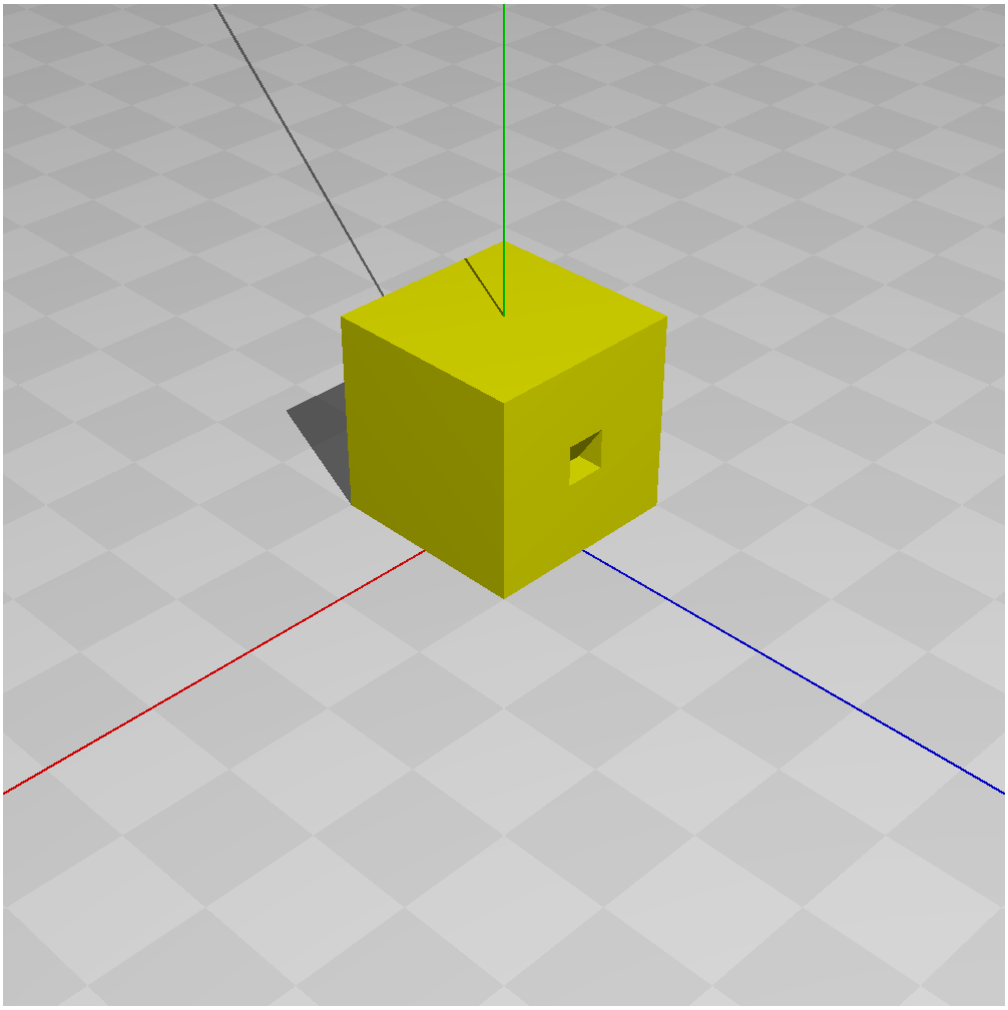
Infographie 2 : TD3

Exercice 0 - Jouer avec povray

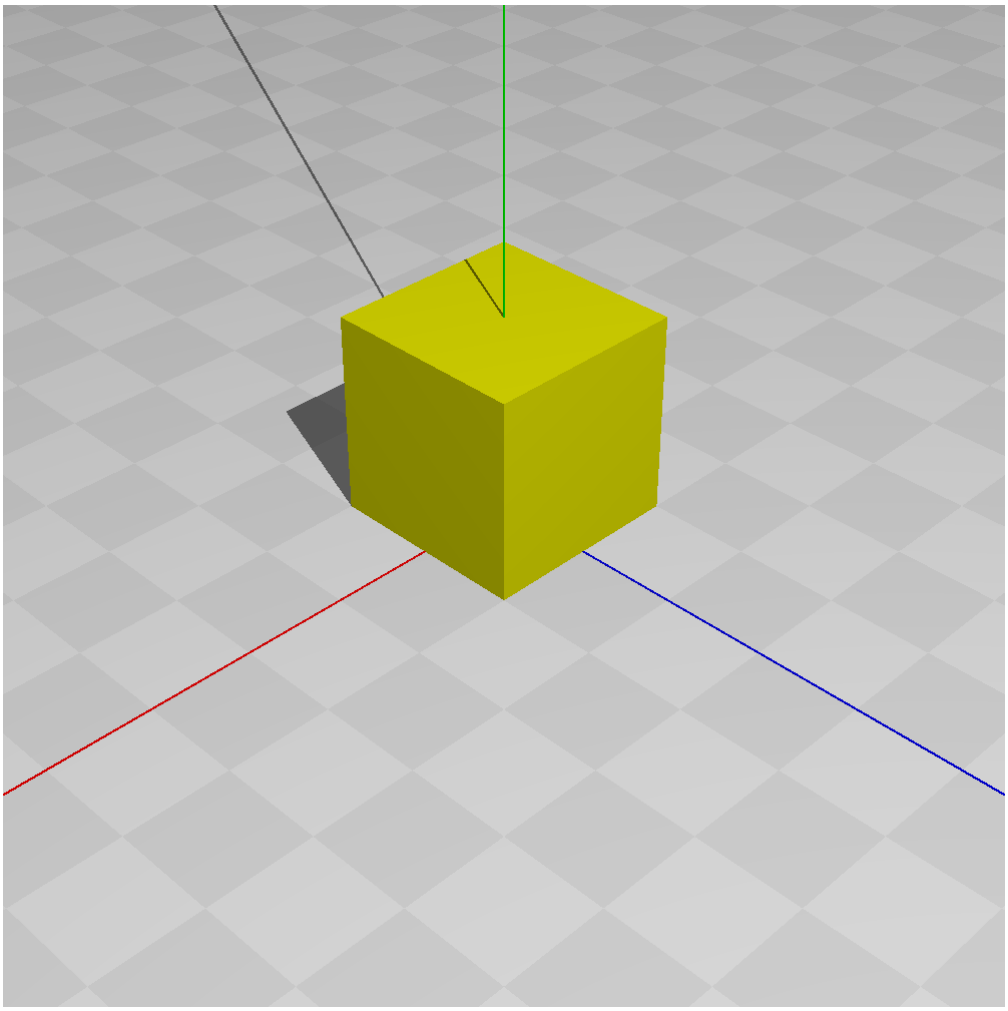
```
#version 3.7;

camera {
    right x * image_width/image_height
    up y
    location <10,10,10>
    look_at<0,0,0>
    angle 30
}

plane {
    <0, 1, 0>,
    0 pigment { checker rgb 1 rgb .9}   // Plan damier
}

light_source {
    <5,15,10>
    rgb <1,1,1>
}

// représentation de l'axe x (rouge)
cylinder { <0, 0, 0>, <1, 0, 0>*10, .01 pigment {rgb <1, 0, 0>}}

// représentation de l'axe y (vert)
cylinder { <0, 0, 0>, <0, 1, 0>*10, .01 pigment {rgb <0, 1, 0>}}

// représentation de l'axe z (bleu)
cylinder { <0, 0, 0>, <0, 0, 1>*10, .01 pigment {rgb <0, 0, 1>}}

difference {
    box { <-1, -1, -1> <1, 1, 1> }

    // Il suffit de très peu pour que les surfaces ne soient plus
    // coplanaire, un changement de 1e-10 creuse déjà un trou
    // FAIRE ATTENTION : Quand on fait des opérations sur des objets,
    // il faut éviter les surfaces confondues
    box { <-1, -1, -1> <1, 1, 1> scale <.2, .2, 1+1e-10>}
    pigment { rgb <1, 1, 0> }
```

```
    translate <0, 1, 0>
}
```



```
camera {
    right x * image_width/image_height
    up y
    location <10,10,10>
    look_at<0,0,0>
    angle 30
}

plane {
    <0, 1, 0>,
    0 pigment { checker rgb 1 rgb .9}   // Plan damier
}

light_source {
    <5,15,10>
    rgb <1,1,1>
}

// représentation de l'axe x (rouge)
```

