) must contain an object (or the face of an object).
BSP and the Painter’s Algorithm
**BSP and the Painter's Algorithm**

**Algorithm** PAINTERSALGORITHM(\(T, p_{\text{view}}\))
1. Let \(v\) be the root of \(T\).
2. if \(v\) is a leaf
3. then Scan-convert the object fragments in \(S(v)\).
4. else if \(p_{\text{view}} \in h^+_v\)
5. then PAINTERSALGORITHM(\(T^-, p_{\text{view}}\))
6. Scan-convert the object fragments in \(S(v)\).
7. PAINTERSALGORITHM(\(T^+, p_{\text{view}}\))
BSP and the Painter’s Algorithm

**Algorithm** `PAINTERSALGORITHM(\(T\), \(p_{\text{view}}\))

1. Let \(v\) be the root of \(T\).
2. if \(v\) is a leaf
3. then Scan-convert the object fragments in \(S(v)\).
4. else if \(p_{\text{view}} \in h_v^+\)
5. then `PAINTERSALGORITHM(\(T^-\), \(p_{\text{view}}\))
6. Scan-convert the object fragments in \(S(v)\).
7. `PAINTERSALGORITHM(\(T^+\), \(p_{\text{view}}\))

set of objects in the region of \(v\)
BSP and the Painter’s Algorithm

**Algorithm** `PAINTERSALGORITHM(\(T, p_{\text{view}}\))`

1. Let \(v\) be the root of \(T\).
2. if \(v\) is a leaf
3. then Scan-convert the object fragments in \(S(v)\).
4. else if \(p_{\text{view}} \in [h^+_v]\)
5. then `PAINTERSALGORITHM(\(T^-, p_{\text{view}}\))`
6. Scan-convert the object fragments in \(S(v)\).
7. `PAINTERSALGORITHM(\(T^+, p_{\text{view}}\))`
8. else if \(p_{\text{view}} \in [h^-_v]\)
9. then `PAINTERSALGORITHM(\(T^+, p_{\text{view}}\))`
10. Scan-convert the object fragments in \(S(v)\).
11. `PAINTERSALGORITHM(\(T^-, p_{\text{view}}\))`
BSP and the Painter’s Algorithm

**Algorithm** `PAINTERSALGORITHM(\mathcal{T}, p_{view})`

1. Let $v$ be the root of $\mathcal{T}$.
2. **if** $v$ is a leaf
3. **then** Scan-convert the object fragments in $S(v)$.
4. **else if** $p_{view} \in [h^+_v]$  
5. **then** `PAINTERSALGORITHM(\mathcal{T}^-, p_{view})`
6. **else if** $p_{view} \in [h^-_v]$  
7. **then** `PAINTERSALGORITHM(\mathcal{T}^+, p_{view})`
8. **else** Scan-convert the object fragments in $S(v)$.
9. `PAINTERSALGORITHM(\mathcal{T}^+, p_{view})`
10. `PAINTERSALGORITHM(\mathcal{T}^-, p_{view})`

**set of objects in the region of $v$**
Algorithm \textsc{PaintersAlgorithm}(\mathcal{T}, p_{\text{view}})
1. Let \nu be the root of \mathcal{T}.
2. \textbf{if} \nu \text{ is a leaf}
3. \textbf{then} Scan-convert the object fragments in \text{S}(\nu).
4. \textbf{else if} \ p_{\text{view}} \in [h_{\nu}^{+}]
5. \textbf{then} \textsc{PaintersAlgorithm}(\mathcal{T}^{-}, p_{\text{view}})
6. \text{Scan-convert the object fragments in } \text{S}(\nu).
7. \textsc{PaintersAlgorithm}(\mathcal{T}^{+}, p_{\text{view}})
8. \textbf{else if} \ p_{\text{view}} \in [h_{\nu}^{-}]
9. \textbf{then} \textsc{PaintersAlgorithm}(\mathcal{T}^{+}, p_{\text{view}})
10. \text{Scan-convert the object fragments in } \text{S}(\nu).
11. \textsc{PaintersAlgorithm}(\mathcal{T}^{-}, p_{\text{view}})
12. \textbf{else} (* p_{\text{view}} \in [h_{\nu}] *)
13. \textsc{PaintersAlgorithm}(\mathcal{T}^{+}, p_{\text{view}})
14. \textsc{PaintersAlgorithm}(\mathcal{T}^{-}, p_{\text{view}})
Algorithm for Computing a BSP in 2d

