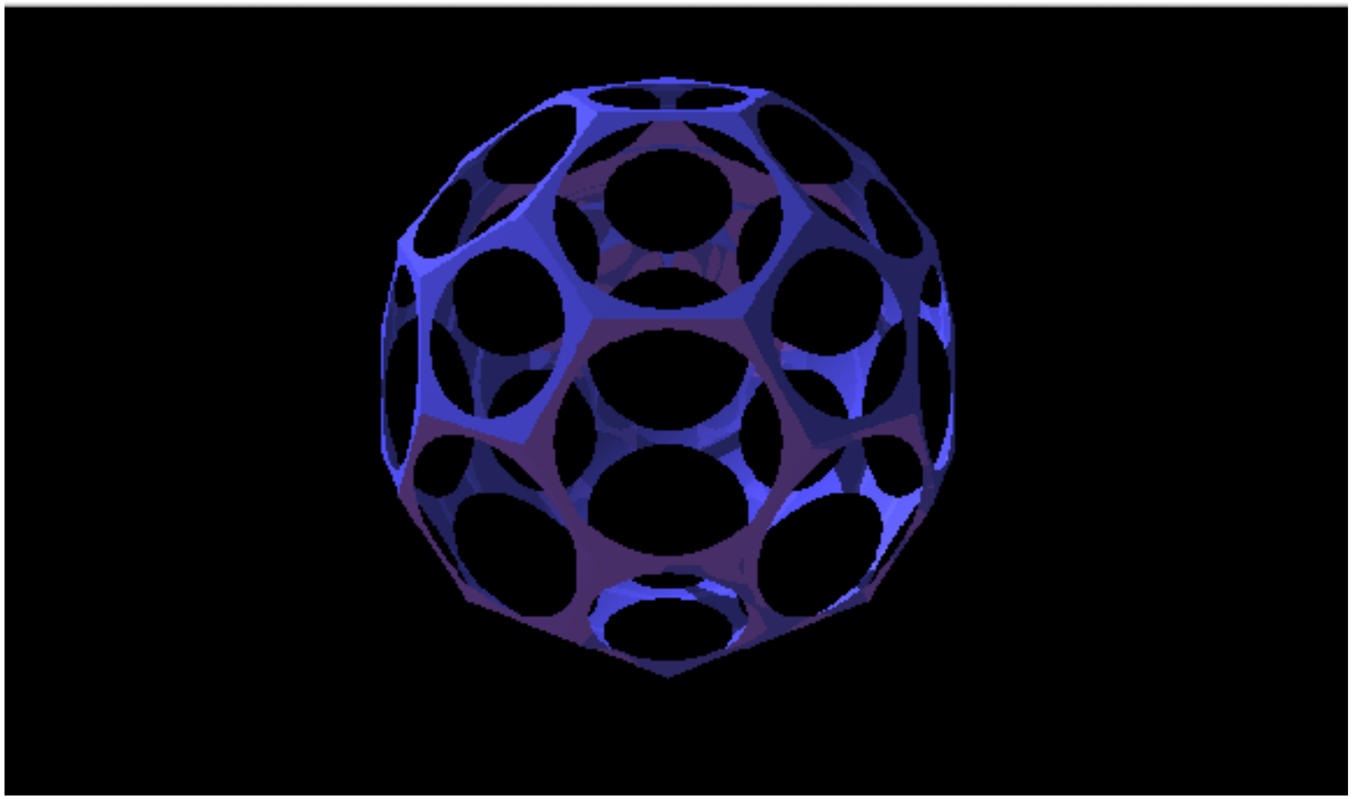
Ray Tracer extensions (part II)

3D Computer Graphics (Lab 10)

Example

Can we render this image with our current rendering framework?

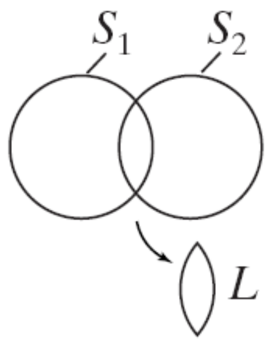


Constructive Solid Geometry

Constructive Solid Geometry

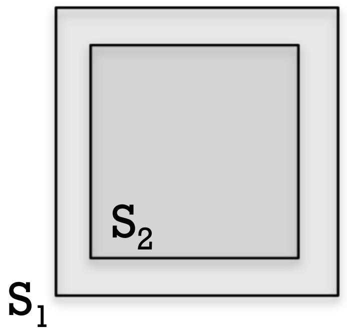
- Constructive Solid Geometry (CSG) is a method to create complex shapes by means of combining simple shapes.
- Arbitrary complex shapes are defined by boolean operations on simple shapes.
- Boolean operators
  - Union
  - Intersection
  - Difference

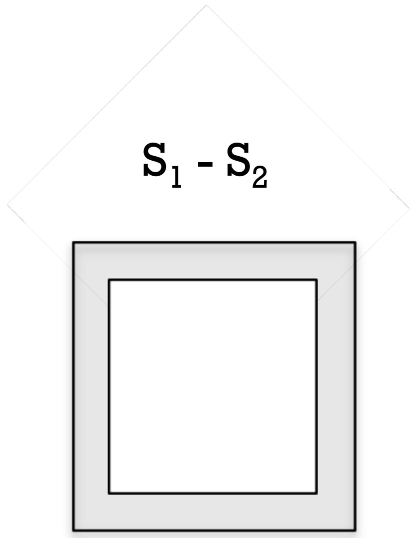
Example: a lens shape constructed as the intersection of two spheres.



- The resulting shapes are called compound, boolean or CSG objects.
- We will add support for the difference operator in this Lab.

Example





sdl file

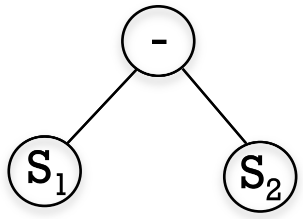
```
square
scale 0.75 0.75 0.75 square
```

sdl file

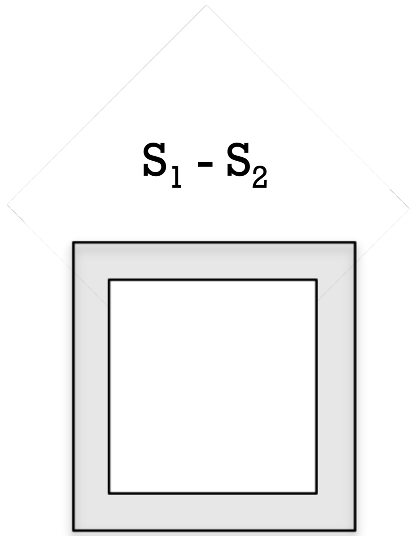
```
difference
square
scale 0.75 0.75 0.75 square
```

Example

How do we represent this boolean object?