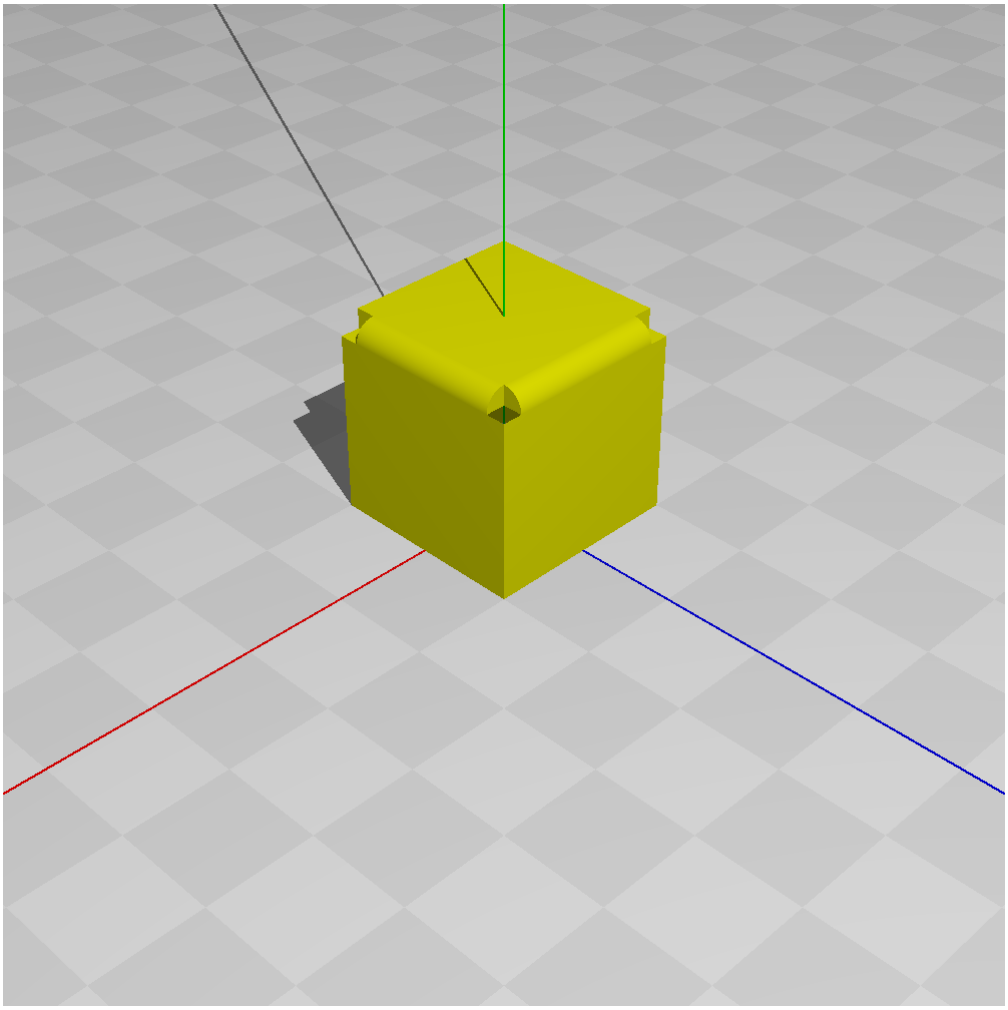
```
cylinder { <0, 0, 0>, <1, 0, 0>*10, .01 pigment {rgb <1, 0, 0>}}

// représentation de l'axe y (vert)
cylinder { <0, 0, 0>, <0, 1, 0>*10, .01 pigment {rgb <0, 1, 0>}}

// représentation de l'axe z (bleu)
cylinder { <0, 0, 0>, <0, 0, 1>*10, .01 pigment {rgb <0, 0, 1>}}

merge {
    box { <-1, -1, -1> <1, 1, 1> }
    box { <-1, -1, -1> <1, 1, 1> scale <.2, .2, 1+1e-10>}
    pigment { rgb <1, 1, 0> }
    translate <0, 1, 0>
}
```



```
#version 3.7;

camera {
    right x * image_width/image_height
    up y
    location <10,10,10>
    look_at<0,0,0>
```

```
    angle 30
}

plane {
    <0, 1, 0>,
    0 pigment { checker rgb 1 rgb .9}   // Plan damier
}

light_source {
    <5,15,10>
    rgb <1,1,1>
}

// représentation de l'axe x (rouge)
cylinder { <0, 0, 0>, <1, 0, 0>*10, .01 pigment {rgb <1, 0, 0>}}

// représentation de l'axe y (vert)
cylinder { <0, 0, 0>, <0, 1, 0>*10, .01 pigment {rgb <0, 1, 0>}}

// représentation de l'axe z (bleu)
cylinder { <0, 0, 0>, <0, 0, 1>*10, .01 pigment {rgb <0, 0, 1>}}
#declare R = .2;
merge {
    merge{
        difference{
            difference{
                box { <-1, -1, -1> <1, 1, 1> }
                box {
                    <-1, -1, -1> <1, 1, 1>
                    translate <2 - R, 2 - R, 0>
                    scale <1, 1, 1+1e-2>
                    }
            }
            box {
                    <-1, -1, -1> <1, 1, 1>
                    translate <0, 2 - R, 2-R>
                    scale <1+1e-2, 1, 1>
                    }
        }
        cylinder {
            <-1, 0, 0>, <1, 0, 0>, 1
            scale <1-R, .2, .2>
            rotate <0,0,0>
            translate <0,1-R,1-R>
        }
    }
```