Algorithm 2DBSP(S)
Input. A set \( S = \{s_1, s_2, \ldots, s_n\} \) of segments.
Output. A BSP tree for \( S \).
1. \textbf{if} \( \text{card}(S) \leq 1 \)
2. \textbf{then} Create a tree \( \mathcal{T} \) consisting of a single leaf node, where the set \( S \) is stored explicitly.
3. \textbf{return} \( \mathcal{T} \)
4. \textbf{else} \hspace{1em} (* Use \( \ell(s_1) \) as the splitting line. *)
5. \( S^+ \leftarrow \{s \cap \ell(s_1)^+: s \in S\}; \quad \mathcal{T}^+ \leftarrow 2\text{DBSP}(S^+) \)
6. \( S^- \leftarrow \{s \cap \ell(s_1)^- : s \in S\}; \quad \mathcal{T}^- \leftarrow 2\text{DBSP}(S^-) \)
7. Create a BSP tree \( \mathcal{T} \) with root node \( v \), left subtree \( \mathcal{T}^- \), right subtree \( \mathcal{T}^+ \), and with \( S(v) = \{s \in S : s \subset \ell(s_1)\} \).
8. \textbf{return} \( \mathcal{T} \)
Algorithm for Computing a BSP in 2d

**Algorithm 2DBSP(S)**

*Input.* A set $S = \{s_1, s_2, \ldots, s_n\}$ of segments, randomly permuted.

*Output.* A BSP tree for $S$.

1. **if** $\text{card}(S) \leq 1$
2. **then** Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set $S$ is stored explicitly.
3. **return** $\mathcal{T}$
4. **else** (*Use $\ell(s_1)$ as the splitting line.*)
5. $S^+ \leftarrow \{s \cap \ell(s_1)^+: s \in S\}$; $\mathcal{T}^+ \leftarrow 2\text{DBSP}(S^+)$
6. $S^- \leftarrow \{s \cap \ell(s_1)^- : s \in S\}$; $\mathcal{T}^- \leftarrow 2\text{DBSP}(S^-)$
7. Create a BSP tree $\mathcal{T}$ with root node $v$, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{s \in S : s \subset \ell(s_1)\}$.
8. **return** $\mathcal{T}$
Algorithm for Computing a BSP in 2d

\underline{Random}

**Algorithm 2DBSP(S)**

*Input.* A set \( S = \{s_1, s_2, \ldots, s_n\} \) of segments, randomly permuted.

*Output.* A BSP tree for \( S \).

1. **if** \( \text{card}(S) \leq 1 \)
2. **then** Create a tree \( \mathcal{T} \) consisting of a single leaf node, where the set \( S \) is stored explicitly.
3. **return** \( \mathcal{T} \)
4. **else** (*Use \( \ell(s_1) \) as the splitting line.*)
5. \( S^+ \leftarrow \{s \cap \ell(s_1)^+ : s \in S\} \); \( \mathcal{T}^+ \leftarrow 2\text{DBSP}(S^+) \)
6. \( S^- \leftarrow \{s \cap \ell(s_1)^- : s \in S\} \); \( \mathcal{T}^- \leftarrow 2\text{DBSP}(S^-) \)
7. Create a BSP tree \( \mathcal{T} \) with root node \( v \), left subtree \( \mathcal{T}^- \), right subtree \( \mathcal{T}^+ \), and with \( S(v) = \{s \in S : s \subset \ell(s_1)\} \).
8. **return** \( \mathcal{T} \)

**Question:** What is the expected size of the autopartition corresponding to \( \mathcal{T} \)?
Algorithm for Computing a BSP in 2d

**Algorithm 2DBSP(S)**

*Input.* A set $S = \{s_1, s_2, \ldots, s_n\}$ of segments, randomly permuted.

*Output.* A BSP tree for $S$.

