Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

MCS 481 Lecture 38
Computational Geometry
Jan Verschelde, 18 April 2025

Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

hidden surface removal

We want to render a scene.

Problem statement:

> Input: a set of objects and a view point.
> Output: a list of visible objects.

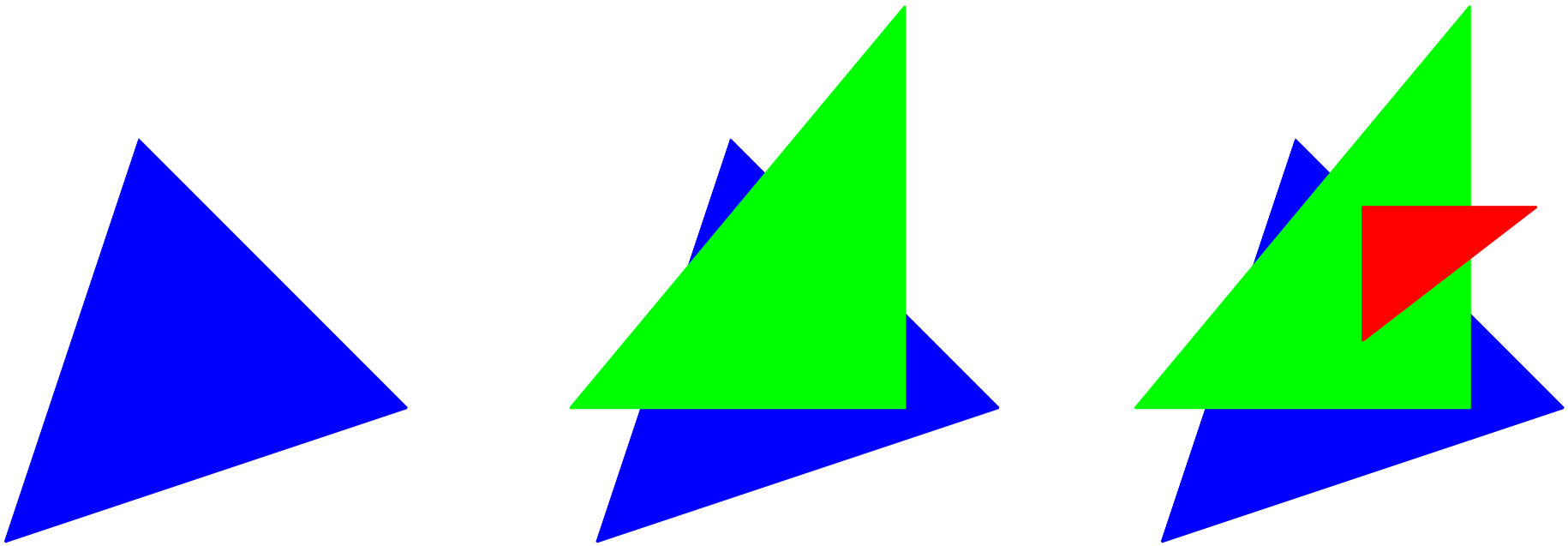
Additional constraint:

the rendering must happen in real time.

Application: rendering landscape, landing strip in a flight simulator.

The removal of nonvisible objects is called *hidden surface removal*.

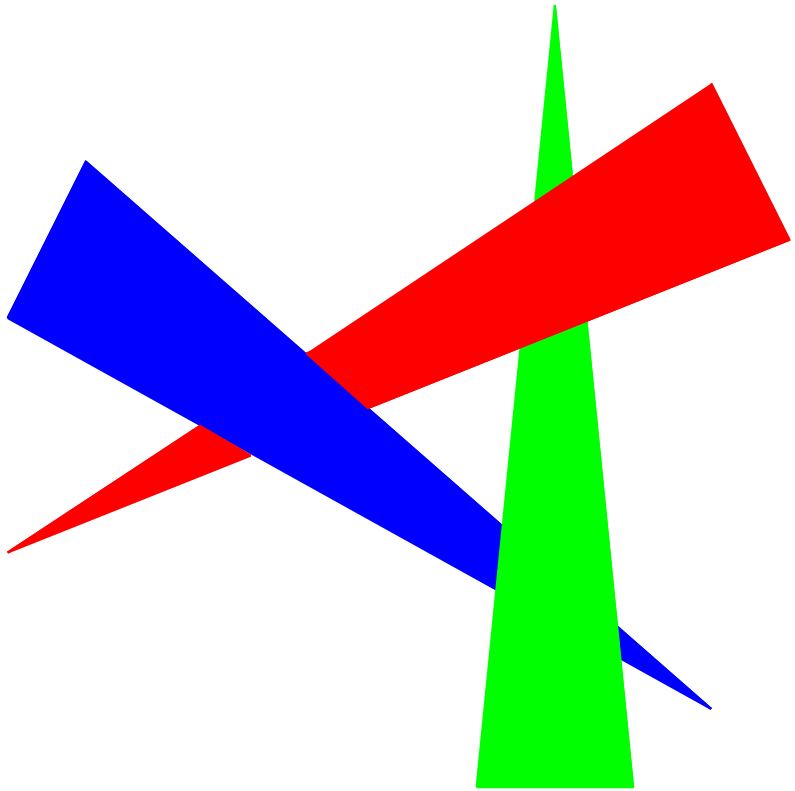
the painter's algorithm



The painter's algorithm:

1. paint first the farthest objects,
2. then overlay the painting with the closer objects.

cyclic overlaps



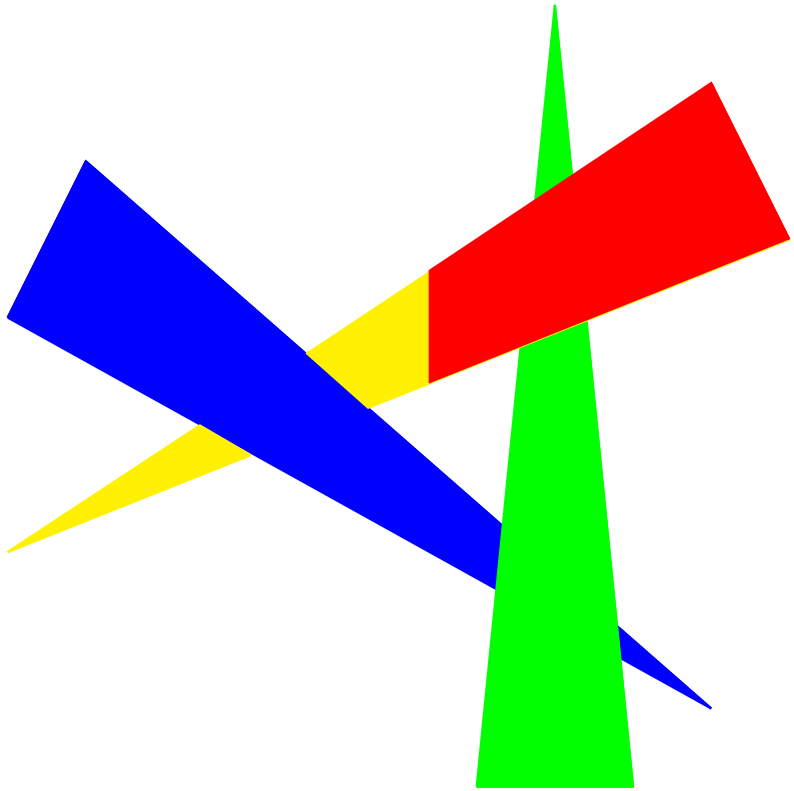
Problem: which object is farthest? Which is closest?

Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

fragmenting objects

Splitting an object resolves the cyclic order.



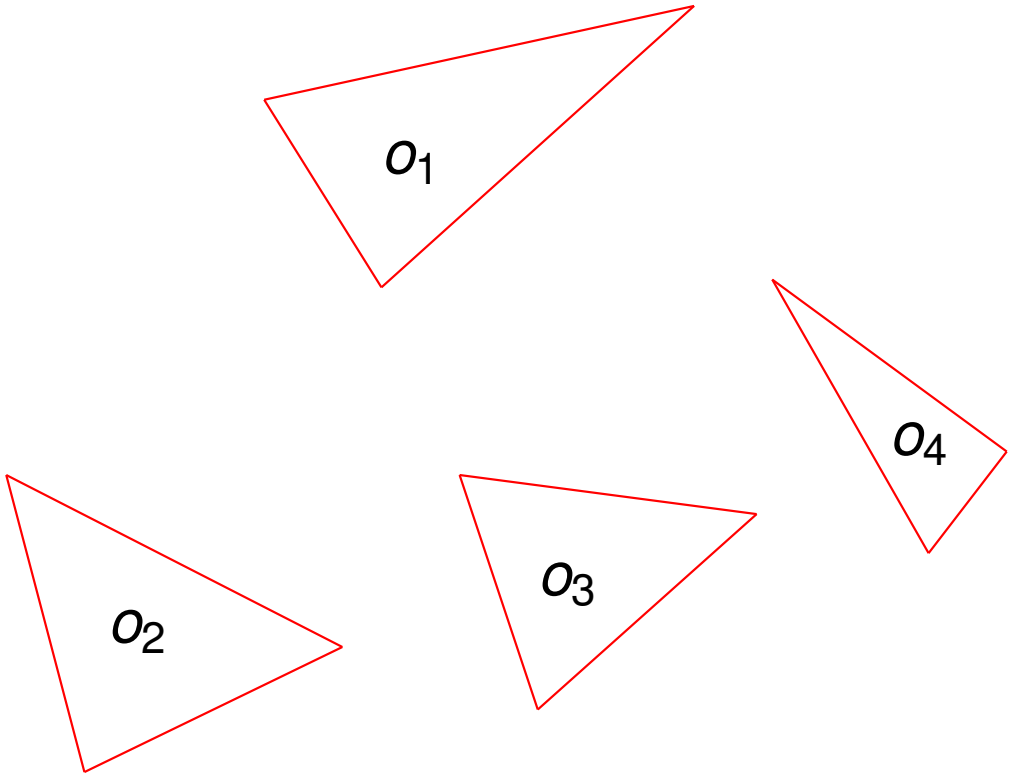
Needed: a data structure to store the fragmented objects.

Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

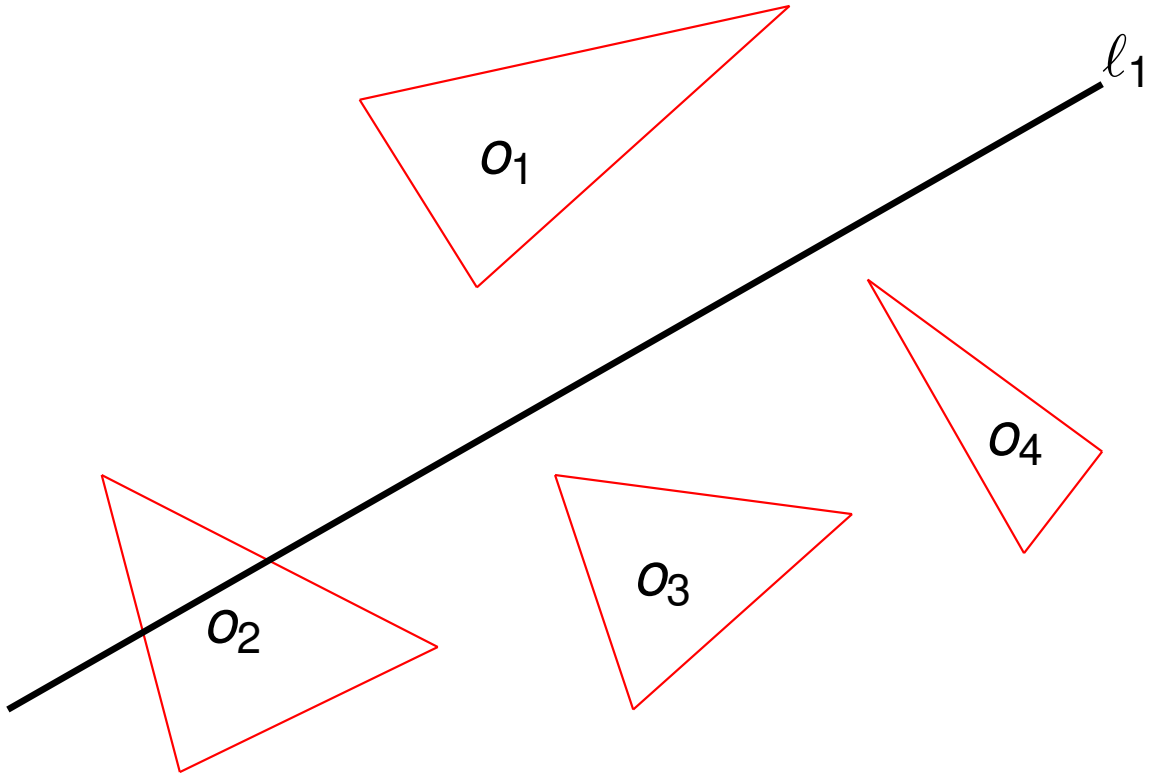
partition four objects

Consider four objects:



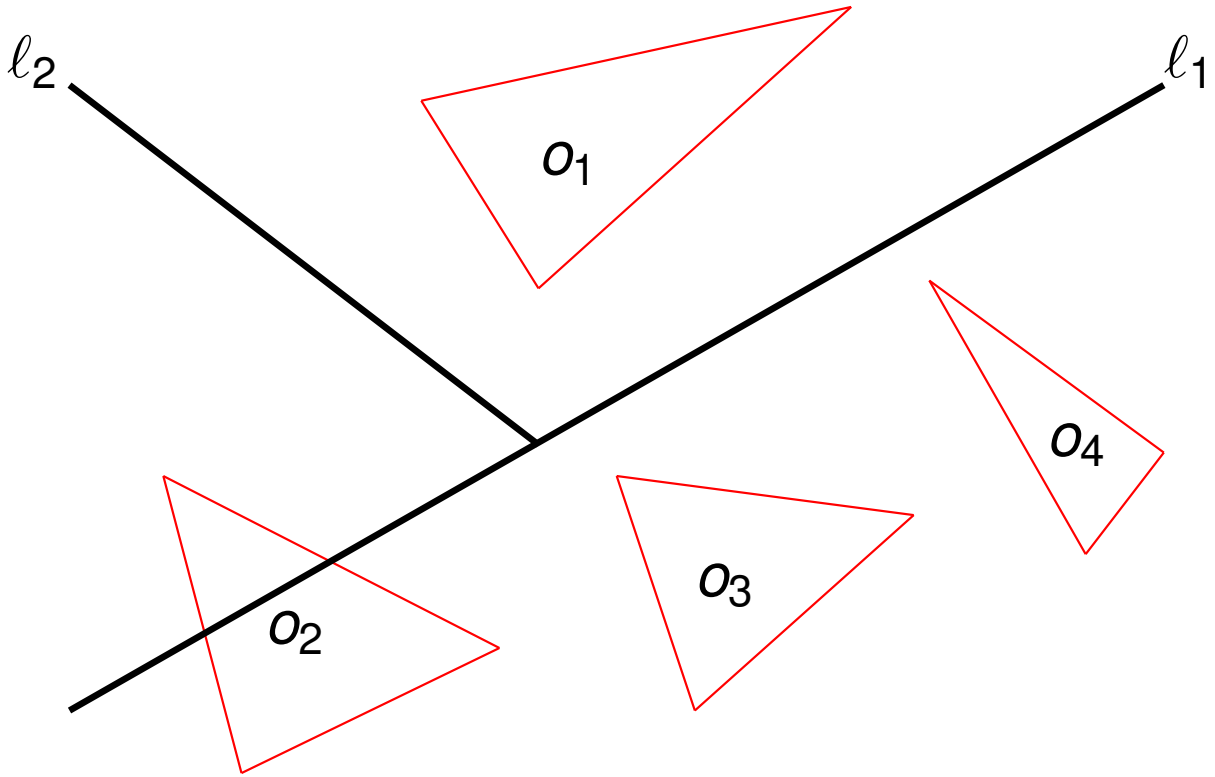
partition four objects

The first split is defined by the line ℓ_1:



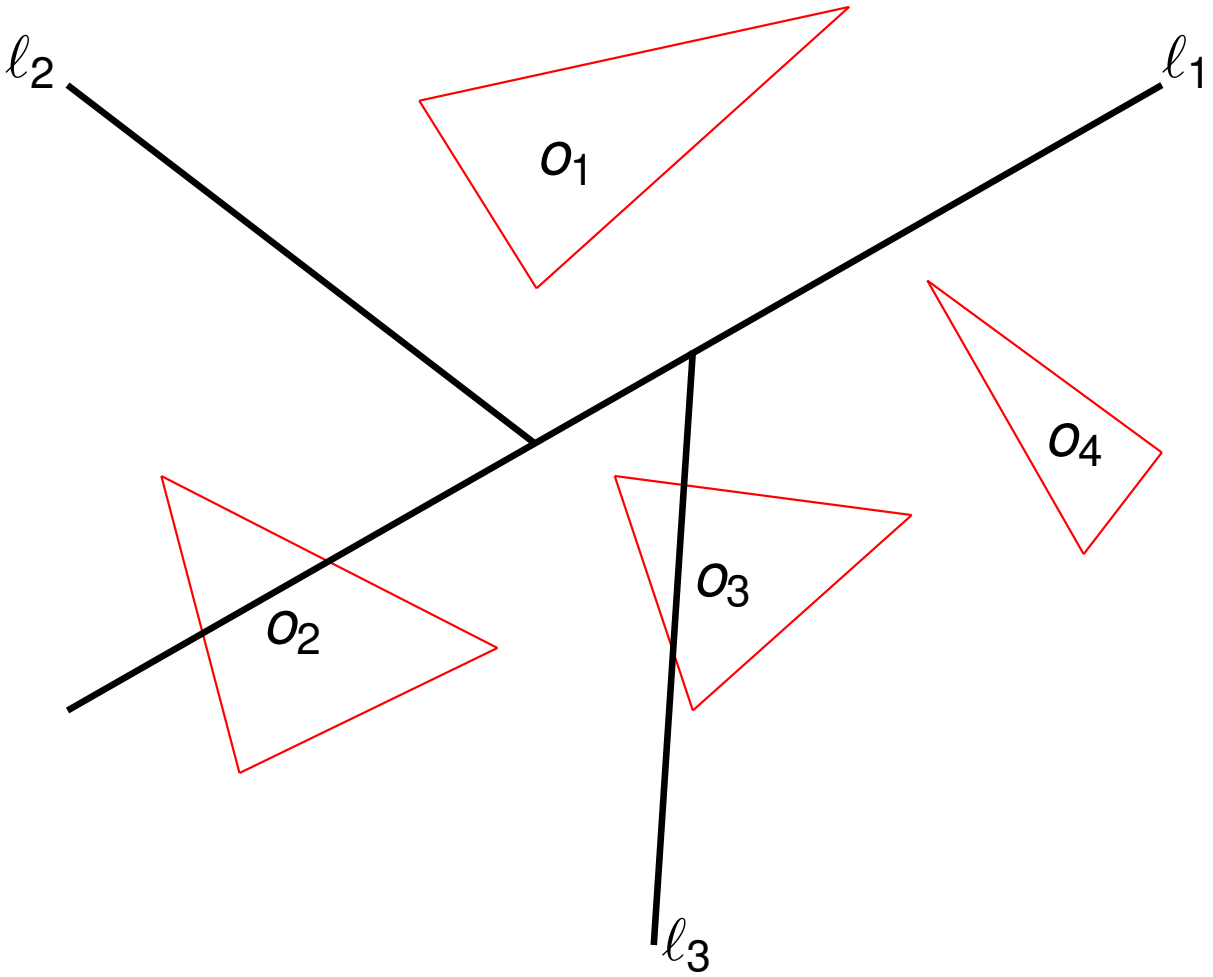
partition four objects

The second split is defined by the half line ℓ_2:



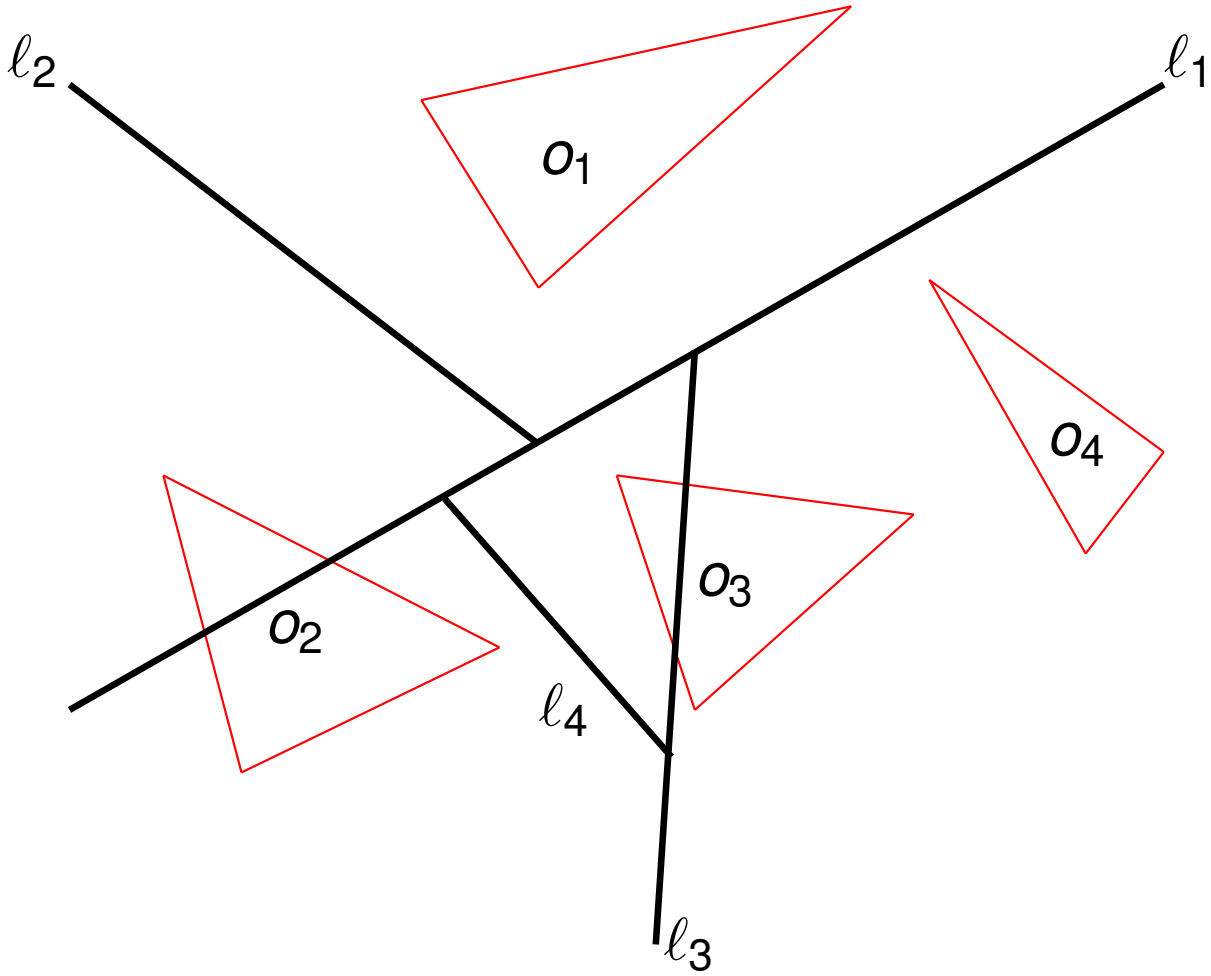
partition four objects

The third split is defined by the half line ℓ_3:



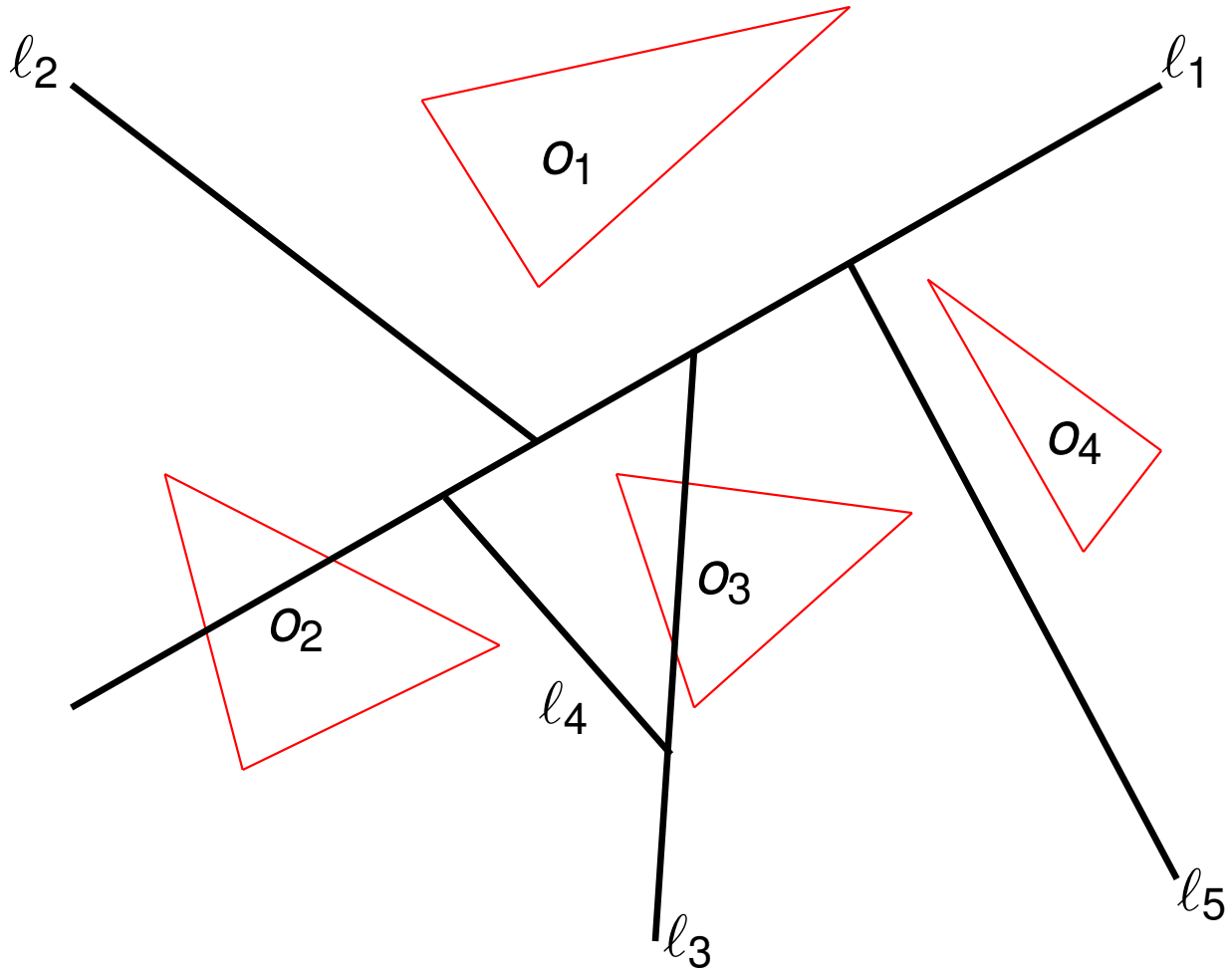
partition four objects

The fourth split is defined by the line segment ℓ_4:

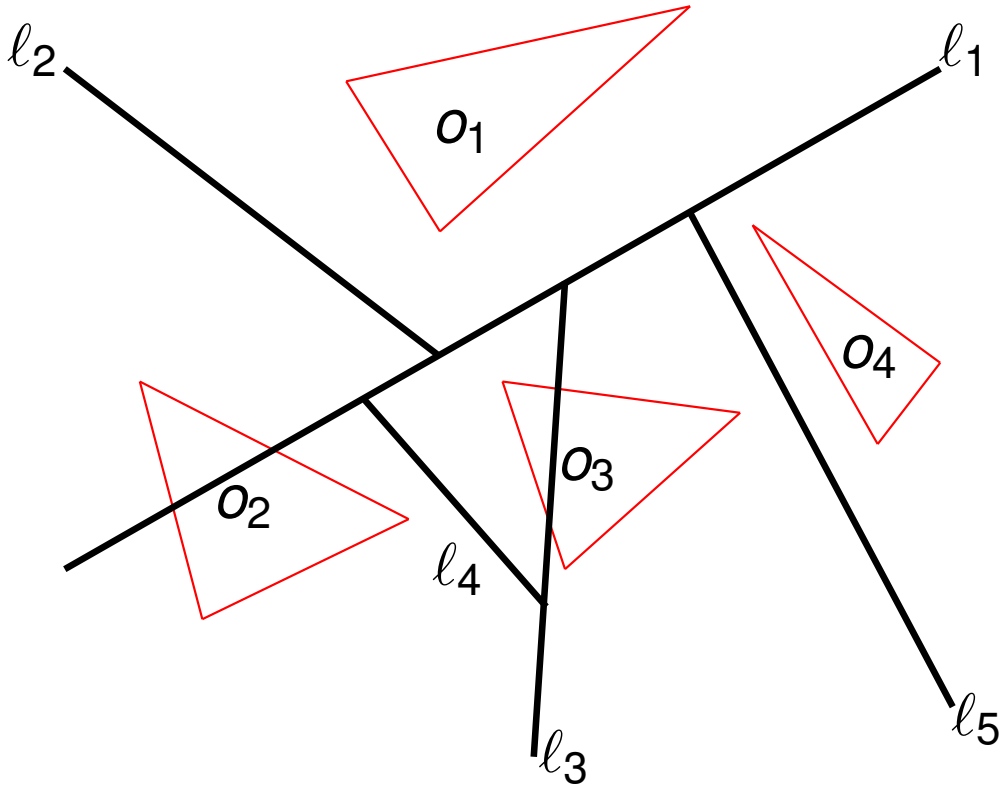


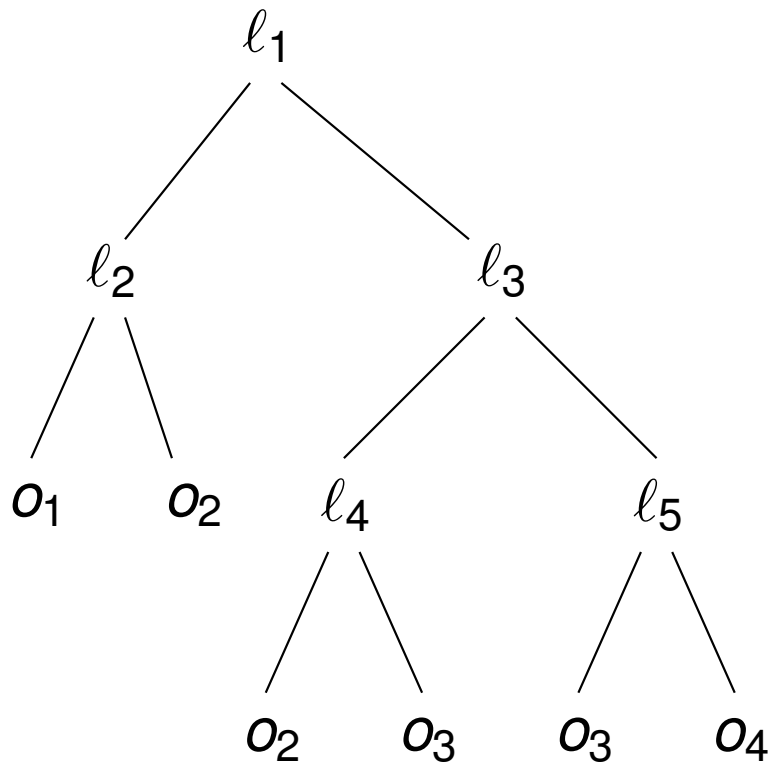
partition four objects

The fifth split is defined by the half line ℓ_5:



the fragments are stored in a tree





A node in the tree is a splitting line, the leaves are the fragments.
The line ℓ is defined by a linear equation $c_0 + c_1 x + c_2 y = 0$:

- the left child stores the negative half plane $c_0 + c_1 x + c_2 y < 0$, and
- the right child stores the positive half plane $c_0 + c_1 x + c_2 y > 0$.

storing a point collection

Let S be a collection of points in 3-space.
The plane h represented by $h(x, y, z) = h_0 + h_1 x + h_2 y + h_3 z$ defines three sets:

1. $S^- = \{ p \in S \mid h(p) < 0 \}$, the negative half plane,
2. $S^h = \{ p \in S \mid h(p) = 0 \}$, the plane,
3. $S^+ = \{ p \in S \mid h(p) < 0 \}$, the positive half plane.

Definition (recursive definition of a binary space partition tree)

For a set S and some random hyperplane h,
the Binary Space Partition tree, or BSP tree, for S is

- the set S as a leaf, if $\#S = 1$, otherwise
- the node stores S^h and
  - the left child is the BSP tree for S^-, and
  - the right child is the BSP tree for S^+.

size of a BSP tree

The running time of any algorithm that constructs and uses a BSP tree depends on the size of the tree.

The more the splitting planes fragment the objects, the larger the size of the BSP tree.

Exercise 1: Consider n nonintersecting line segments in the plane. Show that a BSP tree of size n exists.

An *auto-partition* uses the extension of the line segments to split.

What is the size of a BSP tree of line segments defined by an auto-partition?

Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

construction of a BSP tree

Algorithm CONSTRUCTBSPTREE(S)

Input: $S = \{ t_1, t_2, \ldots, t_n \}$, a set of triangles.
Output: the root of a BSP tree for S.

Define the plane h so h passes through some triangle $t \in S$.

With this choice of h,

- $S \cap h^-$ are all triangles of S,
  intersected with the negative half space h^-,
- $S \cap h$ is the triangle t, and all other triangles of S in h,
- $S \cap h^+$ are all triangles of S,
  intersected with the positive half space h^+.

Quadrilaterals in h^- and h^+ need to be triangulated.

Exercise 2: Adjust the example on slide 15 to triangulate the quadrilaterals arising after the splits.

the algorithm ConstructBSPTree

Algorithm ConstructBSPTree(S)

Input: $S = \{ t_1, t_2, \ldots, t_n \}$, a set of triangles.
Output: the root of a BSP tree for S.

1. if $\#S \leq 1$ then
2. return ConstructLeaf(S)

else

3. let h be the plane through the first triangle t_1 in S
4. $S^- = \{ t \cap h^- \mid t \in S \}$, $S^+ = \{ t \cap h^+ \mid t \in S \}$
5. $S^h = \{ t \subset h \mid t \in S \}$, $v =$ ConstructNode(S^h)
6. LeftChild(v) = ConstructBSPTree(S^-)
7. RightChild(v) = ConstructBSPTree(S^+)
8. return v

special cases

The triangles are the facets of the polyhedral objects in 3-space.

For inputs in general position, no two triangles are coplanar, which implies that $\#S^h = 1$, only one triangle at every node.

For facets that are not triangles, or in the special case that all triangles in S lie in the same plane, then we need a Binary *Plane* Partition tree.

