Presentation writer: Tigran Hayrapetyan

Lecturer | Programmer | Researcher

www.linkedin.com/in/tigran-hayrapetyan-cs/

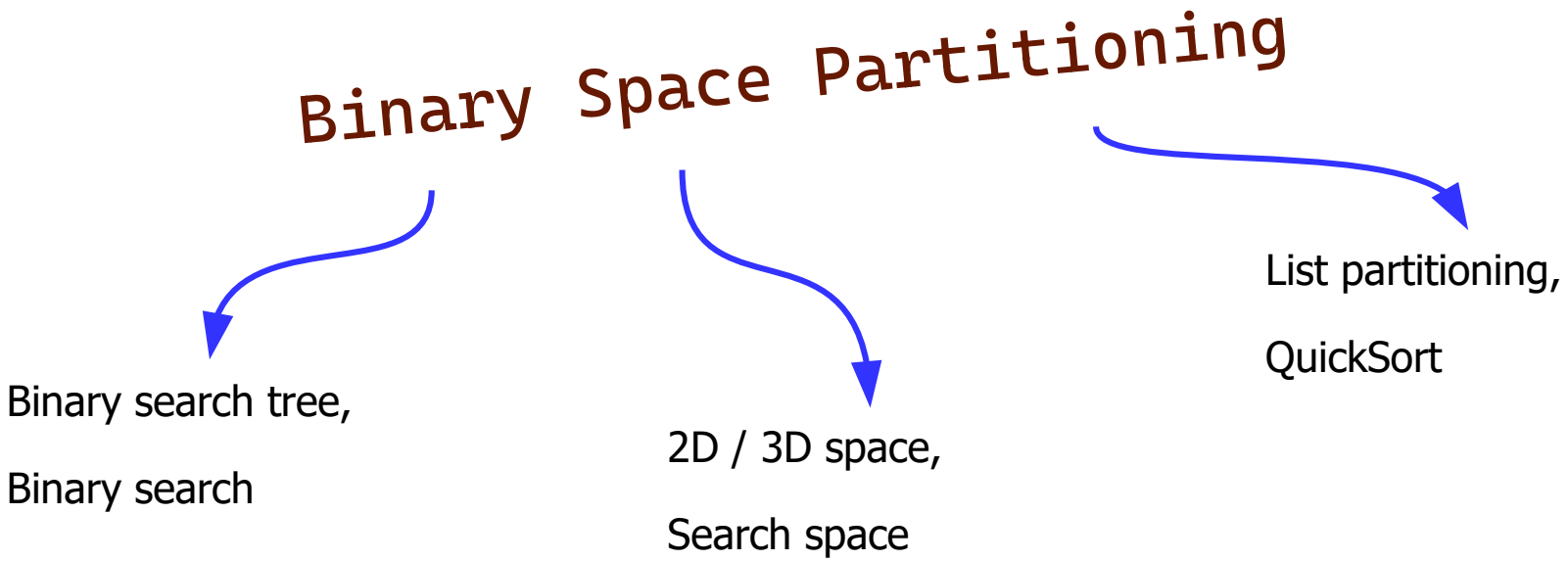
Binary Space Partitioning (BSP)

prerequisites:

- Binary search tree.

Binary Space Partitioning

The naming "Binary Space Partitioning" might look familiar to us, as every word is known:



... but in fact it represents a novel data structure.

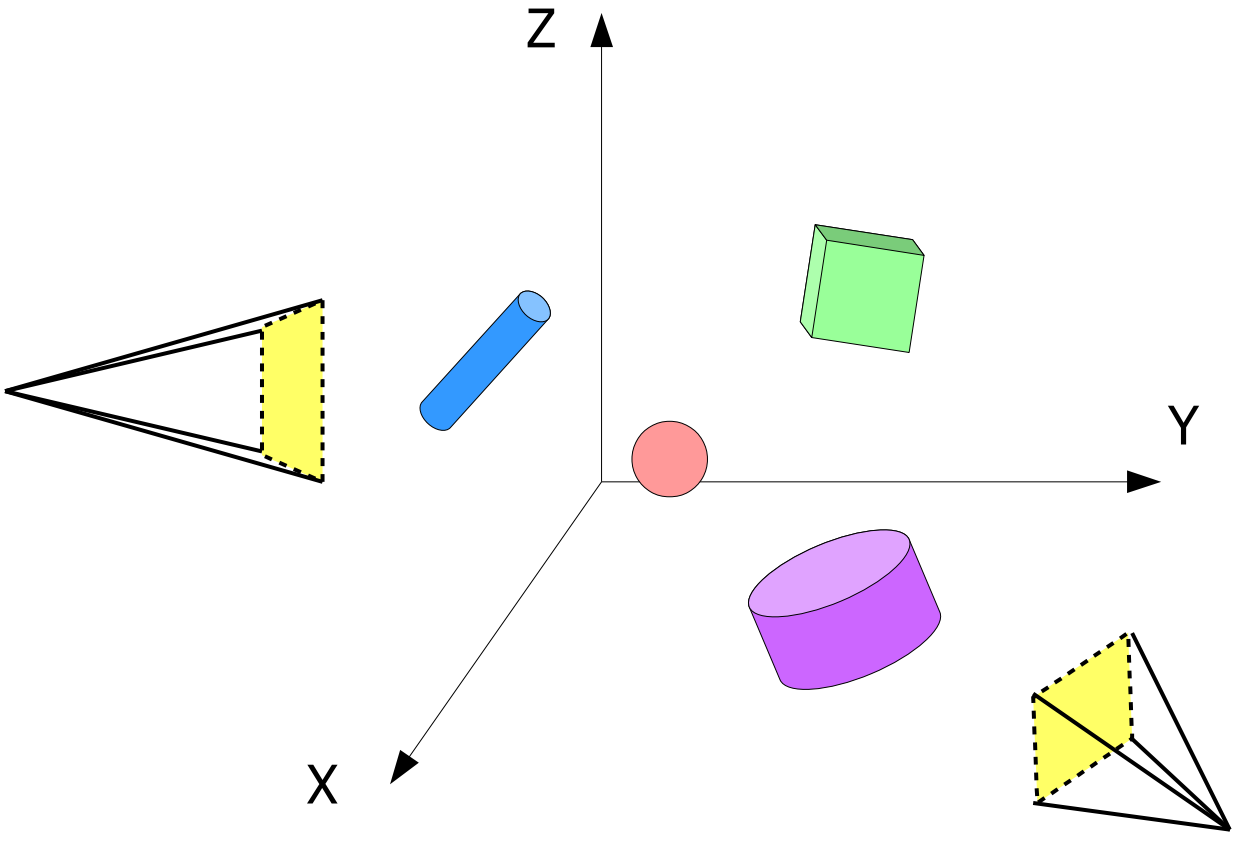
Binary Space Partitioning

It is a multidimensional data structure, and a notable fact is that it works in completely same way, whether we are in **2D** or **3D**.

One usage of BSP comes from Computer graphics, where we:

- have a scene of objects,
- a viewer located at some point, and
- need to render corresponding image.

Binary Space Partitioning



Different viewpoints
-->
different images.

So we <u>need to consider</u>:

- viewer's position,
- angle of view,
- properties of the camera.

Binary Space Partitioning

CG engines like **DirectX** and **OpenGL** work in that way:

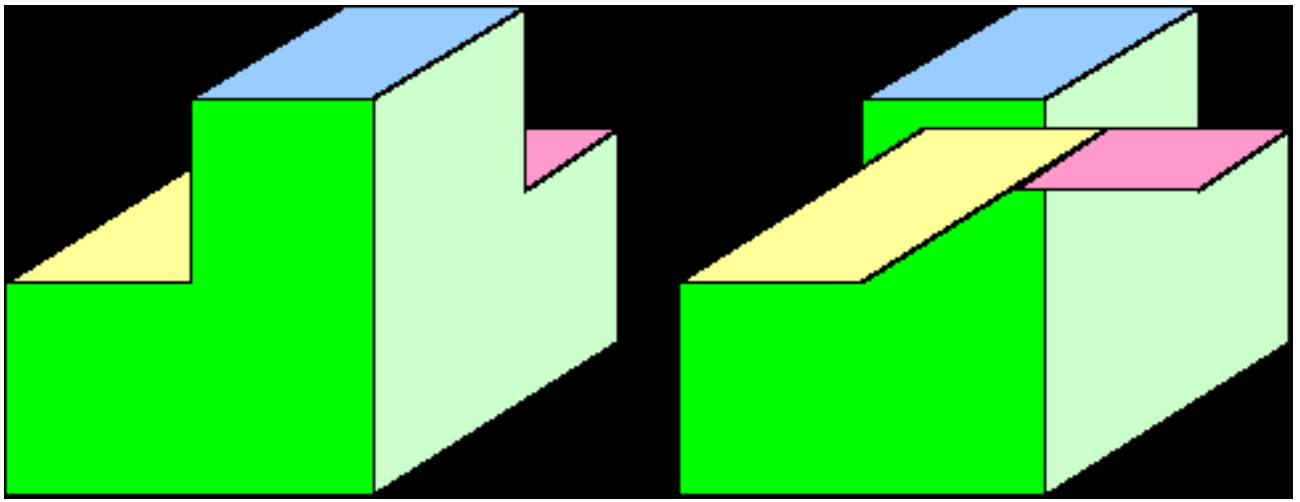
- At first the scene is created and filled with objects,
- Then viewer's coordinates and other properties are provided.

Important fact is that all the objects from scene are drawn more or less independent,

... i.e. rendering is done separately, for every object.

Complete scene rendering

Now if just drawing all the objects of the scene in an independent way, we might get a wrong picture.

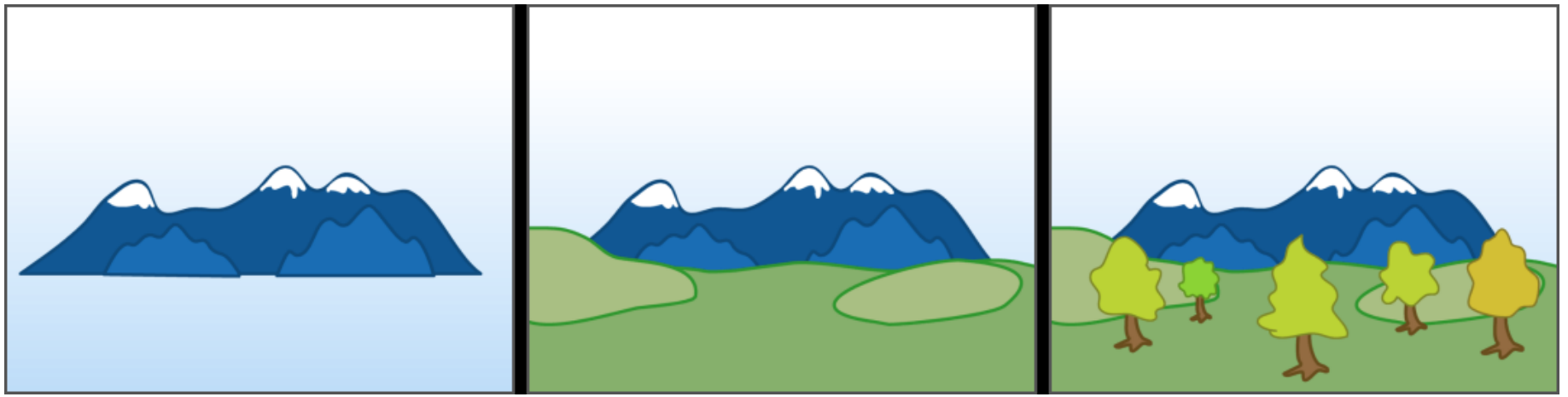


... because the objects which are far might be drawn after the near objects are already rendered.

Complete scene rendering

One solution to this is to initially sort all the objects from farthest one to the closest one, and draw in that order,

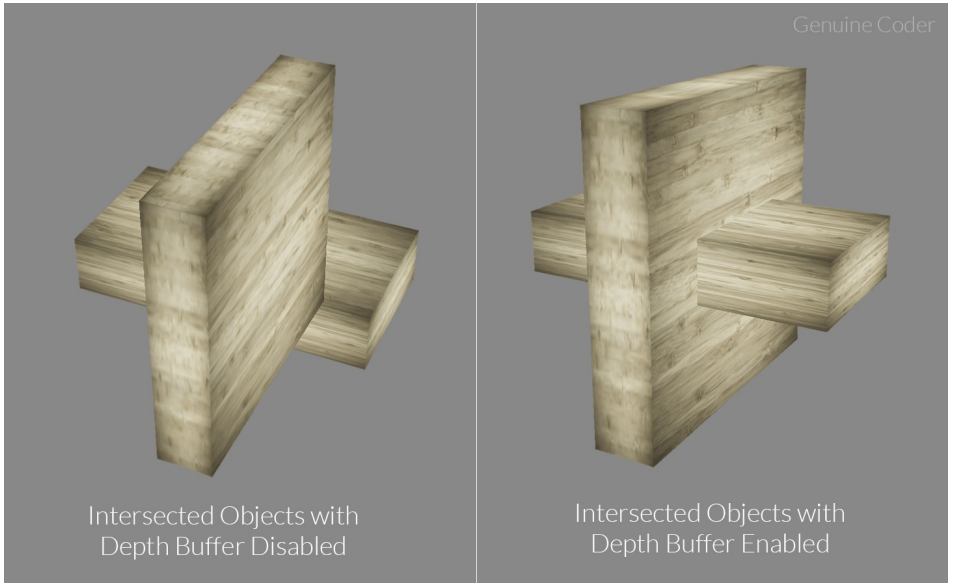
... this is called Painter's algorithm.



... and can be useful for multiple cases.

Complete scene rendering

... but not for all cases:



... as sometimes objects intersect, and must be drawn partially.

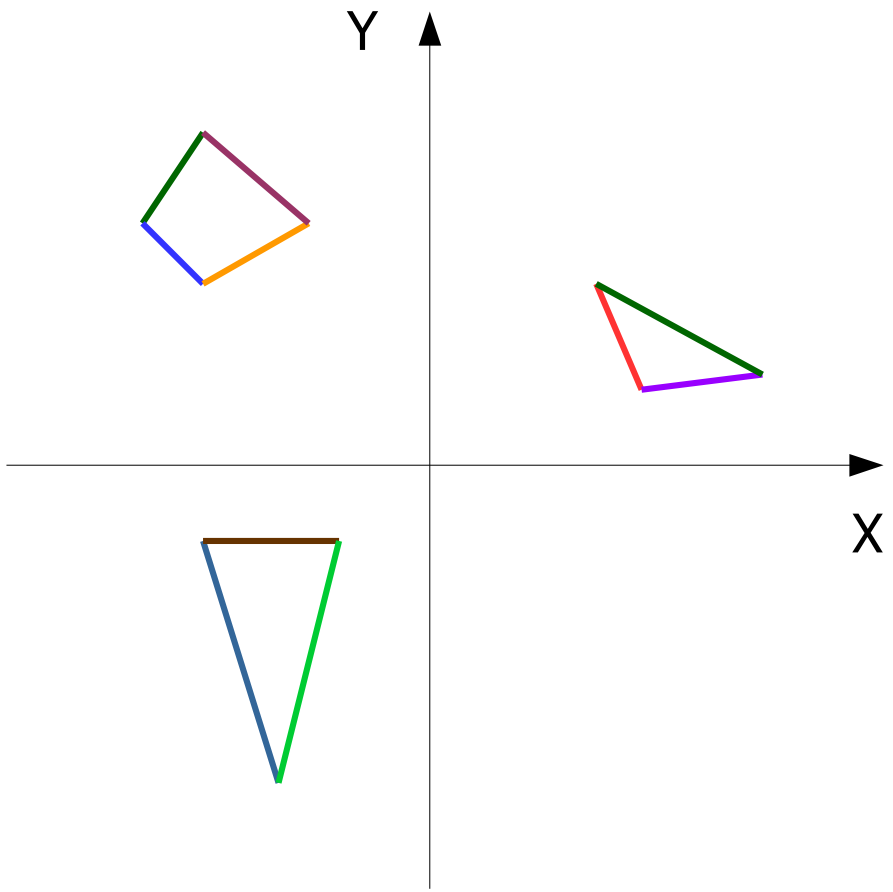
Complete scene rendering

We will study how BSP solves complete scene rendering for the 2D case,

- for simplicity of the geometry, and
- for simplicity of this presentation.

It generalizes to 3D without major modifications.

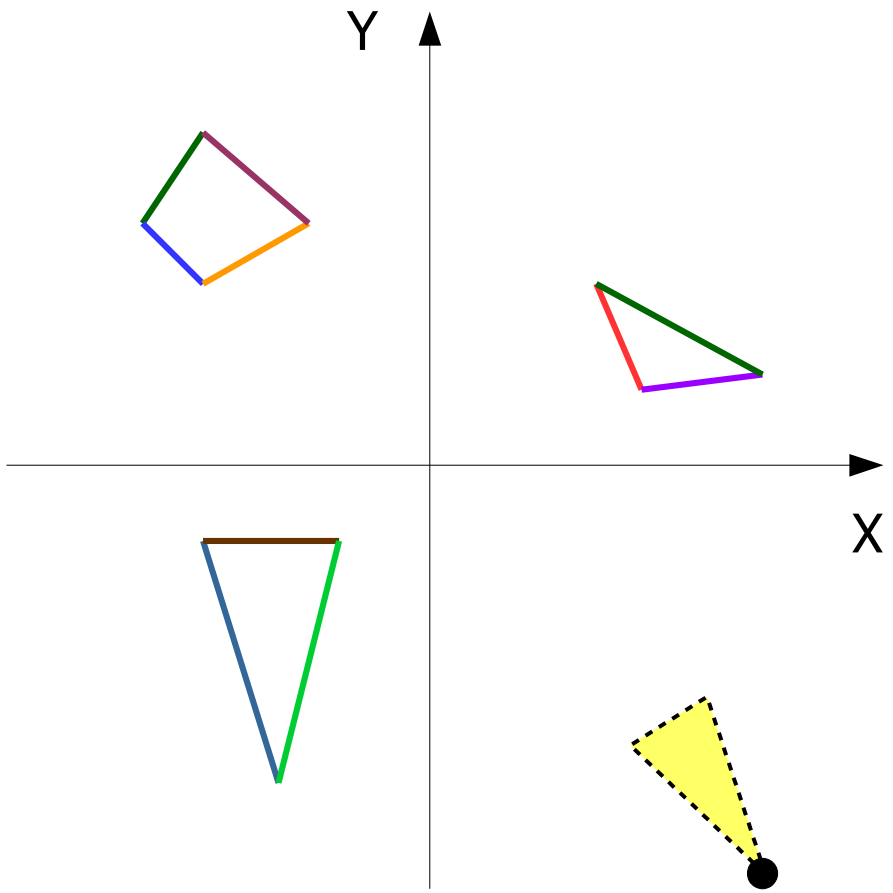
Complete scene rendering



If we are on a plane, the problem turns into this:

- all the objects are represented as sets of segments,

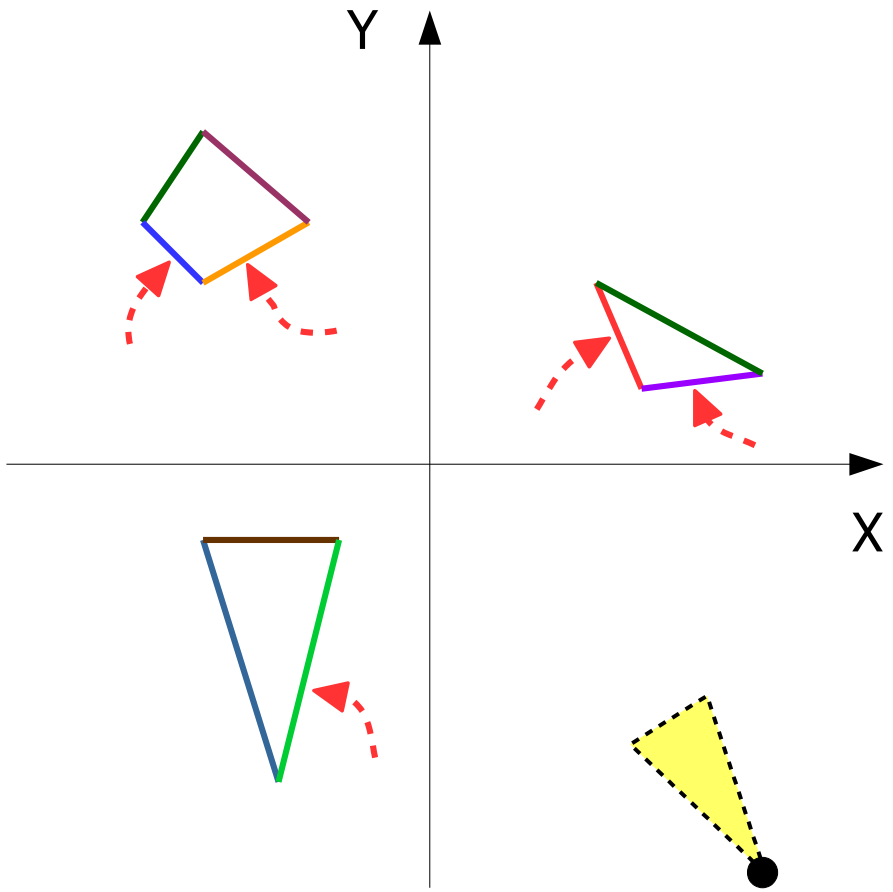
Complete scene rendering



If we are on a plane, the problem turns into this:

- all the objects are represented as sets of segments,
- the viewer is located at some point, and has some direction,

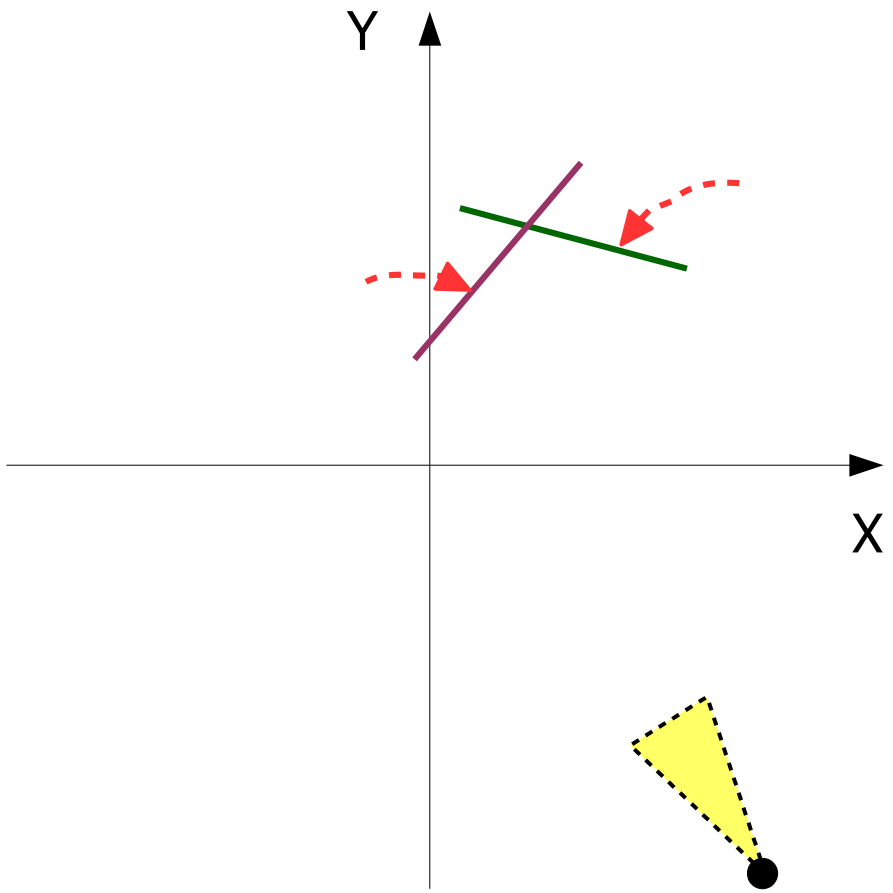
Complete scene rendering



If we are on a plane, the problem turns into this:

- all the objects are represented as sets of segments,
- the viewer is located at some point, and has some direction,
- we need to figure out the segments (or their parts) which will be visible.

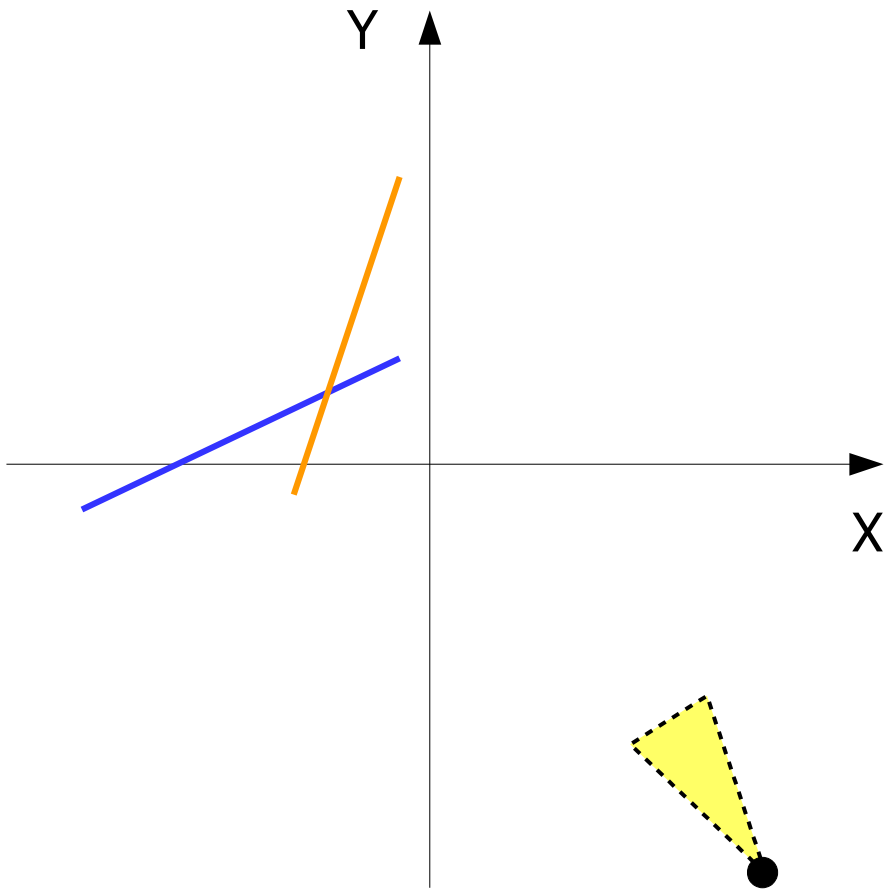
Complete scene rendering



Note, segments still can intersect.

... and the analogy with the problematic 3D case will look like this:

Complete scene rendering

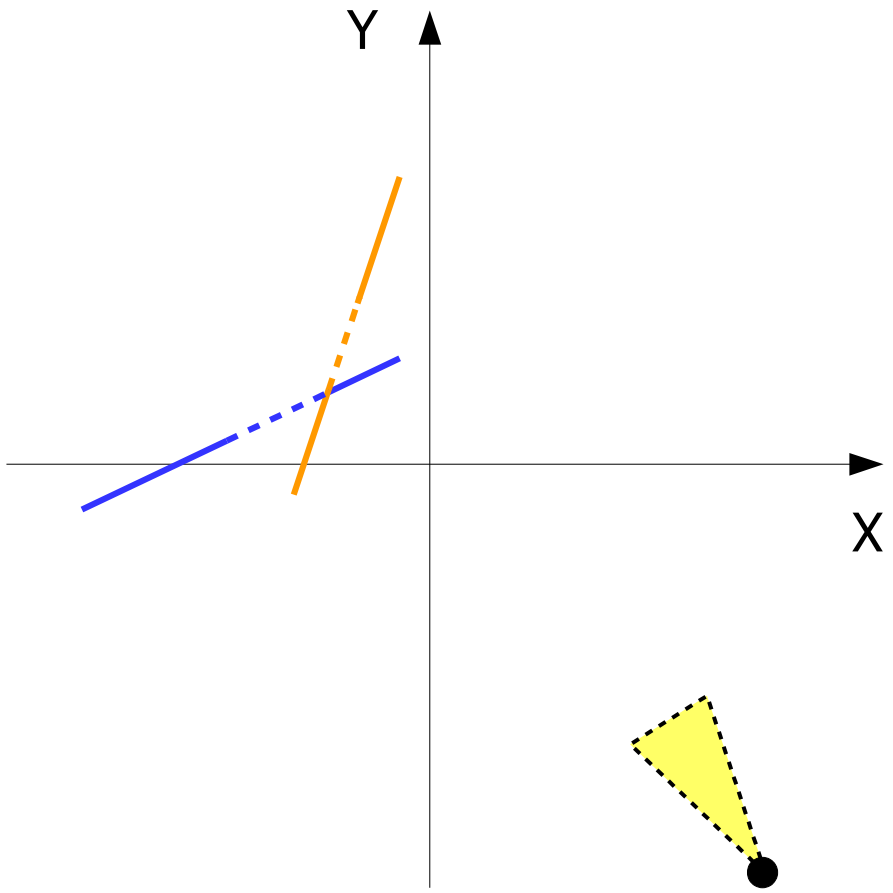


Note, segments still <u>can intersect</u>.

