R 5.A.06

2025 – 2026

Sensibilisation à la programmation multimédia

Corrigé du TP n° 4
Raytracing

ANNE Jean-François

Sensibilisation à la programmation multimédia

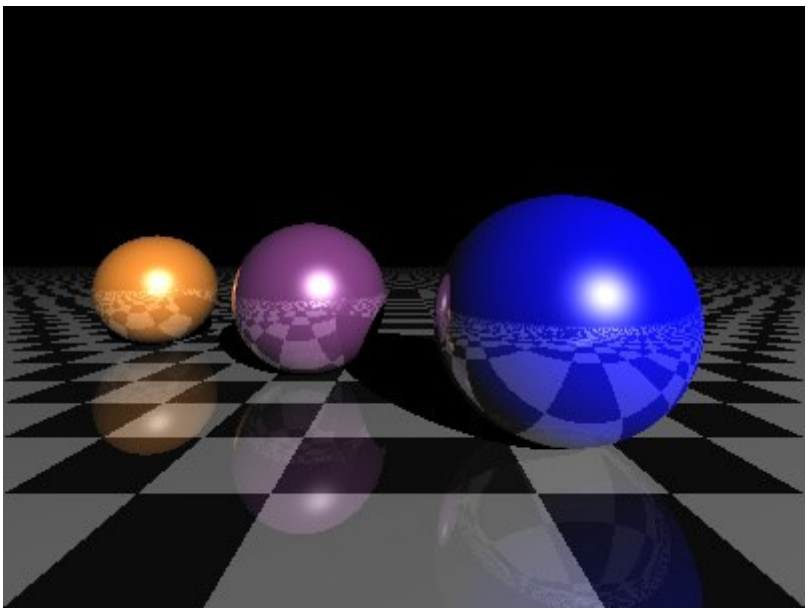
Programmation en 3D

Le but de ce TD est de se familiariser avec le Sensibilisation à la programmation multimédia.

A. *Traceur de rayons Python raisonnablement rapide*

- *https://gist.github.com/rossant/6046463*

Le petit ray-tracer de Cyrille Rossant est un joli programme Python autonome (utilisant NumPy) qui rend cette image 400 × 300 en environ 15 secondes sur un PC rapide :



Vous pourriez en conclure que Python est un langage inapproprié pour un traceur de rayons. Mais cette version est légèrement plus petite et rend la même image en environ 115 millisecondes. C'est plus de 130 fois plus rapide que l'original.

Cette version est rt3.py.

```
from PIL import Image
from functools import reduce
import numpy as np
import time
import numbers


def extract(cond, x):
    if isinstance(x, numbers.Number):
        return x
    else:
        return np.extract(cond, x)


class vec3():
    def __init__(self, x, y, z):
        (self.x, self.y, self.z) = (x, y, z)

    def __mul__(self, other):
        return vec3(self.x * other, self.y * other, self.z * other)
```

```
    def __add__(self, other):
        return vec3(self.x + other.x, self.y + other.y, self.z + other.z)

    def __sub__(self, other):
        return vec3(self.x - other.x, self.y - other.y, self.z - other.z)

    def dot(self, other):
        return (self.x * other.x) + (self.y * other.y) + (self.z * other.z)

    def __abs__(self):
        return self.dot(self)

    def norm(self):
        mag = np.sqrt(abs(self))
        return self * (1.0 / np.where(mag == 0, 1, mag))

    def components(self):
        return (self.x, self.y, self.z)

    def extract(self, cond):
        return vec3(extract(cond, self.x),
                    extract(cond, self.y),
                    extract(cond, self.z))

    def place(self, cond):
        r = vec3(np.zeros(cond.shape), np.zeros(
            cond.shape), np.zeros(cond.shape))
        np.place(r.x, cond, self.x)
        np.place(r.y, cond, self.y)
        np.place(r.z, cond, self.z)
        return r


rgb = vec3

(w, h) = (800, 600)          # Screen size
L = vec3(5, 5, -10)         # Point light position
E = vec3(0, 0.35, -1)      # Eye position
FARAWAY = 1.0e39             # an implausibly huge distance


def raytrace(O, D, scene, bounce=0):
    # O is the ray origin, D is the normalized ray direction
    # scene is a list of Sphere objects (see below)
    # bounce is the number of the bounce, starting at zero for camera rays

    distances = [s.intersect(O, D) for s in scene]
    nearest = reduce(np.minimum, distances)
    color = rgb(0, 0, 0)
    for (s, d) in zip(scene, distances):
        hit = (nearest != FARAWAY) & (d == nearest)
        if np.any(hit):
            dc = extract(hit, d)
```

```
            Oc = O.extract(hit)
            Dc = D.extract(hit)
            cc = s.light(Oc, Dc, dc, scene, bounce)
            color += cc.place(hit)
    return color


class Sphere:
    def __init__(self, center, r, diffuse, mirror=0.5):
        self.c = center
        self.r = r
        self.diffuse = diffuse
        self.mirror = mirror

    def intersect(self, O, D):
        b = 2 * D.dot(O - self.c)
        c = abs(self.c) + abs(O) - 2 * self.c.dot(O) - (self.r * self.r)
        disc = (b ** 2) - (4 * c)
        sq = np.sqrt(np.maximum(0, disc))
        h0 = (-b - sq) / 2
        h1 = (-b + sq) / 2
        h = np.where((h0 > 0) & (h0 < h1), h0, h1)
        pred = (disc > 0) & (h > 0)
        return np.where(pred, h, FARAWAY)

    def diffusecolor(self, M):
        return self.diffuse

    def light(self, O, D, d, scene, bounce):
        M = (O + D * d)                          # intersection point
        N = (M - self.c) * (1. / self.r)         # normal
        toL = (L - M).norm()                     # direction to light
        toO = (E - M).norm()                     # direction to ray origin
        nudged = M + N * .0001                   # M nudged to avoid itself

        # Shadow: find if the point is shadowed or not.
        # This amounts to finding out if M can see the light
        light_distances = [s.intersect(nudged, toL) for s in scene]
        light_nearest = reduce(np.minimum, light_distances)
        seelight = light_distances[scene.index(self)] == light_nearest

        # Ambient
        color = rgb(0.05, 0.05, 0.05)

        # Lambert shading (diffuse)
        lv = np.maximum(N.dot(toL), 0)
        color += self.diffusecolor(M) * lv * seelight

        # Reflection
        if bounce < 2:
            rayD = (D - N * 2 * D.dot(N)).norm()
            color += raytrace(nudged, rayD, scene, bounce + 1) * self.mirror

        # Blinn-Phong shading (specular)
```

```
            phong = N.dot((toL + toO).norm())
            color += rgb(1, 1, 1) * np.power(np.clip(phong, 0, 1), 50) * seelight
            return color


    class CheckeredSphere(Sphere):
        def diffusecolor(self, M):
            checker = ((M.x * 2).astype(int) % 2) == ((M.z * 2).astype(int) % 2)
            return self.diffuse * checker


    scene = [
        Sphere(vec3(.75, .1, 1), .6, rgb(0, 0, 1)),
        Sphere(vec3(-.75, .1, 2.25), .6, rgb(.5, .223, .5)),
        Sphere(vec3(-2.75, .1, 3.5), .6, rgb(1, .572, .184)),
        CheckeredSphere(vec3(0, -99999.5, 0), 99999, rgb(.75, .75, .75), 0.25),
    ]

    r = float(w) / h
    # Screen coordinates: x0, y0, x1, y1.
    S = (-1, 1 / r + .25, 1, -1 / r + .25)
    x = np.tile(np.linspace(S[0], S[2], w), h)
    y = np.repeat(np.linspace(S[1], S[3], h), w)

    t0 = time.time()
    Q = vec3(x, y, 0)
    color = raytrace(E, (Q - E).norm(), scene)
    print("Took", time.time() - t0)

    rgb = [Image.fromarray((255 * np.clip(c, 0, 1).reshape((h, w))
                            ).astype(np.uint8), "L") for c in color.components()]
    Image.merge("RGB", rgb).save("rt3.png")
    img=Image.merge("RGB", rgb)
    display (img)
```