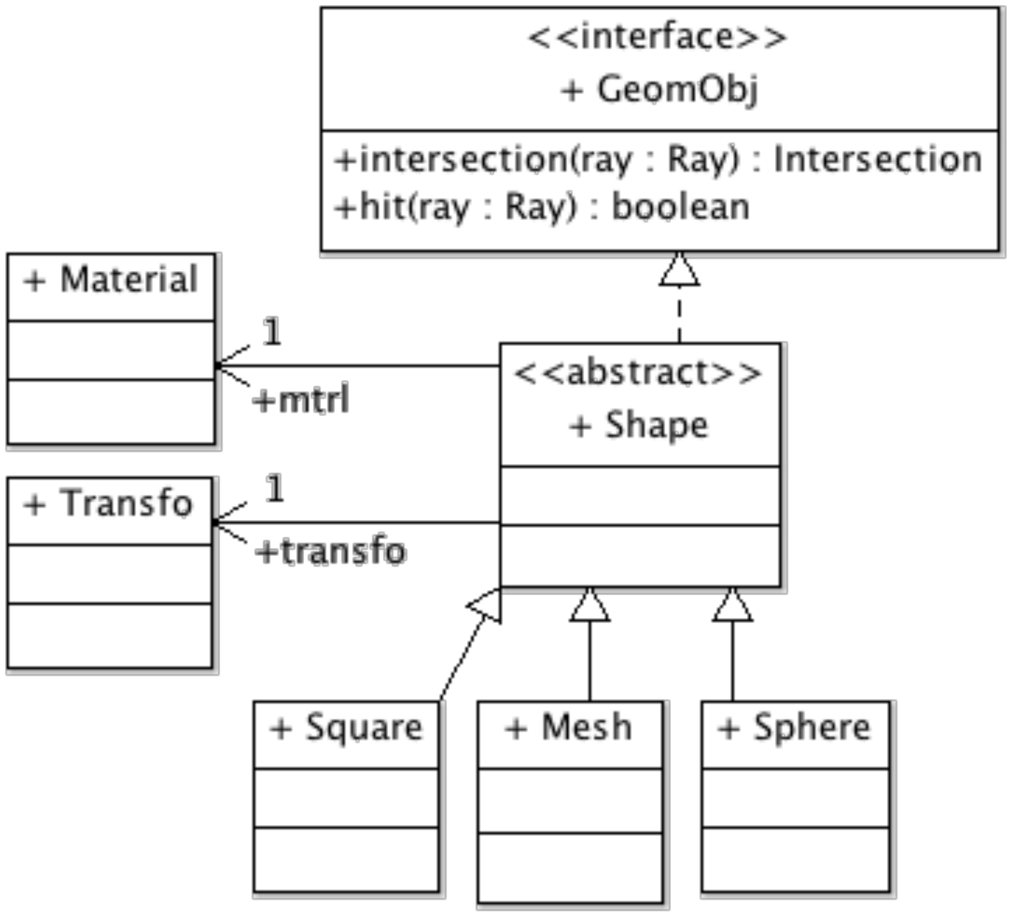
a binary tree structure



sdl file

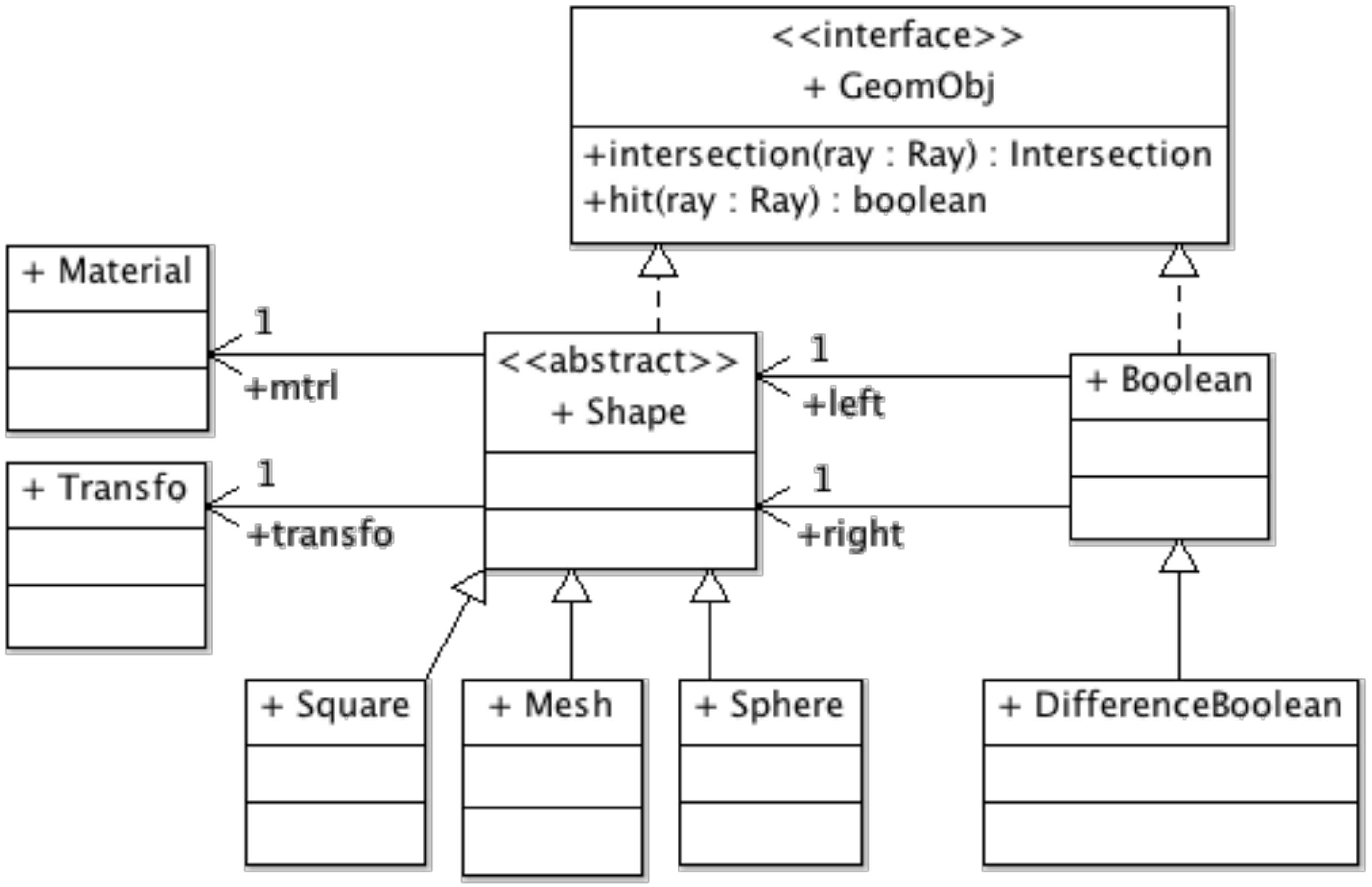
```
difference
square
scale 0.75 0.75 0.75 square
```