... and the analogy with the problematic 3D case will look like this:

... or like this, when both segments must be drawn in **2** parts:

Complete scene rendering



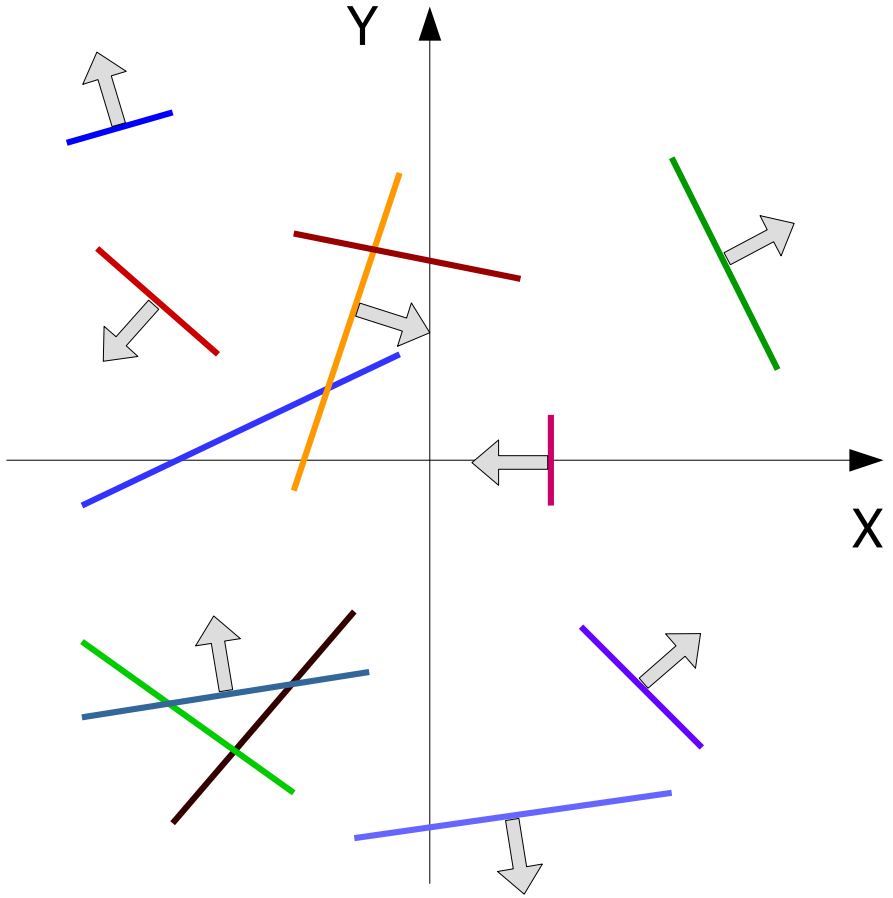
Note, segments still can intersect.

... and the analogy with the problematic 3D case will look like this:

... or like this, when both segments must be drawn in **2** parts:

So just drawing one segment after another will result in wrong picture.

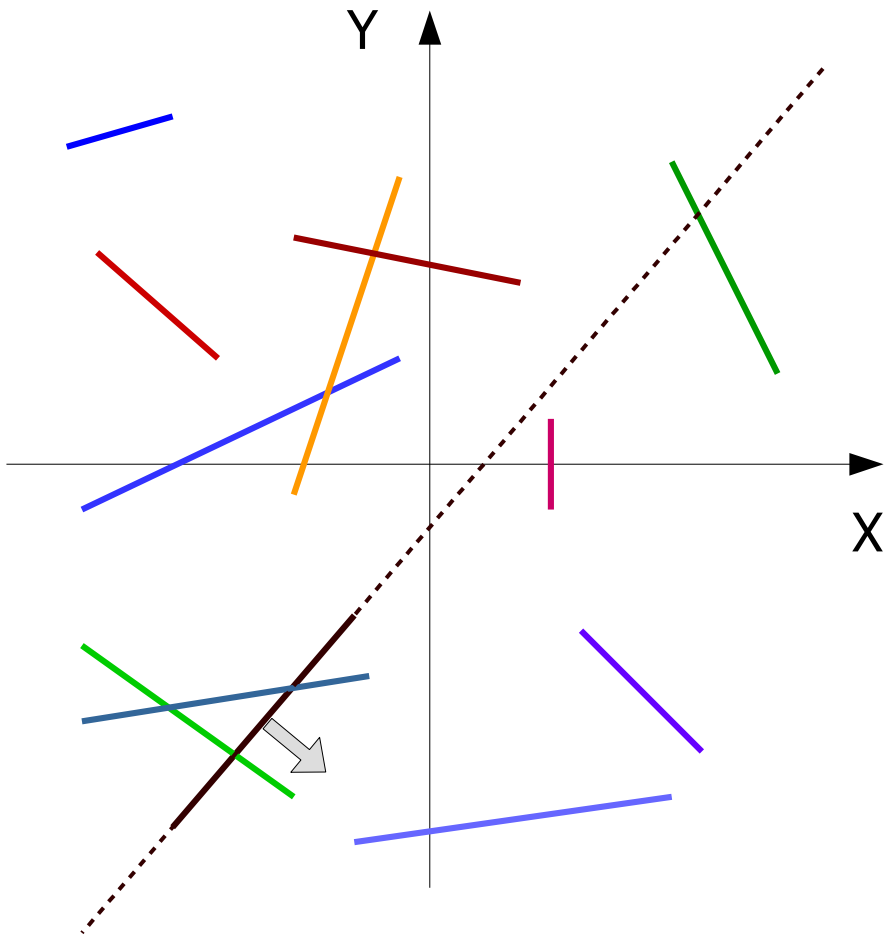
Complete scene rendering



For BSP algorithm to work, every object (segment) must be assigned with front and back faces.

... it doesn't matter at all how will we assign them, as we just need to differentiate.

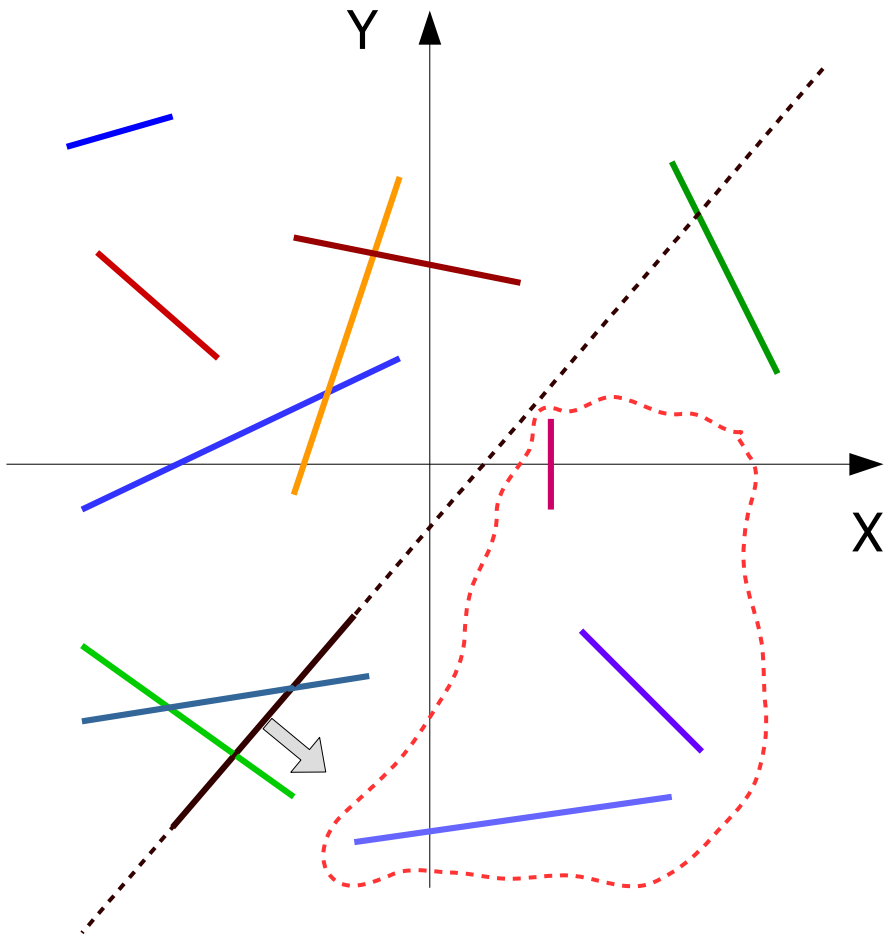
Complete scene rendering



Concept of BSP is to partition all the segments into **2** sets:

- taking one segment as a pivot,

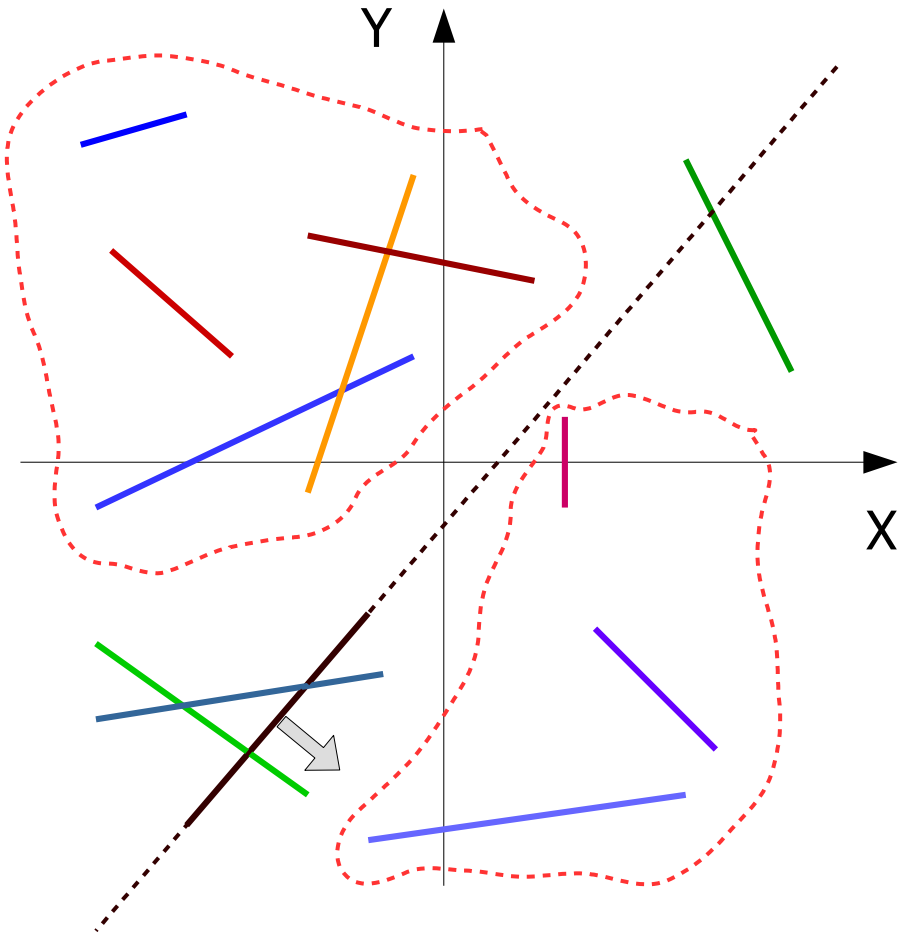
Complete scene rendering



Concept of BSP is to partition all the segments into **2** sets:

- taking one segment as a pivot,
- understand which other segments lie in front of it,

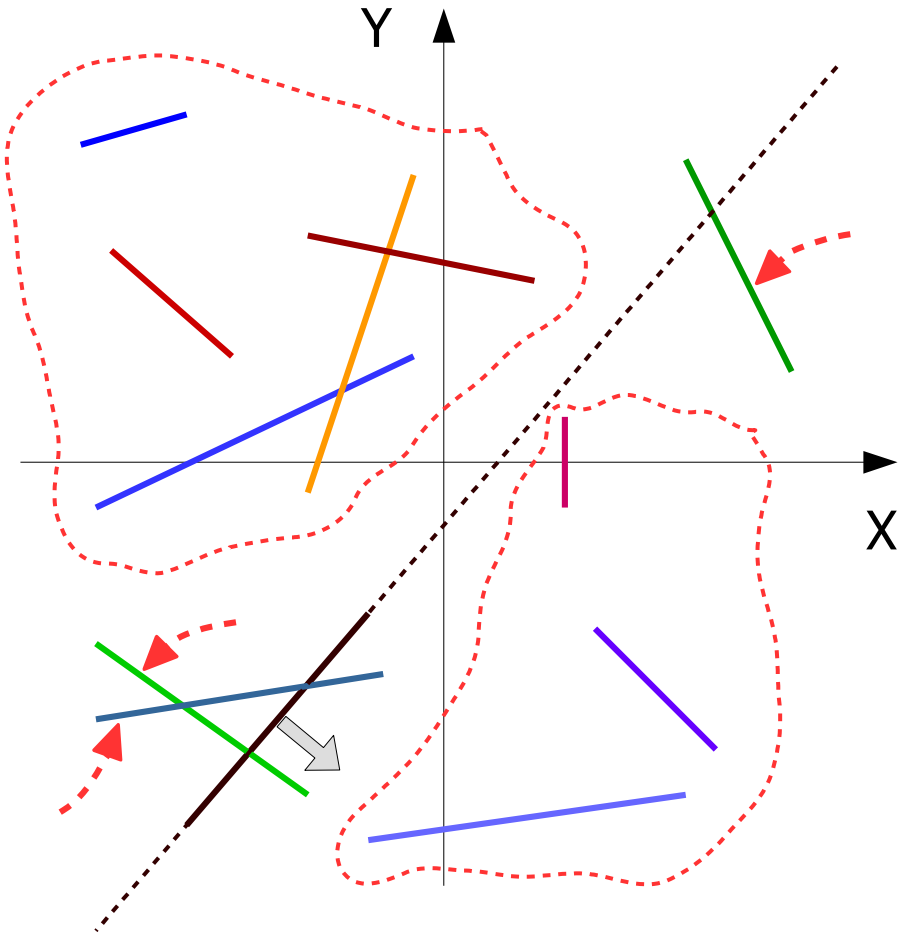
Complete scene rendering



Concept of BSP is to partition all the segments into **2** sets:

- taking one segment as a pivot,
- understand which other segments lie in front of it,
- which ones lie beneath it,

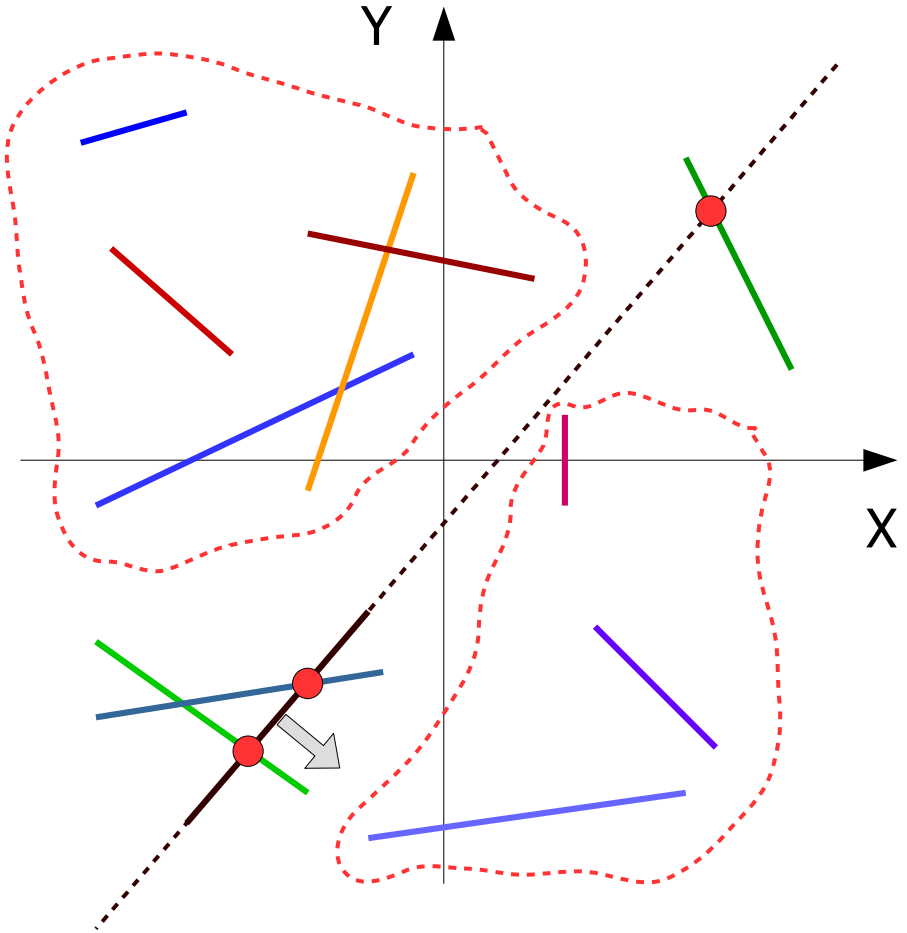
Complete scene rendering



Concept of BSP is to partition all the segments into **2** sets:

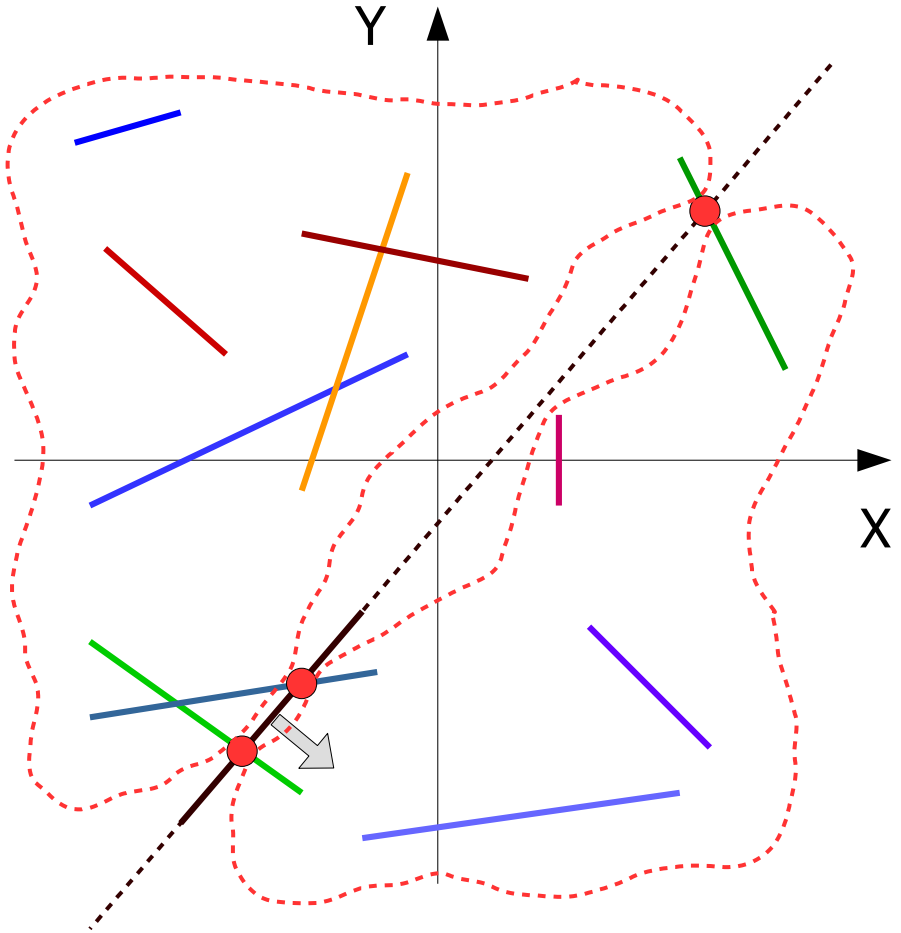
- taking one segment as a pivot,
- understand which other segments lie in front of it,
- which ones lie beneath it,
- and which ones lie on both half-planes.

Complete scene rendering



The segments which intersect partition plane will be cut in **2** independent parts,

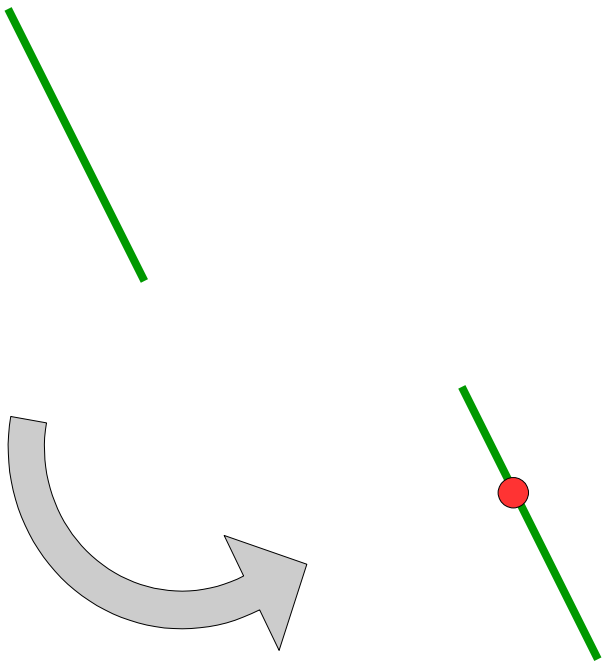
Complete scene rendering



The segments which intersect partition plane will be cut in **2** independent parts,

... and each part will be assumed to belong to one of the half-plane.

Complete scene rendering

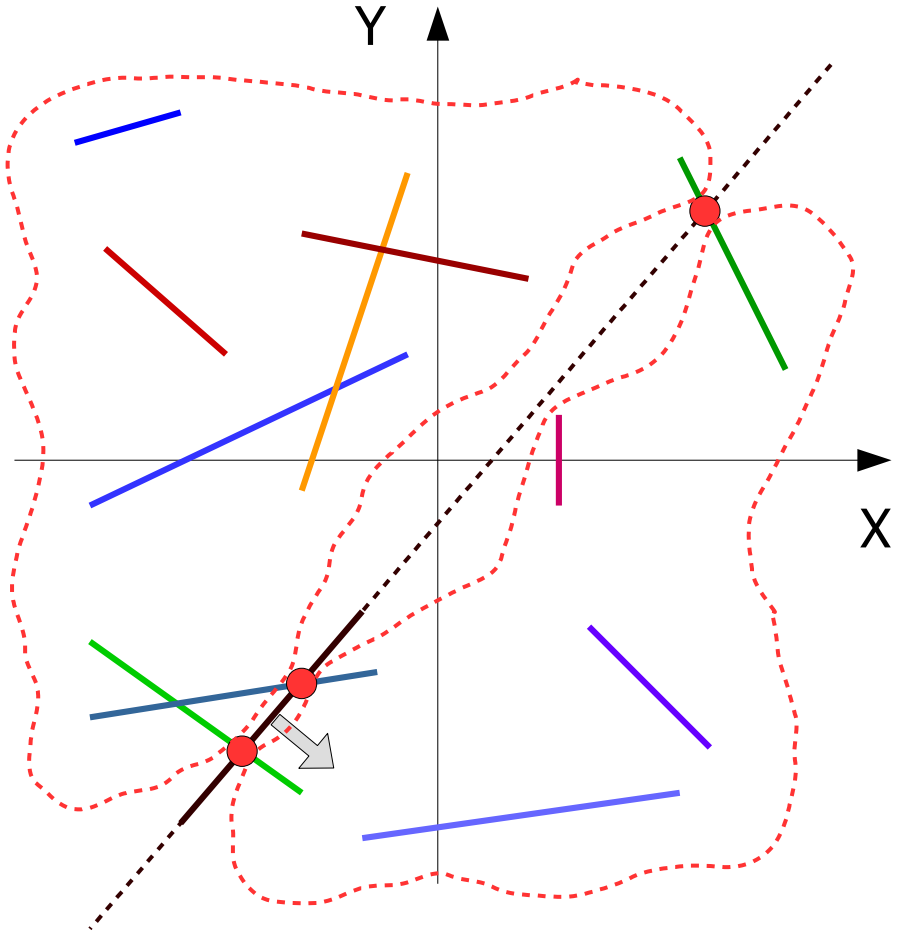


The segments which intersect partition plane will be cut in **2** independent parts,

... and each part will be assumed to belong to one of the half-plane.

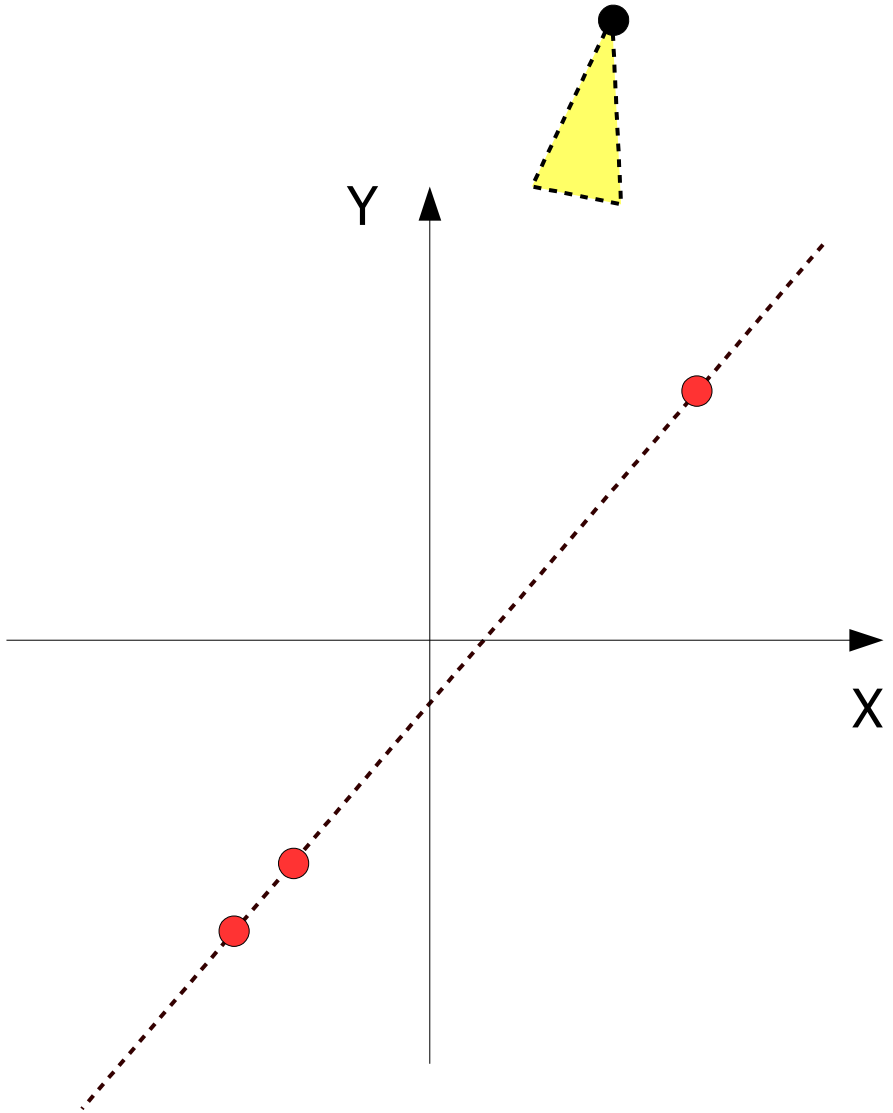
Important note: from rendering perspective it doesn't matter if to draw **1** segment, or **2** properly glued segments.

Complete scene rendering



Also let's note that after such cuts, amount of overall objects increases.

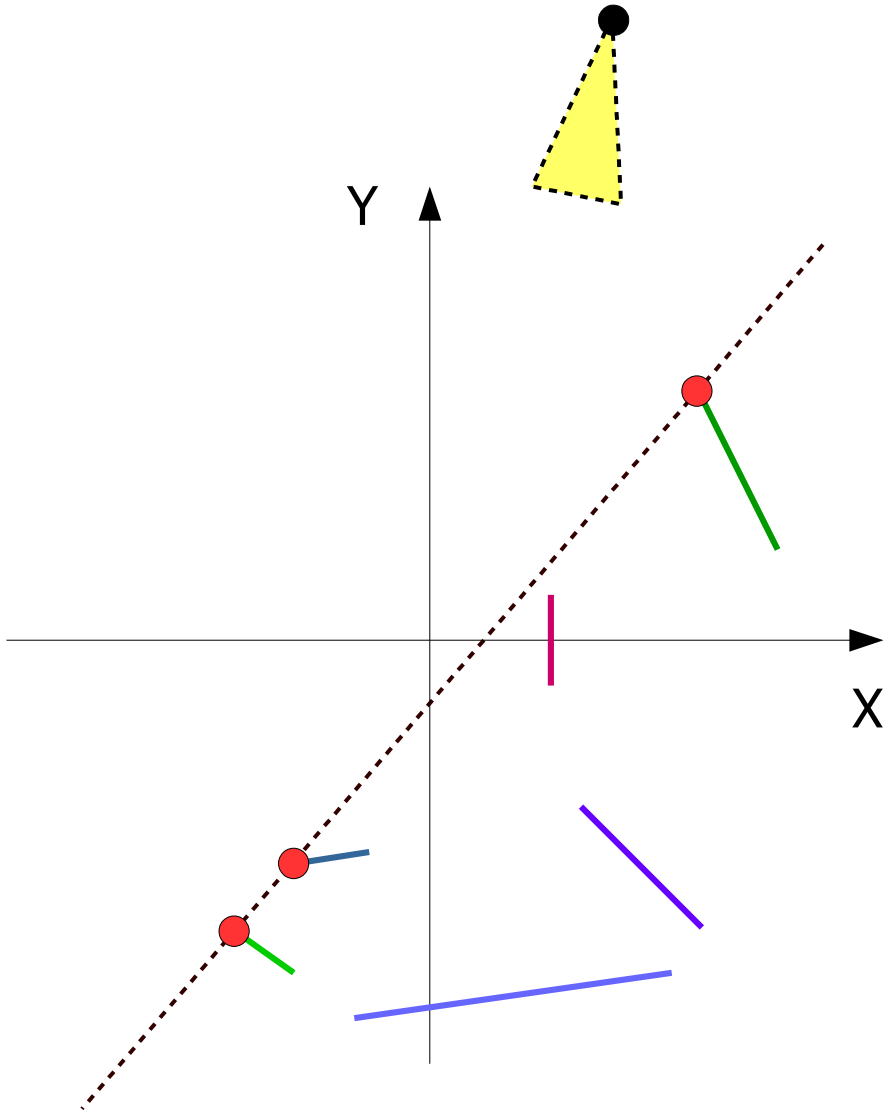
Complete scene rendering



Having such kind of separation, how will we draw necessary objects on the screen?