B. 2ème raytracing :

Tester les programmes de Rafael de la Fuente :

- *https://github.com/rafael-fuente/Python-Raytracer*

1°) Exemple 1 :

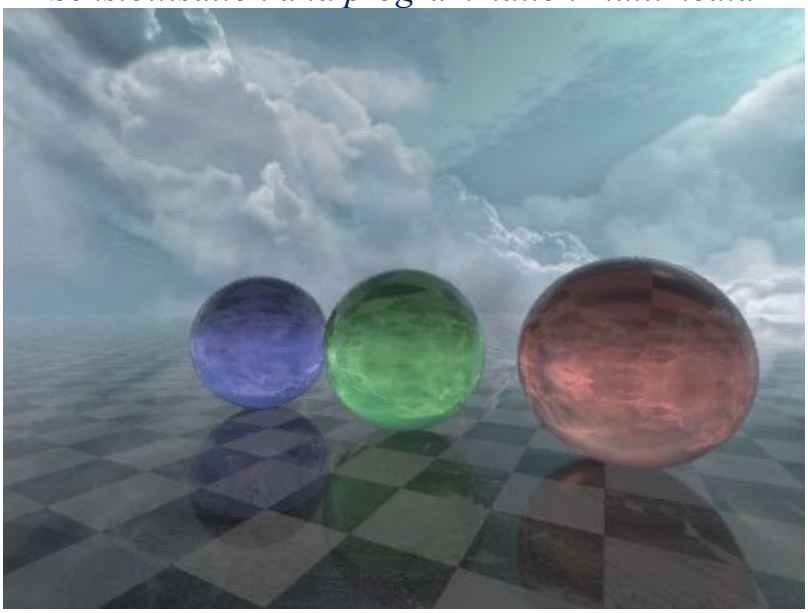
```
from sightpy import *

# define materials to use

gold_metal = Glossy(diff_color = rgb(1., .572, .184), n = vec3(0.15+3.58j,
0.4+2.37j, 1.54+1.91j), roughness = 0.0, spec_coeff = 0.2, diff_coeff= 0.8) # n =
index of refraction
bluish_metal = Glossy(diff_color = rgb(0.0, 0, 0.1), n = vec3(1.3+1.91j, 1.3+1.91j,
1.4+2.91j), roughness = 0.2,spec_coeff = 0.5, diff_coeff= 0.3)

floor =  Glossy(diff_color = image("checkered_floor.png", repeat = 80.),
```

```
                n = vec3(1.2+ 0.3j, 1.2+ 0.3j, 1.1+ 0.3j), roughness = 0.2,
spec_coeff = 0.3, diff_coeff= 0.9 )

# Set Scene
Sc = Scene(ambient_color = rgb(0.05, 0.05, 0.05))

angle = -np.pi/2 * 0.3
Sc.add_Camera(look_from = vec3(2.5*np.sin(angle), 0.25, 2.5*np.cos(angle)  -1.5 ),
              look_at = vec3(0., 0.25, -3.),
              screen_width = 400 ,
              screen_height = 300)

Sc.add_DirectionalLight(Ldir = vec3(0.52,0.45, -0.5),  color = rgb(0.15, 0.15,
0.15))

Sc.add(Sphere(material = gold_metal, center = vec3(-.75, .1, -3.),radius =  .6,
max_ray_depth = 3))
Sc.add(Sphere(material = bluish_metal, center = vec3(1.25, .1, -3.), radius = .6,
max_ray_depth = 3))

Sc.add(Plane(material = floor,  center = vec3(0, -0.5, -3.0), width = 120.0,height =
120.0, u_axis = vec3(1.0, 0, 0), v_axis = vec3(0, 0, -1.0),  max_ray_depth = 3))

#see sightpy/backgrounds
Sc.add_Background("stormydays.png")

# Render
img = Sc.render(samples_per_pixel = 6)

img.save("EXAMPLE1.png")

img.show()
```

```
proccesing checkered_floor.png
proccesing stormydays.png
Rendering...
Render Took 3.1066737174987793
```