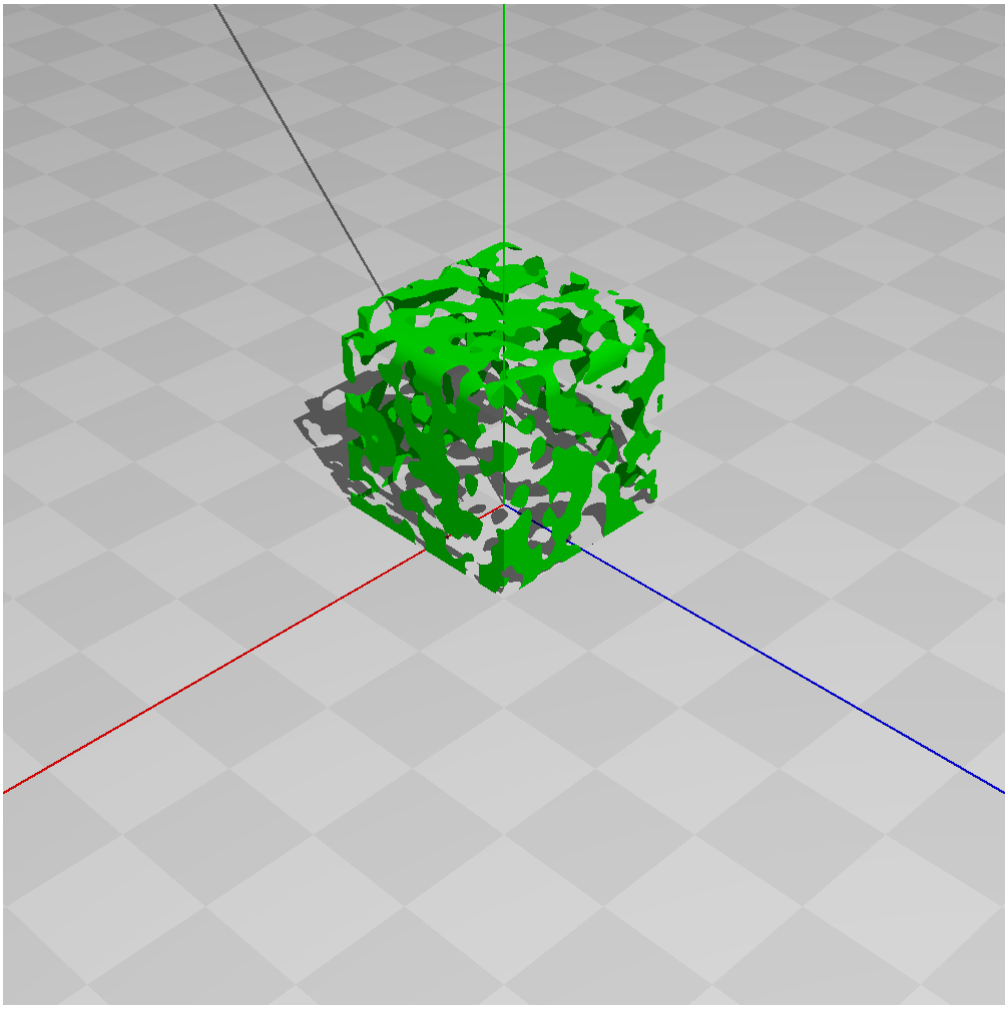
```
    cylinder {
        <-1, 0, 0>, <1, 0, 0>, 1
        scale <1-R, .2, .2>
        rotate <0,90,0>
        translate <1-R,1-R,0>
    }

    pigment { rgb <1, 1, 0> }
    translate <0, 1, 0>
}
```



```
#version 3.7;

camera {
    right x * image_width/image_height
    up y
    location <10,10,10>
    look_at<0,0,0>
    angle 30
}
```

```
plane {
    <0, 1, 0>,
    0 pigment { checker rgb 1 rgb .9}   // Plan damier
}

light_source {
    <5,15,10>
    rgb <1,1,1>
}

// représentation de l'axe x (rouge)
cylinder { <0, 0, 0>, <1, 0, 0>*10, .01 pigment {rgb <1, 0, 0>}}

// représentation de l'axe y (vert)
cylinder { <0, 0, 0>, <0, 1, 0>*10, .01 pigment {rgb <0, 1, 0>}}

// représentation de l'axe z (bleu)
cylinder { <0, 0, 0>, <0, 0, 1>*10, .01 pigment {rgb <0, 0, 1>}}
#declare R = .2;
merge {
    merge{
        difference{
            difference{
                box { <-1, -1, -1> <1, 1, 1> }
                box {
                    <-1, -1, -1> <1, 1, 1>
                    translate <2 - R, 2 - R, 0>
                    scale <1, 1, 1+1e-2>
                    }
            }
            box {
                    <-1, -1, -1> <1, 1, 1>
                    translate <0, 2 - R, 2-R>
                    scale <1+1e-2, 1, 1>
                    }
        }
        cylinder {
            <-1, 0, 0>, <1, 0, 0>, 1
            scale <1-R, .2, .2>
            rotate <0,0,0>
            translate <0,1-R,1-R>
        }
    }

    cylinder {
        <-1, 0, 0>, <1, 0, 0>, 1
```

```
        scale <1-R, .2, .2>
        rotate <0,90,0>
        translate <1-R,1-R,0>
    }

    // Feuillage
    pigment {
        bozo
        scale .2
        color_map {
            [.5 rgbf <1, 1, 1, 1>]
            [.5 rgb <0,1,0>]
        }
    }
    translate <0, 1, 0>
}
```