Current 3D object representation

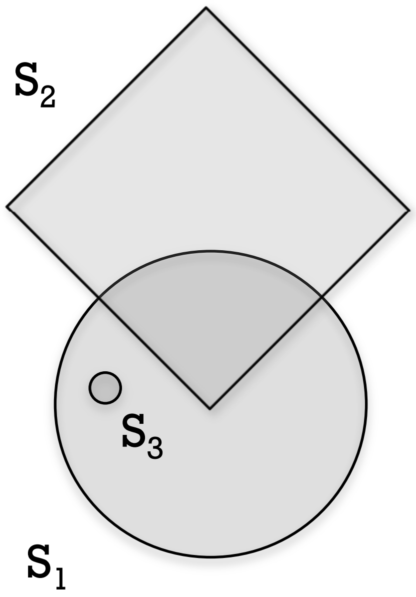


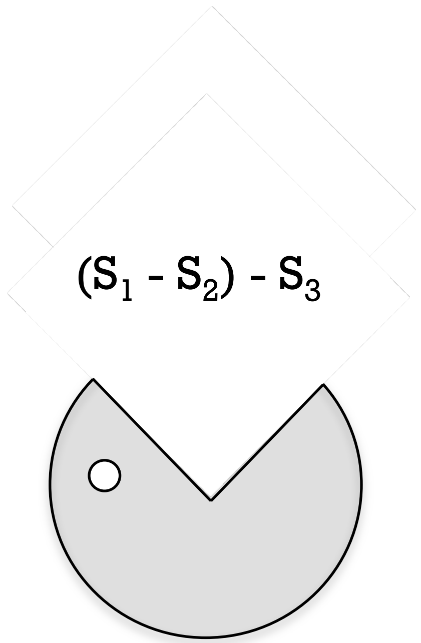
How do we add support for boolean objects?

Support for boolean objects



Example 2





sdl file

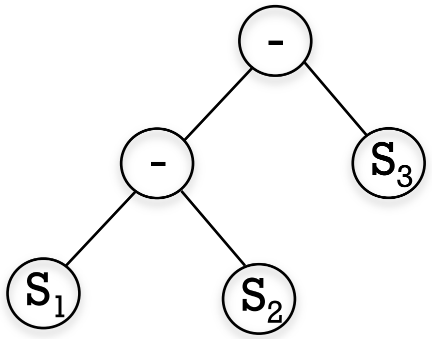
```
sphere
push rotate 45  0 0 1 translate 1 1 0 mesh cube.txt pop
translate -0.75 0.2 0 scale 0.1 0.1 0.1 mesh cylinder.txt
```

sdl file

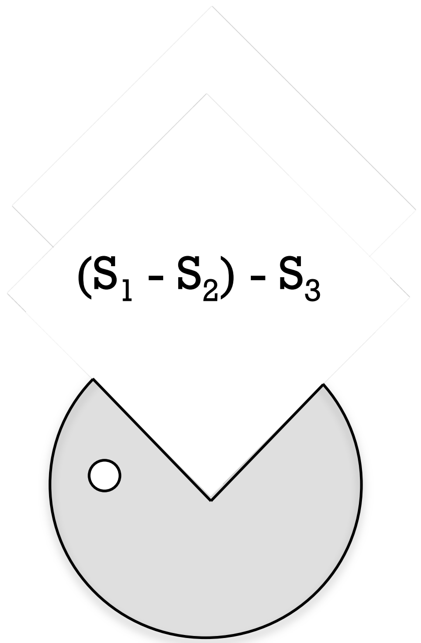
```
difference
difference
sphere
push rotate 45  0 0 1 translate 1 1 0 mesh cube.txt pop
translate -0.75 0.2 0 scale 0.1 0.1 0.1 mesh cylinder.txt
```

Example 2

How do we represent this boolean object?