2°) ***Exemple 2 :***

```
from sightpy import *

# define materials to use

blue_glass = Refractive(n = vec3(1.5 + 4e-8j,1.5 +  4e-8j,1.5 +  0.j)) # n = index
of refraction
green_glass = Refractive(n = vec3(1.5 + 4e-8j,1.5 + 0.j,1.5 + 4e-8j))
red_glass = Refractive(n = vec3(1.5 + 0.j,1.5 +  5e-8j,1.5 +  5e-8j))

floor = Glossy(diff_color = image("checkered_floor.png", repeat = 80.), n =
vec3(1.2+ 0.3j, 1.2+ 0.3j, 1.1+ 0.3j), roughness = 0.2, spec_coeff = 0.3, diff_coeff
= 0.9)

# Set Scene

Sc = Scene(ambient_color = rgb(0.05, 0.05, 0.05))

angle = np.pi/2 * 0.3
Sc.add_Camera(look_from = vec3(2.5*np.sin(angle), 0.25, 2.5*np.cos(angle)  -1.5 ),
              look_at = vec3(0., 0.25, -1.5),
              screen_width = 400 ,
              screen_height =300)

Sc.add_DirectionalLight(Ldir = vec3(0.52,0.45, -0.5),  color = rgb(0.15, 0.15,
0.15))

Sc.add(Sphere(material = blue_glass, center = vec3(-1.2, 0.0, -1.5), radius = .5,
shadow = False ,max_ray_depth = 3))
Sc.add(Sphere(material = green_glass, center = vec3(0., 0.0, -1.5), radius = .5,
shadow = False,max_ray_depth = 3))
Sc.add(Sphere(material = red_glass, center = vec3(1.2, 0.0, -1.5), radius = .5,
shadow = False,max_ray_depth = 3))

Sc.add(Plane(material = floor,  center = vec3(0, -0.5, -3.0), width = 120.0,height =
120.0, u_axis = vec3(1.0, 0, 0), v_axis = vec3(0, 0, -1.0) ,max_ray_depth = 3))

#see sightpy/backgrounds
Sc.add_Background("miramar.jpeg")

# Render
img = Sc.render(samples_per_pixel = 7)

img.save("EXAMPLE2.png")

img.show()
```

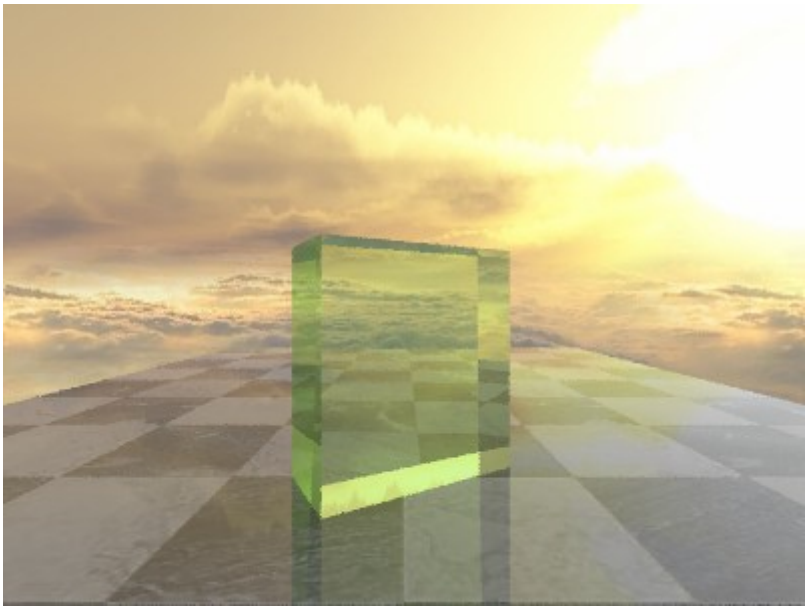


```
proccesing checkered_floor.png
proccesing miramar.jpeg
Rendering...
Render Took 6.993041753768921
```

3°) ***Exemple 3 :***

```
from sightpy import *

# define materials to use

floor = Glossy(diff_color = image("checkered_floor.png", repeat = 2.),  roughness =
0.2, spec_coeff = 0.3, diff_coeff = 0.7, n = vec3(2.2, 2.2, 2.2)) # n = index of
refraction
green_glass = Refractive(n = vec3(1.5 + 4e-8j,1.5 + 0.j,1.5 + 4e-8j))

# Set Scene

Sc = Scene()
Sc.add_Camera(look_from = vec3(0., 0.25, 1. ), look_at = vec3(0., 0.25, -3.),
              screen_width = 400 ,
              screen_height = 300)

Sc.add_DirectionalLight(Ldir = vec3(0.0,0.5, 0.5),  color = rgb(0.5, 0.5, 0.5))

Sc.add(Plane(material = floor,  center = vec3(0, -0.5, -3.0), width = 6.0,height =
6.0, u_axis = vec3(1.0, 0, 0), v_axis = vec3(0, 0, -1.0) , max_ray_depth = 5))

cb = Cuboid( material = green_glass, center = vec3(0.00, 0.0001, -0.8), width =
0.9,height = 1.0, length = 0.4, shadow = False,  max_ray_depth = 5)
cb.rotate(θ = 30, u = vec3(0,1,0))
Sc.add(cb)

#see sightpy/backgrounds
Sc.add_Background("stormydays.png")

# Render
img = Sc.render(samples_per_pixel = 4)

img.save("EXAMPLE3.png")
```

```
img.show()
```



```
proccesing checkered_floor.png
proccesing stormydays.png
Rendering...
Render Took 2.0629539489746094
```

4°) ***Exemple 4 :***

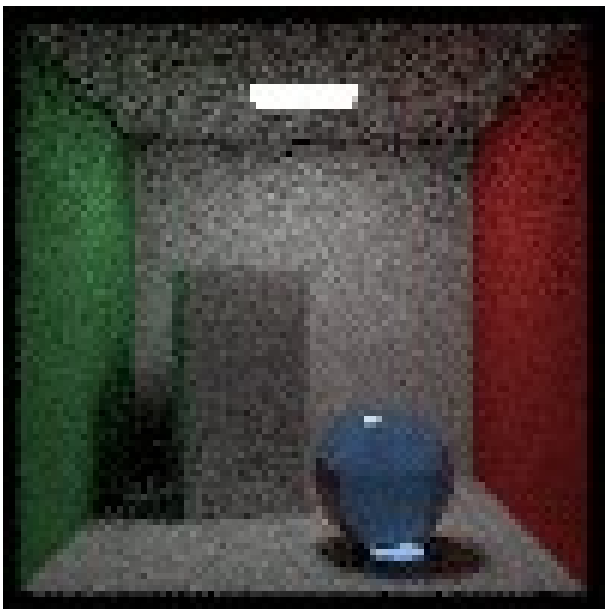
```
from sightpy import *

# Set Scene

Sc = Scene(ambient_color = rgb(0.01, 0.01, 0.01))

angle = -np.pi*0.5
Sc.add_Camera(screen_height =300,  screen_width = 400 ,
              look_from = vec3(4.0*np.sin(angle), 0.00, 4.0*np.cos(angle)  ),
              look_at = vec3(0., 0.05, 0.0))

soap_bubble = ThinFilmInterference(thickness = 330, noise = 60.)
Sc.add(Sphere(material = soap_bubble, center = vec3(1., 0.0, 1.5), radius = 1.7,
shadow = False, max_ray_depth = 5))

Sc.add_Background("lake.png", light_intensity = 5., blur = 10.)

# Render
img = Sc.render(samples_per_pixel = 10)

img.save("EXAMPLE4.png")

img.show()
```

```
proccesing lake.png
blurring lake.png
Rendering...
Render Took 2.6920993328094482
```