Considering that viewer is located at one part,

Complete scene rendering

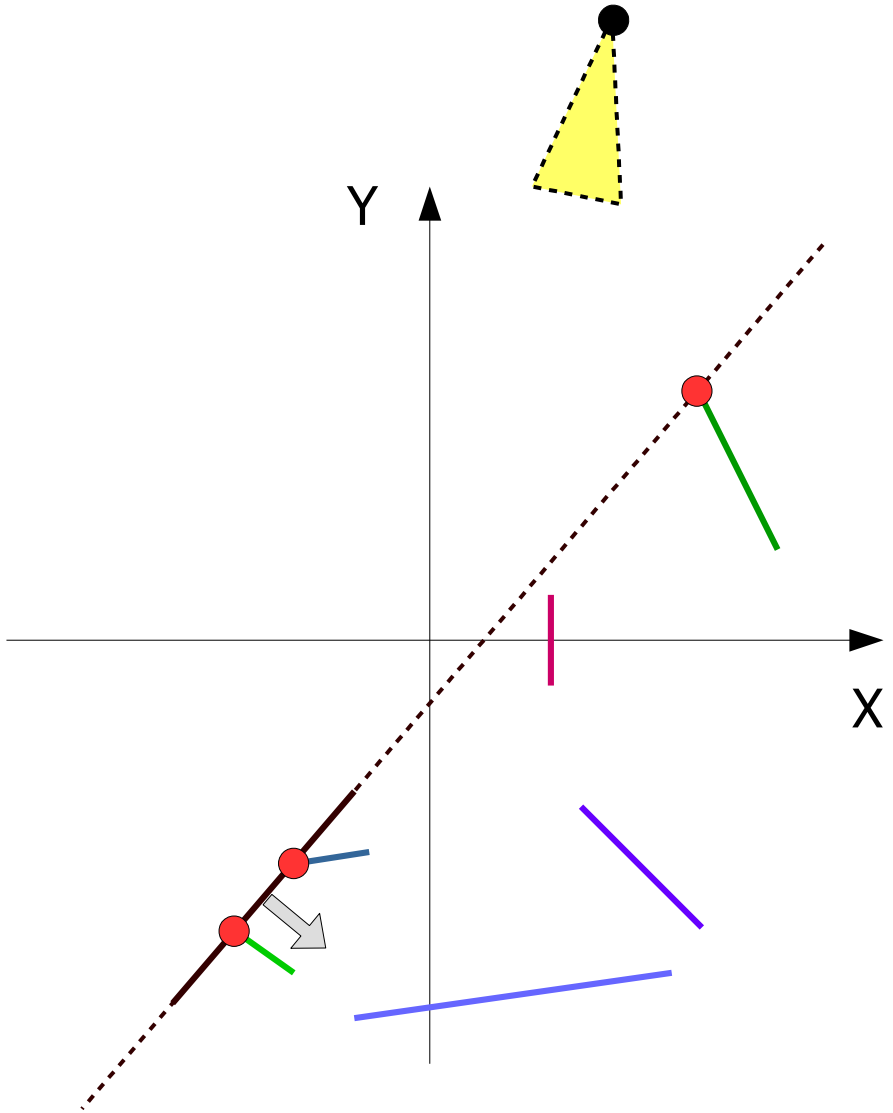


Having such kind of separation, how will we draw necessary objects on the screen?

Considering that viewer is located at one part,

- At first we will properly draw objects from the other part,

Complete scene rendering

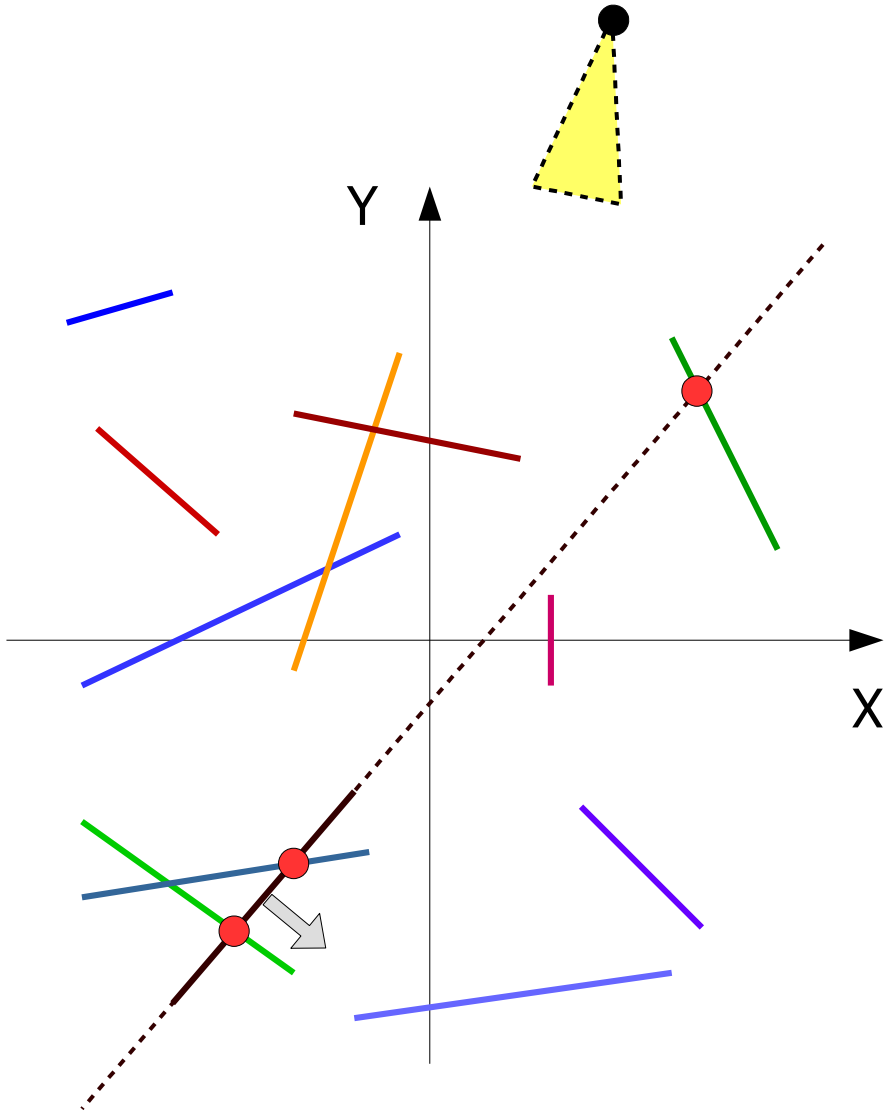


Having such kind of separation, how will we draw necessary objects on the screen?

Considering that viewer is located at one part,

- At first we will properly draw objects from the other part,
- Then we will draw the object(s) of split-plane,

Complete scene rendering

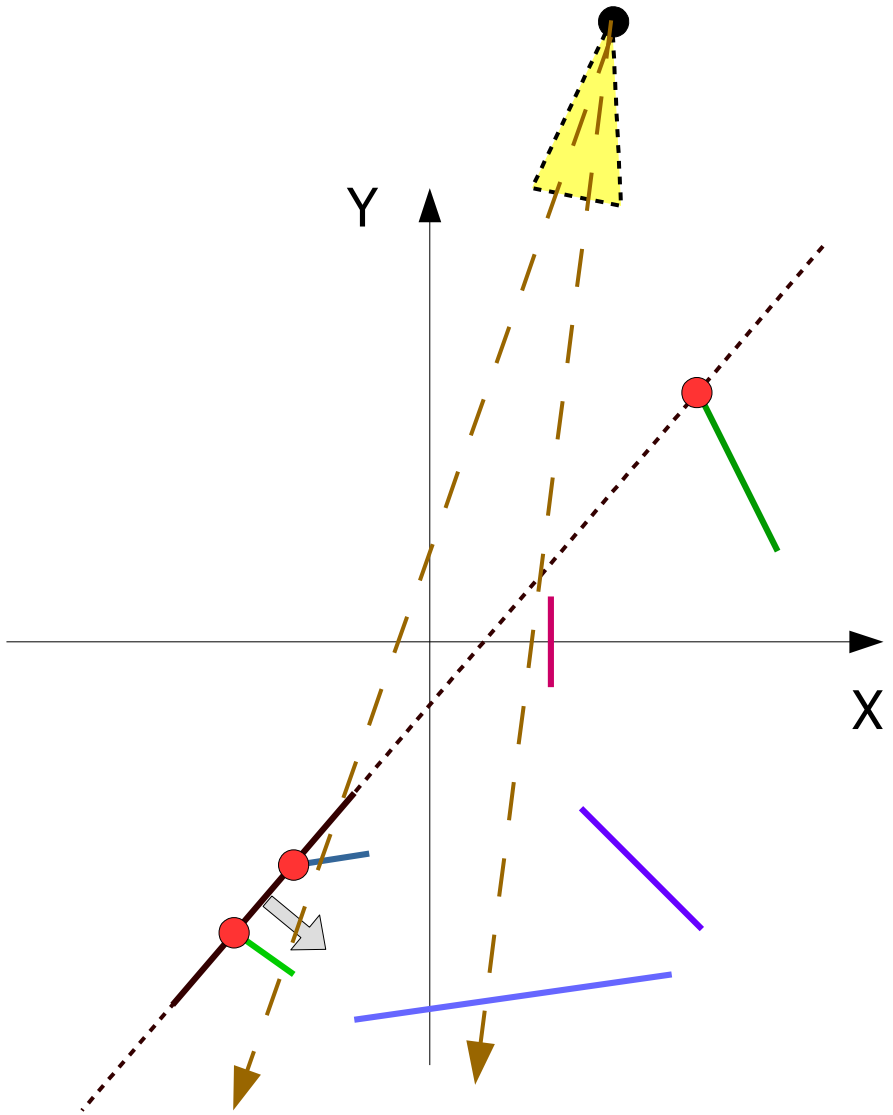


Having such kind of separation, how will we draw necessary objects on the screen?

Considering that viewer is located at one part,

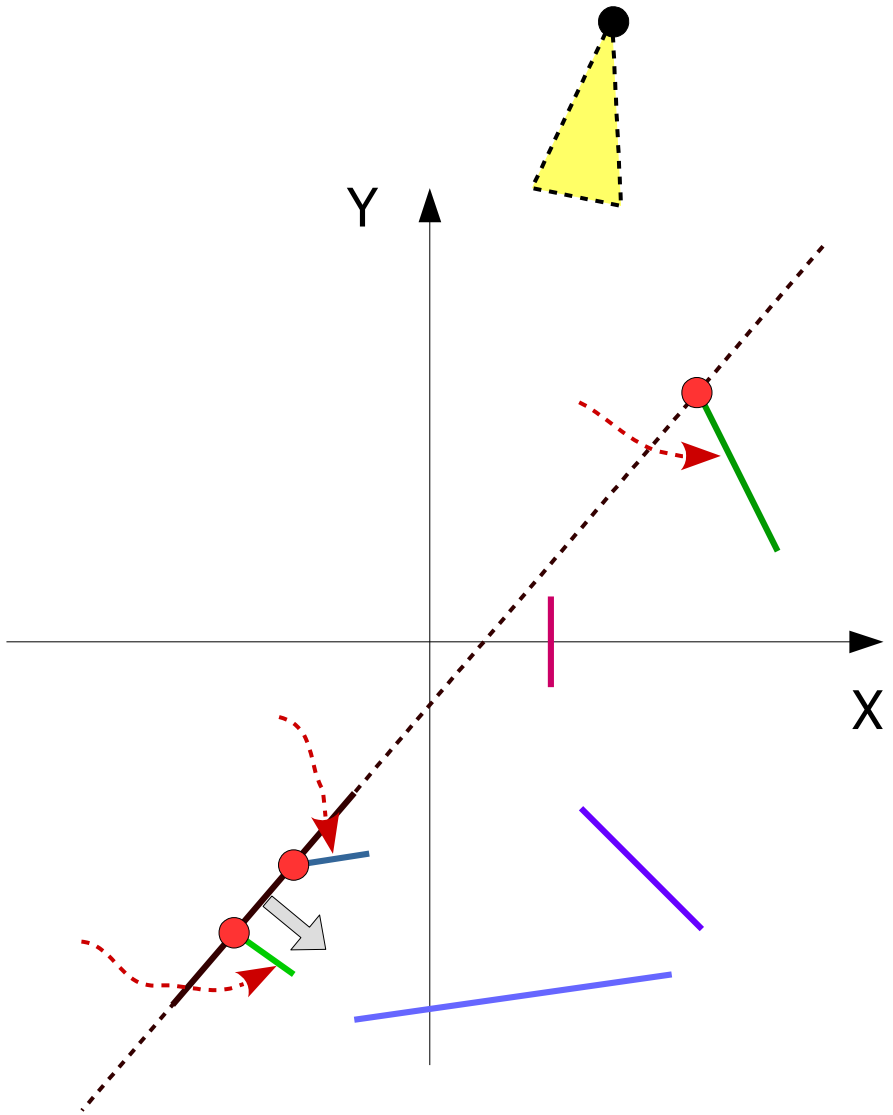
- At first we will properly draw objects from the other part,
- Then we will draw the object(s) of split-plane,
- And at the end we will properly draw objects from this part.

Complete scene rendering



This will ensure that looking by any direction, farther objects are being drawn before nearer ones.

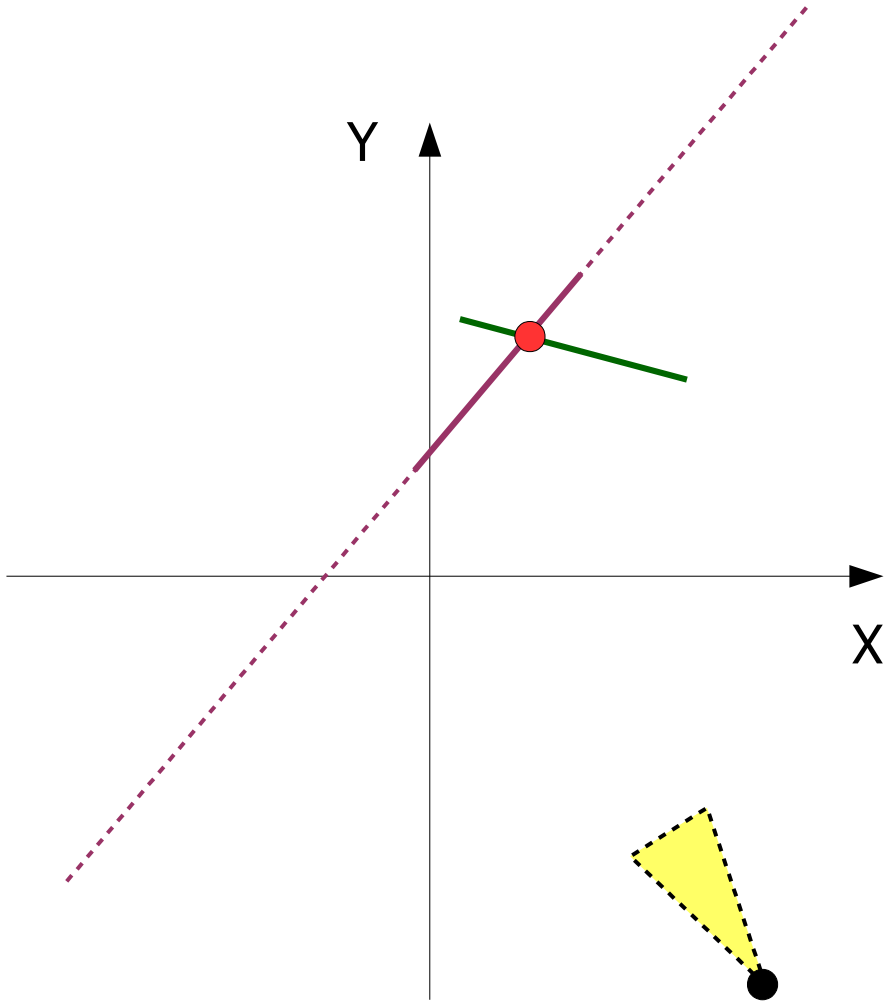
Complete scene rendering



This will ensure that looking by any direction, farther objects are being drawn before nearer ones.

Also let's note that those objects which were cut are being drawn by steps.

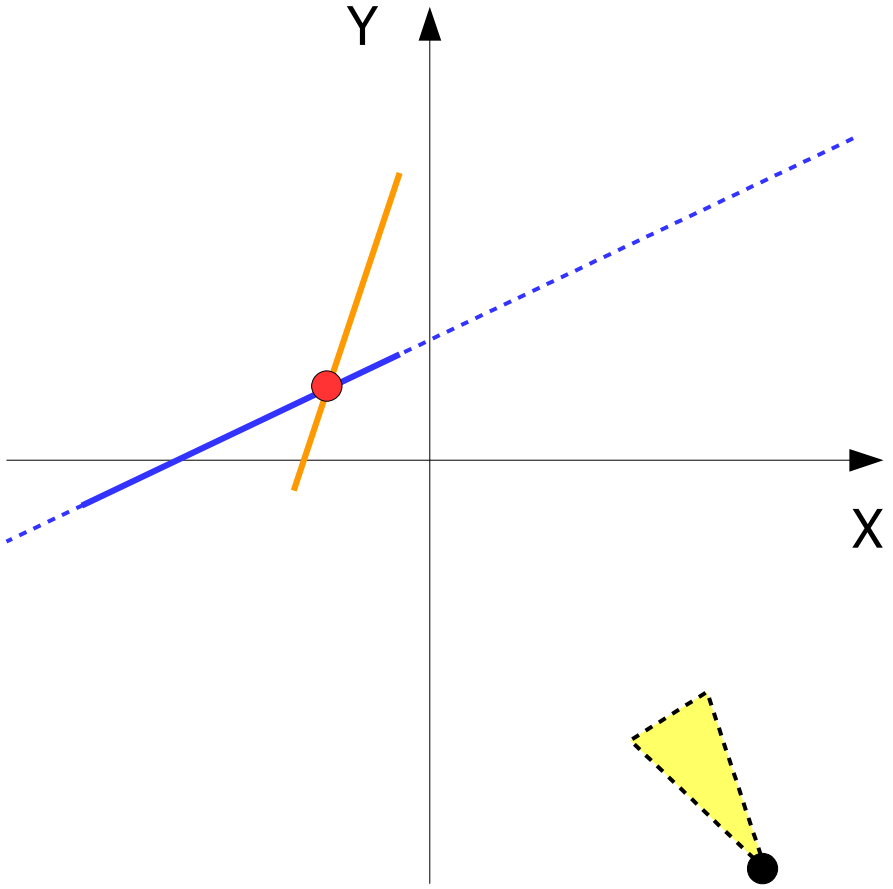
Complete scene rendering



Let's pay attention that the problematic cases mentioned earlier are resolved by this:

... case **1**,

Complete scene rendering

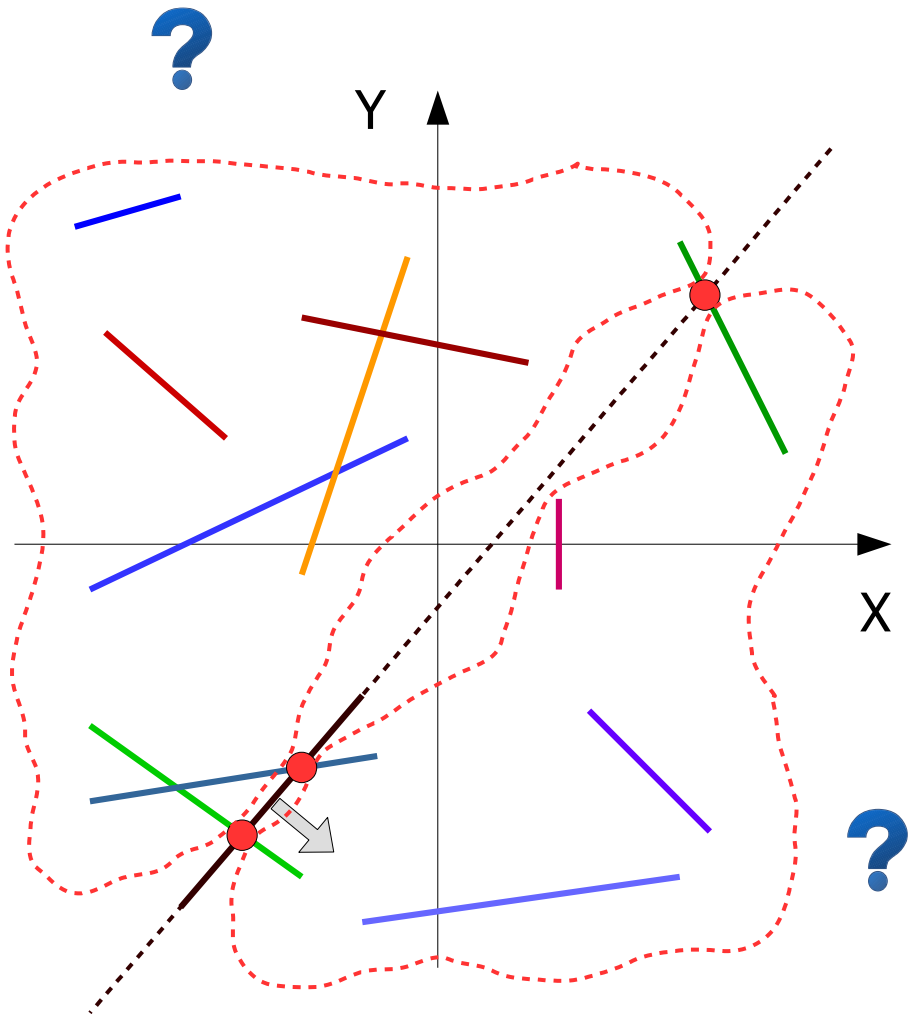


Let's pay attention that the problematic cases mentioned earlier are resolved by this:

... case **1**,

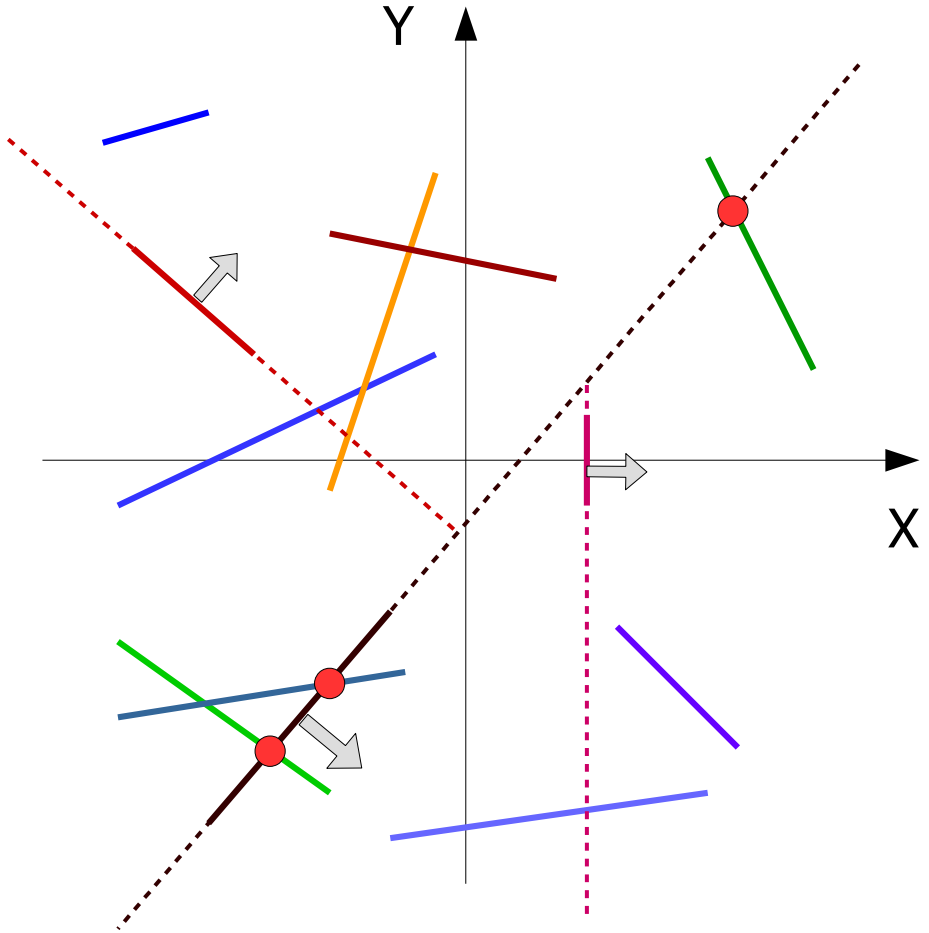
... case **2**.

Complete scene rendering



Fine, now how are we going to properly draw content of the both parts?

Complete scene rendering

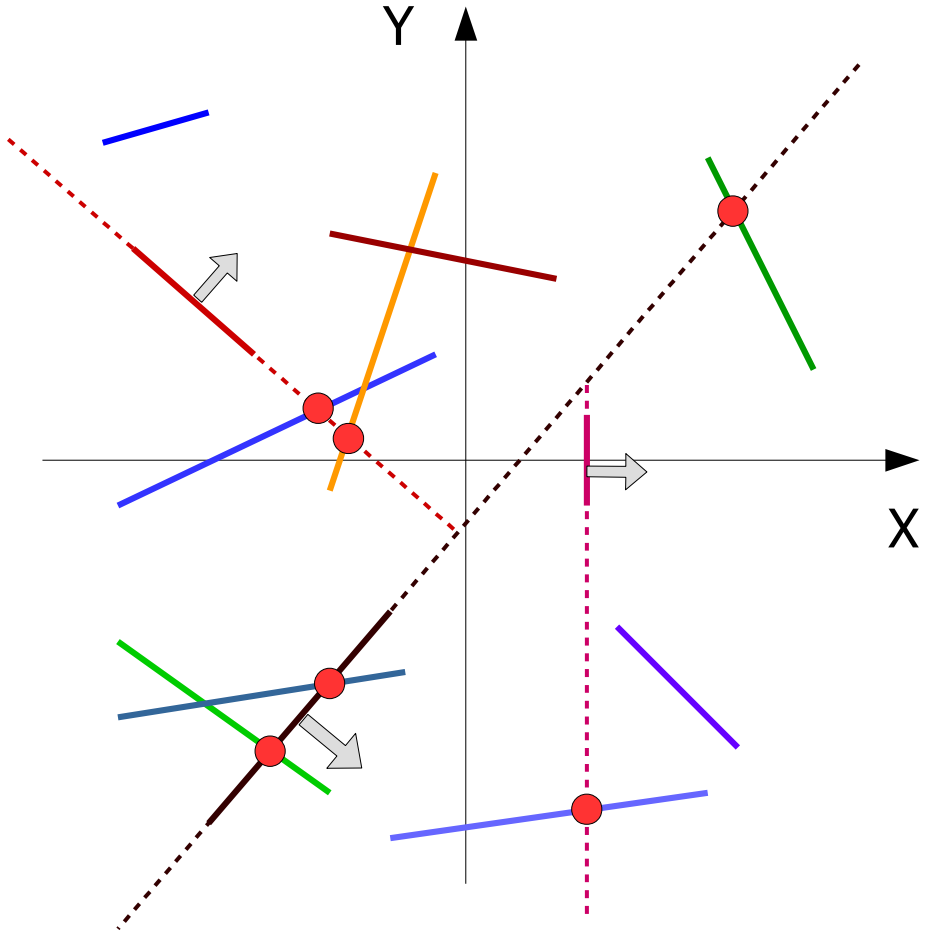


Fine, now how are we going to properly draw content of the both parts?

The answer is – recursively.

- So we will subdivide content of each part separately,

Complete scene rendering

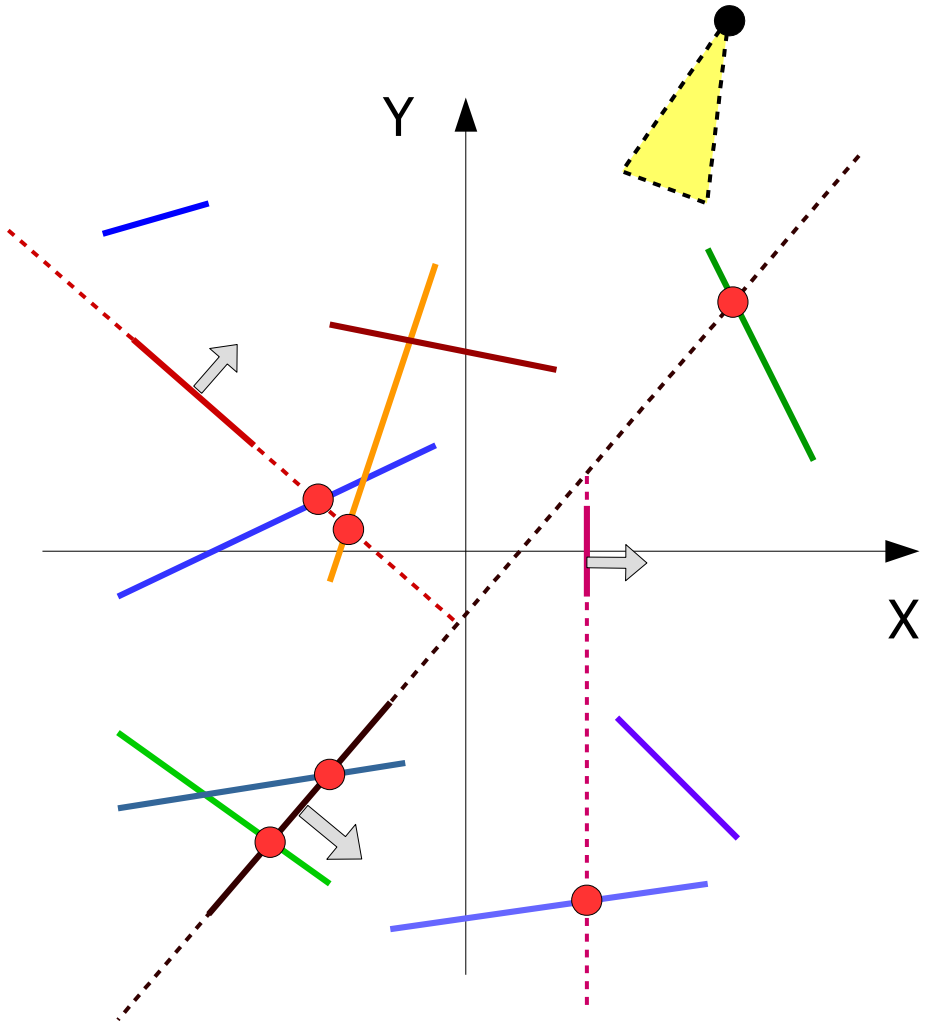


Fine, now how are we going to properly draw content of the both parts?

The answer is – recursively.