a binary tree structure

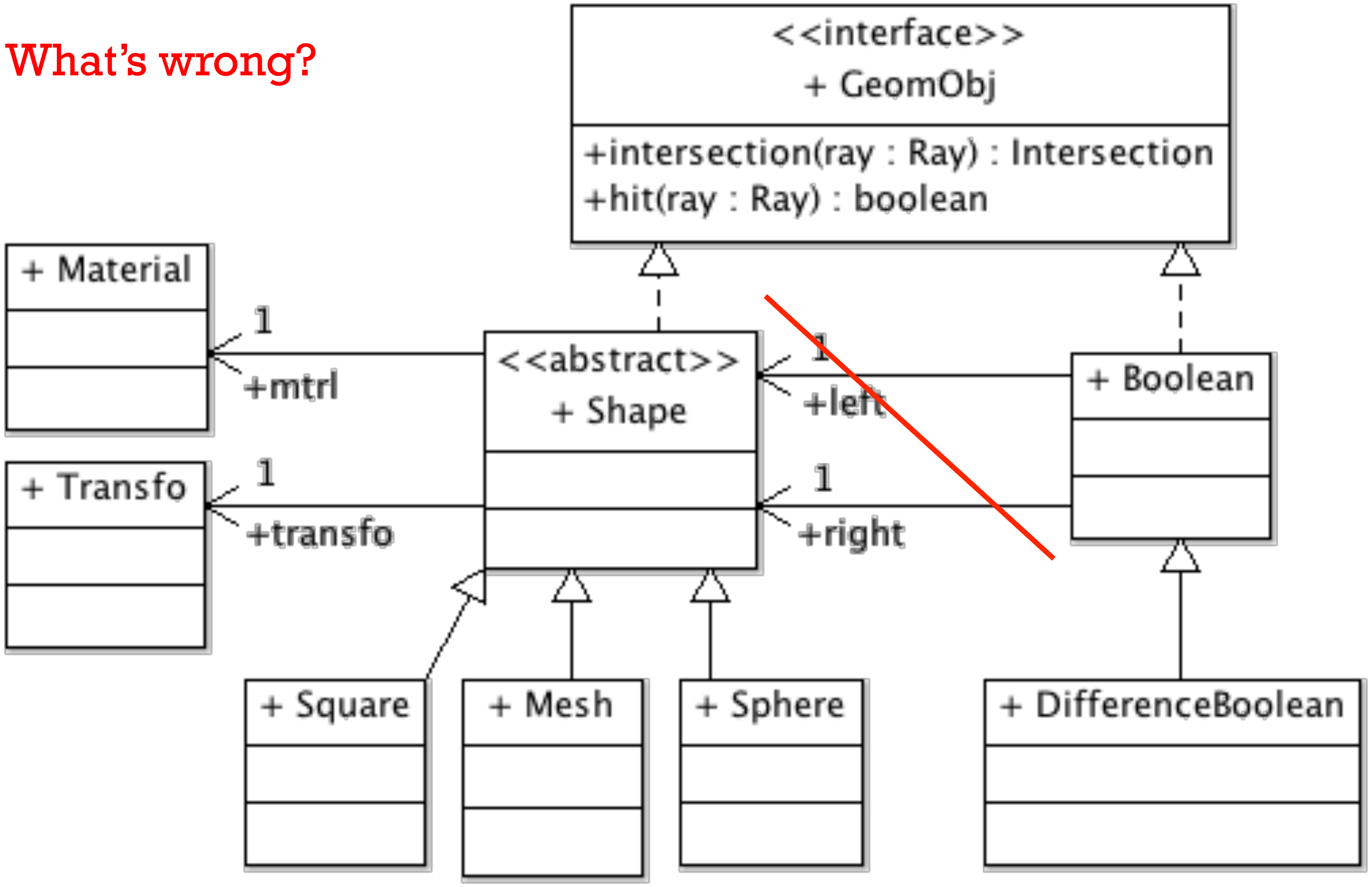


sdl file

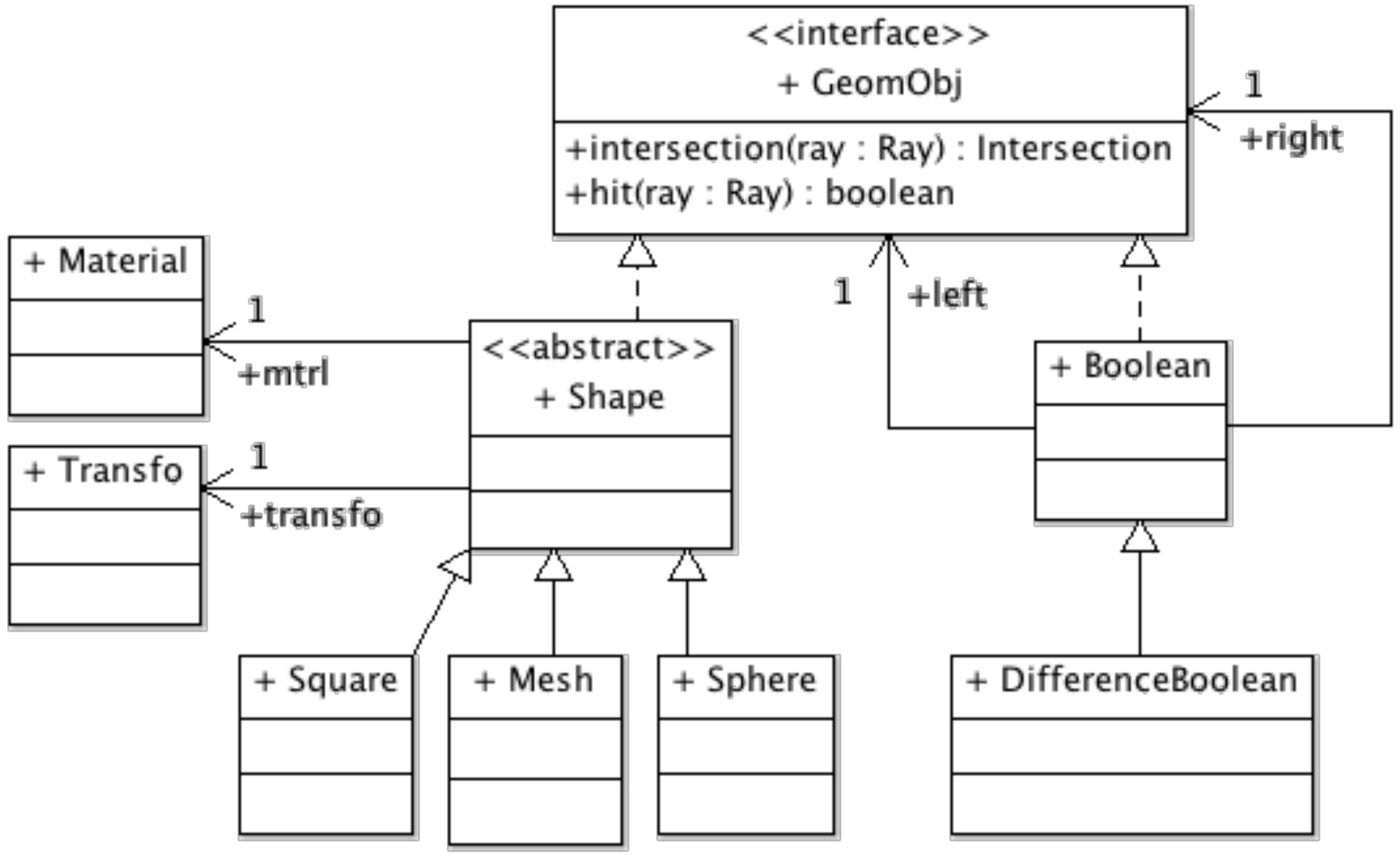
```
difference
difference
sphere
push rotate 45  0 0 1 translate 1 1 0 mesh cube.txt pop
translate -0.75 0.2 0 scale 0.1 0.1 0.1 mesh cylinder.txt
```

Support for boolean objects

What's wrong?