5°) ***Exemple 5 : CornellBox***

```
from sightpy import *

# Set Scene

Sc = Scene(ambient_color = rgb(0.00, 0.00, 0.00))

angle = -0

Sc.add_Camera(screen_width = 100 ,screen_height = 100,
              look_from = vec3(278, 278, 800), look_at = vec3(278,278,0),
              focal_distance= 1., field_of_view= 40)

# define materials to use

green_diffuse=Diffuse(diff_color = rgb(.12, .45, .15))
red_diffuse=Diffuse(diff_color = rgb(.65, .05, .05))
white_diffuse=Diffuse(diff_color = rgb(.73, .73, .73))
emissive_white =Emissive(color = rgb(15., 15., 15.))
emissive_blue =Emissive(color = rgb(2., 2., 3.5))
blue_glass =Refractive(n = vec3(1.5 + 0.05e-8j,1.5 +  0.02e-8j,1.5 +  0.j))

# this is the light
Sc.add(Plane(material = emissive_white,  center = vec3(213 + 130/2, 554, -227.0 -
105/2), width = 130.0, height = 105.0, u_axis = vec3(1.0, 0.0, 0), v_axis =
vec3(0.0, 0, 1.0)),
                     importance_sampled = True)

Sc.add(Plane(material = white_diffuse,  center = vec3(555/2, 555/2, -555.0), width =
555.0,height = 555.0, u_axis = vec3(0.0, 1.0, 0), v_axis = vec3(1.0, 0, 0.0)))

Sc.add(Plane(material = green_diffuse,  center = vec3(-0.0, 555/2, -555/2), width =
555.0,height = 555.0,  u_axis = vec3(0.0, 1.0, 0), v_axis = vec3(0.0, 0, -1.0)))
```

```
Sc.add(Plane(material = red_diffuse,  center = vec3(555.0, 555/2, -555/2), width =
555.0,height = 555.0,  u_axis = vec3(0.0, 1.0, 0), v_axis = vec3(0.0, 0, -1.0)))

Sc.add(Plane(material = white_diffuse,  center = vec3(555/2, 555, -555/2), width =
555.0,height = 555.0,  u_axis = vec3(1.0, 0.0, 0), v_axis = vec3(0.0, 0, -1.0)))

Sc.add(Plane(material = white_diffuse,  center = vec3(555/2, 0., -555/2), width =
555.0,height = 555.0,  u_axis = vec3(1.0, 0.0, 0), v_axis = vec3(0.0, 0, -1.0)))

cb = Cuboid( material = white_diffuse, center = vec3(182.5, 165, -285-160/2), width
= 165,height = 165*2, length = 165, shadow = False)
cb.rotate(θ = 15, u = vec3(0,1,0))
Sc.add(cb)

Sc.add(Sphere( material = blue_glass, center = vec3(370.5, 165/2, -65-185/2), radius
= 165/2, shadow = False, max_ray_depth = 3),
                     importance_sampled = True)
# Render

img = Sc.render(samples_per_pixel = 10, progress_bar = True)
# you are going to need more than 10 samples to remove the noise. At least 1000 for
a nice image.

img.save("cornell_box.png")

img.show()
```



```
  0% |                                                                          |
Rendering...
100% |##########################################################################|
Render Took 30.267064571380615
```