- So we will subdivide content of each part separately,
- Cut other segments which lie on the next borders,

Complete scene rendering

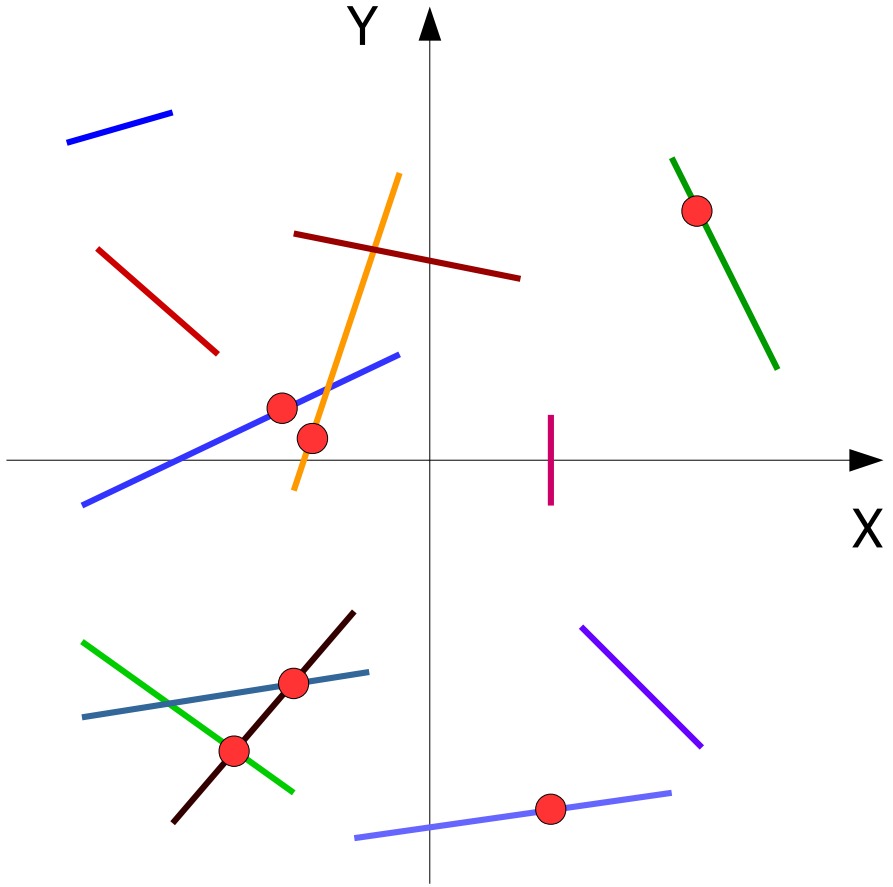


Fine, now how are we going to properly draw content of the both parts?

The answer is – recursively.

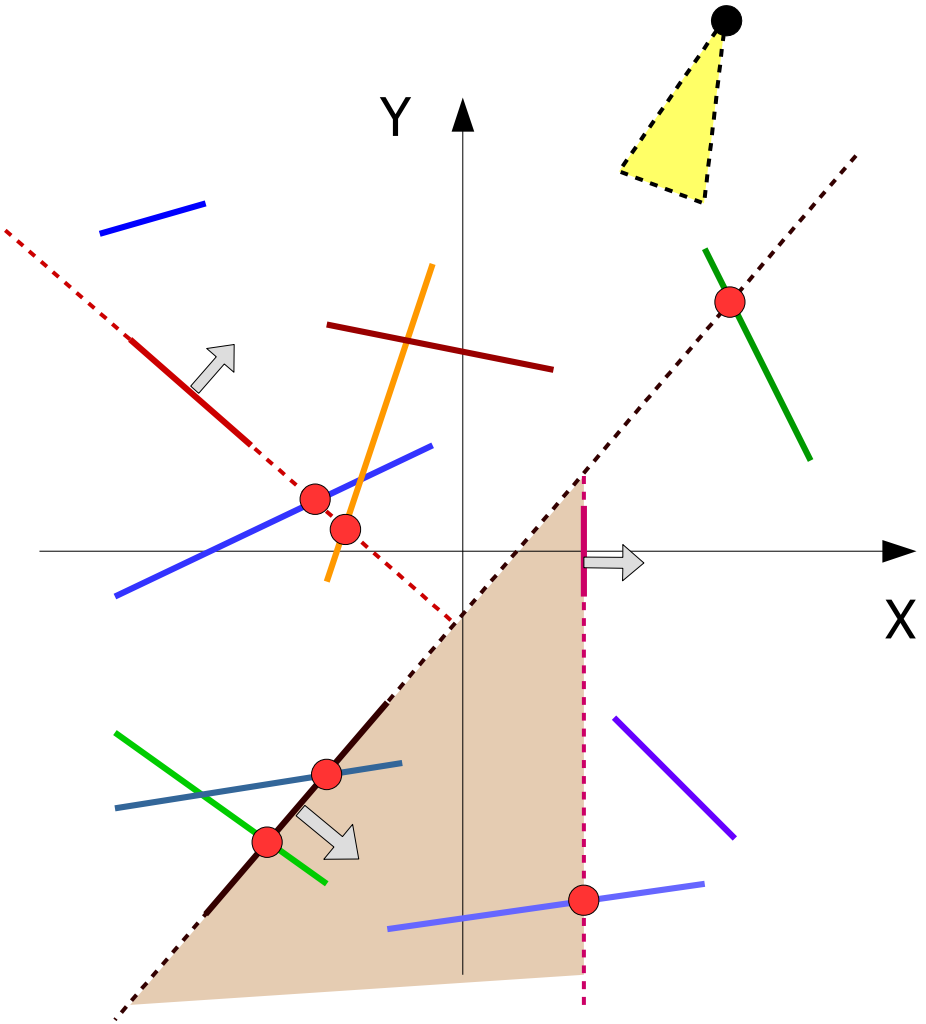
- So we will subdivide content of each part separately,
- Cut other segments which lie on the next borders,
- And will draw content of every half in proper order, depending on user's viewpoint.

Complete scene rendering



Note how amount of objects has increased again.

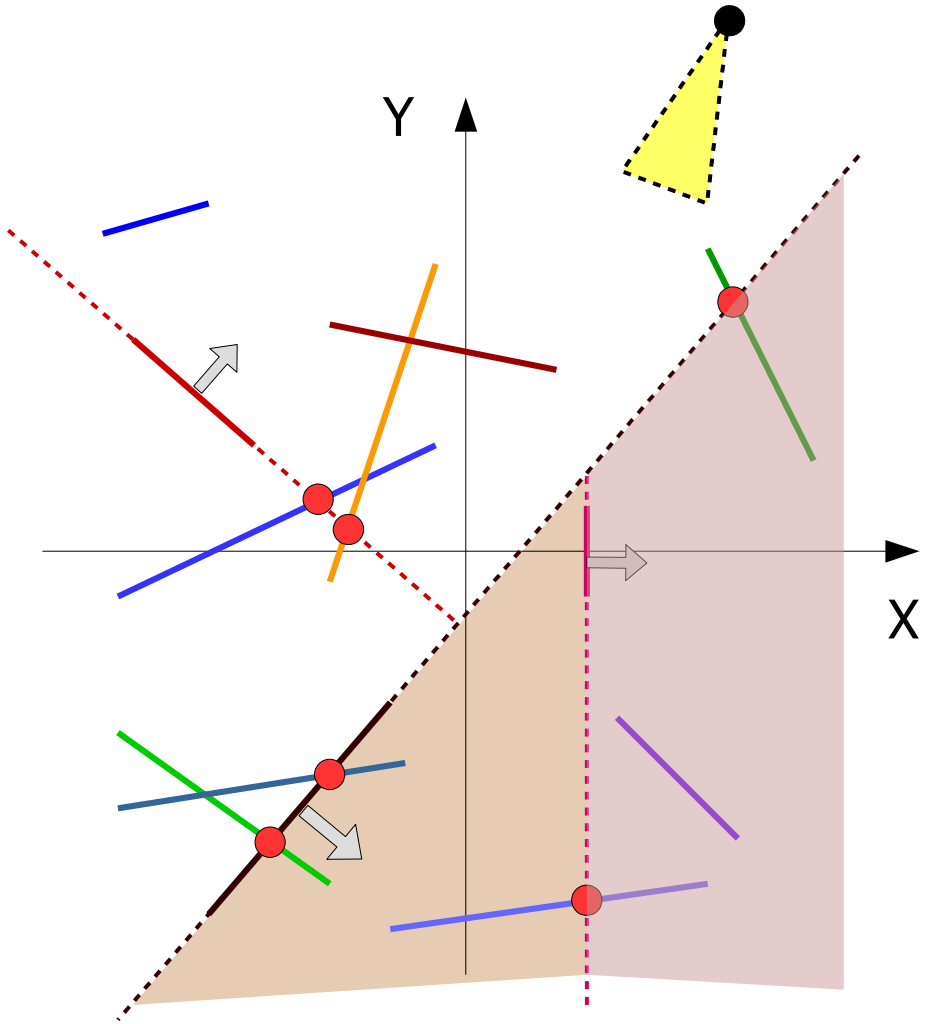
Complete scene rendering



So for current view-point the overall order will be:

... quater 1,

Complete scene rendering

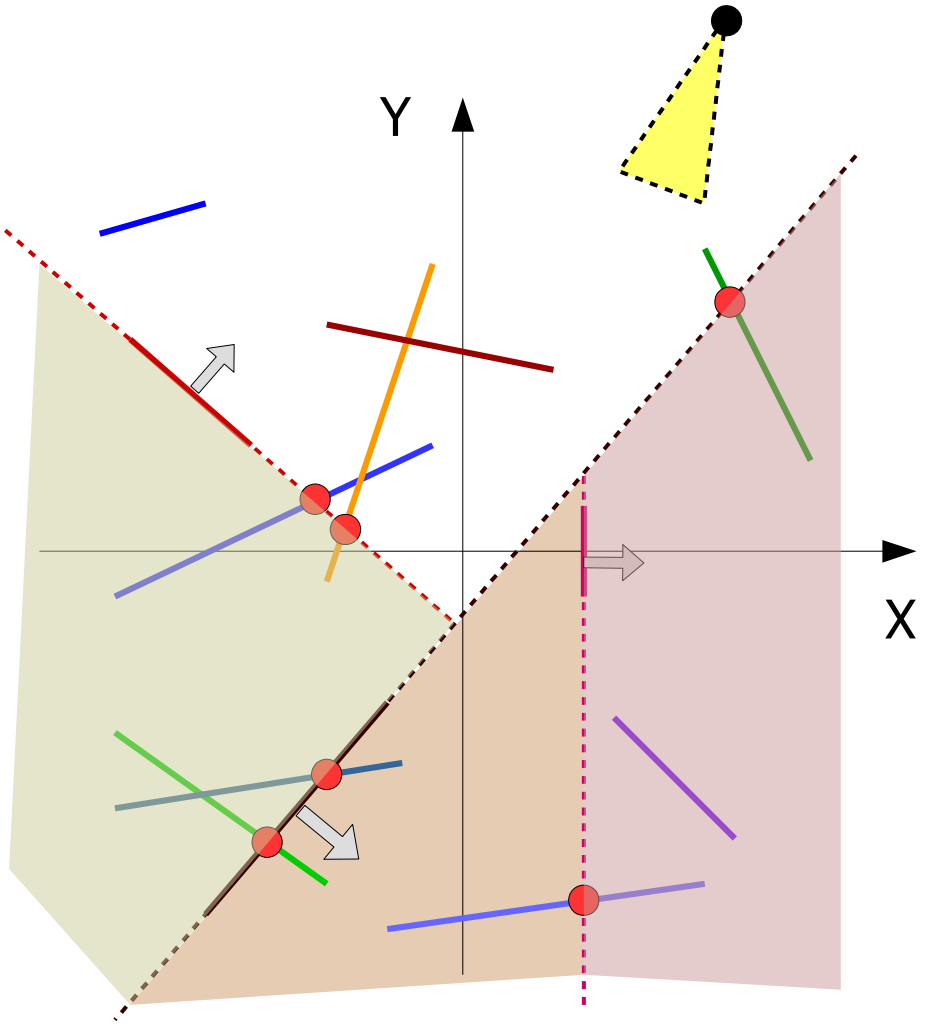


So for current view-point the overall order will be:

... quater 1,

... quater 2,

Complete scene rendering



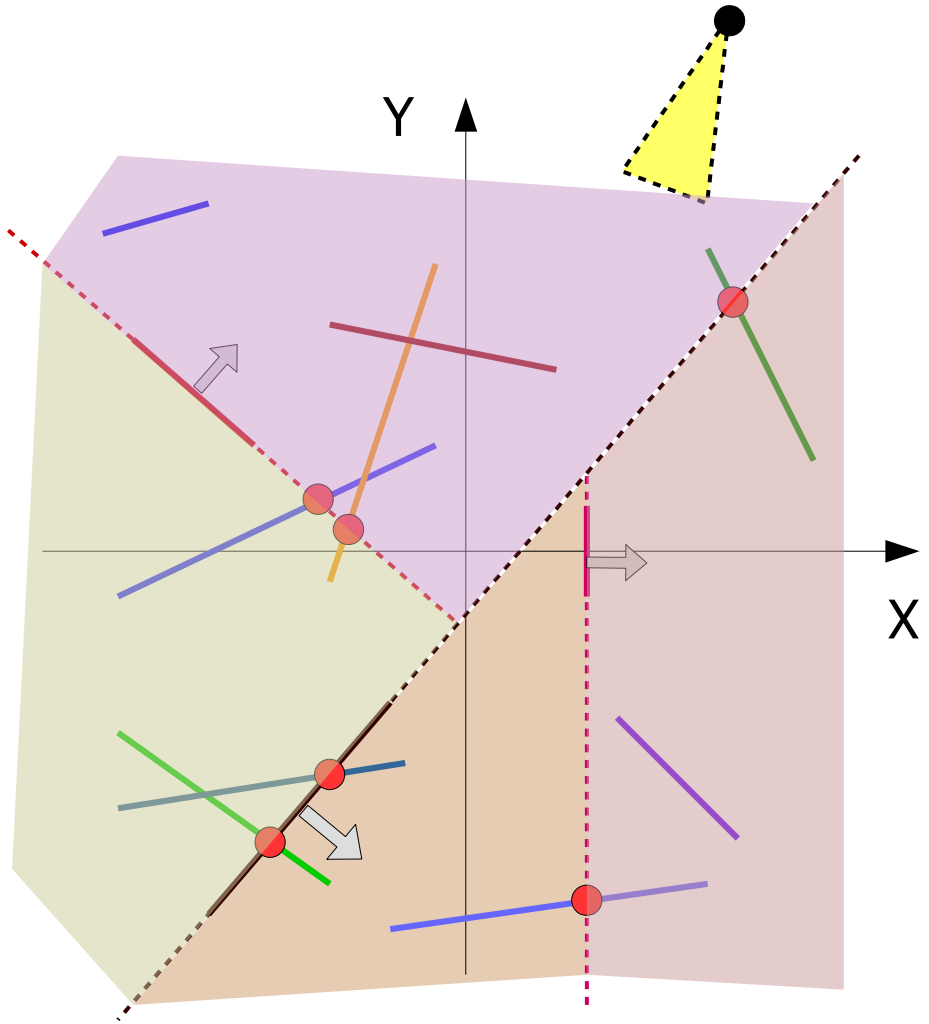
So for current view-point the overall order will be:

... quater 1,

... quater 2,

... quater 3,

Complete scene rendering



So for current view-point the overall order will be:

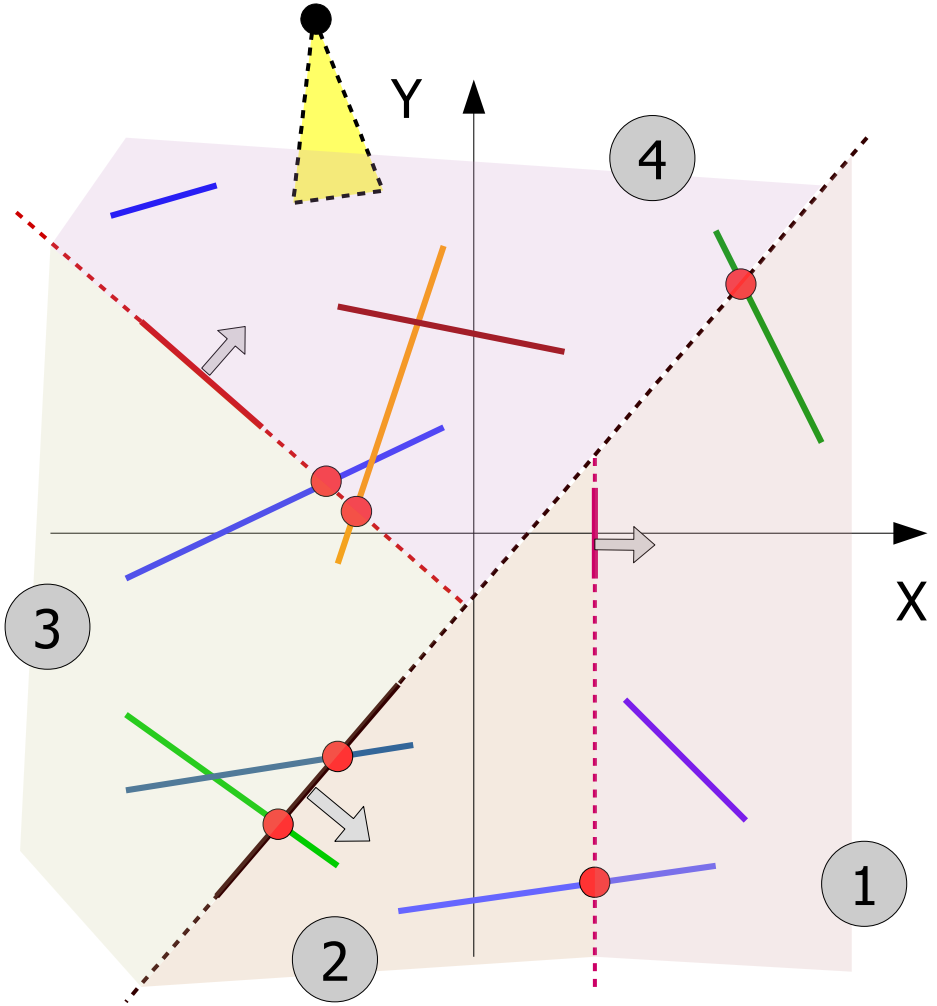
... quater 1,

... quater 2,

... quater 3,

... quater 4,

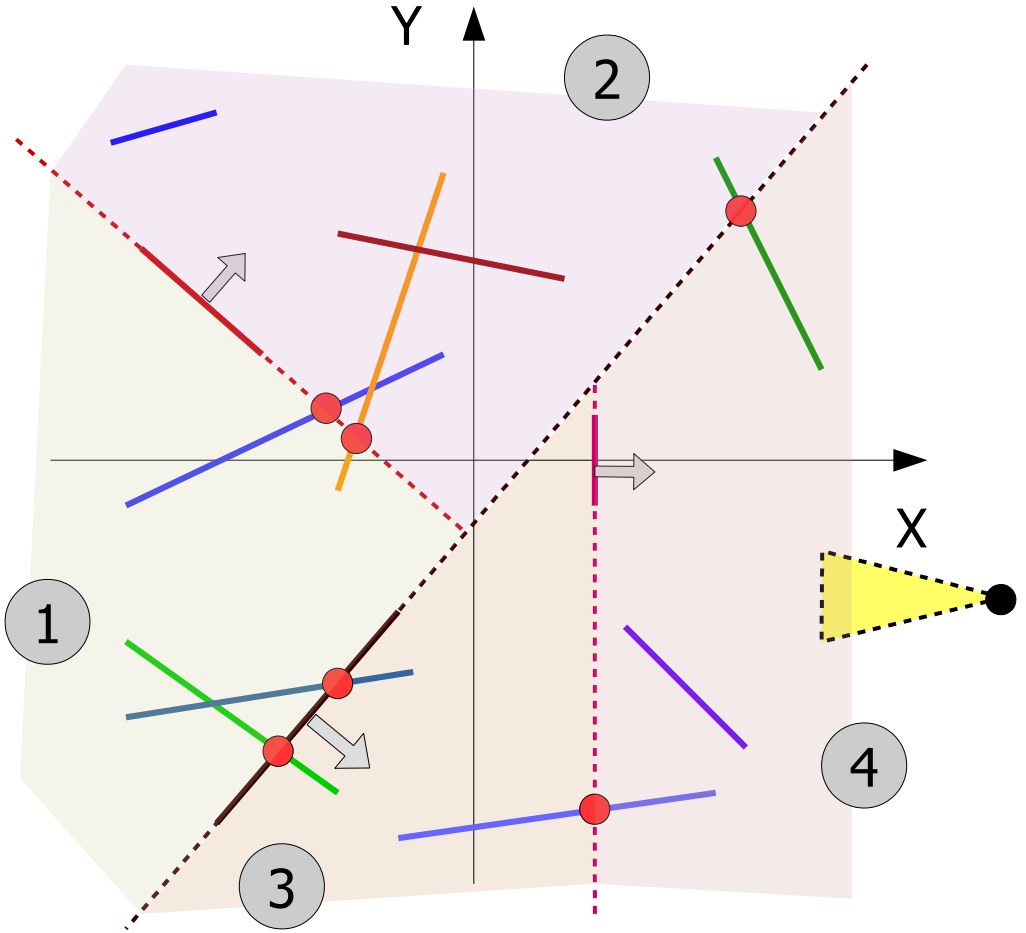
Complete scene rendering



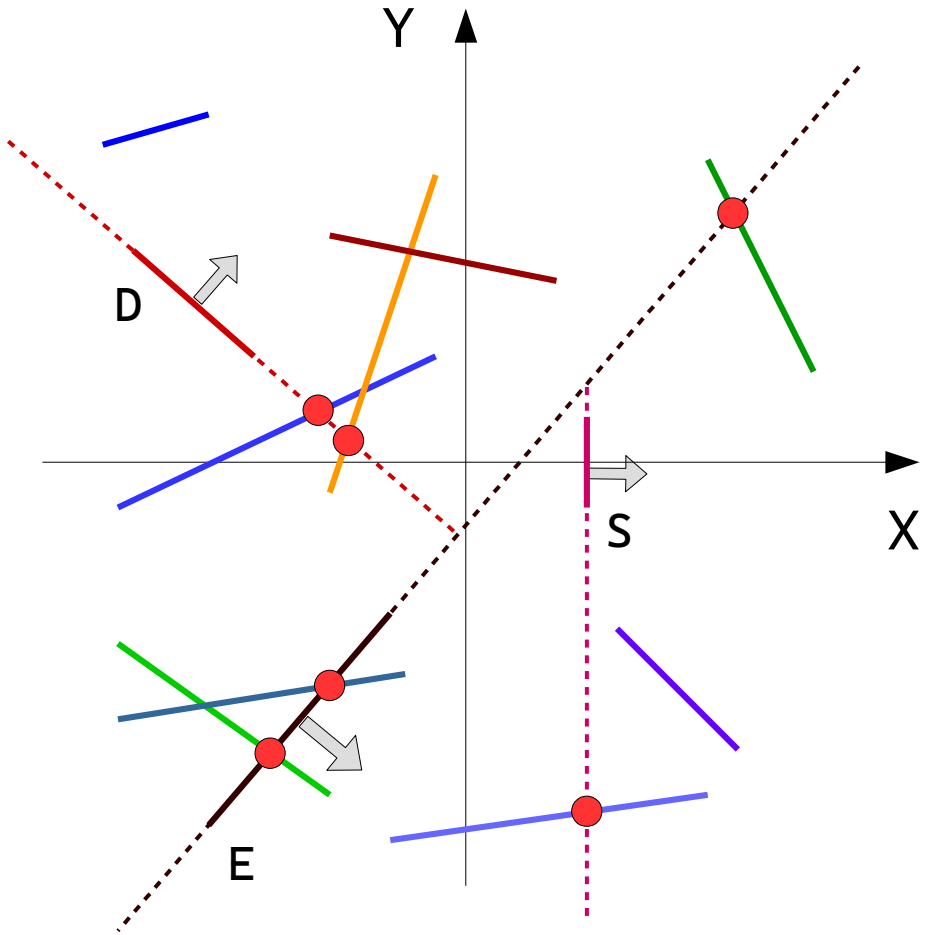
What if the viewpoint will reside a bit away?

Complete scene rendering

One more case?

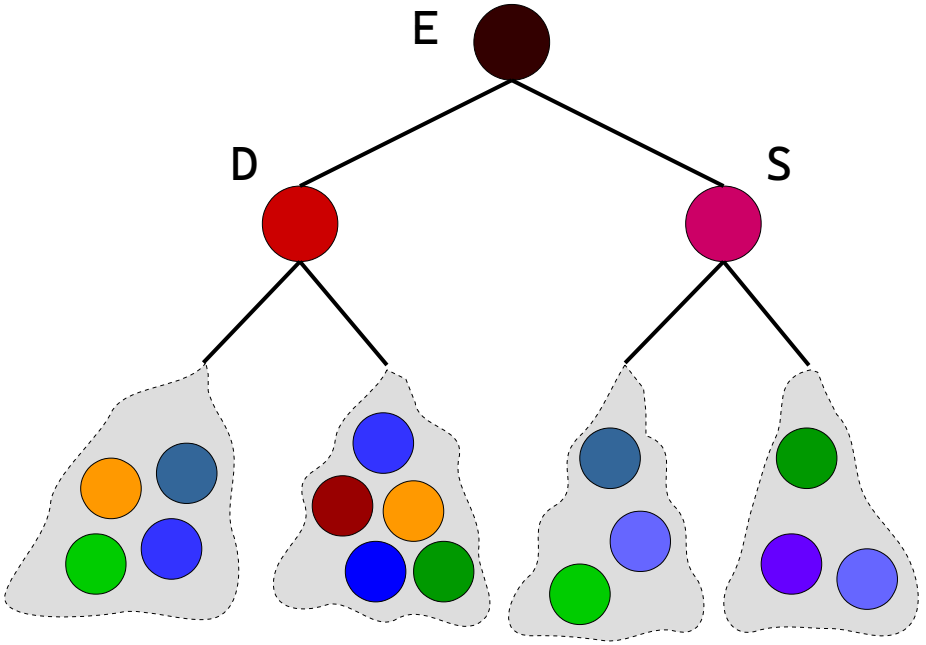


Complete scene rendering

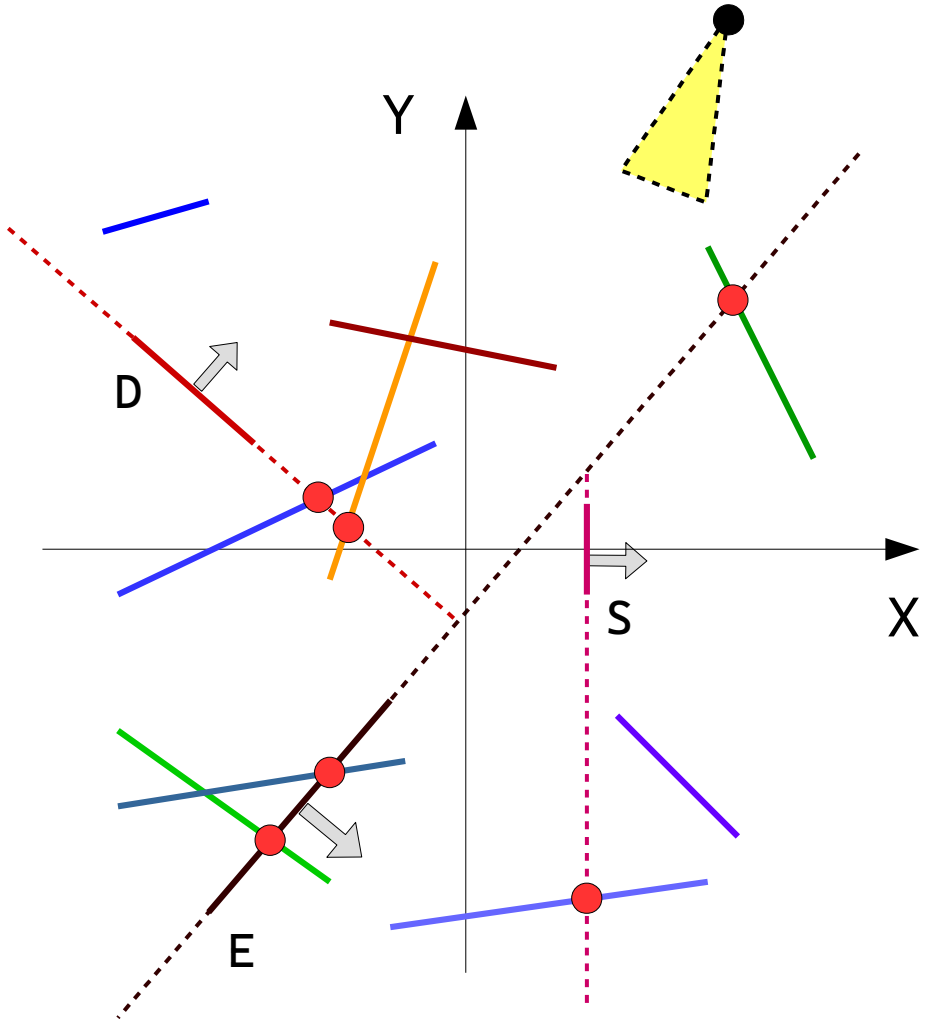


Alright, how it all is being arranged as a data structure?

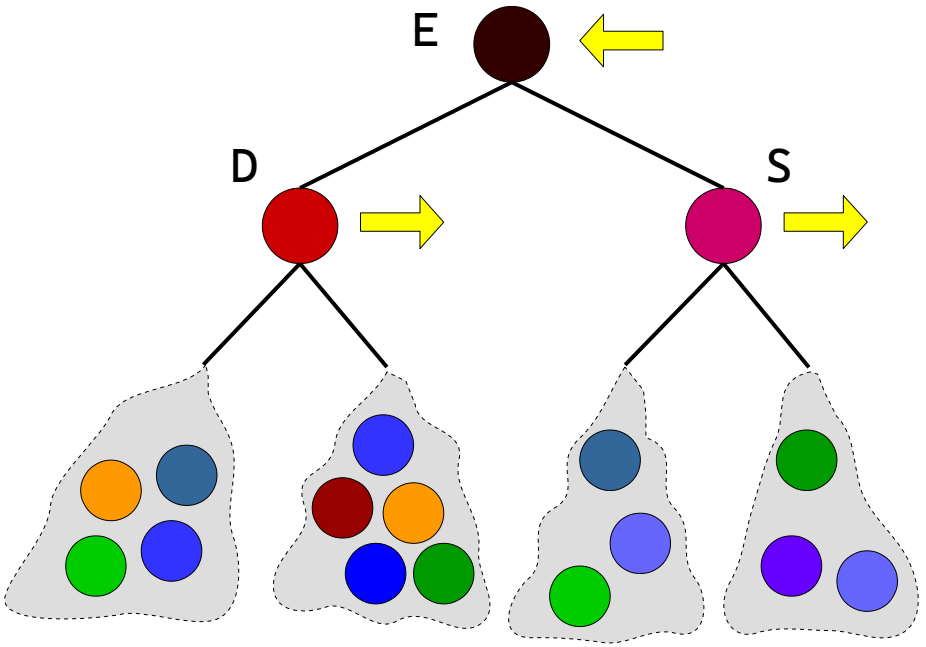
It is arranged in form of binary tree.