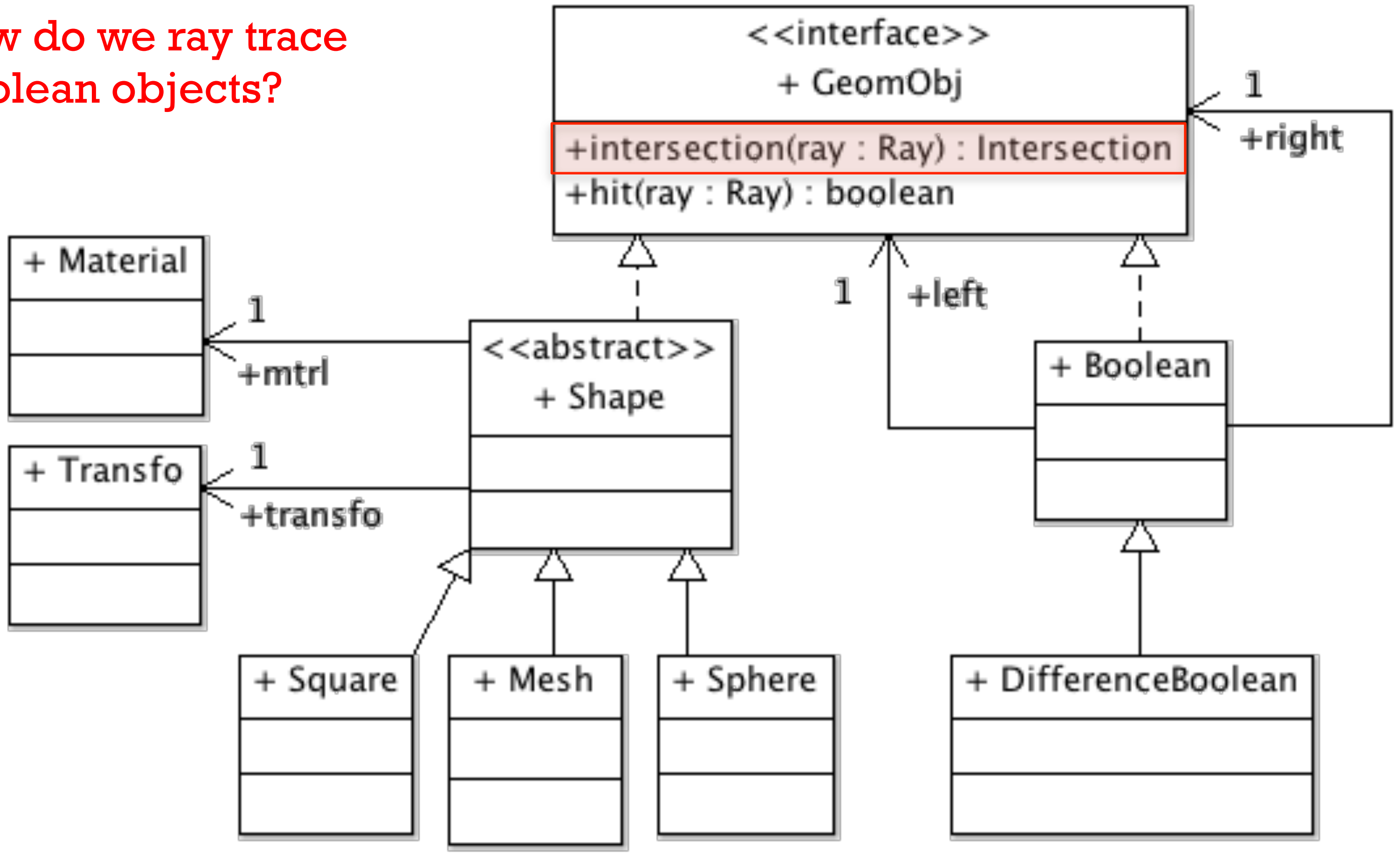


Support for boolean objects

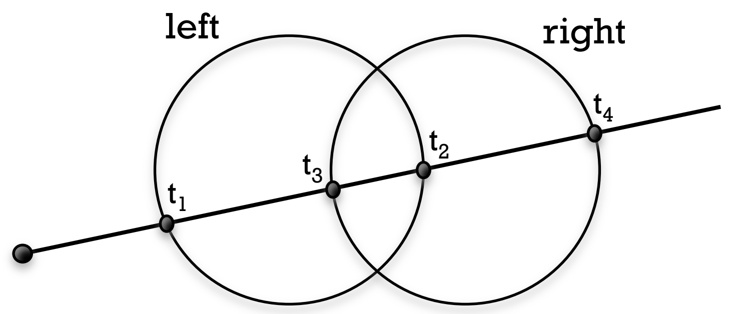


Support for boolean objects

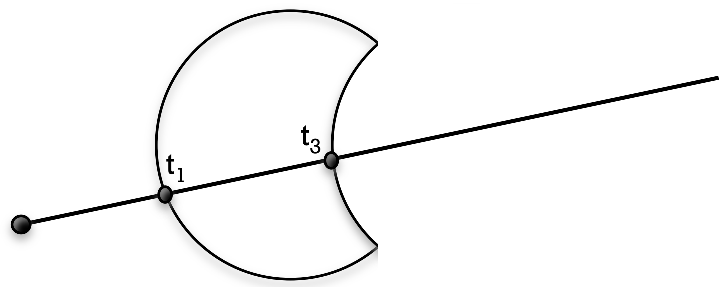
How do we ray trace boolean objects?



Example



comb = left - right



leftInter = left.intersection(ray) $\boxed{t_1}$ $\boxed{t_2}$

rightInter = right.intersection(ray) $\boxed{t_3}$ $\boxed{t_4}$

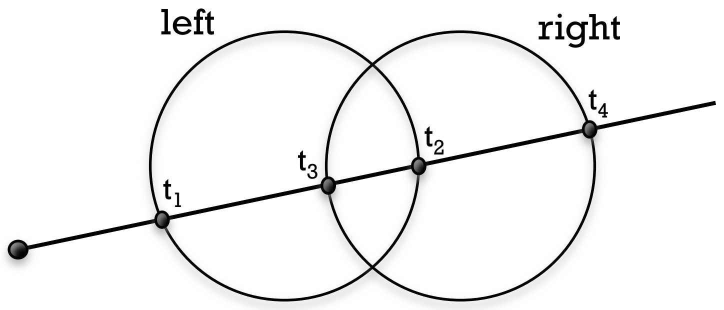
$\xrightarrow{?}$

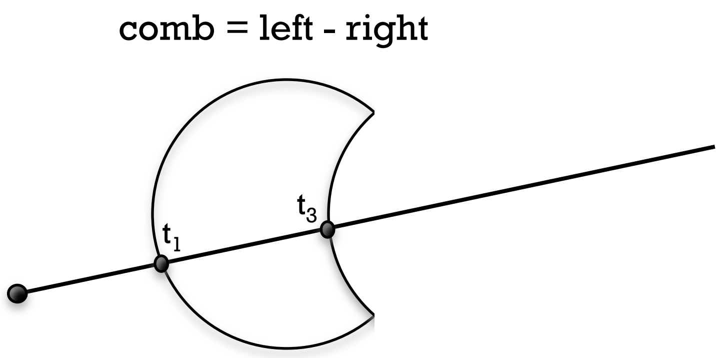
combInter: $\boxed{t_1}$ $\boxed{t_3}$

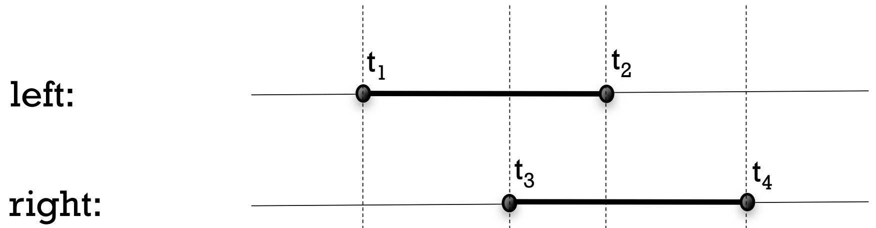
Pseudocode for DifferenceBoolean

```
Public Intersection intersection(Ray ray){
    Intersection combInter = new Intersection();
    Intersection leftInter, rightInter;      // and initialize values

                  ?

    return combInter;

}
```

Example



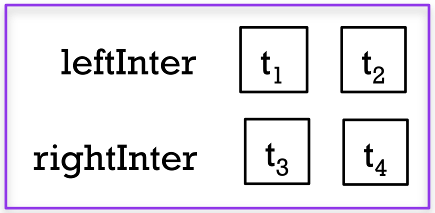




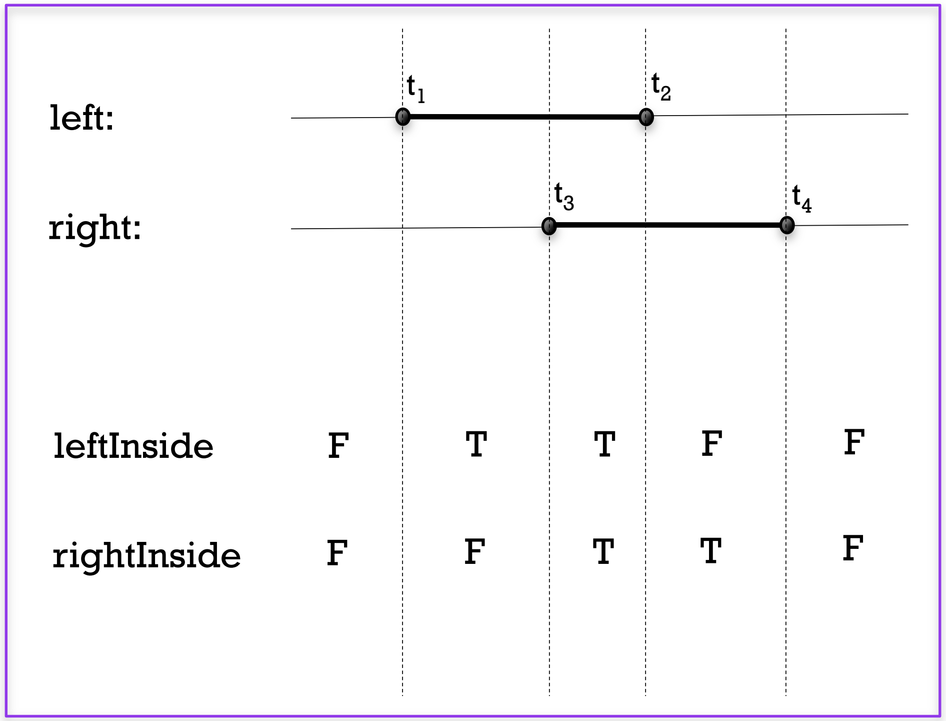
leftInside	F	T	T	F	F
rightInside	F	F	T	T	F