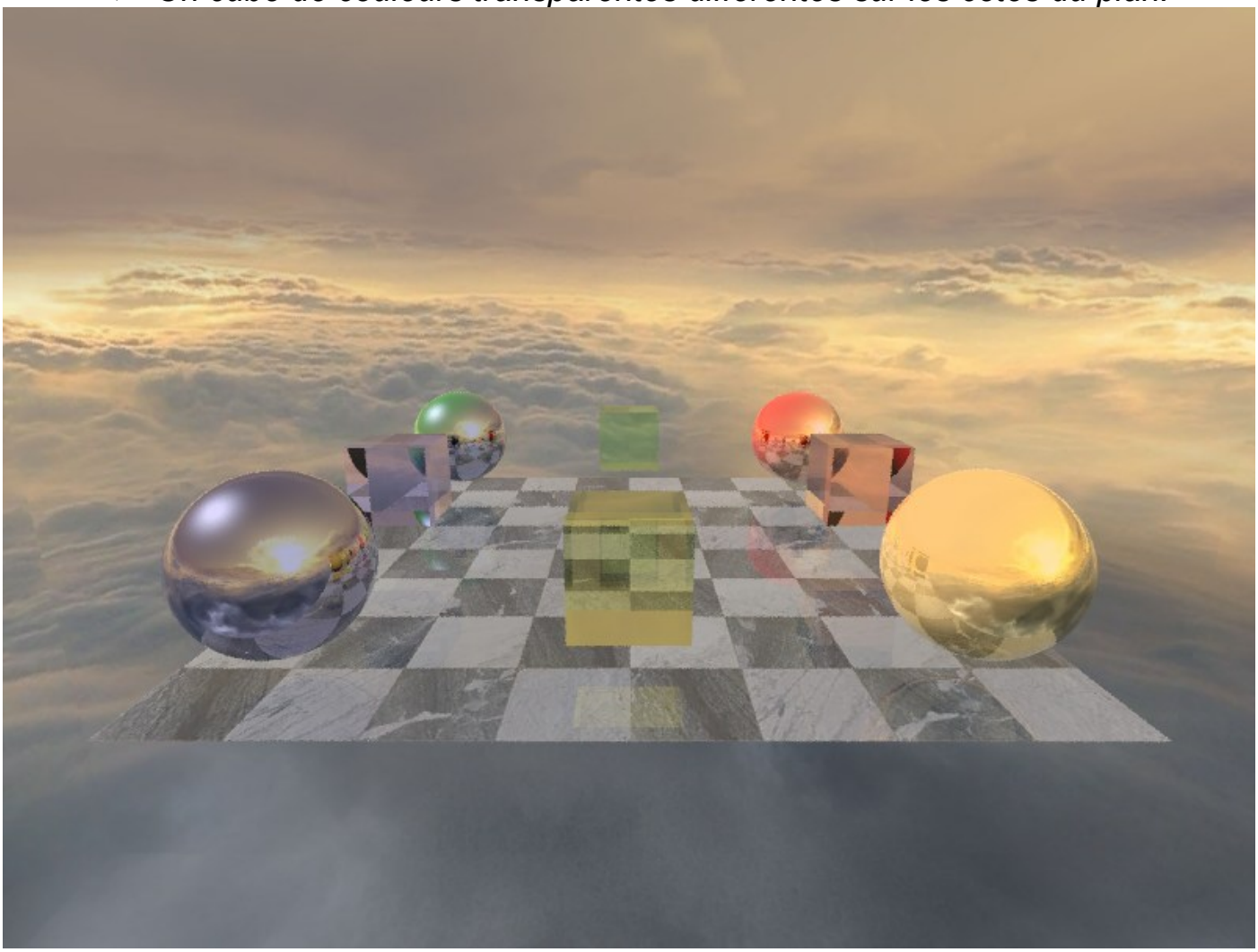
C. ***Synthèse :***

1°) ***Exercice 1 :***

A partir des exemples fournis sur le site de Rafael, créer la scène suivante :

- *Une sphère de couleurs métalliques différentes à chaque coin du plan de base,*

➢ *Un cube de couleurs transparentes différentes sur les côtés du plan.*



```
from IPython import display
from sightpy import *
from PIL import Image, ImageSequence
import math
import os
import numpy as np
from sightpy.geometry.primitive import Primitive
from sightpy.geometry.collider import Collider

width = 800
height = 600
num_frames = 1  # Nombre de frames pour effectuer une rotation complète
# define materials to use
floor = Glossy(diff_color=image("checkered_floor.png", repeat=2.),  roughness=0.2,
               spec_coeff=0.3, diff_coeff=0.7, n=vec3(2.2, 2.2, 2.2))


# n = index of refraction
blue_glass = Refractive(n=vec3(1.5 + 4e-8j, 1.5 + 4e-8j, 1.5 + 0j))
blue_cyan_glass = Refractive(n=vec3(1.5 + 5e-8j, 1.5 + + 5e-8j, 1.5 + 0j))
green_glass = Refractive(n=vec3(1.5 + 4e-8j, 1.5 + 0.j, 1.5 + 4e-8j))
red_glass = Refractive(n=vec3(1.5 + 0.j, 1.5 + 5e-8j, 1.5 + 5e-8j))
# Calcul de la couleur jaune
# yellow_glass = Refractive(n=green_glass.n + red_glass.n)
```