1. \textbf{if} $\text{card}(S) \leq 1$
2. \textbf{then} Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set $S$ is stored explicitly.
3. \textbf{return} $\mathcal{T}$
4. \textbf{else} (* Use $\ell(s_1)$ as the splitting line, *)
5. $S^+ \leftarrow \{s \cap \ell(s_1)^+: s \in S\}$; \quad $\mathcal{T}^+ \leftarrow 2DBSP(S^+)$
6. $S^- \leftarrow \{s \cap \ell(s_1)^- : s \in S\}$; \quad $\mathcal{T}^- \leftarrow 2DBSP(S^-)$
7. Create a BSP tree $\mathcal{T}$ with root node $v$, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{s \in S : s \subset \ell(s_1)\}$.
8. \textbf{return} $\mathcal{T}$

**Question:** What is the expected size of the autopartition corresponding to $\mathcal{T}$?

**Theorem.** $2D\text{RANDOMBSD}$ produces an autopartition of expected complexity $O(n \log n)$. 
Analysis

**Thm.** $\text{2dRandomBsp}$ produces an autopartition of complexity $O(n \log n)$.

*Proof.*

![Diagram showing lines intersecting at various points.](image)
Analysis

**Thm.** \(2d\text{RANDOM}\text{BSP}\) produces an autopartition of complexity \(O(n \log n)\).

**Proof.**
Let \(s_i \in S\) be a fixed segment.
Thm. \textbf{2dRandomBsp} produces an autopartition of complexity $O(n \log n)$.

\textbf{Proof.}

Let $s_i \in S$ be a fixed segment.

We analyze the expected number of line segments that are split when we insert $\ell(s_i)$. 
Analysis

**Thm.** \(2d\text{RANDOMBSP} \) produces an autopartition of complexity \(O(n \log n)\).

**Proof.**

Let \(s_i \in S\) be a fixed segment.

We analyze the expected number of line segments that are split when we insert \(\ell(s_i)\).

Consider \(s_j \in S\) with \(s_j \cap \ell(s_i) \neq \emptyset\):
Analysis

**Thm.** $2d$RandomBsp produces an autopartition of complexity $O(n \log n)$. 

**Proof.**
Let $s_i \in S$ be a fixed segment.

We analyze the expected number of line segments that are split when we insert $\ell(s_i)$.

Consider $s_j \in S$ with $s_j \cap \ell(s_i) \neq \emptyset$: $s_j$ is split only if all segments between $s_i$ and $s_j$ have index $> i$. 

![Diagram](image-url)
Analysis

**Thm.** \(2\text{dRandomBsp}\) produces an autopartition of complexity \(O(n \log n)\).

**Proof.**

Let \(s_i \in S\) be a fixed segment.

We analyze the expected number of line segments that are split when we insert \(\ell(s_i)\).

Consider \(s_j \in S\) with \(s_j \cap \ell(s_i) \neq \emptyset\):

\(s_j\) is split only if all segments between \(s_i\) and \(s_j\) have index \(> i\).

\[
\text{dist}_i(s_j) := \begin{cases} 
\# \text{ segments that intersect } \ell(s_i) \text{ between } s_i \text{ and } s_j \\
\infty, \text{ otherwise}
\end{cases}
\]
Analysis

Thm. \textsc{2dRandomBsp} produces an autopartition of complexity $O(n \log n)$.

\begin{proof}
Let $s_i \in S$ be a fixed segment.

We analyze the expected number of line segments that are split when we insert $\ell(s_i)$.

Consider $s_j \in S$ with $s_j \cap \ell(s_i) \neq \emptyset$:

$s_j$ is split only if all segments between $s_i$ and $s_j$ have index $> i$.

\[ \text{dist}_i(s_j) := \begin{cases} 
\# \text{ segments that intersect } \ell(s_i) \text{ between } s_i \text{ and } s_j \\
\infty, \text{ otherwise}
\end{cases} \]
Analysis

**Thm.** \(2d\text{RANDOMBSP} \) produces an autopartition of complexity \(O(n \log n)\).

**Proof.**
Let \(s_i \in S\) be a fixed segment.
We analyze the expected number of line segments that are split when we insert \(\ell(s_i)\).