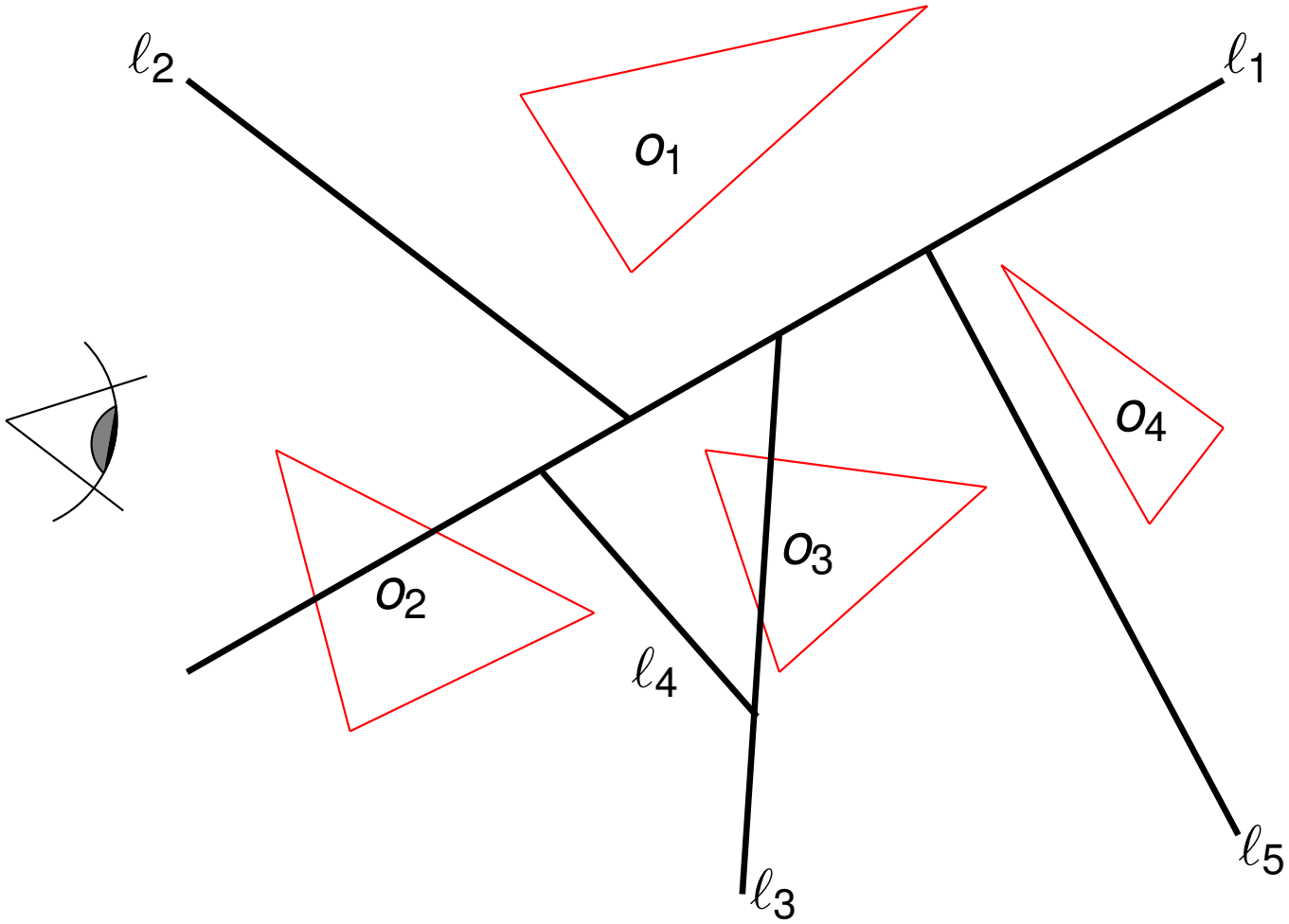
In the plane, the partitioning is done with splitting lines. Then each splitting line passes through an edge of a triangle.

Is there a special case to this case?

Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

ordering the objects with a view point



recursive tree traversal

The painter paints the farthest objects first.

We traverse the BSP tree as follows:

1. visit all nodes at the other half space as the view point,
2. visit the objects stored at the node,
3. visit all notes at the same half space as the view point.

This leads to a classic recursive tree traversal algorithm.

the painter's algorithm

Algorithm PAINTERSALGORITHM(T, p_{view})

Input: T is a BSP tree, p_{view} is a view point.
Output: objects in T listed in the view point order.

1. if ISLEAF(T) then
2. visit all fragments at the leaf

else

3. let h be the hyperplane at the node T
4. if $p_{\text{view}} \in h^+$ then
5. PAINTERSALGORITHM(LEFTCHILD(T), p_{view})
6. visit all fragments at the node
7. PAINTERSALGORITHM(RIGHTCHILD(T), p_{view})

PAINTERSALGORITHM continued

8. if $p_{\text{view}} \in h^-$ then
9. PAINTERSALGORITHM(RIGHTCHILD(T), p_{view})
10. visit all fragments at the node
11. PAINTERSALGORITHM(LEFTCHILD(T), p_{view})

else (we have: $p_{\text{view}} \in h$)

12. PAINTERSALGORITHM(RIGHTCHILD(T), p_{view})
13. PAINTERSALGORITHM(LEFTCHILD(T), p_{view})

If $p_{\text{view}} \in h$, then the objects at the node are not visible.

Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

performance of the BSP tree

The performance of the BSP tree is determined by its size.

- The size of the BSP tree depends on the number of fragments.
- The fragmentation is set by the choice of the splitting planes.

To limit the size of the BSP tree, we want to minimize the fragmentation.

Randomization of the input avoids the worst possible case.

Theorem (size of a BSP tree)

For a set of n nonintersecting triangles in 3-space,
there is a BSP tree of size $O(n^2)$.
There are inputs for which the size of any BSP tree is $\Omega(n^2)$.

For low density inputs, the size is bounded by $O(n \log(n))$.

Binary Space Partitions

1. Problem Statement
   - hidden surface removal
   - fragmenting objects
2. Binary Space Partition Trees
   - an example and definition
   - construction of a BSP tree
   - the painter's algorithm
3. Cost Analysis
   - performance of the BSP tree
   - generalization of the kd tree

generalization of the kd tree — summary

The Binary Space Partition tree is a geometric data structure, which generalizes the kd tree.

In a kd tree, the half planes are either horizontal or vertical, so we may view the BSP tree as a generalization of the kd tree.

The cost analysis computes the expected size of a BSP tree.

While the quadratic bounds may be discouraging, for low density inputs, the size of a BSP tree is $O(n \log(n))$.

BSP trees are applied in computer graphics.
Related to BSP trees are the *AABB trees*, available in CGAL.
AABB abbreviates Axis-Aligned Bounding Box.