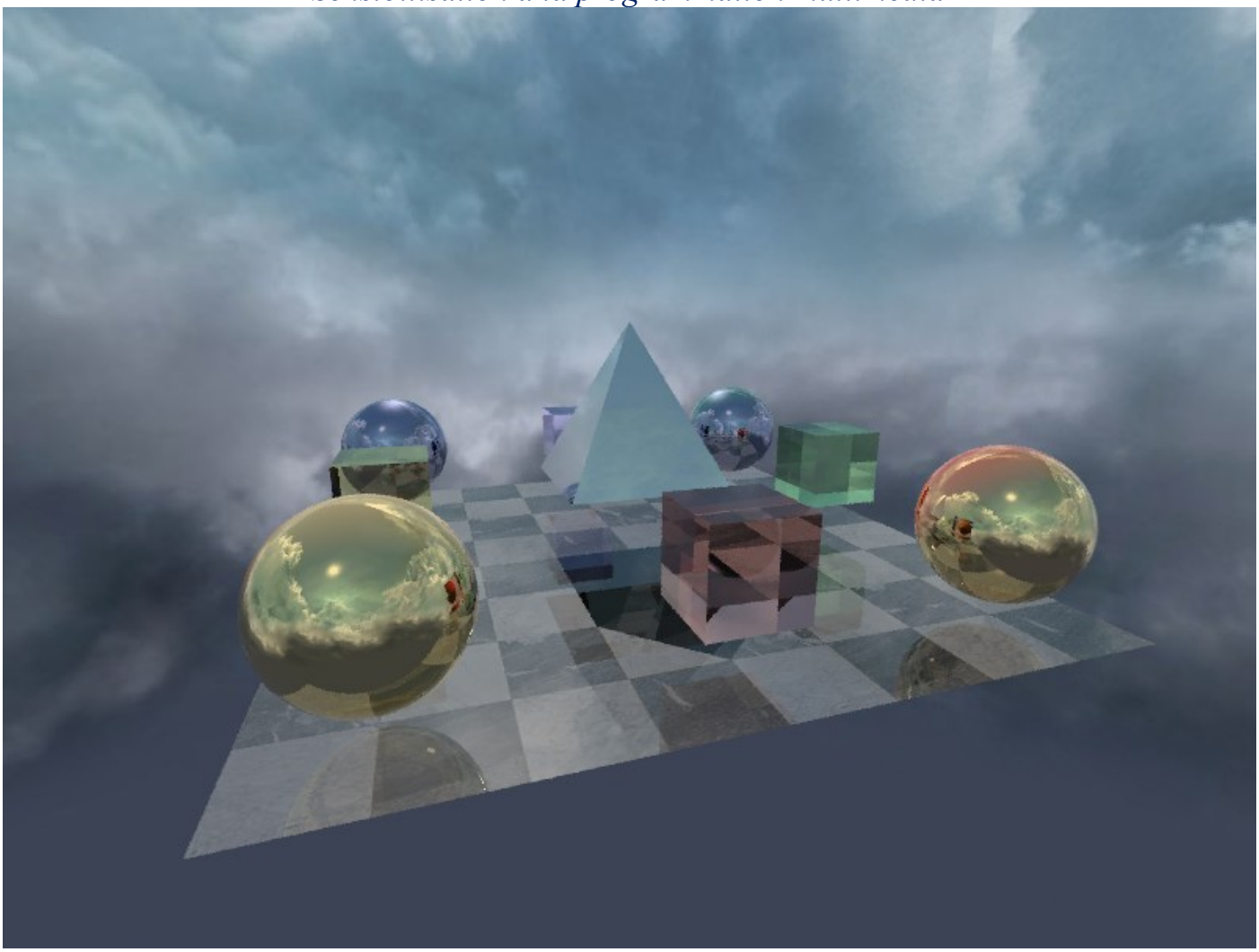
```
yellow_glass = Refractive(n=vec3(1.5 + 0.j, 1.5 + 0j, 1.5 + 5e-8j))

gold_metal = Glossy(diff_color=rgb(1., .572, .184), n=vec3(0.15+3.58j, 0.4+2.37j,
1.54+1.91j),
                    roughness=0.0, spec_coeff=0.2, diff_coeff=0.8)  # n = index of
refraction
bluish_metal = Glossy(diff_color=rgb(0.0, 0, 0.1), n=vec3(
    1.3+1.91j, 1.3+1.91j, 1.4+2.91j), roughness=0.2, spec_coeff=0.5, diff_coeff=0.3)
silver_metal = Glossy(diff_color=rgb(0.8, 0.8, 0.8), n=vec3(
    0.2+3.1j, 0.2+3.1j, 0.2+3.1j), roughness=0.1, spec_coeff=0.8, diff_coeff=0.2)
# red_metal = Glossy(diff_color=rgb(1., 0., 0.), n=vec3(0.15+3.58j, 0.4+2.37j,
1.54+1.91j), roughness=0.0, spec_coeff=0.2, diff_coeff=0.8)
# red_metal = Glossy(    diff_color=rgb(1., 0., 0.),    n=vec3(0.2+3.4j, 1.01+2.73j,
1.34+2.27j),    roughness=0.0,    spec_coeff=0.2,    diff_coeff=0.8)
# red_metal = Glossy(    diff_color=rgb(1., 0., 0.),    n=vec3(0.3+2.5j, 0.2+3j,
0.2+3.2j),    roughness=0.05,    spec_coeff=0.4,    diff_coeff=0.6)
red_metal = Glossy(
    diff_color=rgb(1.0, 0.0, 0.0),
    n=vec3(
        0.25 + 2.0j,   # Rouge peu absorbé
        0.20 + 4.0j,   # Vert fortement absorbé
        0.15 + 4.5j    # Bleu encore davantage absorbé
    ),
    roughness=0.05,      # Micro-flou pour élargir le rouge
    spec_coeff=0.4,
    diff_coeff=0.6
)

green_metal = Glossy(diff_color=rgb(0., 1., 0.), n=vec3(
    1.3+1.91j, 1.3+1.91j, 1.4+2.91j), roughness=0.2, spec_coeff=0.5, diff_coeff=0.3)


# Créez une liste d'images que vous souhaitez inclure dans l'animation
images = []

frame_angle = 360 / num_frames  # Angle entre chaque frame

# Coordonnées de l'objet à filmer
obj_x = 0.0
obj_y = 0.0
obj_z = 0.0

# Coordonnées de la caméra
cam_x = 0.0
cam_y = 0.25
cam_z = 3.0  # Distance de la caméra par rapport à l'objet à filmer

for pos_Cam in range(num_frames):

    # Calcul des coordonnées de la caméra
    theta = math.radians(pos_Cam * frame_angle)  # Convertir l'angle en radians
    cam_x = math.cos(theta)*4
    # Distance de la caméra par rapport à l'objet à filmer
    cam_y = 1.0
```

```
    cam_z = math.sin(theta)*4

    # Calcul des coordonnées de look_at
    # look_at_x = obj_x - cam_x
    # look_at_y = obj_y - cam_y
    # look_at_z = obj_z - cam_z
    look_at_x = 0
    look_at_y = 0
    look_at_z = 0

    # Normalisation du vecteur look_at
    # look_at_length = math.sqrt(look_at_x**2 + look_at_y**2 + look_at_z**2)
    look_at_length = 3
    # look_at_x /= look_at_length
    # look_at_y /= look_at_length
    # look_at_z /= look_at_length

    # Set Scene
    Sc = Scene()
    Sc.add_Camera(look_from=vec3(cam_x, cam_y, cam_z), look_at=vec3(look_at_x,
look_at_y, look_at_z),
                  screen_width=width,
                  screen_height=height)

    Sc.add_DirectionalLight(Ldir=vec3(0.0, 0.5, 0.5),
                            color=rgb(0.5, 0.5, 0.5))

    Sc.add(Plane(material=floor,  center=vec3(0, -0.5, 0), width=4.0, height=4.0,
                 u_axis=vec3(1.0, 0, 0), v_axis=vec3(0, 0, -1.0), max_ray_depth=5))

    cb = Cuboid(material=green_glass, center=vec3(-1.5, 0, 0),
                width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb)

    cb1 = Cuboid(material=yellow_glass, center=vec3(1.5, 0, 0),
                 width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb1.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb1)

    cb2 = Cuboid(material=blue_glass, center=vec3(0, 0, 1.5),
                 width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb2.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb2)

    cb3 = Cuboid(material=red_glass, center=vec3(0, 0, -1.5),
                 width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb3.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb3)

    center = vec3(0, 0, 0)
    material = silver_metal
    max_ray_depth = 3
```

```
        Sc.add(Sphere(material=bluish_metal, center=vec3(1.5, 0.0, 1.5),
                      radius=.4, shadow=False, max_ray_depth=3))
        Sc.add(Sphere(material=red_metal, center=vec3(-1.5, 0.0, -1.5),
                      radius=.4, shadow=False, max_ray_depth=3))
        Sc.add(Sphere(material=green_metal, center=vec3(-1.5, 0.0, 1.5),
                      radius=.4, shadow=False, max_ray_depth=3))
        Sc.add(Sphere(material=gold_metal, center=vec3(1.5, 0.0, -1.5),
                      radius=.4, shadow=False, max_ray_depth=3))

        # see sightpy/backgrounds
        Sc.add_Background("stormydays.png")

        # Render
        img = Sc.render(samples_per_pixel=4)
        # Vérifier si le fichier existe déjà
        if os.path.exists("scene0_lum_JFA"+str(pos_Cam)+".png"):
            os.remove("scene0_lum_JFA"+str(pos_Cam)+".png")
        img.save("scene0_lum_JFA"+str(pos_Cam)+".png")
        # img.show()
        display.Image("scene0_lum_JFA"+str(pos_Cam)+".png")
        image = Image.open("scene0_lum_JFA"+str(pos_Cam)+".png")
        images.append(image)
        print("Image "+str(pos_Cam)+" créée avec succès!")

    # Créer une animation GIF à partir des images
    print("Animation  créée avec succès!")
```

2°) ***Exercice 2 :***

Rajouter un triangle de couleur argent au milieu de la scène précédente :



Merci à clément BARATIN pour son aide.