Consider \(s_j \in S\) with \(s_j \cap \ell(s_i) \neq \emptyset\):

\(s_j\) is split only if all segments between \(s_i\) and \(s_j\) have index \(> i\).

\[
\text{dist}_i(s_j) := \begin{cases} 
\# \text{ segments that intersect } \ell(s_i) \text{ between } s_i \text{ and } s_j \n\infty, \text{ otherwise}
\end{cases}
\]
Analysis

**Thm.** \texttt{2dRandomBsp} produces an autopartition of complexity $O(n \log n)$.

**Proof.**

Let $s_i \in S$ be a fixed segment.

We analyze the expected number of line segments that are split when we insert $\ell(s_i)$.

Consider $s_j \in S$ with $s_j \cap \ell(s_i) \neq \emptyset$:

$s_j$ is split only if all segments between $s_i$ and $s_j$ have index $> i$.

$$\text{dist}_i(s_j) := \begin{cases} 
\# \text{ segments that intersect } \ell(s_i) \text{ between } s_i \text{ and } s_j \\
\infty, \text{ otherwise}
\end{cases}$$
**Thm.** \(2d\text{RANDOMBSP}\) produces an autopartition of complexity \(O(n \log n)\).

**Proof.** (cont’d)
Analysis

\textbf{Thm.} \texttt{2dRandomBsp} produces an autopartition of complexity $O(n \log n)$.

\textit{Proof.} (cont’d)
Let $k = \text{dist}_i(s_j)$. Let $s_{j_1}, \ldots, s_{j_k}$ be the segments between $s_i$ and $s_j$. 
Thm. \( 2dRand\text{omBsp} \) produces an autopartition of complexity \( O(n \log n) \).

Proof. (cont’d)

Let \( k = \text{dist}_i(s_j) \). Let \( s_{j_1}, \ldots, s_{j_k} \) be the segments between \( s_i \) and \( s_j \).

What is the probability that \( \ell(s_i) \) splits \( s_j \)?
Analysis

**Thm.** \(2\text{D\textsc{RandomBsp}} \) produces an autopartition of complexity \(O(n \log n)\).

**Proof.** (cont’d)

Let \(k = \text{dist}_i(s_j)\). Let \(s_{j_1}, \ldots, s_{j_k}\) be the segments between \(s_i\) and \(s_j\).

What is the probability that \(\ell(s_i)\) splits \(s_j\)?

For this to happen, \(s_i\) must be first among \(\{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\}\).
Analysis

**Thm.** \( \text{2dRandomBsp} \) produces an autopartition of complexity \( O(n \log n) \).

**Proof.** (cont’d)

Let \( k = \text{dist}_i(s_j) \). Let \( s_{j_1}, \ldots, s_{j_k} \) be the segments between \( s_i \) and \( s_j \).

What is the probability that \( \ell(s_i) \) splits \( s_j \)?

For this to happen, \( s_i \) must be first among \( \{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\} \).

So \( i \) must be the minimum of \( \{i, j, j_1, \ldots, j_k\} \).
Analysis

**Thm.** *2dRandomBsp* produces an autopartition of complexity $O(n \log n)$.

**Proof.** (cont’d)

Let $k = \text{dist}_i(s_j)$. Let $s_{j_1}, \ldots, s_{j_k}$ be the segments between $s_i$ & $s_j$.

What is the probability that $\ell(s_i)$ splits $s_j$?

For this to happen, $s_i$ must be first among $\{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\}$.

So $i$ must be the minimum of $\{i, j, j_1, \ldots, j_k\}$.

$\Rightarrow P[\ell(s_i) \text{ splits } s_j] \leq $
**Analysis**

### Thm.

2dRandomBsp produces an autopartition of complexity $O(n \log n)$.

**Proof.** (cont’d)

Let $k = \text{dist}_i(s_j)$. Let $s_{j_1}, \ldots, s_{j_k}$ be the segments betw. $s_i$ & $s_j$.

What is the probability that $\ell(s_i)$ splits $s_j$?

For this to happen, $s_i$ must be first among $\{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\}$.