Complete scene rendering

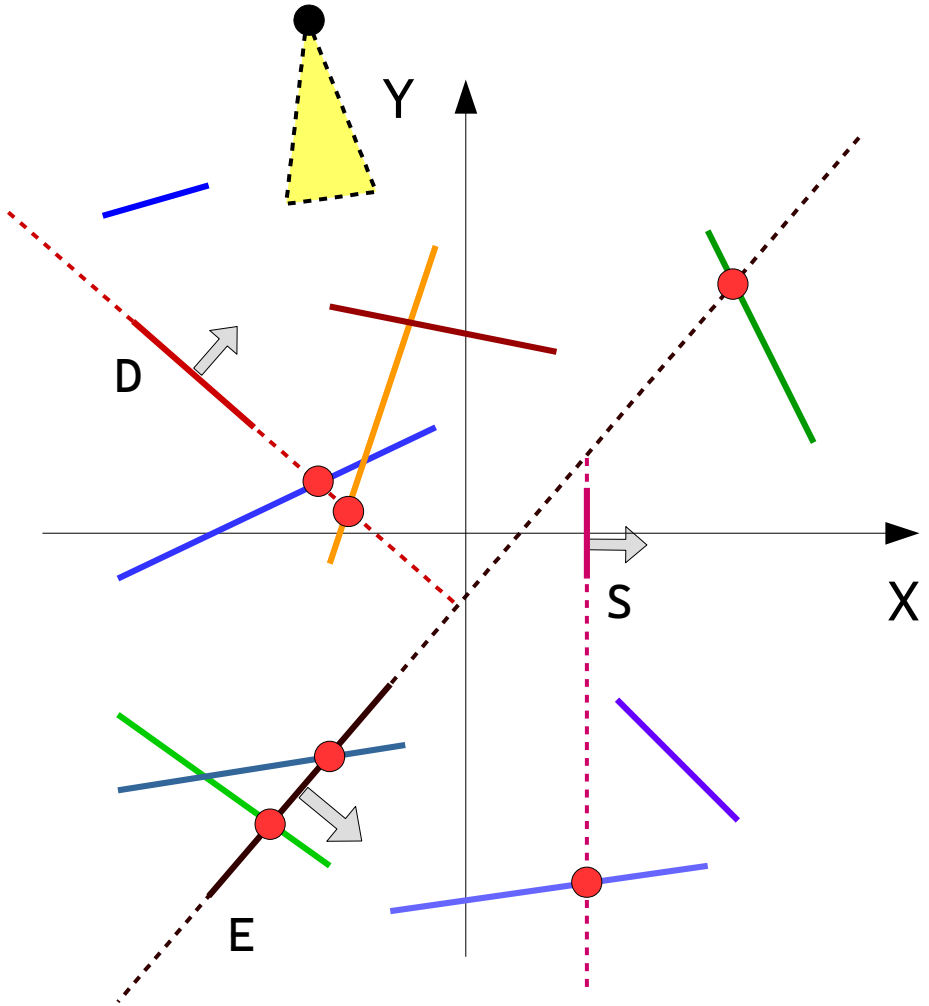


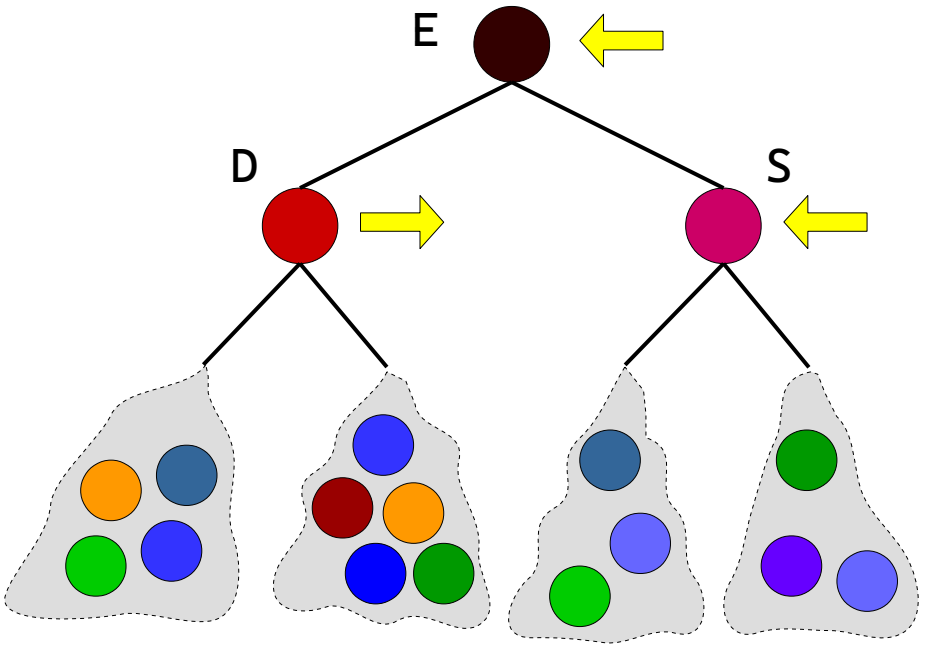
And for each rendering we must decide the traversal order <u>for every node separately</u>.



Complete scene rendering

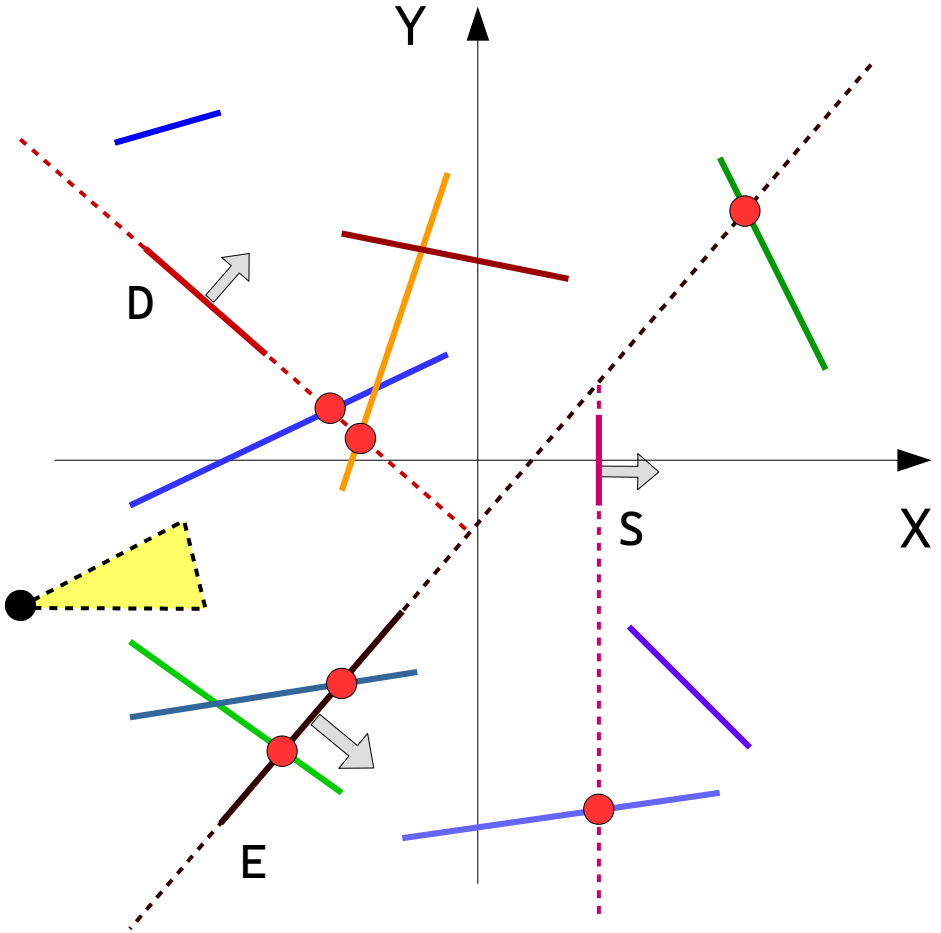
... the other example.

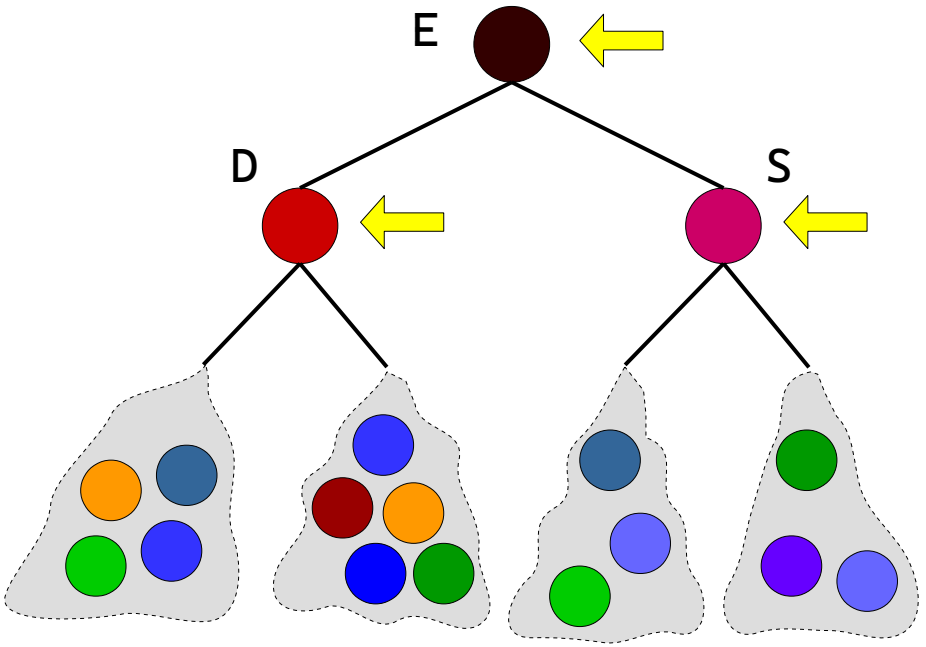




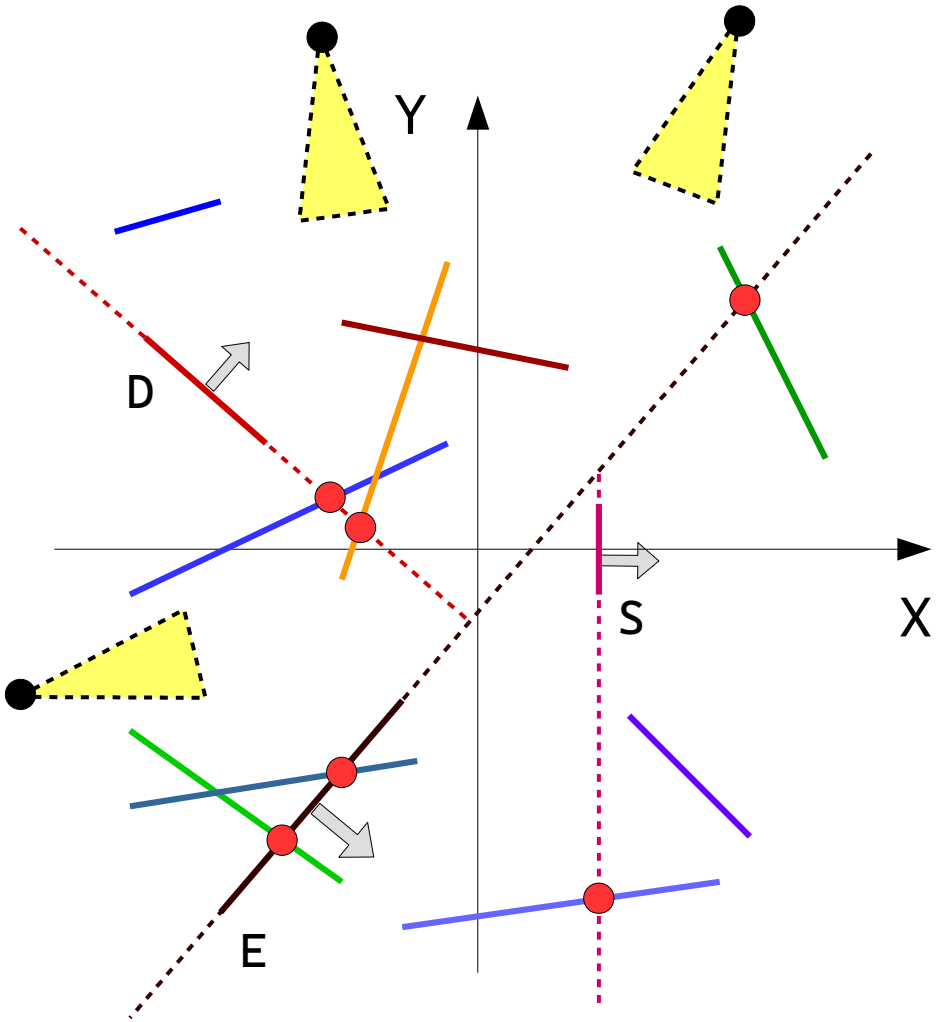
Complete scene rendering

... one more example.





Complete scene rendering

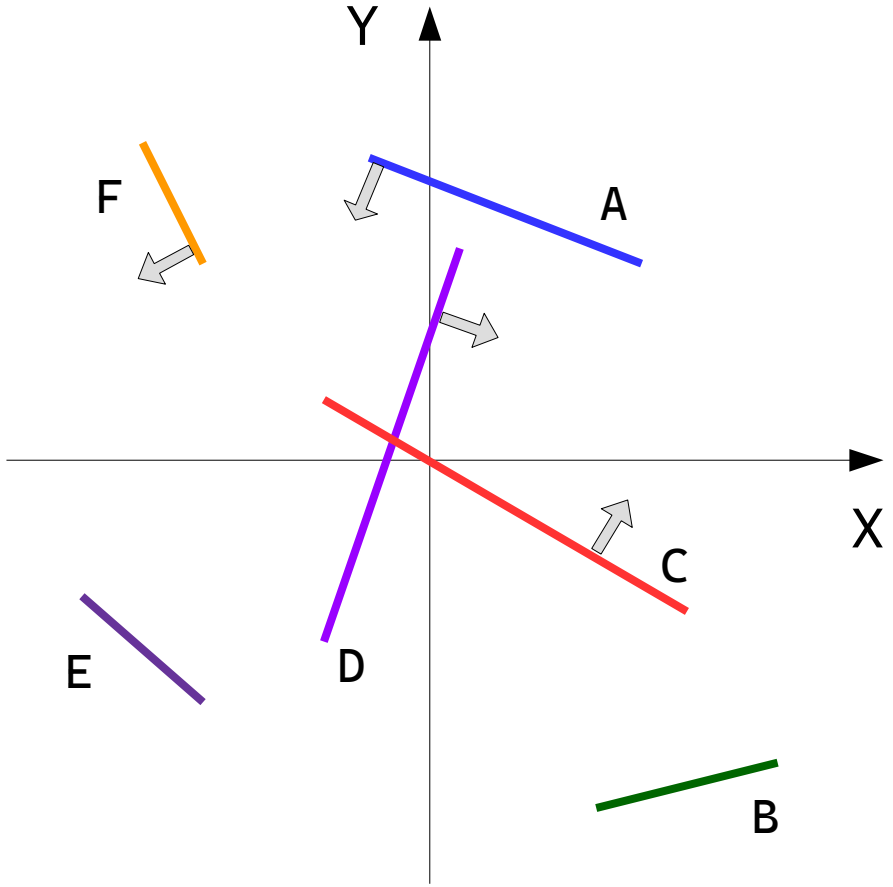


As we see, BSP allows us to properly render from any viewpoint,

... and the tree must be constructed only once.

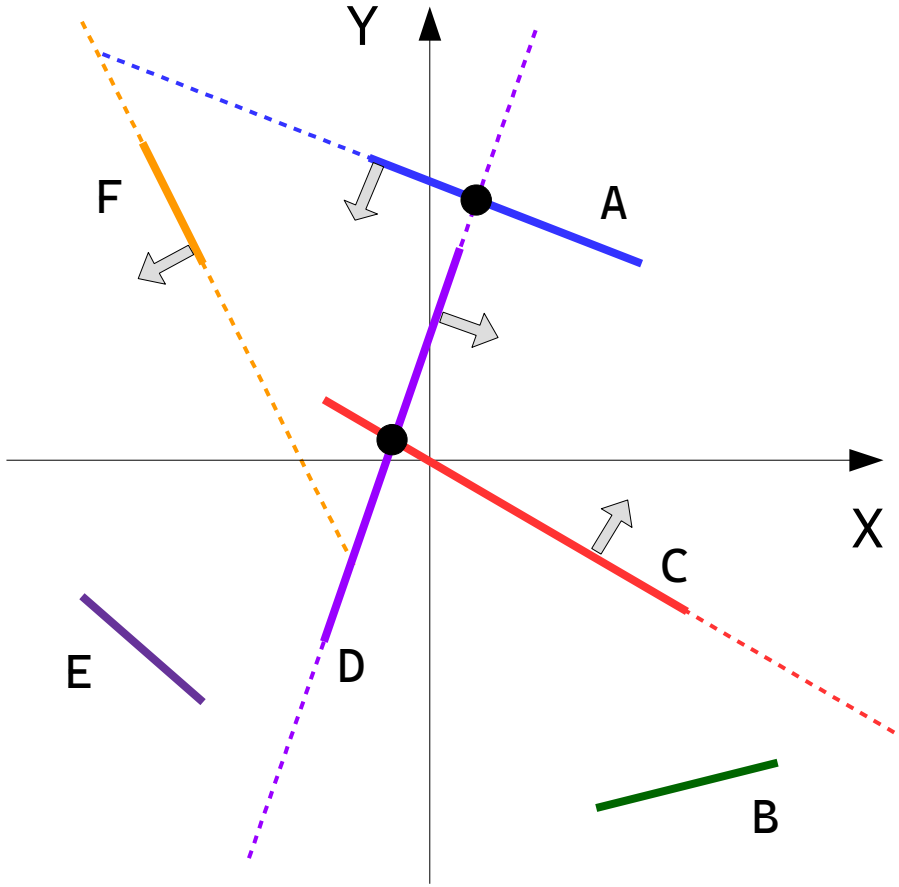
Exercise

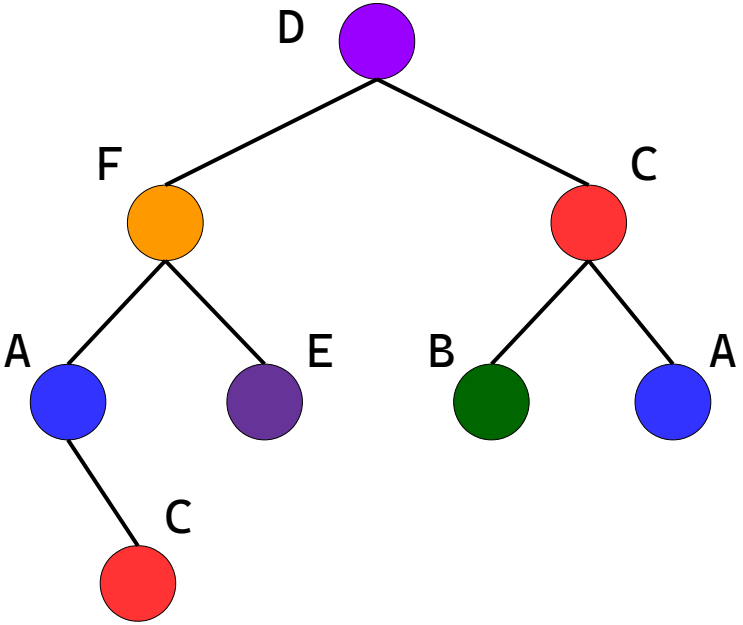
Construct BSP tree for the following segments:



Exercise
(solution)

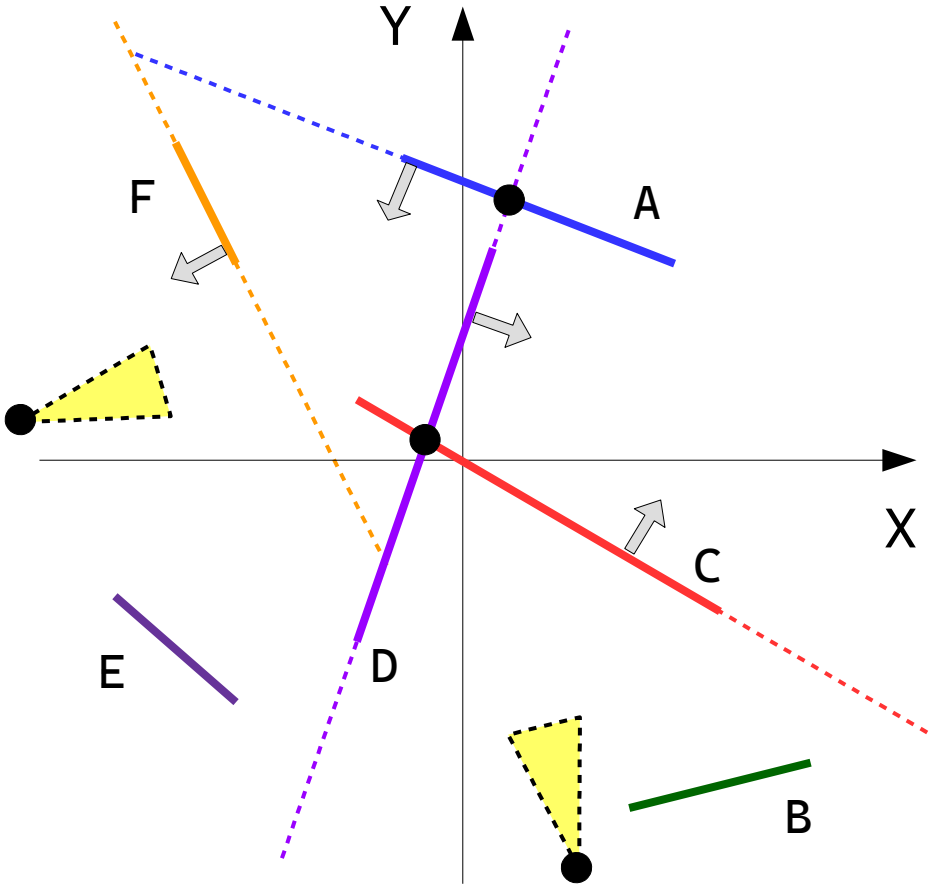
Construct BSP tree for the following segments:

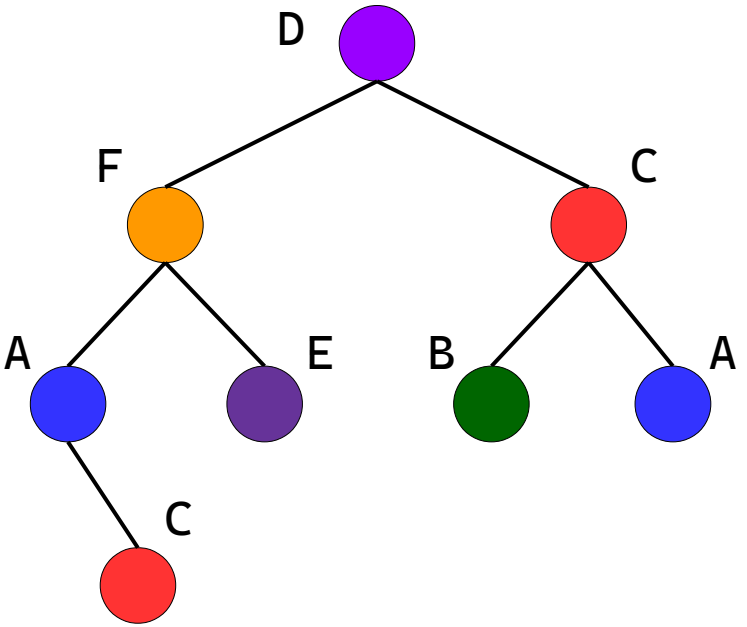




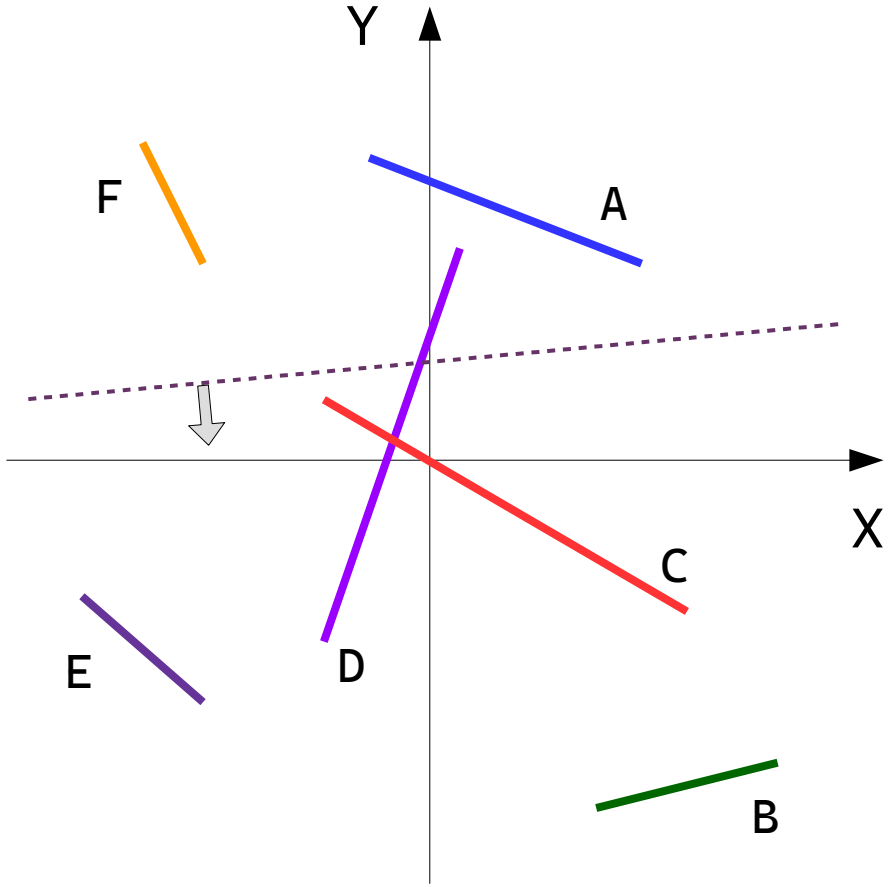
Exercise

Having the BSP tree, describe its traversals needed for render from the following viewpoints:



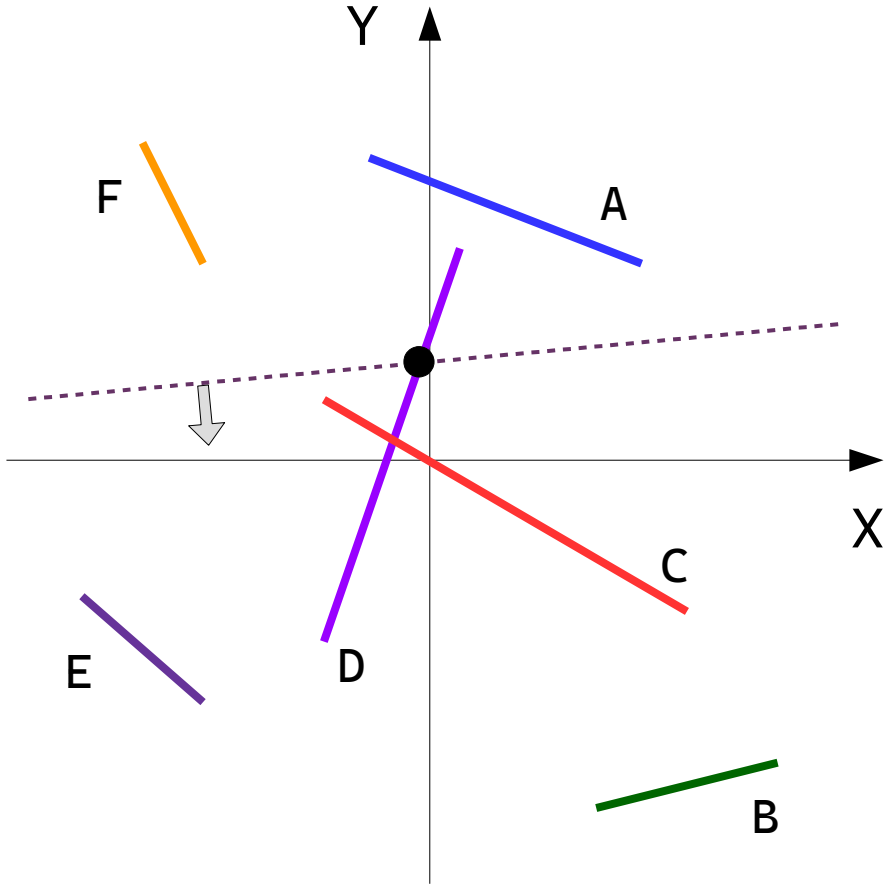


Complete scene rendering



Question: Are we always forced to take an existing segment as pivot for partitioning? Can't we take an arbitrary line instead?

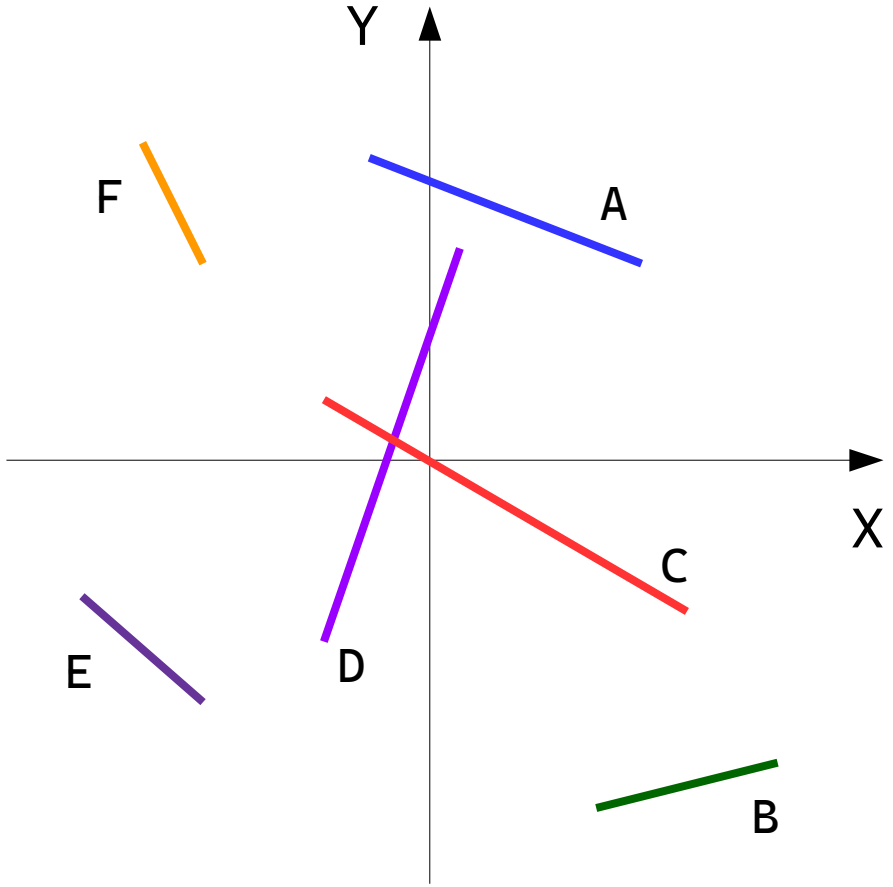
Complete scene rendering



Question: Are we always forced to take an existing segment as pivot for partitioning? Can't we take an arbitrary line instead?