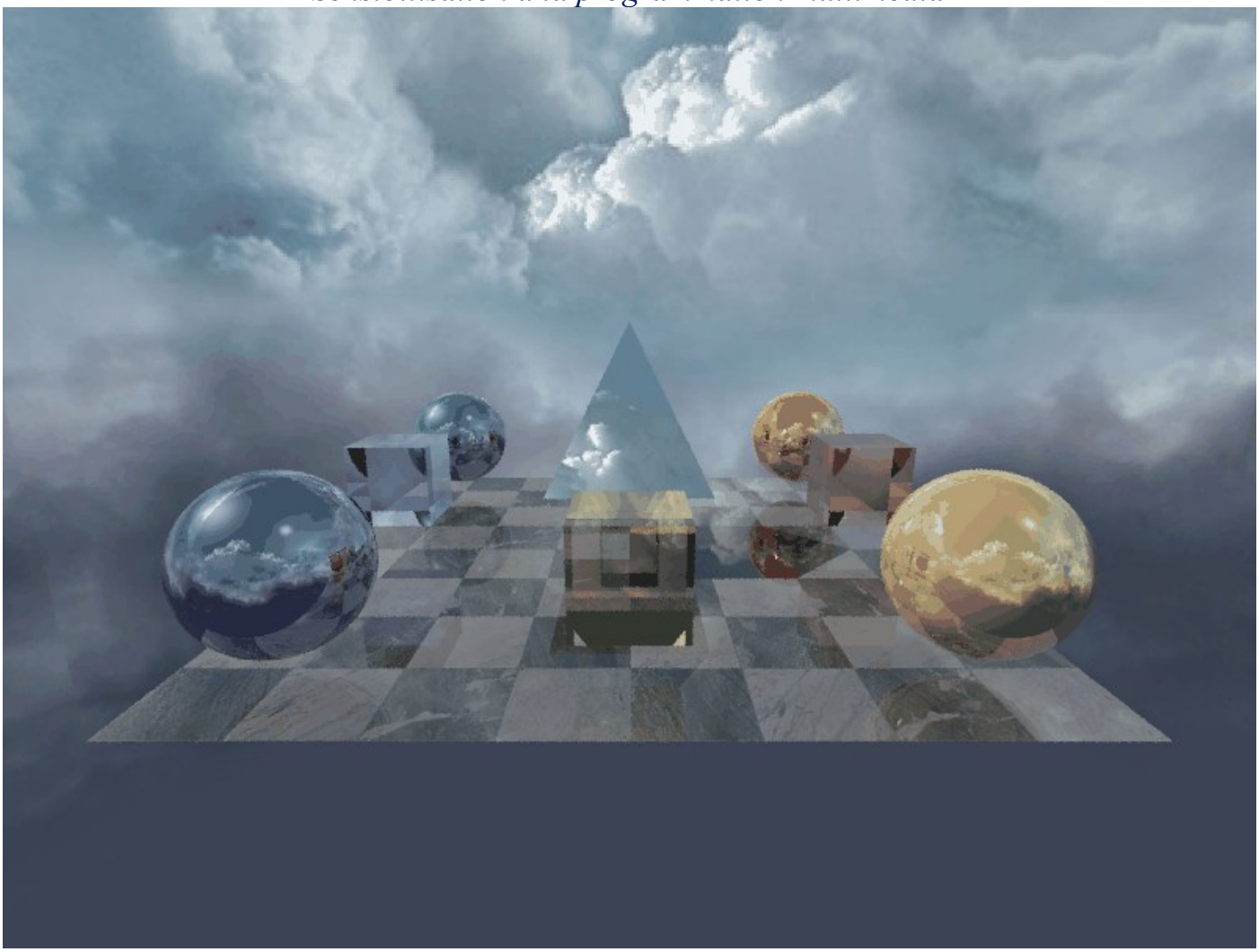
```
from IPython import display
from sightpy import *
from PIL import Image, ImageSequence
import math
import os
import numpy as np
from sightpy.geometry.primitive import Primitive
from sightpy.geometry.collider import Collider

width = 800
height = 600
num_frames = 10  # Nombre de frames pour effectuer une rotation complète
# define materials to use
floor = Glossy(diff_color=image("checkered_floor.png", repeat=2.),  roughness=0.2,
               spec_coeff=0.3, diff_coeff=0.7, n=vec3(2.2, 2.2, 2.2))

# n = index of refraction
silver_metal = Glossy(diff_color=rgb(0.8, 0.8, 0.8), n=vec3(
    0.2+3.1j, 0.2+3.1j, 0.2+3.1j), roughness=0.1, spec_coeff=0.8, diff_coeff=0.2)

# Créez une liste d'images que vous souhaitez inclure dans l'animation
images = []
frame_angle = 360 / num_frames  # Angle entre chaque frame
```

```
# Coordonnées de l'objet à filmer
obj_x = 0.0
obj_y = 0.0
obj_z = 0.0

# Coordonnées de la caméra
cam_x = 0.0
cam_y = 0.25
cam_z = 3.0  # Distance de la caméra par rapport à l'objet à filmer

for pos_Cam in range(num_frames):

    # Calcul des coordonnées de la caméra
    theta = math.radians(pos_Cam * frame_angle)  # Convertir l'angle en radians
    cam_x = math.cos(theta)*4
    # Distance de la caméra par rapport à l'objet à filmer
    cam_y = 1.0
    cam_z = math.sin(theta)*4

    # Calcul des coordonnées de look_at
    # look_at_x = obj_x - cam_x
    # look_at_y = obj_y - cam_y
    # look_at_z = obj_z - cam_z
    look_at_x = 0
    look_at_y = 0
    look_at_z = 0

    # Normalisation du vecteur look_at
    # look_at_length = math.sqrt(look_at_x**2 + look_at_y**2 + look_at_z**2)
    look_at_length = 3
    # look_at_x /= look_at_length
    # look_at_y /= look_at_length
    # look_at_z /= look_at_length

    # Set Scene
    Sc = Scene()
    Sc.add_Camera(look_from=vec3(cam_x, cam_y, cam_z), look_at=vec3(look_at_x,
look_at_y, look_at_z),
                  screen_width=width,
                  screen_height=height)

    Sc.add_DirectionalLight(Ldir=vec3(0.0, 0.5, 0.5),
                            color=rgb(0.5, 0.5, 0.5))

    Sc.add(Plane(material=floor,  center=vec3(0, -0.5, 0), width=4.0, height=4.0,
                 u_axis=vec3(1.0, 0, 0), v_axis=vec3(0, 0, -1.0), max_ray_depth=5))

    center = vec3(0, 0, 0)
    material = silver_metal
    max_ray_depth = 3

    p1 = vec3(-0.5, 0, -0.5)
    p2 = vec3(0.5, 0, -0.5)
    p3 = vec3(0.5, 0, 0.5)
```

```
    # Define additional points for the other three triangles of the pyramid
    p4 = vec3(-0.5, 0, 0.5)  # Point at the base of the triangle
    p5 = vec3(0, 1, 0)  # Point at the base of the right side of the triangle
    p6 = vec3(0, 0, 0)  # Point at the base of the left side of the triangle

    # Add the other three triangles to form the pyramid
    Sc.add(Triangle(center, material, p1, p2, p5, max_ray_depth, shadow=True))
    Sc.add(Triangle(center, material, p2, p3, p5, max_ray_depth, shadow=True))
    Sc.add(Triangle(center, material, p3, p4, p5, max_ray_depth, shadow=True))
    Sc.add(Triangle(center, material, p4, p1, p5, max_ray_depth, shadow=True))

# Make sure to update the intersection logic inside the Triangle class to handle
these new triangles properly.    # Sc.add(Sphere(material=silver_metal,
center=vec3(0, 0.0, 0), radius=.4, shadow=False, max_ray_depth=3))
    # see sightpy/backgrounds
    Sc.add_Background("miramar.jpeg")

    # Render
    img = Sc.render(samples_per_pixel=4)
    # Vérifier si le fichier existe déjà
    if os.path.exists("scene1_JFA"+str(pos_Cam)+".png"):
        os.remove("scene1_JFA"+str(pos_Cam)+".png")
    img.save("scene1_JFA"+str(pos_Cam)+".png")
    # img.show()
    display.Image("scene1_JFA"+str(pos_Cam)+".png")
    image = Image.open("scene1_JFA"+str(pos_Cam)+".png")
    images.append(image)
    print("Image "+str(pos_Cam)+" créée avec succès!")

# Vérifier si le fichier existe déjà
if os.path.exists("animation1_JFA.gif"):
    os.remove("animation1_JFA.gif")
# Créez un nouveau fichier GIF en utilisant la première image comme image de fond
gif_path = 'animation1_JFA.gif'
images[0].save(gif_path, save_all=True, append_images=images[1:],
               optimize=False, duration=200, loop=0)

print("Animation GIF créée avec succès!")
```