We will process the t-values in ascending order and keep track of whether we are currently inside the left object and/or inside the right object.

Pseudocode for DifferenceBoolean

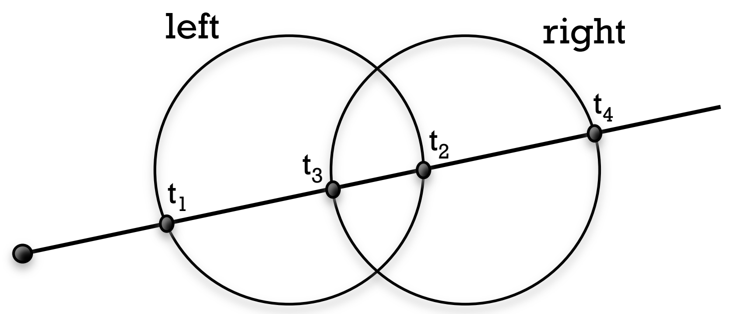


```
Public Intersection intersection(Ray ray){
    Intersection combInter = new Intersection();
    Intersection leftInter, rightInter;      // and initialize values
    boolean leftInside, rightInside;         // and initialize values
    while (there are still unprocessed hitPoints in leftInter and rightInter){
        if(next unprocessed hitPoint in leftInter is closer than next unprocesed hitPoint in rightInter){
            change leftInside
        } else {
            change rightInside
        }
    }
    return combInter;
}
```

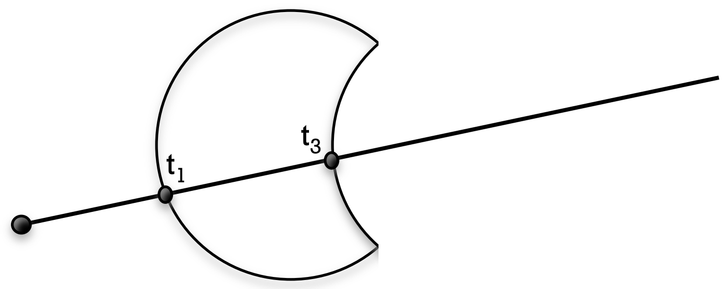


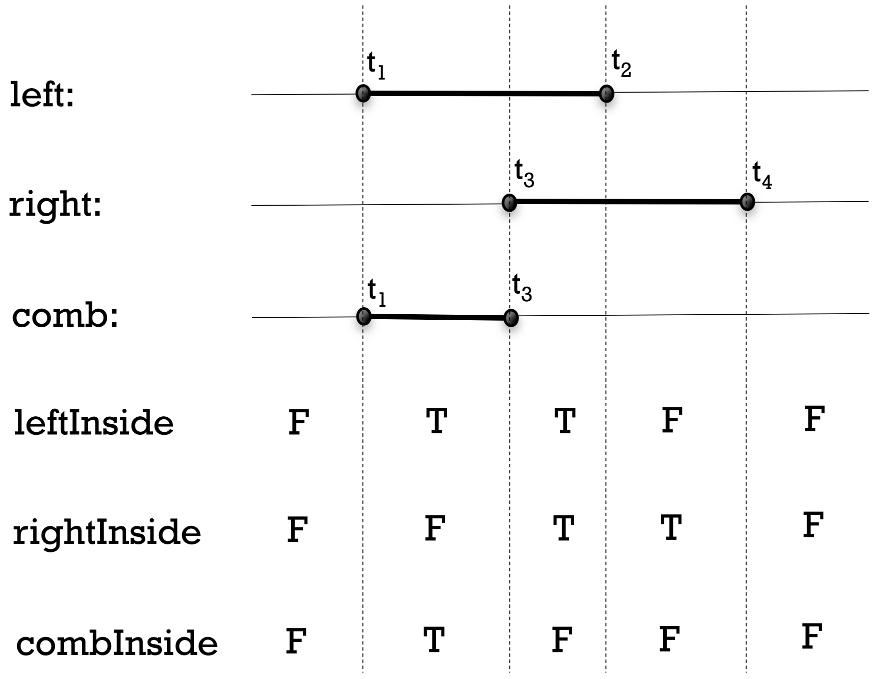
leftInside	F	T	T	F	F
rightInside	F	F	T	T	F

Example



comb = left - right





leftInside	F	T	T	F	F
rightInside	F	F	T	T	F
combInside	F	T	F	F	F

- Apart from leftInside and rightInside, we will also keep track of combInside.
- How can we compute combInside if we know leftInside and rightInside?

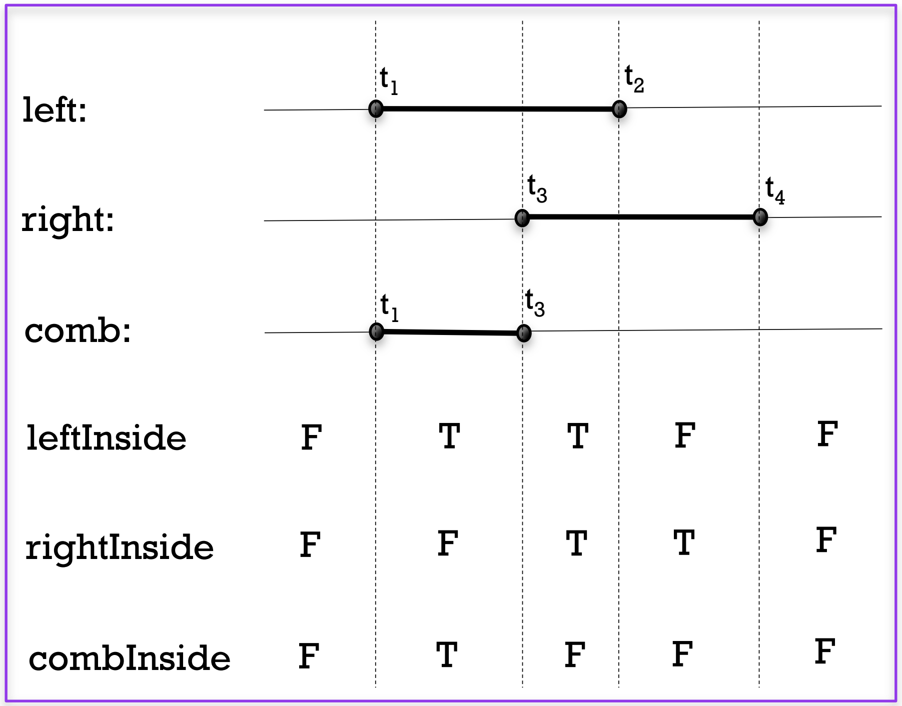
  combInside = leftInside && !rightInside

- Why is the value of combInside useful?