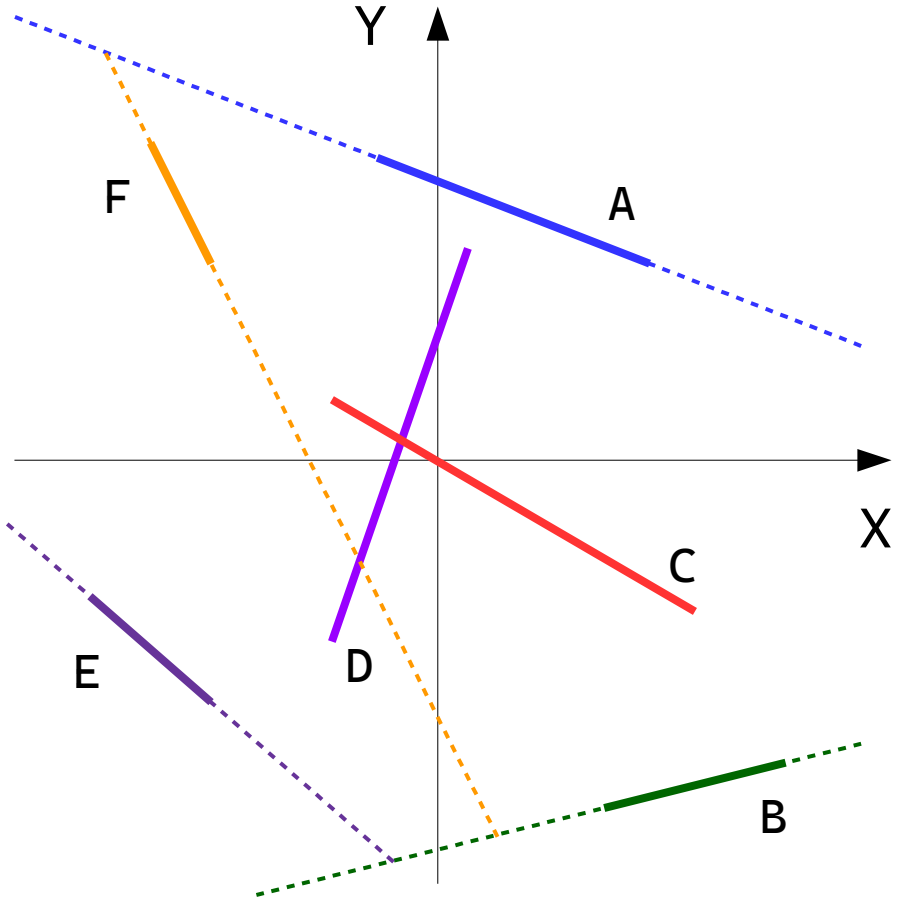
Answer: Yes, we can, but taking an existing segment the a pivot at least eliminates the risk for that segment to be cut later, down the tree.

Complete scene rendering



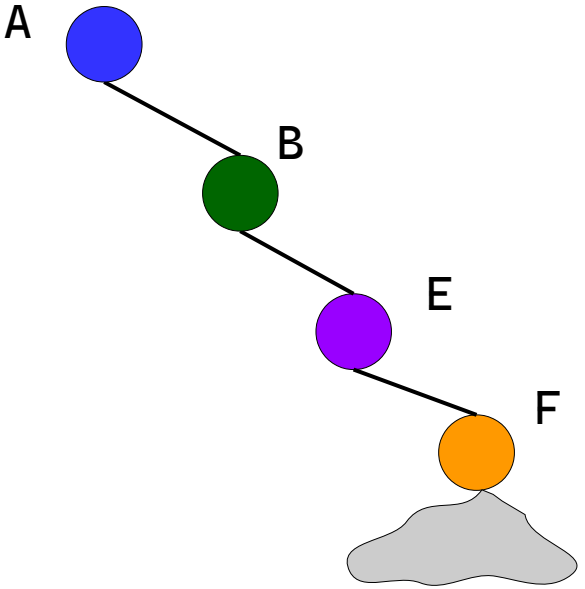
Question: Can there be "good" and "bad" partitionings in BSP tree?

Complete scene rendering



Question: Can there be "good" and "bad" partitionings in BSP tree?

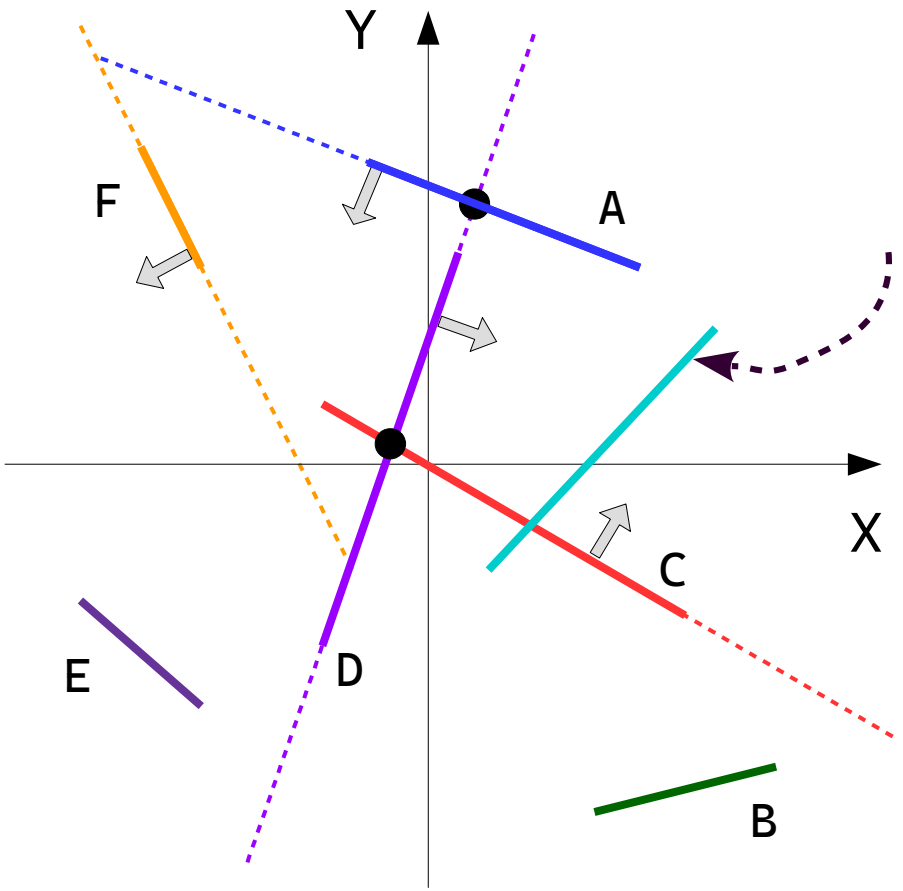
Answer: Yes, there can be. So for constructing an efficient tree it is a good practice <u>to shuffle</u> at first all the objects.



A variant of insertion

Can we insert new segment in already constructed BSP tree?

There can be **4** cases, traversing down the tree:

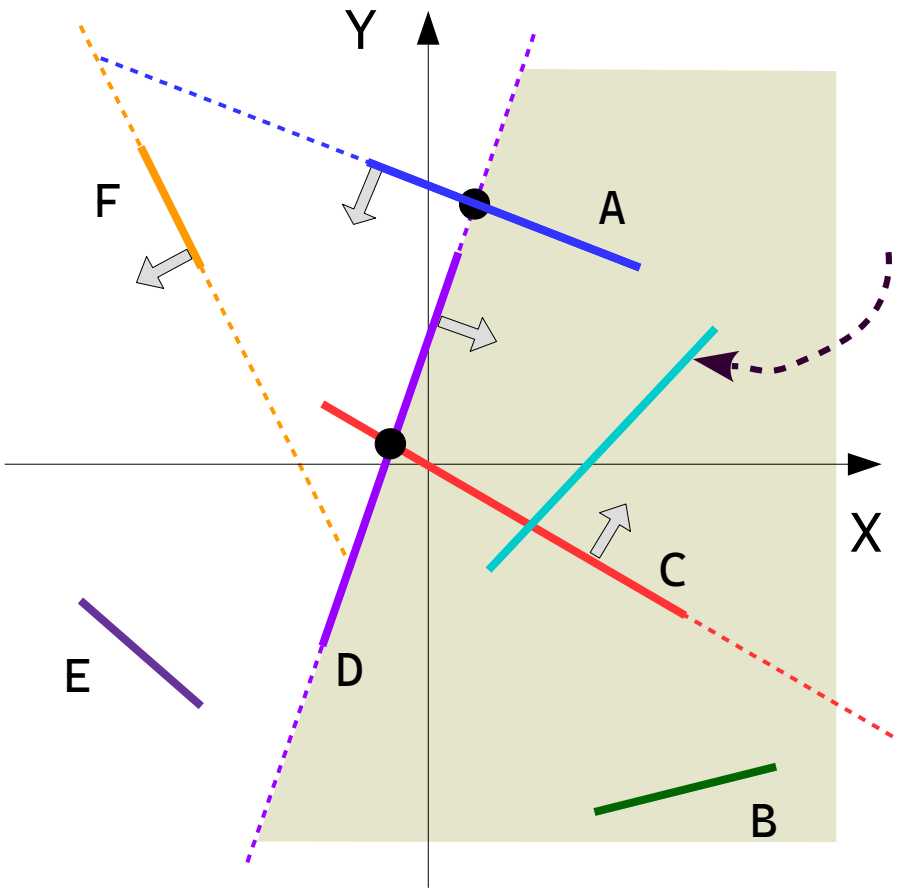


A variant of insertion

Can we insert new segment in already constructed BSP tree?

There can be **4** cases, traversing down the tree:

- '**x**' is completely in the front part,
  ... insert it in right subtree

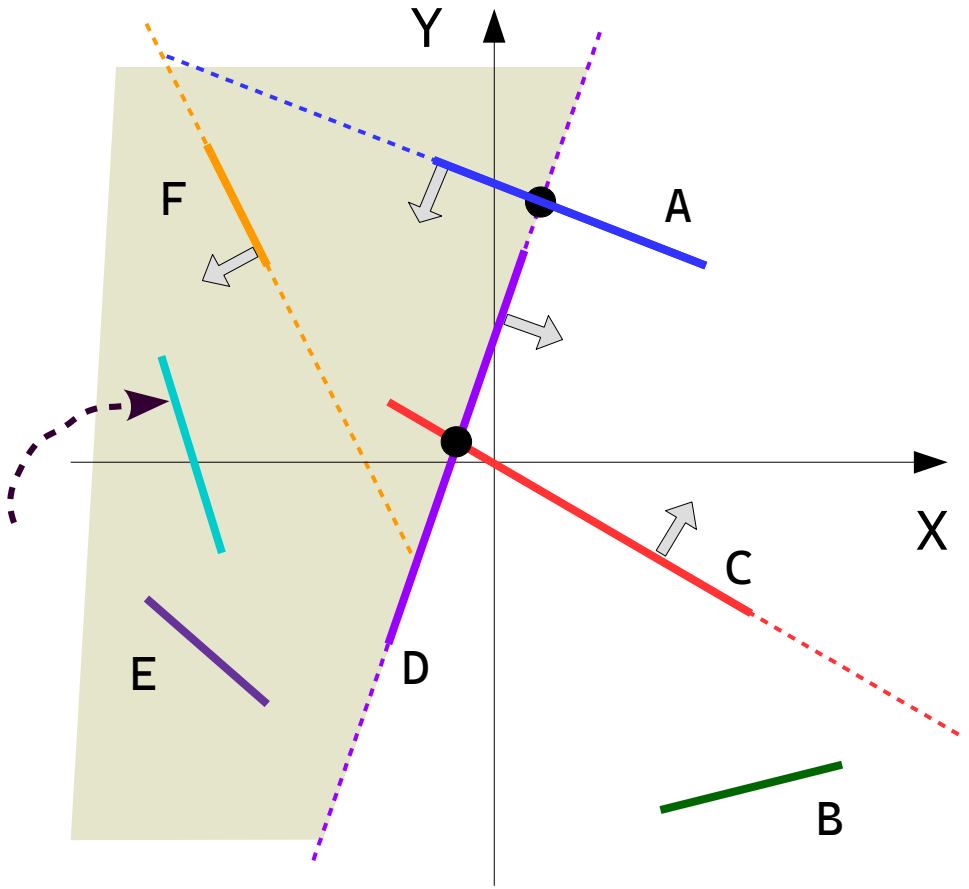


A variant of insertion

Can we insert new segment in already constructed BSP tree?

There can be **4** cases, traversing down the tree:

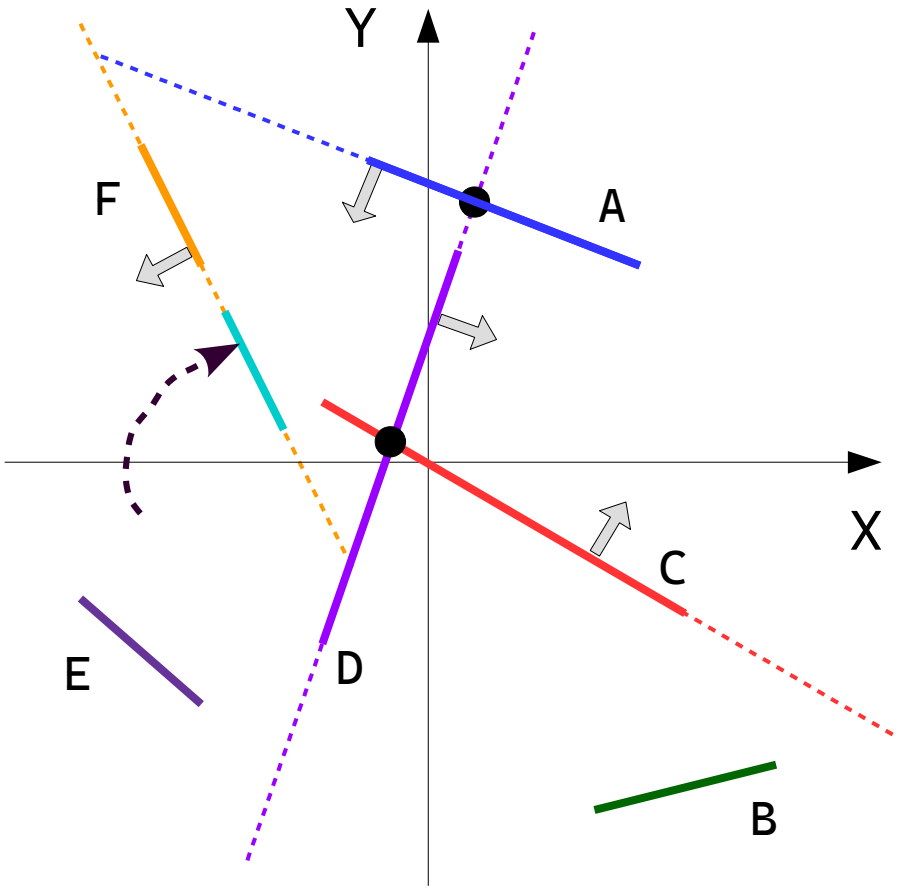
- '**x**' is completely in the front part,
  ... insert it in right subtree
- '**x**' is completely in the back part,
  ... insert it in left subtree



A variant of insertion

...

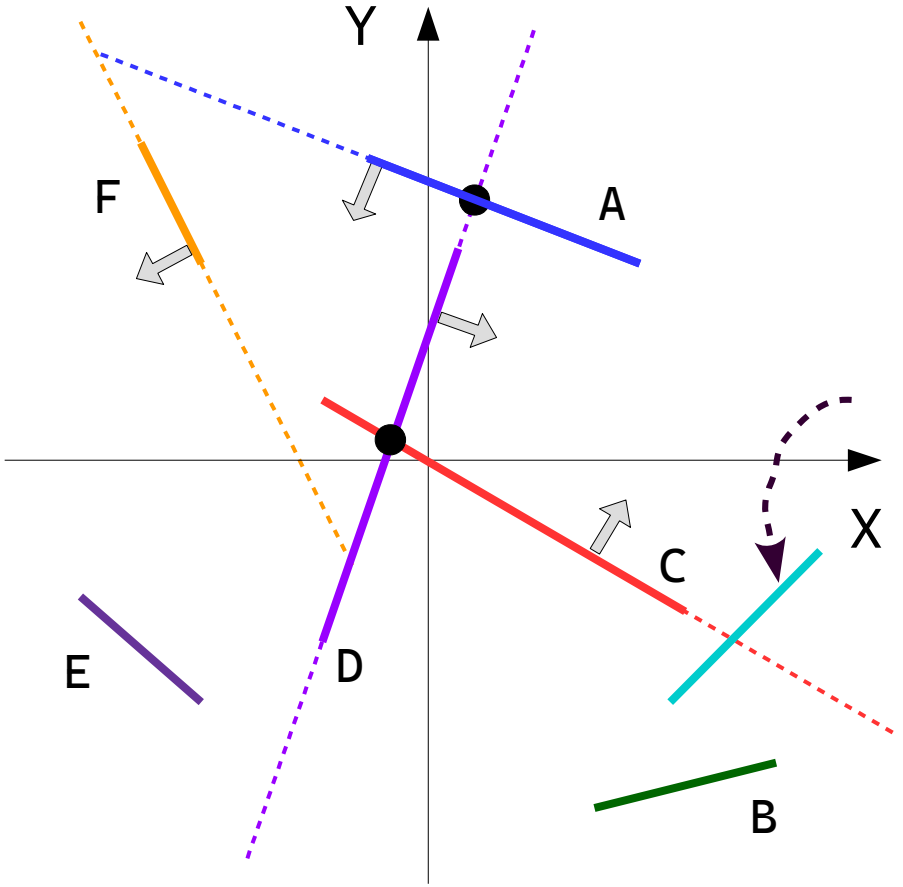
- '**x**' completely aligns on the split line of pivot segment,
  ... keep it in the same node of pivot



A variant of insertion

...

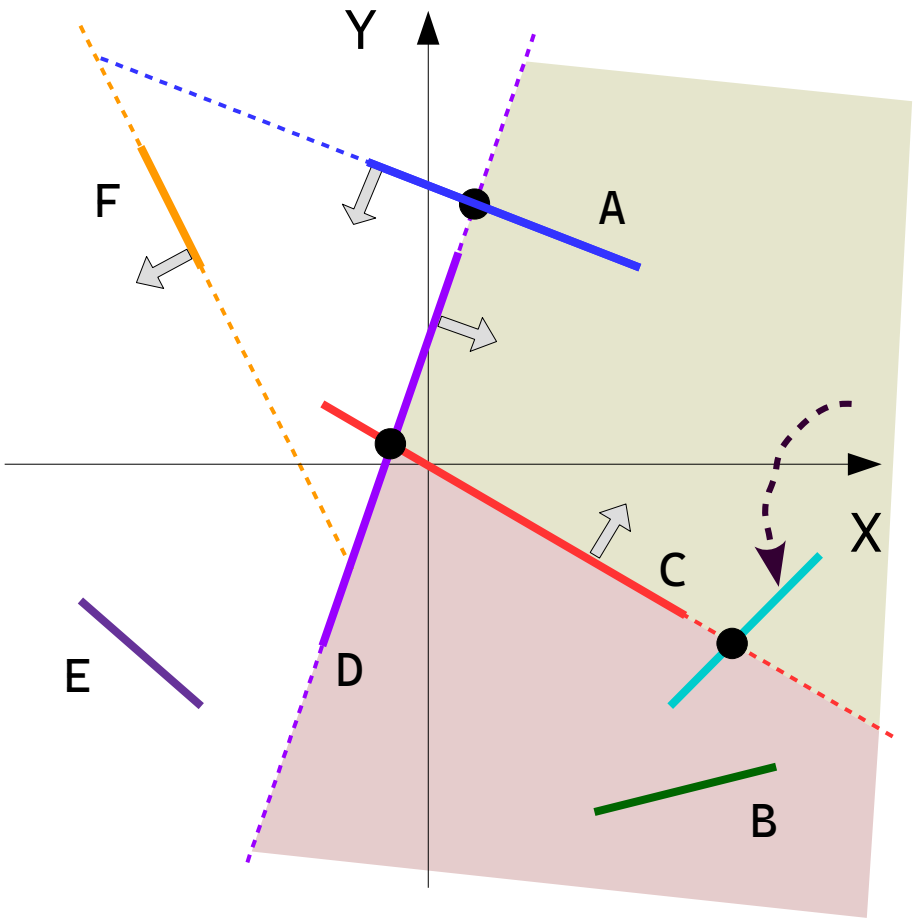
- '**x**' completely aligns on the split line of pivot segment,
  ... keep it in the same node of pivot
- '**x**' intersects split line,
  ... recursively <u>insert its parts</u> in the two subtrees.



A variant of insertion

...

- '**x**' completely aligns on the split line of pivot segment,
  ... keep it in the same node of pivot
- '**x**' intersects split line,
  ... recursively insert its parts in the two subtrees.



So we see that during insertion the new segment '**x**' can also be cut several times.

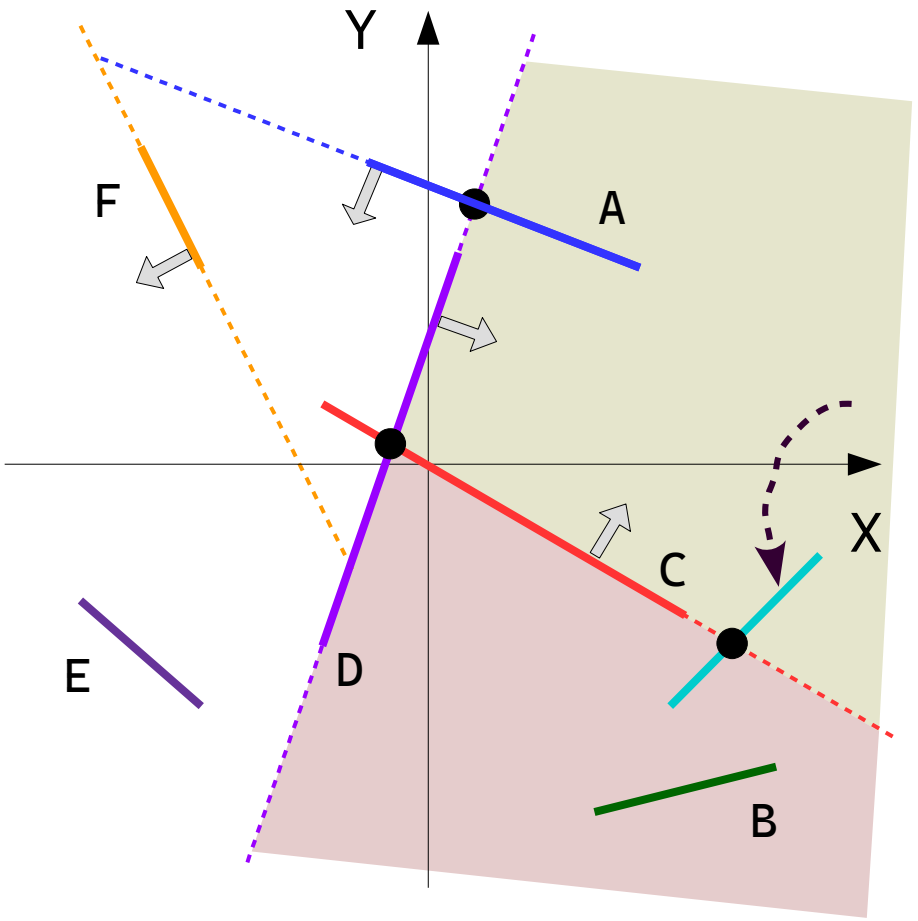
A variant of insertion

This means that time complexity of insertion is larger than **O(h)**,

... where '**h**' is height of the BSP tree.

Actually it is **O(kh)**,

... where '**k**' is number of parts of the new segment.



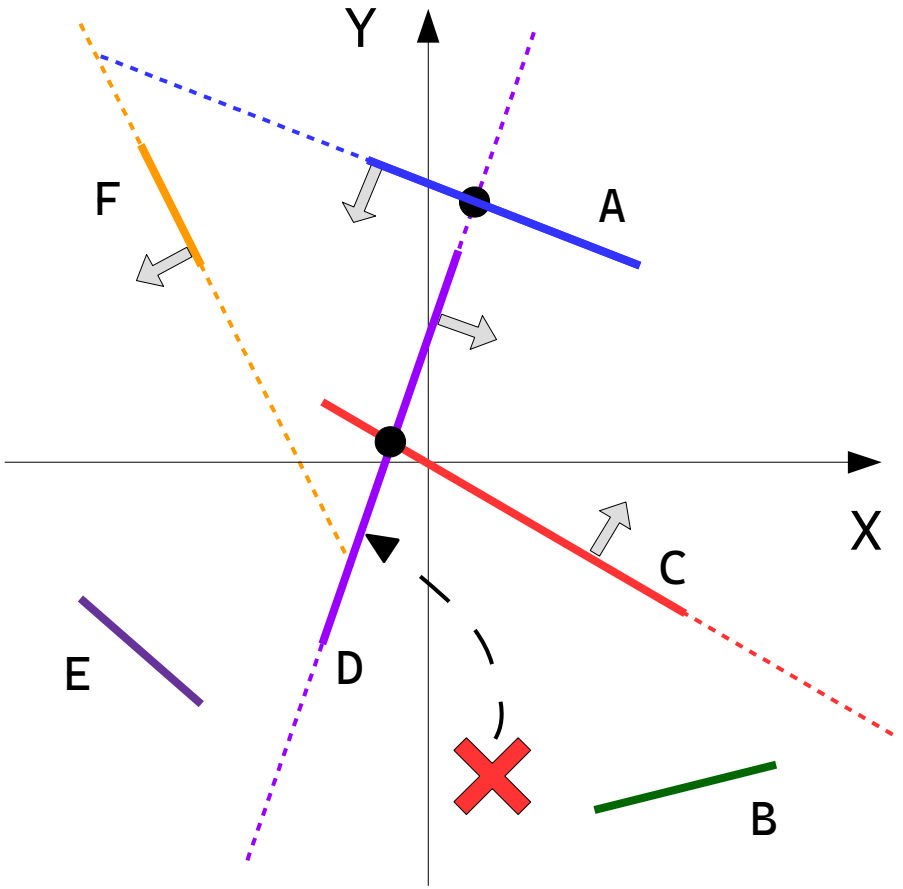
A variant of removal

We can't just remove a segment,

... because it will cause multiple unifications,

... then other segment must be put on its place in the tree,

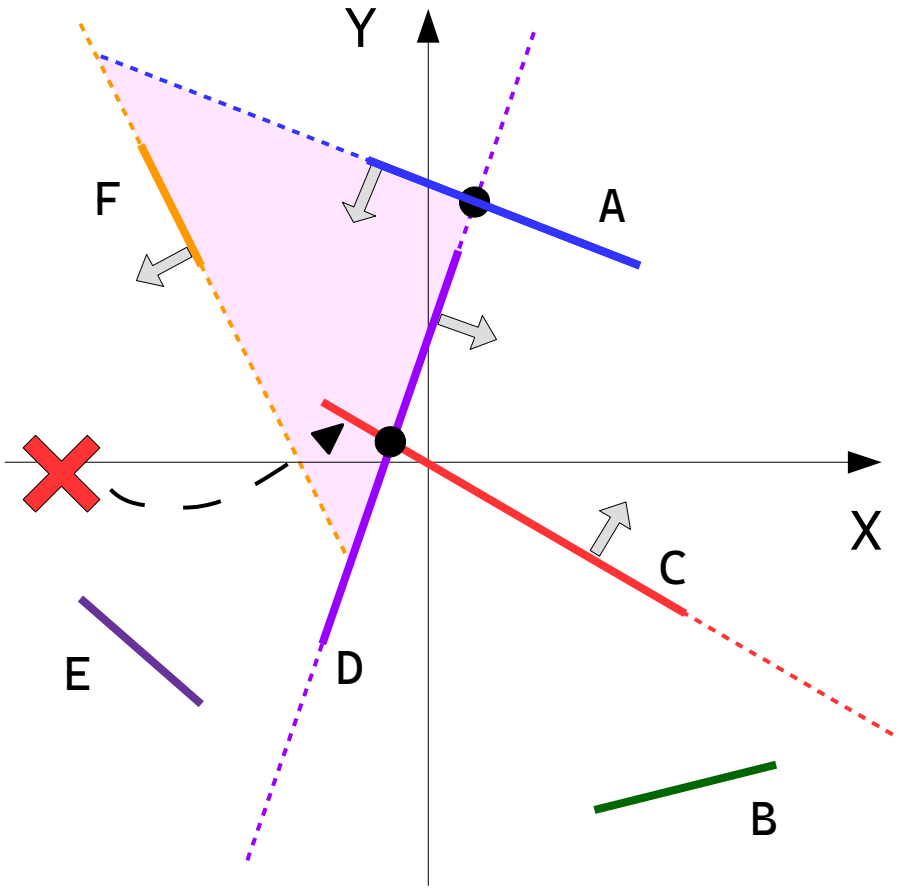
... which will itself cause multiple new cuts.