So $i$ must be the minimum of $\{i, j, j_1, \ldots, j_k\}$.

$\Rightarrow P[\ell(s_i) \text{ splits } s_j] \leq \frac{1}{\text{dist}_i(s_j)+2}$
Analysis

**Thm.** \( \text{2dRandomBsp} \) produces an autopartition of complexity \( O(n \log n) \).

**Proof.** (cont’d)

Let \( k = \text{dist}_i(s_j) \). Let \( s_{j_1}, \ldots, s_{j_k} \) be the segments betw. \( s_i \) & \( s_j \).

What is the probability that \( \ell(s_i) \) splits \( s_j \)?

For this to happen, \( s_i \) must be first among \( \{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\} \).

So \( i \) must be the minimum of \( \{i, j, j_1, \ldots, j_k\} \).

\[ \Rightarrow P[\ell(s_i) \text{ splits } s_j] \leq \frac{1}{\text{dist}_i(s_j) + 2} \quad ("\leq" \text{ since a line through some other segment can "save" } s_j) \]
Analysis

**Thm.** \(2\text{dRandomBsp} \) produces an autopartition of complexity \(O(n \log n)\).

**Proof.** (cont’d)

Let \( k = \text{dist}_i(s_j) \). Let \( s_{j_1}, \ldots, s_{j_k} \) be the segments betw. \( s_i \) & \( s_j \).

What is the probability that \( \ell(s_i) \) splits \( s_j \)?

For this to happen, \( s_i \) must be first among \( \{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\} \).

So \( i \) must be the minimum of \( \{i, j, j_1, \ldots, j_k\} \).

\[ \Rightarrow \text{P}[\ell(s_i) \text{ splits } s_j] \leq \frac{1}{\text{dist}_i(s_j)+2} \quad (\text{“} \leq \text{” since a line through some other segment can “save” } s_j) \]

\[ \Rightarrow \text{E}[\# \text{ segments that } s_i \text{ splits}] \leq \]
Analysis

**Thm.** $\texttt{2dRandomBsp}$ produces an autopartition of complexity $O(n \log n)$.

**Proof.** (cont’d)

Let $k = \text{dist}_i(s_j)$. Let $s_{j_1}, \ldots, s_{j_k}$ be the segments betw. $s_i$ & $s_j$.

What is the probability that $\ell(s_i)$ splits $s_j$?

For this to happen, $s_i$ must be first among $\{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\}$.

So $i$ must be the minimum of $\{i, j, j_1, \ldots, j_k\}$.

$$\Rightarrow \textbf{P}[\ell(s_i) \text{ splits } s_j] \leq \frac{1}{\text{dist}_i(s_j)+2} \quad ("\leq" \text{ since a line through some other segment can "save" } s_j)$$

$$\Rightarrow \textbf{E}[\# \text{ segments that } s_i \text{ splits}] \leq \sum_{j \neq i} \frac{1}{\text{dist}_i(s_j)+2} \leq$$
Analysis

**Thm.** \( \text{2dRandomBsp} \) produces an autopartition of complexity \( O(n \log n) \).

**Proof.** (cont’d)

Let \( k = \text{dist}_i(s_j) \). Let \( s_{j1}, \ldots, s_{jk} \) be the segments betw. \( s_i \) & \( s_j \).

What is the probability that \( \ell(s_i) \) splits \( s_j \)?

For this to happen, \( s_i \) must be first among \( \{s_i, s_j, s_{j1}, \ldots, s_{jk}\} \).

So \( i \) must be the minimum of \( \{i, j, j_1, \ldots, j_k\} \).

\( \Rightarrow \) \( P[\ell(s_i) \text{ splits } s_j] \leq \frac{1}{\text{dist}_i(s_j)+2} \) ("\( \leq \)" since a line through some other segment can "save" \( s_j \))

\( \Rightarrow \) \( E[\# \text{ segments that } s_i \text{ splits}] \leq \sum_{j \neq i} \frac{1}{\text{dist}_i(s_j)+2} \leq 2 \sum_{k=0}^{n-2} \frac{1}{k+2} \)
Analysis

Thm.  \texttt{2dRandomBsp} produces an autopartition of complexity $O(n \log n)$.