Exercice 1

> **Algorithme de Warnock** : algorithme de subdivisions récursives jusqu'à une sous-fenêtre simple (0, 1 ou 2 objets ou bien un objet qui cache tous les autres).

Sujet

Triangle : structure qui contient les 3 sommets. Sous-fenêtre : rectangle défini par 2 points.

Espace écran :

- x, y : position sur l'image
- z : profondeur

Entrées :

- L : liste de triangles
- W : sous-fenêtre

Sorties :

- T : liste de triangles de $L \in W$

1. Quels triangles appartiennent à la sous-fenêtre W ?

Entrées :

- A, B (triangles)

Sorties :

- $-1\ (A < B)$
- $0\ (A = B)$
- $1\ (A > B)$

2. Comment comparer deux triangles ?

3. Trier les objets (triangles) par distance (décroissante)

4. Déterminer les objets en commençant par les plus éloignés.

Résolution

Question 1

```
structure point2D
    x, y : réel

structure point3D
    x, y, z : réel

structure triangle
    p1, p2, p3 : point3D

structure fenetre
    pmin, pmax : point2D

debut
```

```
    pour T dans L faire
        si (T.p1.x < F.pmin.x et T.p2.x < F.pmin.x et T.p3.x < F.pmin.x) ou
           (T.p1.x > F.pmax.x et T.p2.x > F.pmax.x et T.p3.x > F.pmax.x) ou
           (T.p1.y < F.pmin.y et T.p2.y < F.pmin.y et T.p3.y < F.pmin.y) ou
           (T.p1.y > F.pmax.y et T.p2.y > F.pmax.y et T.p3.y > F.pmax.y) alo
            retourner faux

        si (T.p1 > F.pmin et T.p1 < F.pmax) ou
           (T.p2 > F.pmin et T.p2 < F.pmax) ou
           (T.p3 > F.pmin et T.p3 < F.pmax) alors
            retourner vrai

        // Test d'intersection avec la boite -> retourne vrai

        // Test si la fenetre est dans le triangle -> retourne vrai

    fin pour
    retourner faux
fin
```