A variant of removal

Even if the segment is a leaf, we still can't remove it in a straightforward way,

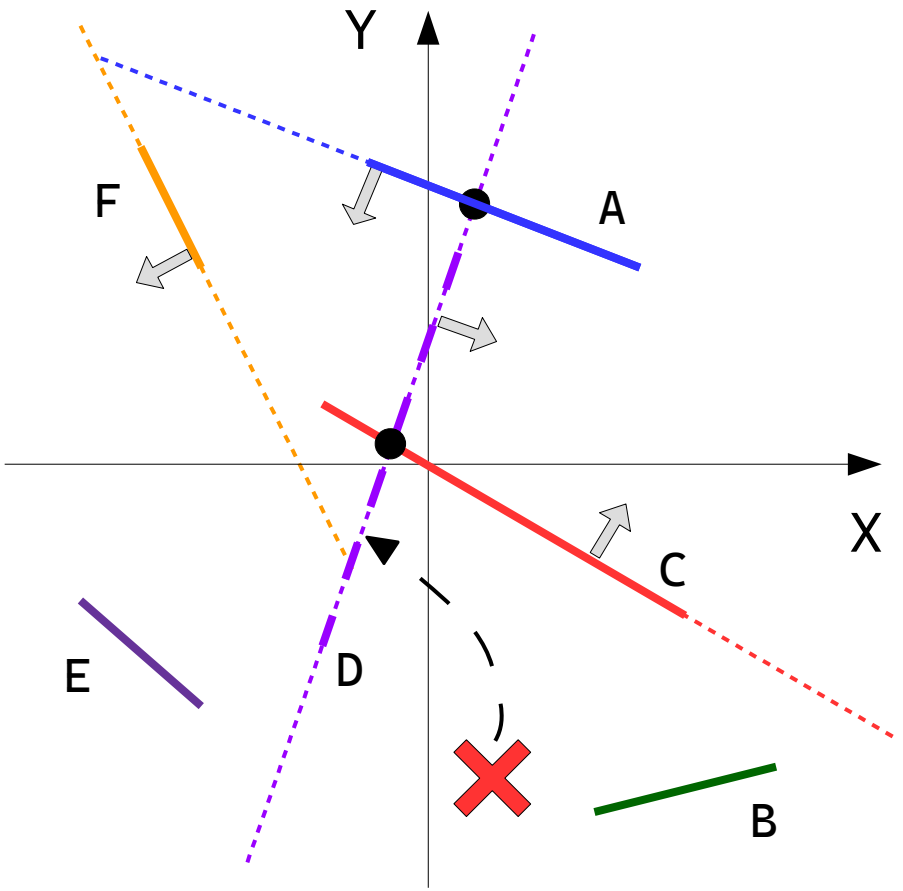
... because it might be present also in other parts of the BSP tree as a non-leaf node.



A variant of removal

So the simplest way is to mark the segment as deleted.

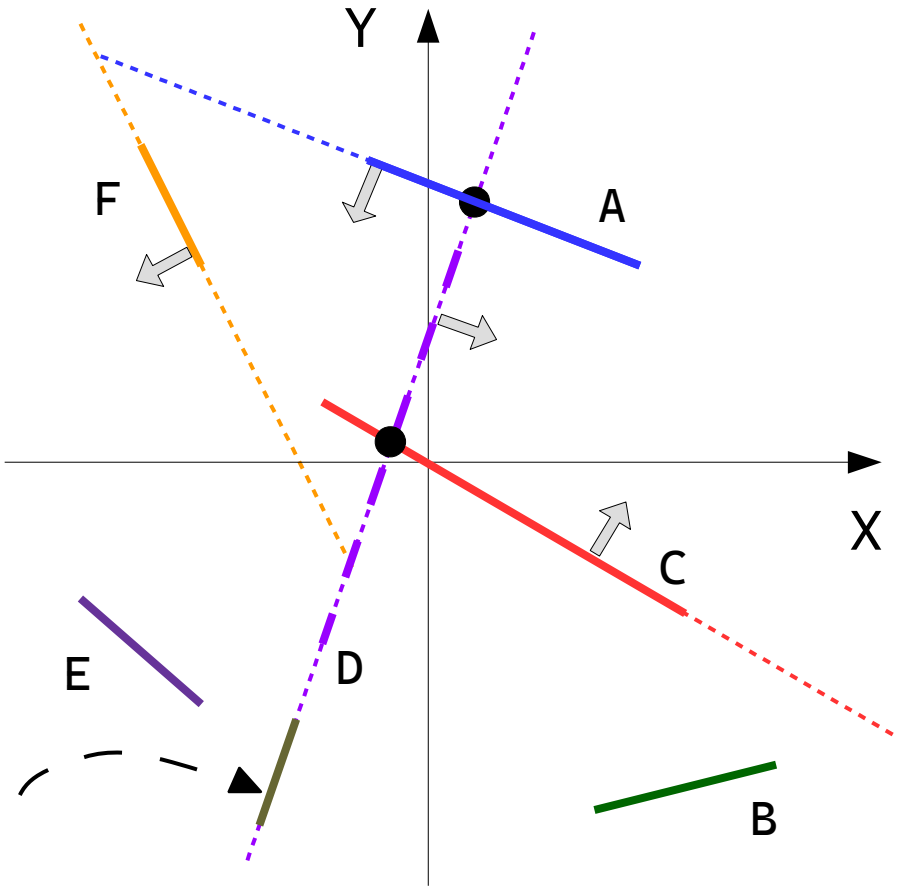
... this will preserve entire structure of the BSP.



A variant of removal

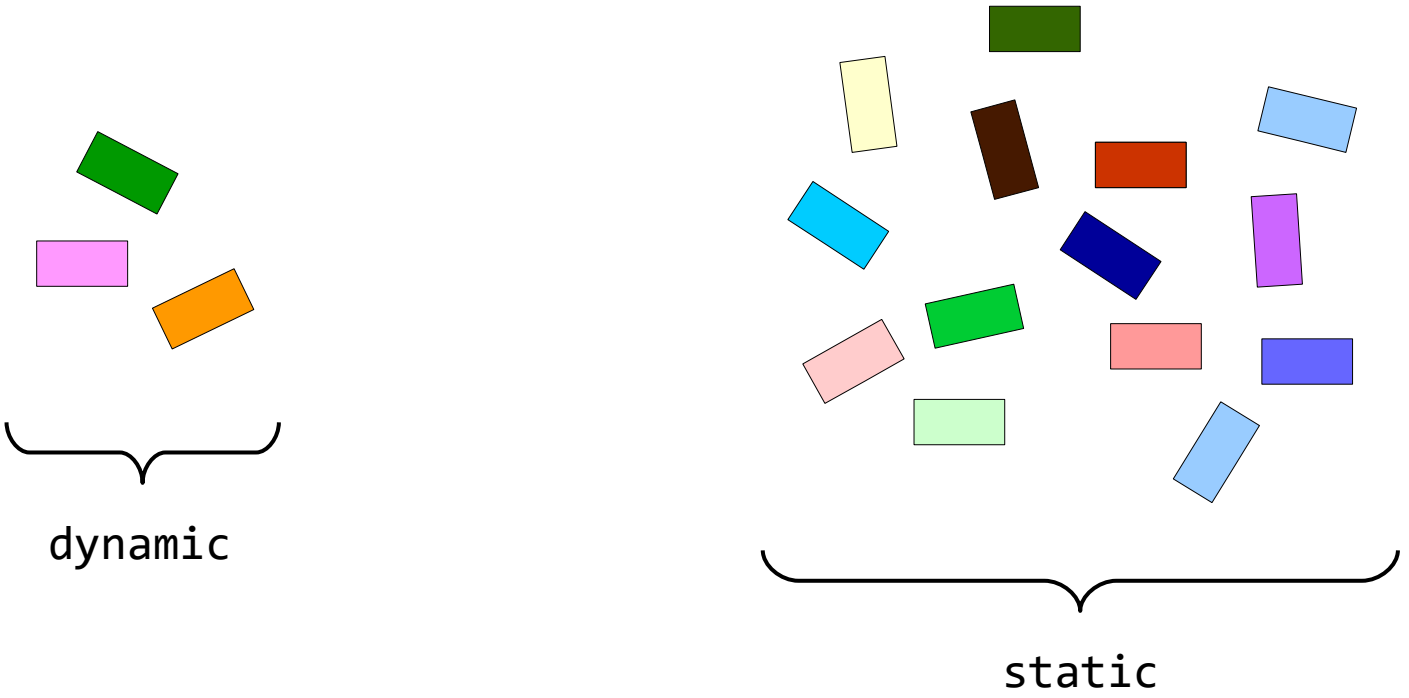
Later if another segment is being added, which is completely aligned with the removed one,

... it can be placed at the same node.



A variant of removal

In many practical applications we have large set of static objects and small set of dynamic objects.



Then we can construct BSP tree for the static objects, and add/remove dynamic objects upon need.

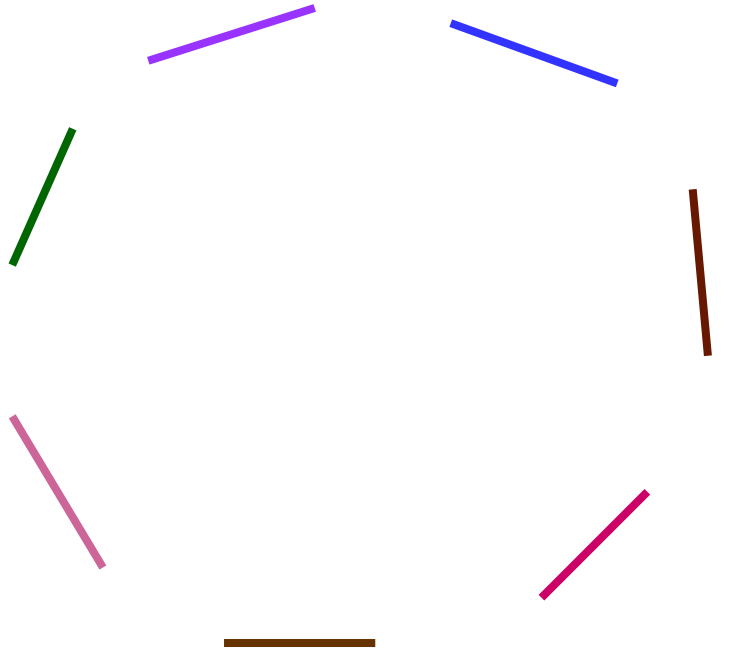
Batch construction

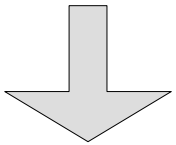
Question: How do you think, is it always possible to construct a balanced BSP tree?

Batch construction

Question: How do you think, is it always possible to construct a balanced BSP tree?

Answer: Not always.

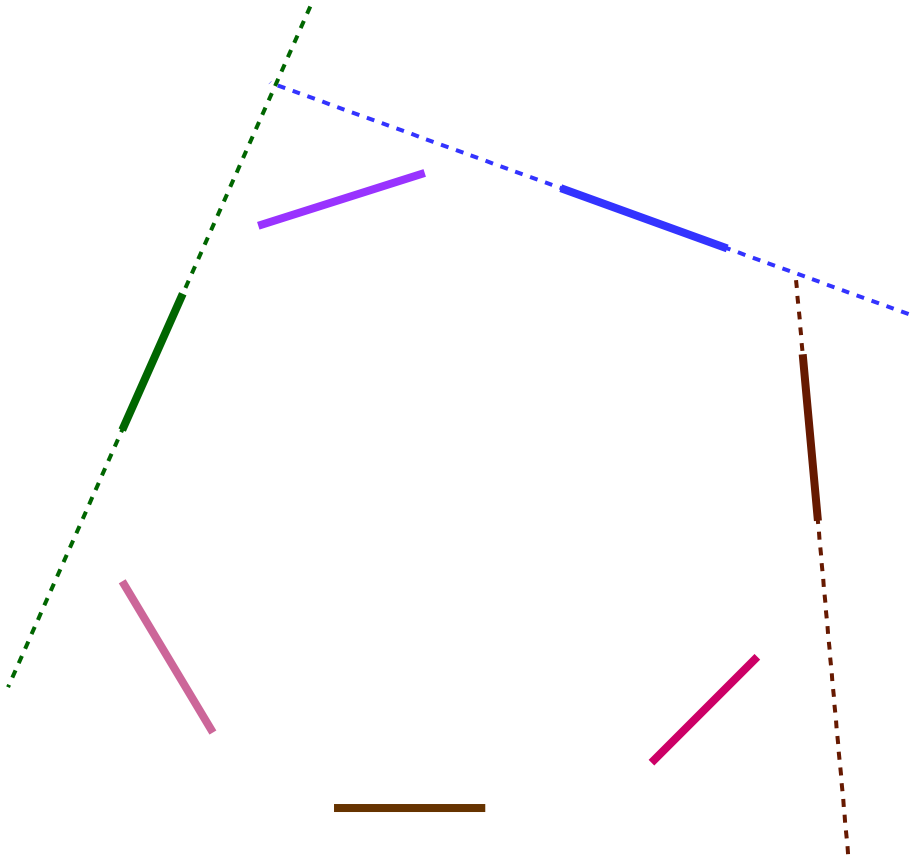




Batch construction

Question: How do you think, is it always possible to construct a balanced BSP tree?

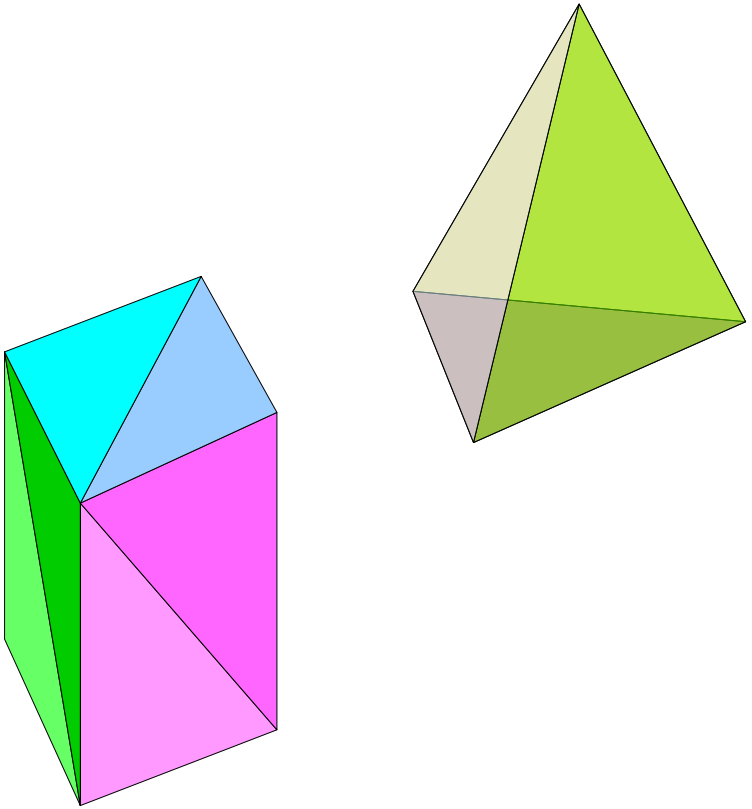
Answer: Not always.



BST in 3D space

What is being changed in BSP if we move to 3D space?

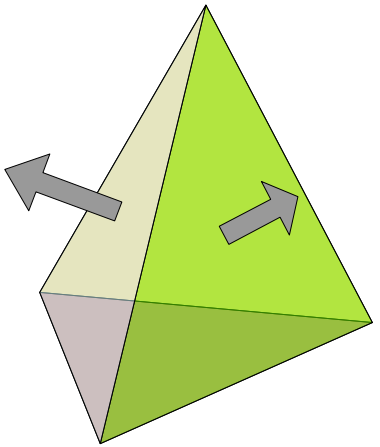
- Objects are now triangles, and not segments.



BST in 3D space

What is being changed in BSP if we move to 3D space?

- Objects are now triangles, and not segments.
- Every triangle has its front and back faces.

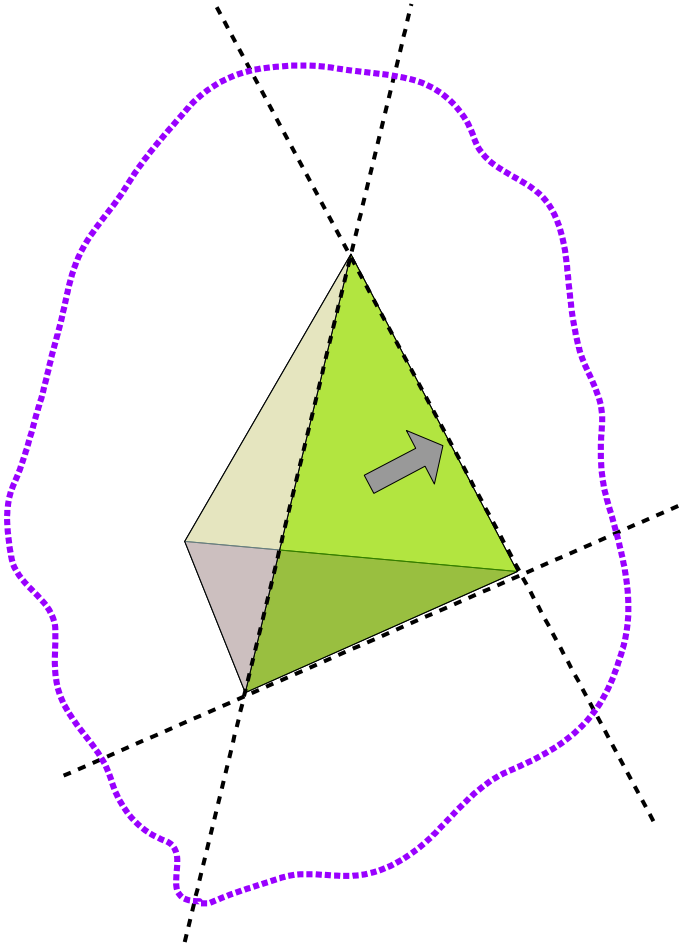


BST in 3D space

What is being changed in BSP if we move to 3D space?

- Objects are now triangles, and not segments.
- Every triangle has its front and back faces.
- Each triangle partitions its space in **2** parts,

  ... still cutting many other objects in **2** parts.



More realistic rendering

Scene rendering is OK, as long as we want speed, and can compromise on image quality.

... this is often the case of computer games.

But there are situations when we want the most realistic images, and can wait for hours for them to be generated.

... such need arises when rendering 3D industrial designs.

More realistic rendering

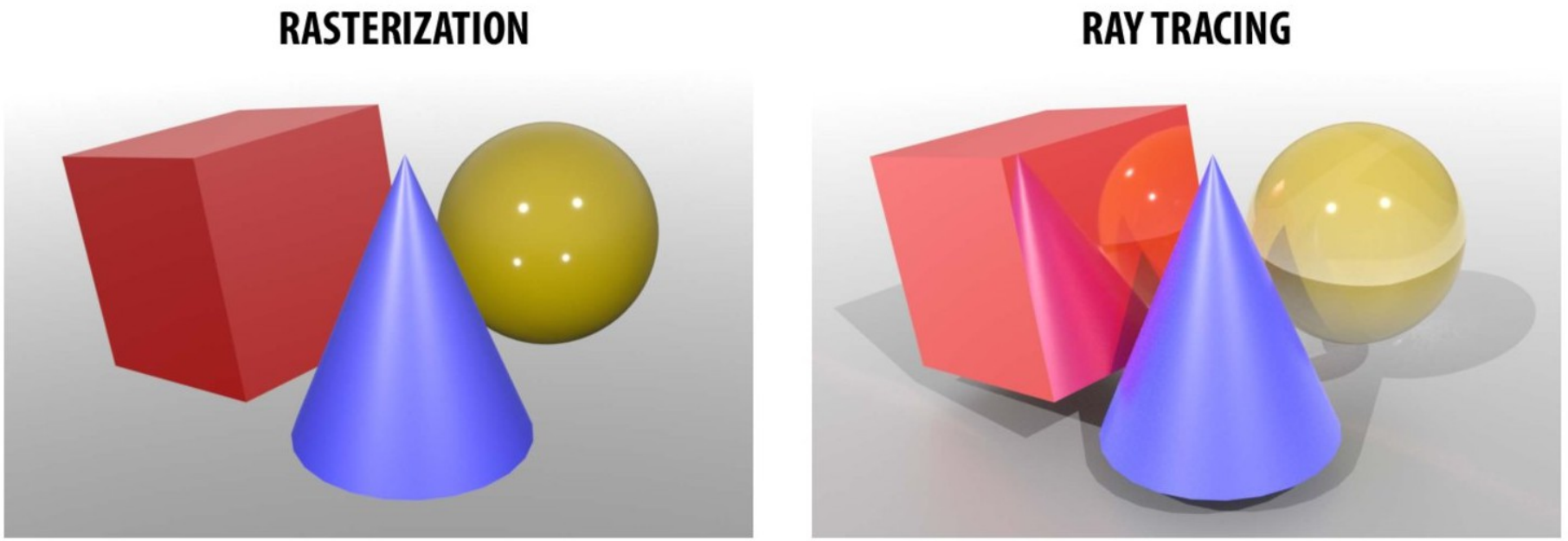
You might ask:

"what's wrong with traditional rendering?"

Rasterization generally doesn't take into account that presence of some objects might alter look of another objects. This includes:

- one object dropping shadow on another,
- one object being reflected at the surface of another,
- reflection surfaces are not always plain.

More realistic rendering



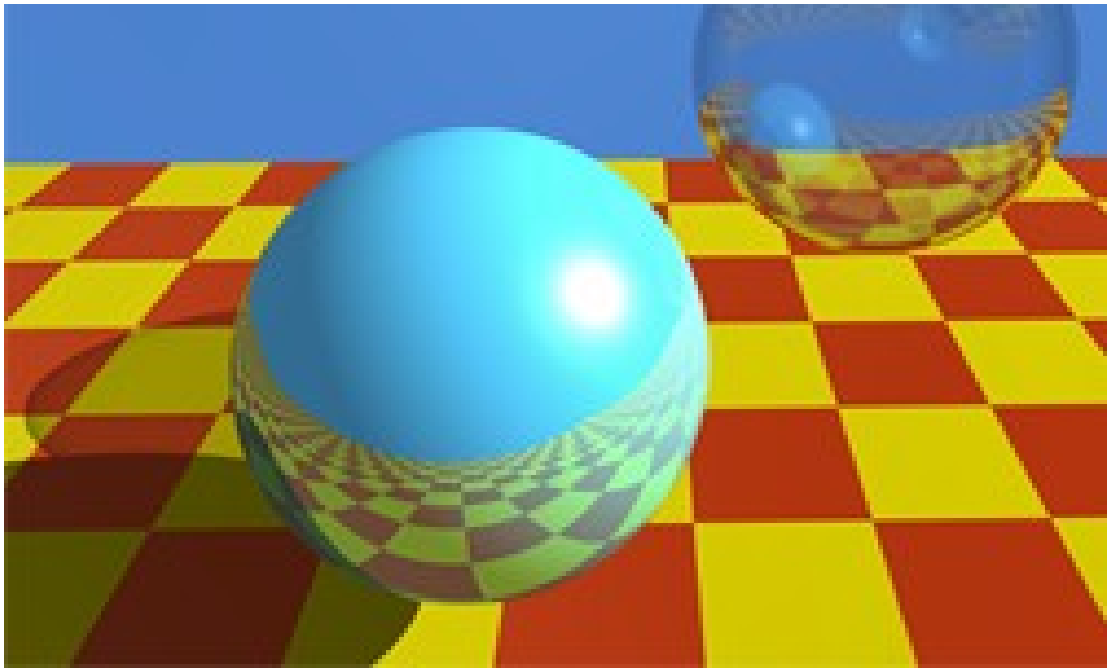
Here is a difference between rasterized picture and more realistic render. We see how:

- cone drops shadows,
- ball being reflected on the cube.

More realistic rendering

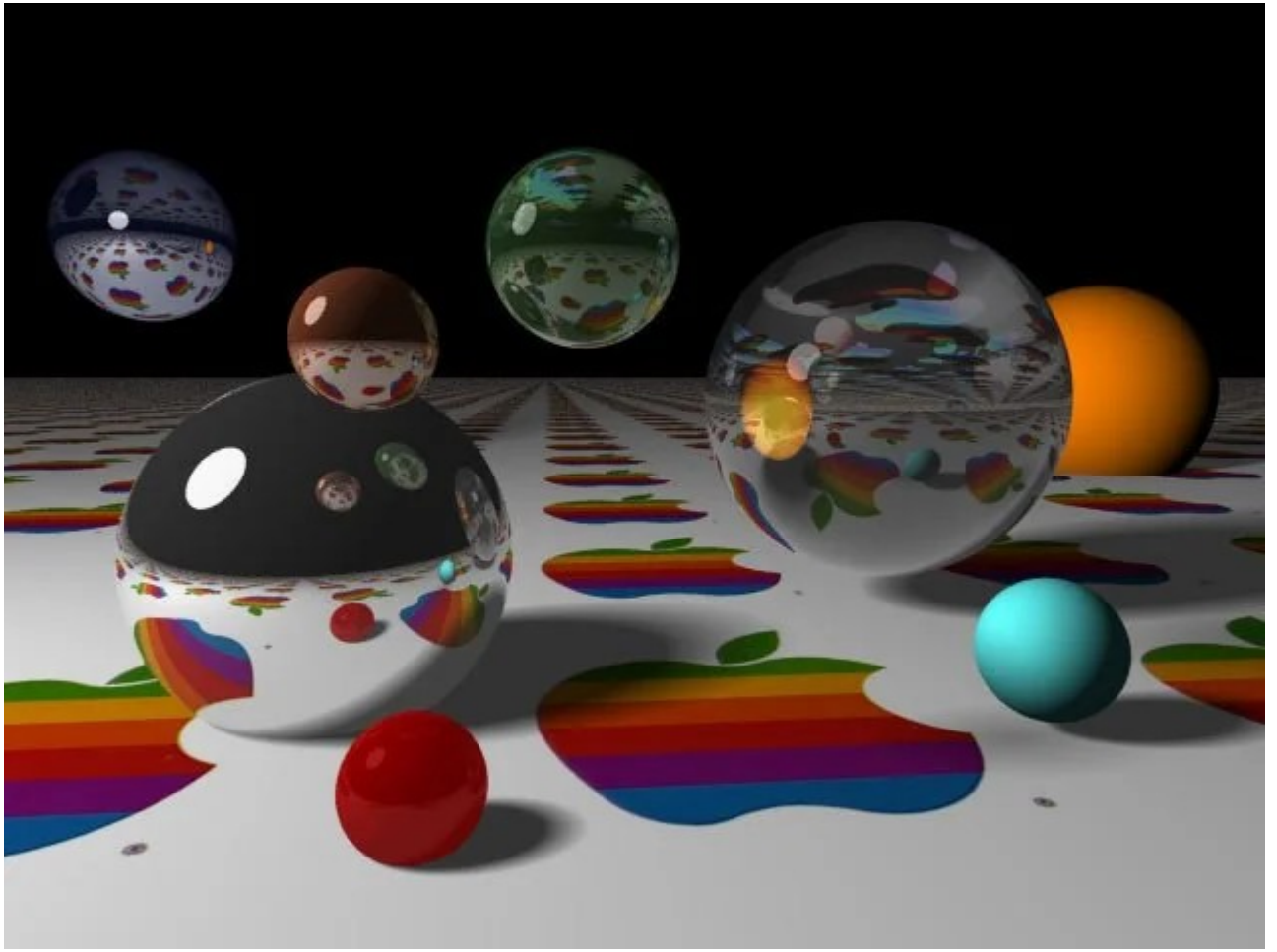
Another example with a lot of reflections.

More realistic rendering



During reflection, image of objects can be distorted.

More realistic rendering



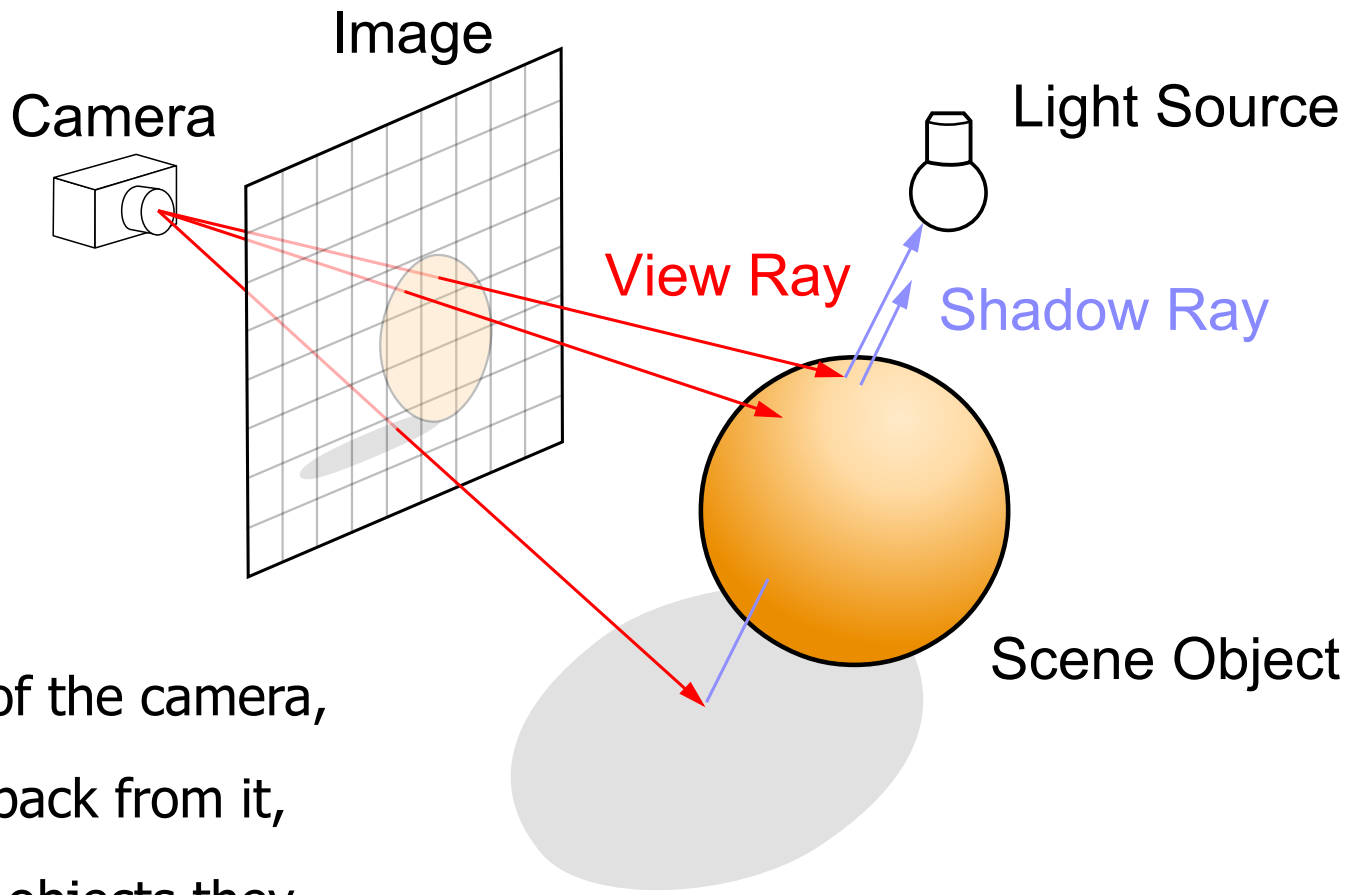
And finally, the same object can be reflected several times.