\textbf{Proof.} (cont’d)

Let $k = \text{dist}_i(s_j)$. Let $s_{j_1}, \ldots, s_{j_k}$ be the segments between $s_i$ & $s_j$.

What is the probability that $\ell(s_i)$ splits $s_j$?

For this to happen, $s_i$ must be first among $\{s_i, s_j, s_{j_1}, \ldots, s_{j_k}\}$.

So $i$ must be the minimum of $\{i, j, j_1, \ldots, j_k\}$.

$\Rightarrow \mathbb{P}[\ell(s_i) \text{ splits } s_j] \leq \frac{1}{\text{dist}_i(s_j)+2}$ \hspace{1cm} (“$\leq$” since a line through some other segment can “save” $s_j$)

$\Rightarrow \mathbb{E}[\# \text{ segments that } s_i \text{ splits}] \leq \sum_{j \neq i} \frac{1}{\text{dist}_i(s_j)+2} \leq 2 \sum_{k=0}^{n-2} \frac{1}{k+2} \leq 2 \ln n$
Algorithm for Computing a BSP in 2d

Algorithm $\text{2DBSP}(S)$

Input. A set $S = \{s_1, s_2, \ldots, s_n\}$ of segments.

Output. A BSP tree for $S$.

1. if $\text{card}(S) \leq 1$
2. then Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set $S$ is stored explicitly.
3. return $\mathcal{T}$
4. else (* Use $\ell(s_1)$ as the splitting line. *)
5. $S^+ \leftarrow \{s \cap \ell(s_1)^+: s \in S\};$ $\mathcal{T}^+ \leftarrow \text{2DBSP}(S^+)$
6. $S^- \leftarrow \{s \cap \ell(s_1)^-: s \in S\};$ $\mathcal{T}^- \leftarrow \text{2DBSP}(S^-)$
7. Create a BSP tree $\mathcal{T}$ with root node $v$, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{s \in S: s \subseteq \ell(s_1)\}$.
8. return $\mathcal{T}$
Algorithm for Computing a BSP in 3d

**Algorithm** $\text{3DBSP}(S)$

*Input.* A set $S = \{t_1, t_2, \ldots, t_n\}$ of triangles in $\mathbb{R}^3$.

*Output.* A BSP tree for $S$.

1. if $\text{card}(S) \leq 1$
2. then Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set $S$ is stored explicitly.
3. return $\mathcal{T}$
4. else (*Use $h(t_1)$ as the splitting plane.*)
5. $S^+ \leftarrow \{t \cap h(t_1)^+ : t \in S\}$; $\mathcal{T}^+ \leftarrow \text{3DBSP}(S^+)$
6. $S^- \leftarrow \{t \cap h(t_1)^- : t \in S\}$; $\mathcal{T}^- \leftarrow \text{3DBSP}(S^-)$
7. Create a BSP tree $\mathcal{T}$ with root node $v$, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{t \in S : t \subset h(t_1)\}$.
8. return $\mathcal{T}$
Algorithm for Computing a BSP in 3d

\begin{algorithm}
\begin{algorithmic}
\State {\bf 3DBSP(S)}
\Input A set $S = \{t_1, t_2, \ldots, t_n\}$ of triangles in $\mathbb{R}^3$.
\Output A BSP tree for $S$.
\State \textbf{if} card($S$) $\leq$ 1
\State \textbf{then} Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set $S$ is stored explicitly.
\State \textbf{return} $\mathcal{T}$
\State \textbf{else} (* Use $h(t_1)$ as the splitting plane. *)
\State $S^+ \leftarrow \{ t \cap h(t_1)^+ : t \in S \}$; \hspace{1em} $\mathcal{T}^+ \leftarrow \text{3DBSP}(S^+)$
\State $S^- \leftarrow \{ t \cap h(t_1)^- : t \in S \}$; \hspace{1em} $\mathcal{T}^- \leftarrow \text{3DBSP}(S^-)$
\State Create a BSP tree $\mathcal{T}$ with root node $v$, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{ t \in S : t \subset h(t_1) \}$.
\State \textbf{return} $\mathcal{T}$
\end{algorithmic}
\end{algorithm}