  If the value of combInside changes, we have to add the current hitpoint to combInter.

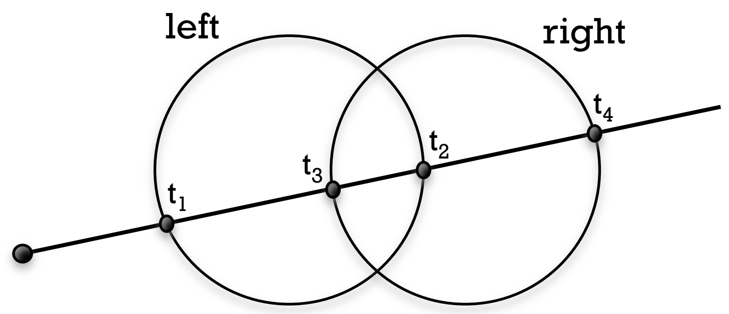
Pseudocode for DifferenceBoolean

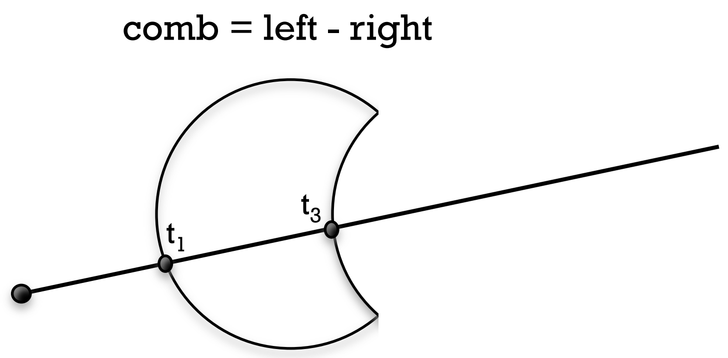
```
Public Intersection intersection(Ray ray){
    Intersection combInter = new Intersection();
    Intersection leftInter, rightInter;       // and initialize values
    boolean leftInside, rightInside, combInside;       // and initialize values
    while (there are still unprocessed hitPoints in leftInter and rightInter){
            boolean combInsideNew;
            if(next unprocessed hitPoint in leftInter is closer than next unprocesed hitPoint in rightInter){
                    change leftInside
                    combInsideNew = leftInside && !rightInside
                     if(combInsideNew != combInside)
                         update combInside
                         add the current hitPoint of leftInter to combInter
            } else {
                     change rightInside
                     combInsideNew = leftInside && !rightInside
                      if(combInsideNew != combInside)
                          update combInside
                          add the current hitPoint of rightInter to combInter
             }
    }

    return combInter;
}
```



leftInside	F	T	T	F	F
rightInside	F	F	T	T	F
combInside	F	T	F	F	F

Example





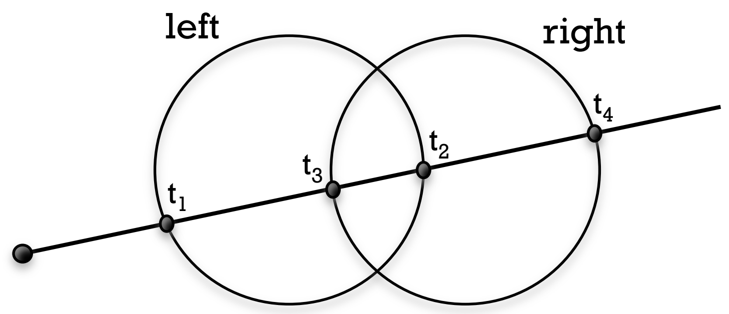
leftInter = left.intersection(ray) $\boxed{t_1}$ $\boxed{t_2}$

rightInter = right.intersection(ray) $\boxed{t_3}$ $\boxed{t_4}$

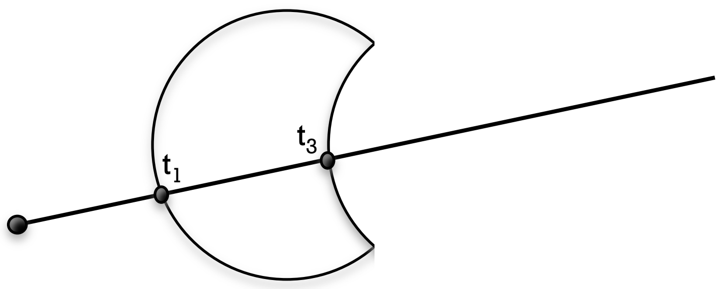
$\Rightarrow$ combInter: $\boxed{t_1}$ $\boxed{t_3}$

- The intersection method returns an Intersection object.
- An Intersection object contains a list of HitInfo objects.
- A HitInfo object contains more information than a t-value!

Example



comb = left - right



leftInter:

t_1
hitMaterialLeft
hitPoint_1
hitNormal_1
$\text{isEntering}_1 = \text{T}$

t_2
hitMaterialLeft
hitPoint_2
hitNormal_2
$\text{isEntering}_2 = \text{F}$

rightInter:

t_3
hitMaterialRight
hitPoint_3
hitNormal_3
$\text{isEntering}_3 = \text{T}$

t_4
hitMaterialRight
hitPoint_4
hitNormal_4
$\text{isEntering}_4 = \text{F}$

combInter:

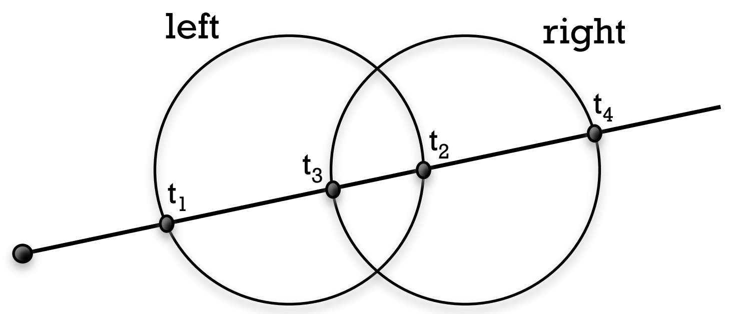
t_1
hitMaterialLeft
hitPoint_1
hitNormal_1
$\text{isEntering}_1 = \text{T}$

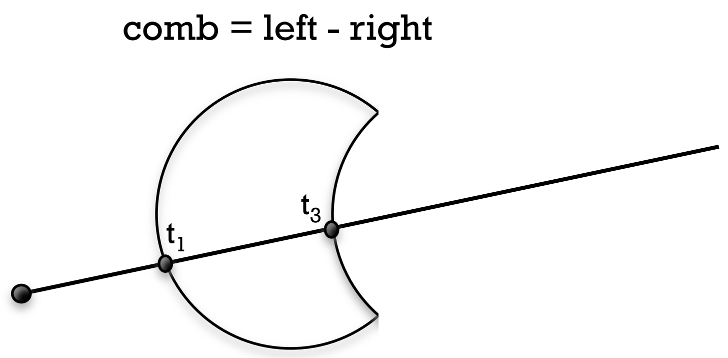
Wrong

~~t_3~~
~~hitMaterialRight~~
~~hitPoint_3~~
~~hitNormal_3~~
~~$\text{isEntering}_3 = \text{T}$~~

t_3
hitMaterialLeft
hitPoint_3
reverse of hitNormal_3
$\text{isEntering}_3 = \text{F}$

Example





Changeset A

Changes which have to be made to HitInfo objects before they are added to combInter:

1) All HitInfo objects of combInter have a hit-Material equal to the material of the left shape.