Ray tracing

One method of achieving such realistic images is called "Ray tracing".

- It simulates the principle by which "images" are generated in real world.
- We know that our eyes percieve the rays which fall on them.
- So in order to know those rays and their colors, in ray tracing we do backward tracing of rays.

Ray tracing



Knowing position of the camera,

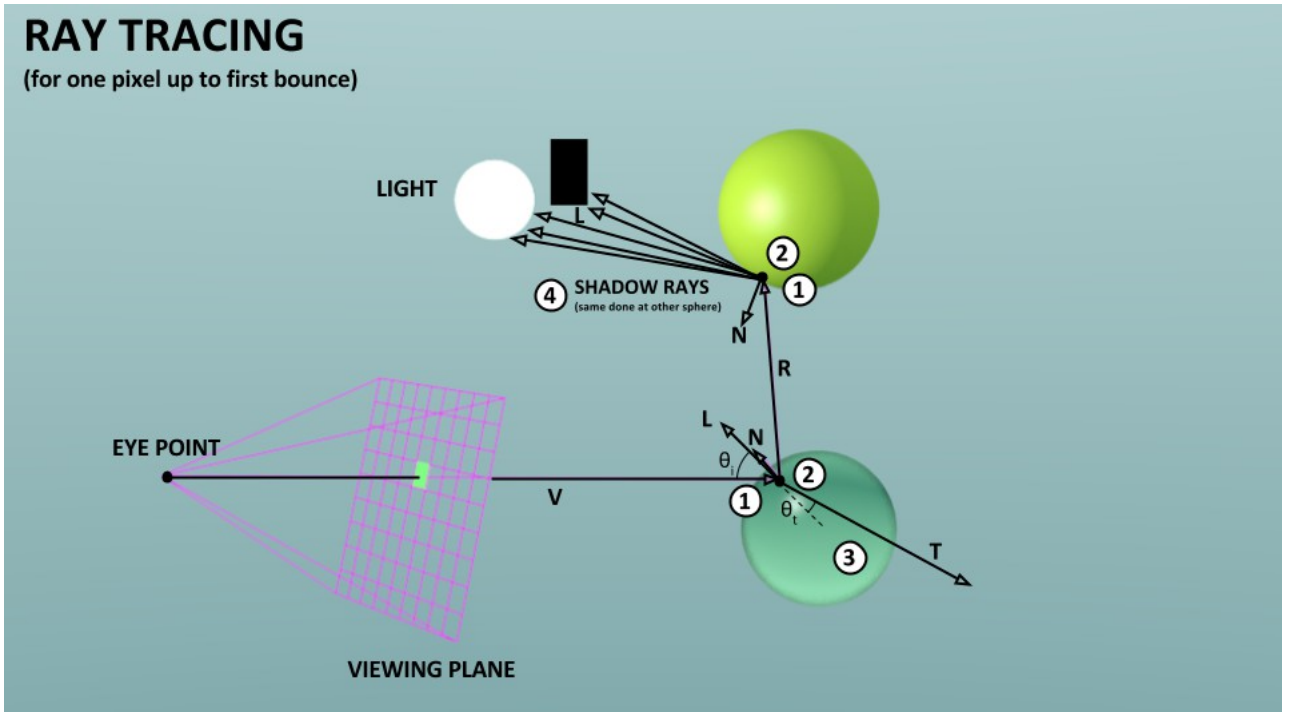
- we trace rays back from it,
- and look what objects they will intersect.

Color and properties of those objects affect the final color of that pixel.

Ray tracing

Let's note that depending on surface type, one ray can be split into several other rays,

... so the overall number of casted rays will probably increase.



Ray tracing

It is not difficult to do all those geometrical calculations.

The only difficulty is that for example, for **FullHD** monitor **(1920*1080)** we will have around **2M** rays,

... each of which might be split into several rays, during the case.

More than that, if we want to display a video with **60 FPS**, we must trace around

$$\mathbf{2*10^6*60 = 0.12*10^9}$$

rays per second!

... which is the reason why ray tracing is rarely used in practice.

Ray tracing

Question: Can you suggest some ideas on how to efficiently trace rays?

... either in 2D or in 3D.

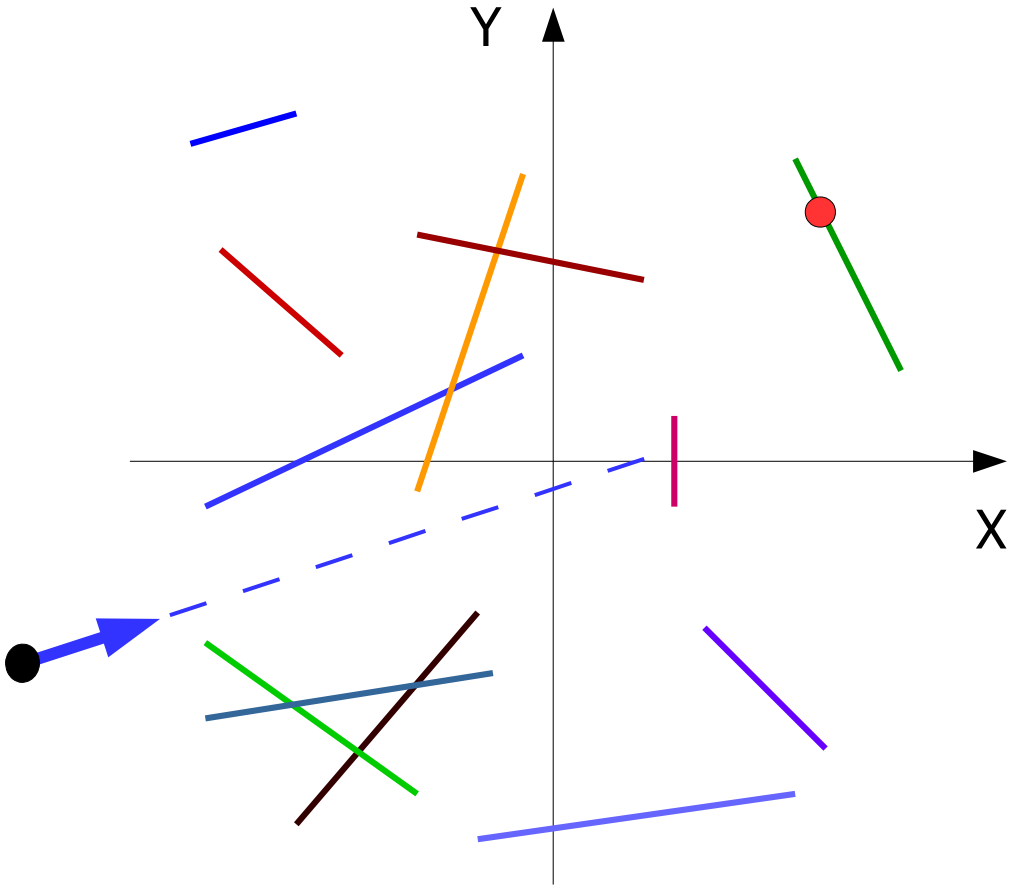
BSP in ray tracing

Let's understand if BSP can help us in tracing rays.

We will address the problem for 2D case, as 3D case is again solved similarly.

So the problem is:

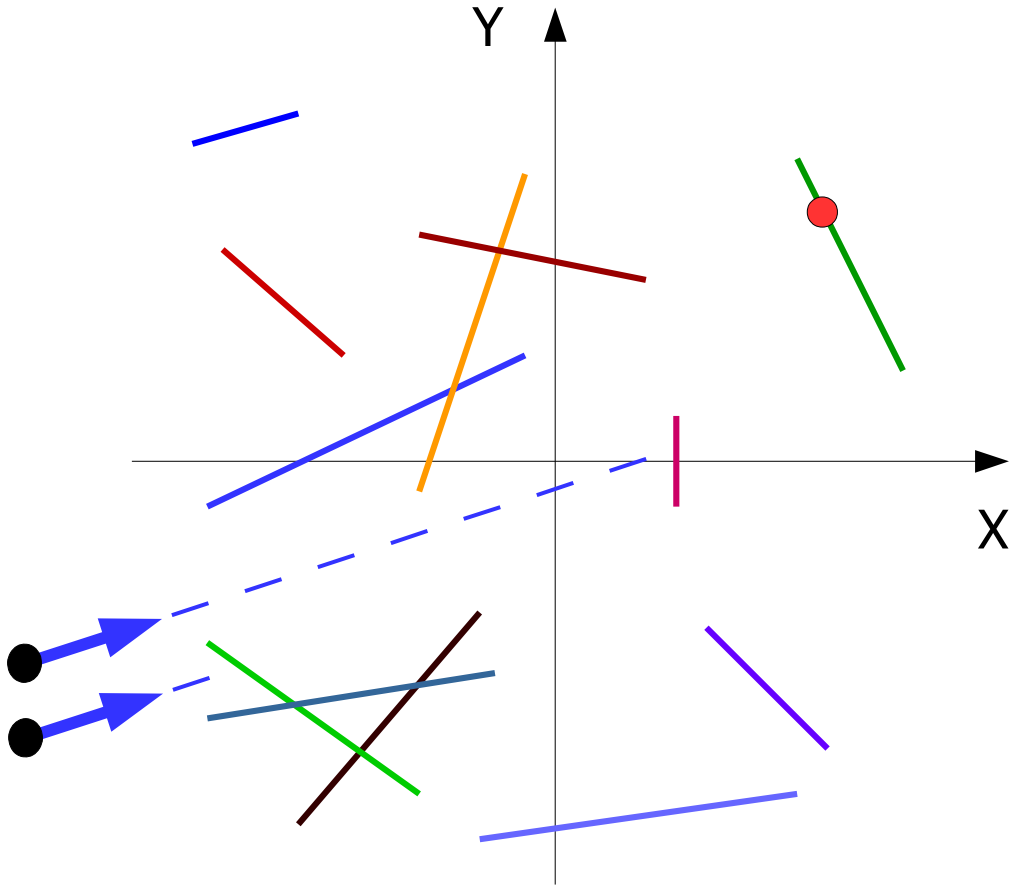
- having a static scene,
- and a ray coming from some point,
- determine the first object which will be hit by it.



BSP in ray tracing

At first let's pay attention that it is not an easy problem at all:

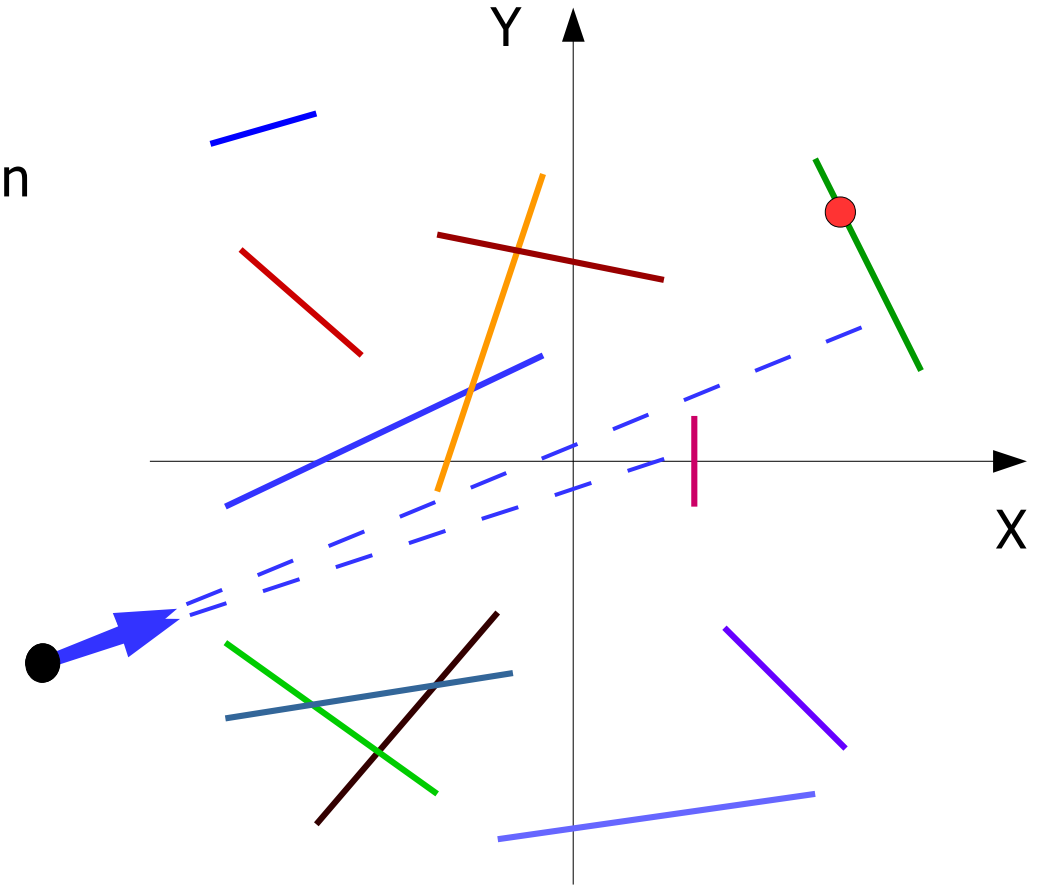
- Slight change in the start point can bring to a completely different result.



BSP in ray tracing

At first let's pay attention that it is not an easy problem at all:

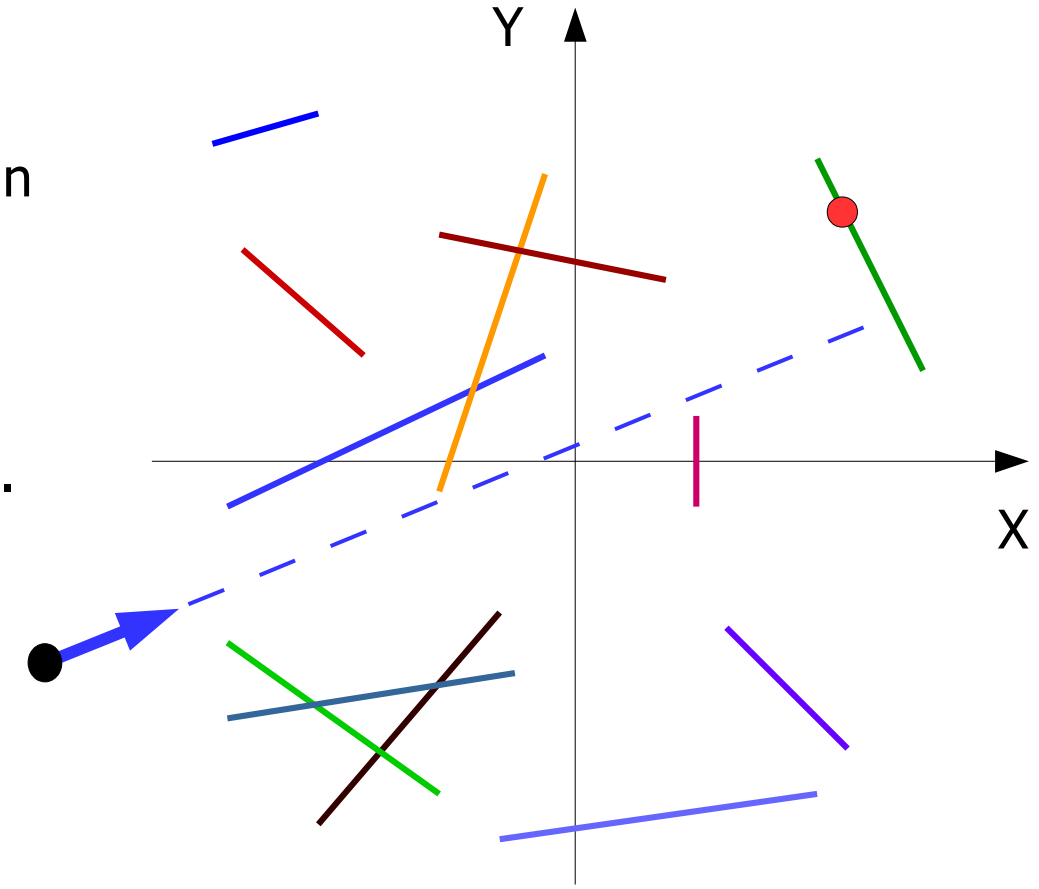
- Slight change in the start point can bring to a completely different result.
- Slight change in the angle can also bring to a completely different result.



BSP in ray tracing

At first let's pay attention that it is not an easy problem at all:

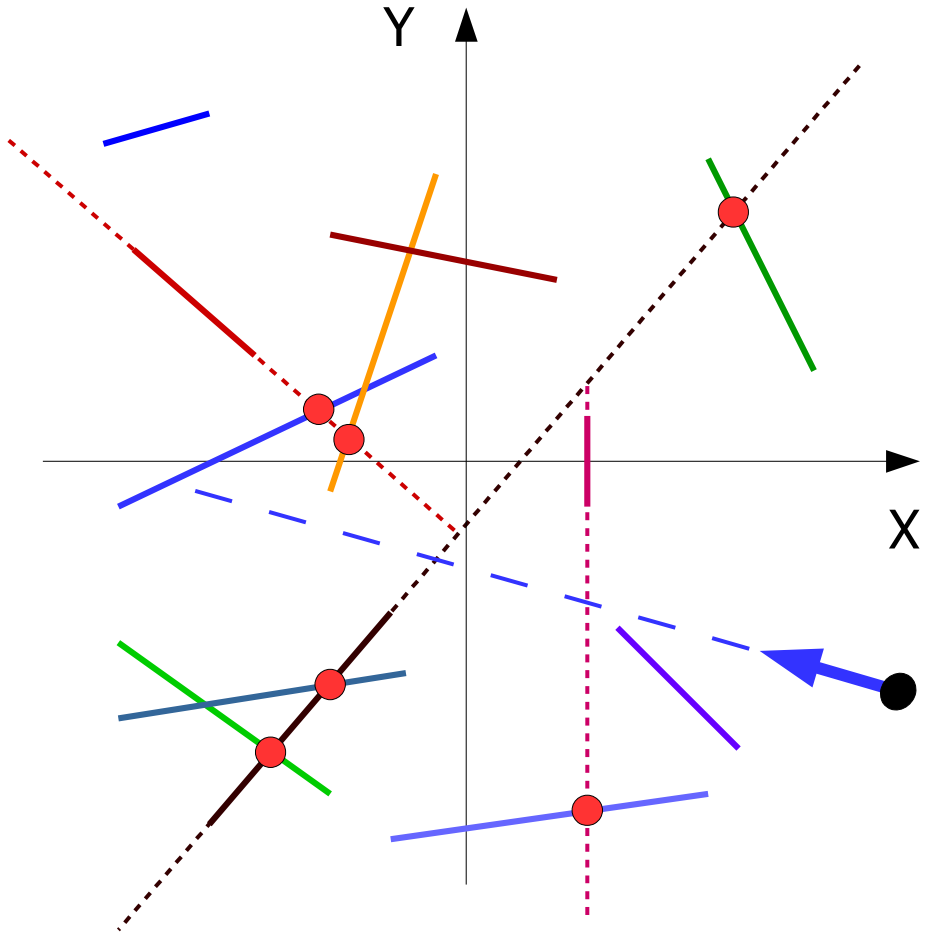
- Slight change in the start point can bring to a completely different result.
- Slight change in the angle can also bring to a completely different result.
- A ray can get too close to objects, without hitting them.



BSP in ray tracing

BSP efficiently solves the ray tracing problem.

... we will use the same partitioned scene.



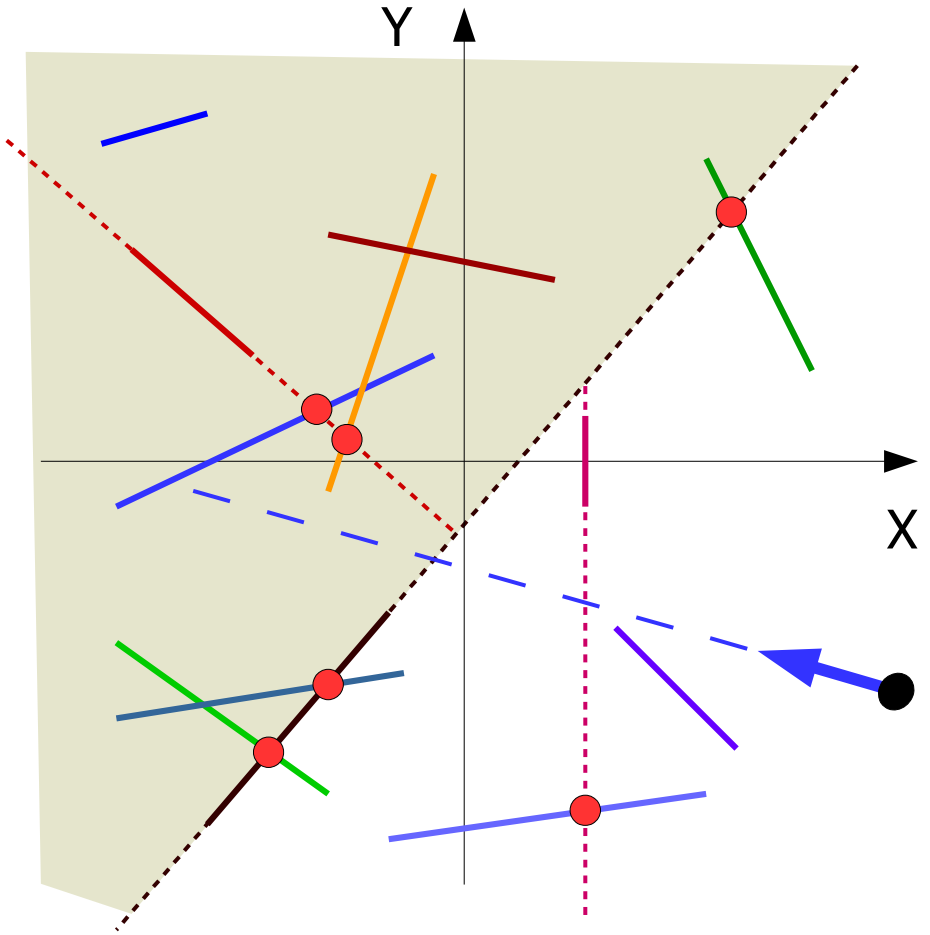
What does mean to cast a ray in such scene?

- The start point is in one of the **2** halves.
- Ray might hit an object from its half or from the other half.
- It might even hit no object at all.

BSP in ray tracing

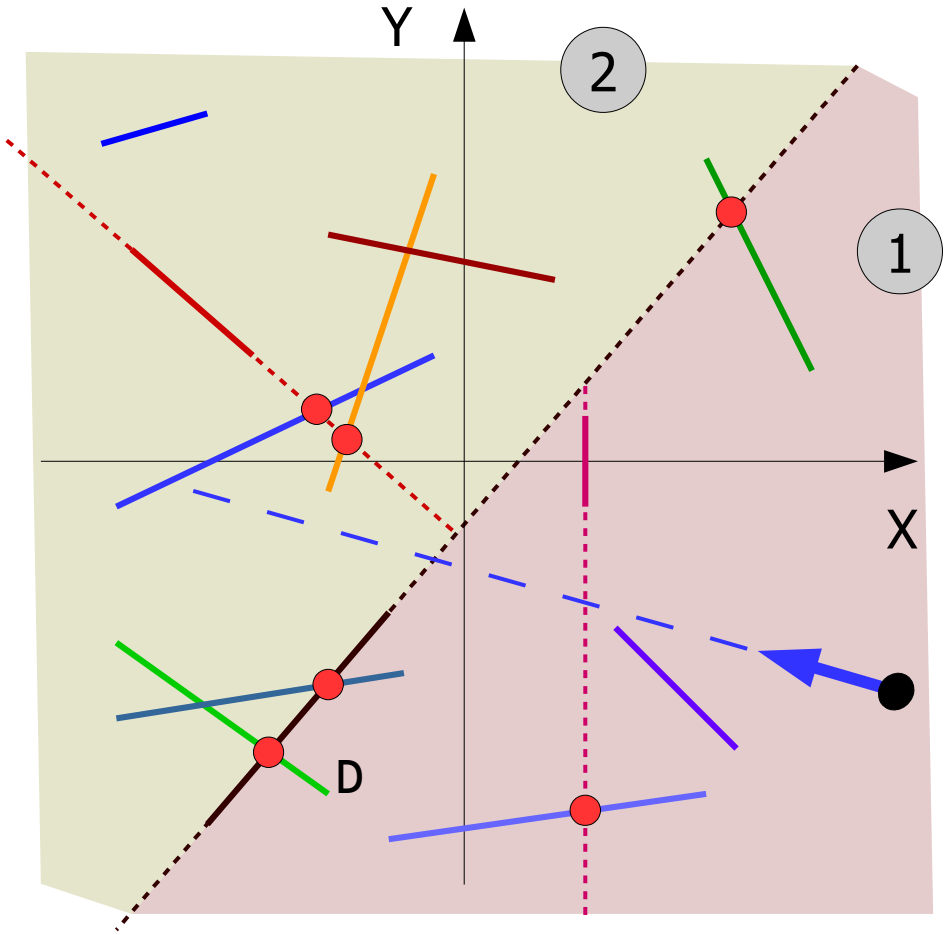
What does mean to cast a ray in such scene?

- The start point is in one of the **2** halves.
- Ray might hit an object from its half or from the other half.
- It might even hit no object at all.
- But it will hit an object from other half only if it hasn't hit any in its half.



BSP in ray tracing

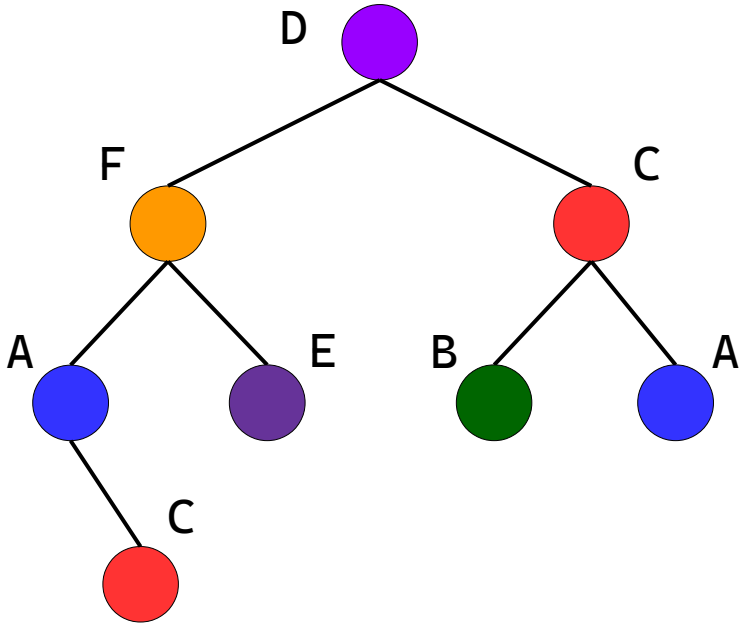
This brings us to the idea that at first we can check one half, and only after, if needed, we will check the other half.

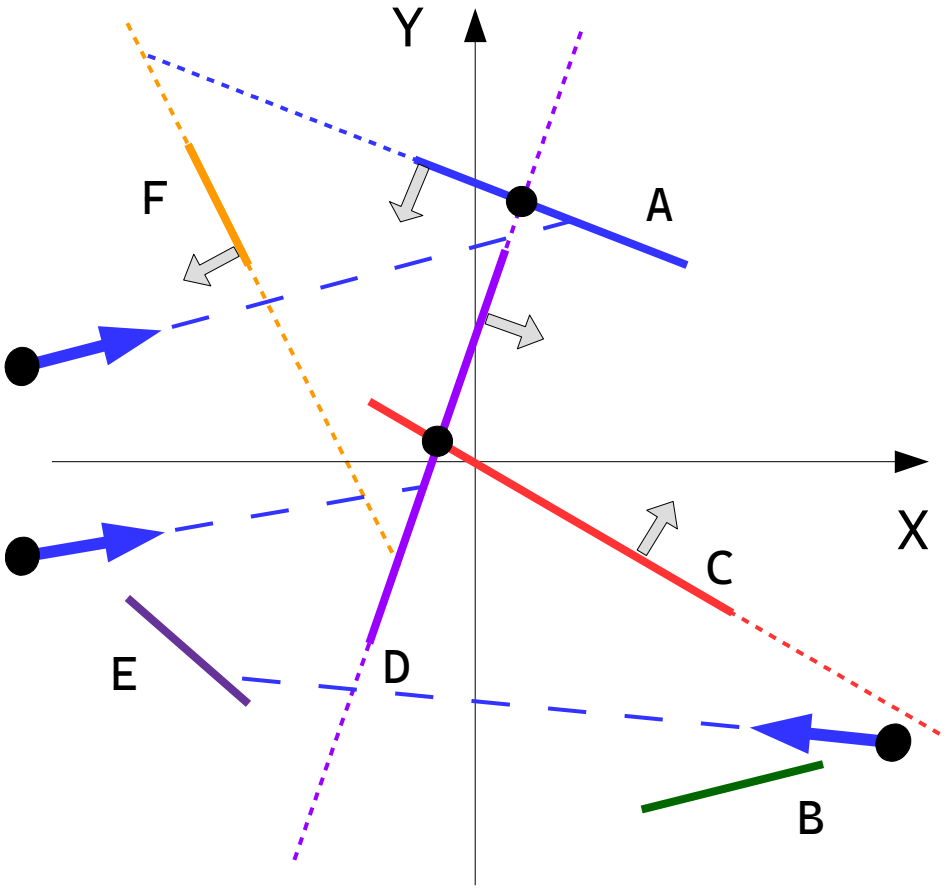


To be continued... vvv

Exercise

Cast the following rays, and describe the order of traversed subtrees.





Presentation writer: Tigran Hayrapetyan

Lecturer | Programmer | Researcher

www.linkedin.com/in/tigran-hayrapetyan-cs/

Thank you!

Binary Space Partitioning