No similar analysis known...
Useful Splits and Free Splits

cell of a 2d BSP
Useful Splits and Free Splits

cell of a 2d BSP

useful split
Useful Splits and Free Splits

cell of a 2d BSP

useful split

cell of a 3d BSP
Useful Splits and Free Splits

cell of a 2d BSP

useful split

input triangle

cell of a 3d BSP
Useful Splits and Free Splits

Useful split

Free split

Input triangle
A Variant of the 3d Algorithm

**Algorithm** `3dRANDOMBSP2(S)`

*Input.* A set $S = \{t_1, t_2, \ldots, t_n\}$ of triangles in $\mathbb{R}^3$.

*Output.* A BSP tree for $S$.

1. Generate a random permutation $t_1, \ldots, t_n$ of the set $S$.
2. **for** $i \leftarrow 1$ **to** $n$
3. **do** Use $h(t_i)$ to split every cell where the split is useful.
4. **end**
5. **for** $i \leftarrow 1$ **to** $n$
6. **do** Make all possible free splits.
7. **end**

---

**Notes:**

- $t_i$ represents a triangle in the set $S$.
- $h(t_i)$ is a function that determines how to split the cell.
- The algorithm generates a random permutation to ensure diversity in the splits.
- The final step involves making all possible free splits to fully utilize the BSP tree.
A Variant of the 3d Algorithm

Algorithm 3DRandomBsp2(S)

Input. A set $S = \{t_1, t_2, \ldots, t_n\}$ of triangles in $\mathbb{R}^3$.

Output. A BSP tree for $S$.

1. Generate a random permutation $t_1, \ldots, t_n$ of the set $S$.
2. for $i \leftarrow 1$ to $n$
3. do Use $h(t_i)$ to split every cell where the split is useful.
4. Make all possible free splits.

Thm. 3DRandomBsp2 produces an autopartition of complexity $O(n^2)$.
### A Variant of the 3d Algorithm

**Algorithm** $\text{3DRandomBsp2}(S)$

*Input.* A set $S = \{t_1, t_2, \ldots, t_n\}$ of triangles in $\mathbb{R}^3$.

*Output.* A BSP tree for $S$.

1. Generate a random permutation $t_1, \ldots, t_n$ of the set $S$.
2. **for** $i \leftarrow 1$ **to** $n$
3. **do** Use $h(t_i)$ to split every cell where the split is useful.
4. Make all possible free splits.

**Thm.** $\text{3DRandomBsp2}$ produces an autopartition of complexity $O(n^2)$.

**Proof.** See Section 12.4 of the Book [Comp Geom A&A].
Bad examples

**Thm.** There are sets of pairwise disjoint triangles in $\mathbb{R}^3$ for which any autopartition has size $\Omega(n^2)$. 
Bad examples

**Thm.** There are sets of pairwise disjoint triangles in $\mathbb{R}^3$ for which any autopartition has size $\Omega(n^2)$.
Bad examples

**Thm.** There are sets of pairwise disjoint triangles in $\mathbb{R}^3$ for which any autopartition has size $\Omega(n^2)$.

$$\{y = i, z = ix \mid 1 \leq i \leq n/2\} \cup \{x = i, z = iy + \varepsilon \mid 1 \leq i \leq n/2\}, \quad \varepsilon > 0$$
Bad examples

Thm. There are sets of pairwise disjoint triangles in $\mathbb{R}^3$ for which any autopartition has size $\Omega(n^2)$.

$\{y = i, z = ix \mid 1 \leq i \leq n/2\} \cup \{x = i, z = iy + \varepsilon \mid 1 \leq i \leq n/2\}, \varepsilon > 0$
Bad examples

**Thm.** There are sets of pairwise disjoint triangles in $\mathbb{R}^3$ for which any autopartition has size $\Omega(n^2)$.

\[{y = i, z = ix \mid 1 \leq i \leq n/2} \cup \{x = i, z = iy + \varepsilon \mid 1 \leq i \leq n/2\}, \varepsilon > 0\]