2) If a HitInfo object of the right shape is added to combInter, the hitNormal should be reversed if combInside ≠ rightInside.

3) The isEntering variable of all HitInfo objects of combInter should be set to combInside.

combInter:

t_1
hitMaterialLeft
hitPoint$_1$
hitNormal$_1$
isEntering$_1$ = T

~~t_3~~
~~hitMaterialRight~~
~~hitPoint$_3$~~
~~hitNormal$_3$~~
~~isEntering$_3$ = T~~

t_3
hitMaterialLeft
hitPoint$_3$
reverse of hitNormal$_3$
isEntering$_3$ = F

Pseudocode for DifferenceBoolean

```
Public Intersection intersection(Ray ray){
    Intersection combInter = new Intersection();
    Intersection leftInter, rightInter;       // and initialize values
    boolean leftInside, rightInside, combInside;       // and initialize values
    while (there are still unprocessed hitPoints in leftInter and rightInter){
        boolean combInsideNew;
        if(next unprocessed hitPoint in leftInter is closer than next unprocesed hitPoint in rightInter){
            change leftInside
            combInsideNew = leftInside && !rightInside
            if(combInsideNew != combInside)
                update combInside
                add the current hitPoint of leftInter to combInter
        } else {
            change rightInside
            combInsideNew = leftInside && !rightInside
            if(combInsideNew != combInside)
                update combInside
                add the current hitPoint of rightInter to combInter
        }
    }

    return combInter;
}
```

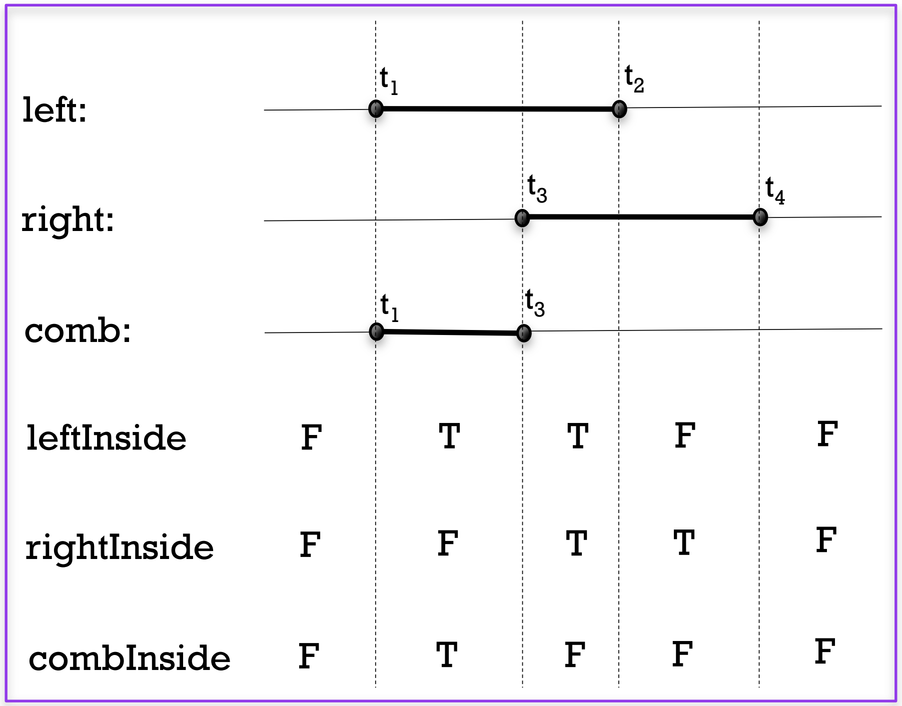
When does the while loop end?

If the list of HitInfo objects of leftInter OR rightInter is completely processed.

Should we still process the other list?

Yes, if it is the list of the left object.

Changeset B:
Process the remaining HitInfo objects of the left shape.



leftInside	F	T	T	F	F
rightInside	F	F	T	T	F
combInside	F	T	F	F	F

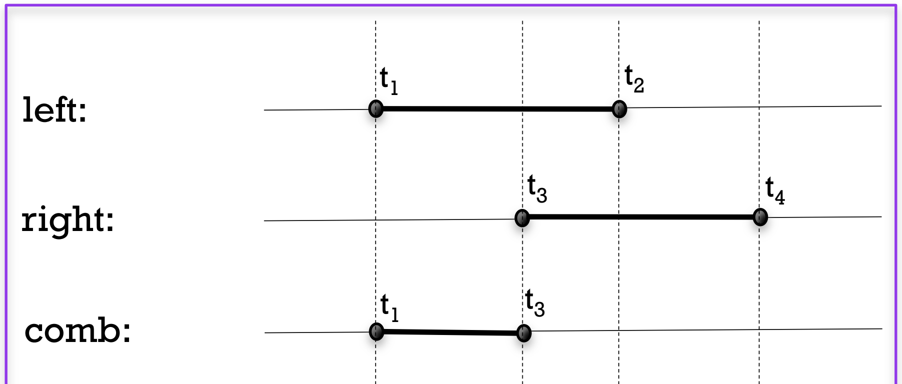
Pseudocode for DifferenceBoolean

Changeset C

What if the list of HitInfo objects of leftInter or rightInter is empty?

- If leftInter is empty,
  return an empty combInter immediately.
- If rightInter is empty,
  return leftInter immediately.

```
Public Intersection intersection(Ray ray){
    Intersection combInter = new Intersection();
    Intersection leftInter, rightInter;        // and initialize values
    boolean leftInside, rightInside, combInside;        // and initialize values
    while (there are still unprocessed hitPoints in leftInter and rightInter){
        boolean combInsideNew;
        if(next unprocessed hitPoint in leftInter is closer than next unprocesed hitPoint in rightInter){
            change leftInside
            combInsideNew = leftInside && !rightInside
            if(combInsideNew != combInside)
                update combInside
                add the current hitPoint of leftInter to combInter
        } else {
            change rightInside
            combInsideNew = leftInside && !rightInside
            if(combInsideNew != combInside)
                update combInside
                add the current hitPoint of rightInter to combInter
        }
    }
    return combInter;
}
```



leftInside	F	T	T	F	F
rightInside	F	F	T	T	F
combInside	F	T	F	F	F

Questions?