3°) ***Exercice 3 :***

A partir de la scène précédente, réaliser une animation panoramique sur 360° en .gif :



```
from IPython import display
from sightpy import *
from PIL import Image, ImageSequence
import math
import os
import numpy as np
from sightpy.geometry.primitive import Primitive
from sightpy.geometry.collider import Collider

width = 800
height = 600
num_frames = 360  # Nombre de frames pour effectuer une rotation complète
# define materials to use
floor = Glossy(diff_color=image("checkered_floor.png", repeat=2.),  roughness=0.2,
               spec_coeff=0.3, diff_coeff=0.7, n=vec3(2.2, 2.2, 2.2))


# n = index of refraction
blue_glass = Refractive(n=vec3(1.5 + 4e-8j, 1.5 + 4e-8j, 1.5 + 0j))
blue_cyan_glass = Refractive(n=vec3(1.5 + 5e-8j, 1.5 + + 5e-8j, 1.5 + 0j))
green_glass = Refractive(n=vec3(1.5 + 4e-8j, 1.5 + 0.j, 1.5 + 4e-8j))
red_glass = Refractive(n=vec3(1.5 + 0.j, 1.5 + 5e-8j, 1.5 + 5e-8j))
# Calcul de la couleur jaune
# yellow_glass = Refractive(n=green_glass.n + red_glass.n)
yellow_glass = Refractive(n=vec3(1.5 + 0.j, 1.5 + 0j, 1.5 + 5e-8j))
```

```
gold_metal = Glossy(diff_color=rgb(1., .572, .184), n=vec3(0.15+3.58j, 0.4+2.37j,
1.54+1.91j),
                    roughness=0.0, spec_coeff=0.2, diff_coeff=0.8)  # n = index of
refraction
bluish_metal = Glossy(diff_color=rgb(0.0, 0, 0.1), n=vec3(
    1.3+1.91j, 1.3+1.91j, 1.4+2.91j), roughness=0.2, spec_coeff=0.5, diff_coeff=0.3)
silver_metal = Glossy(diff_color=rgb(0.8, 0.8, 0.8), n=vec3(
    0.2+3.1j, 0.2+3.1j, 0.2+3.1j), roughness=0.1, spec_coeff=0.8, diff_coeff=0.2)
red_metal = Glossy(diff_color=rgb(1., 0., 0.), n=vec3(
    0.15+3.58j, 0.4+2.37j, 1.54+1.91j), roughness=0.0, spec_coeff=0.2,
diff_coeff=0.8)

green_metal = Glossy(diff_color=rgb(0., 1., 0.), n=vec3(
    1.3+1.91j, 1.3+1.91j, 1.4+2.91j), roughness=0.2, spec_coeff=0.5, diff_coeff=0.3)


# Créez une liste d'images que vous souhaitez inclure dans l'animation
images = []

frame_angle = 360 / num_frames  # Angle entre chaque frame

# Coordonnées de l'objet à filmer
obj_x = 0.0
obj_y = 0.0
obj_z = 0.0

# Coordonnées de la caméra
cam_x = 0.0
cam_y = 0.25
cam_z = 3.0  # Distance de la caméra par rapport à l'objet à filmer

for pos_Cam in range(num_frames):

    # Calcul des coordonnées de la caméra
    theta = math.radians(pos_Cam * frame_angle)  # Convertir l'angle en radians
    cam_x = math.cos(theta)*4
    # Distance de la caméra par rapport à l'objet à filmer
    cam_y = 1.0
    cam_z = math.sin(theta)*4

    # Calcul des coordonnées de look_at
    # look_at_x = obj_x - cam_x
    # look_at_y = obj_y - cam_y
    # look_at_z = obj_z - cam_z
    look_at_x = 0
    look_at_y = 0
    look_at_z = 0

    # Normalisation du vecteur look_at
    # look_at_length = math.sqrt(look_at_x**2 + look_at_y**2 + look_at_z**2)
    look_at_length = 3
    # look_at_x /= look_at_length
    # look_at_y /= look_at_length
```

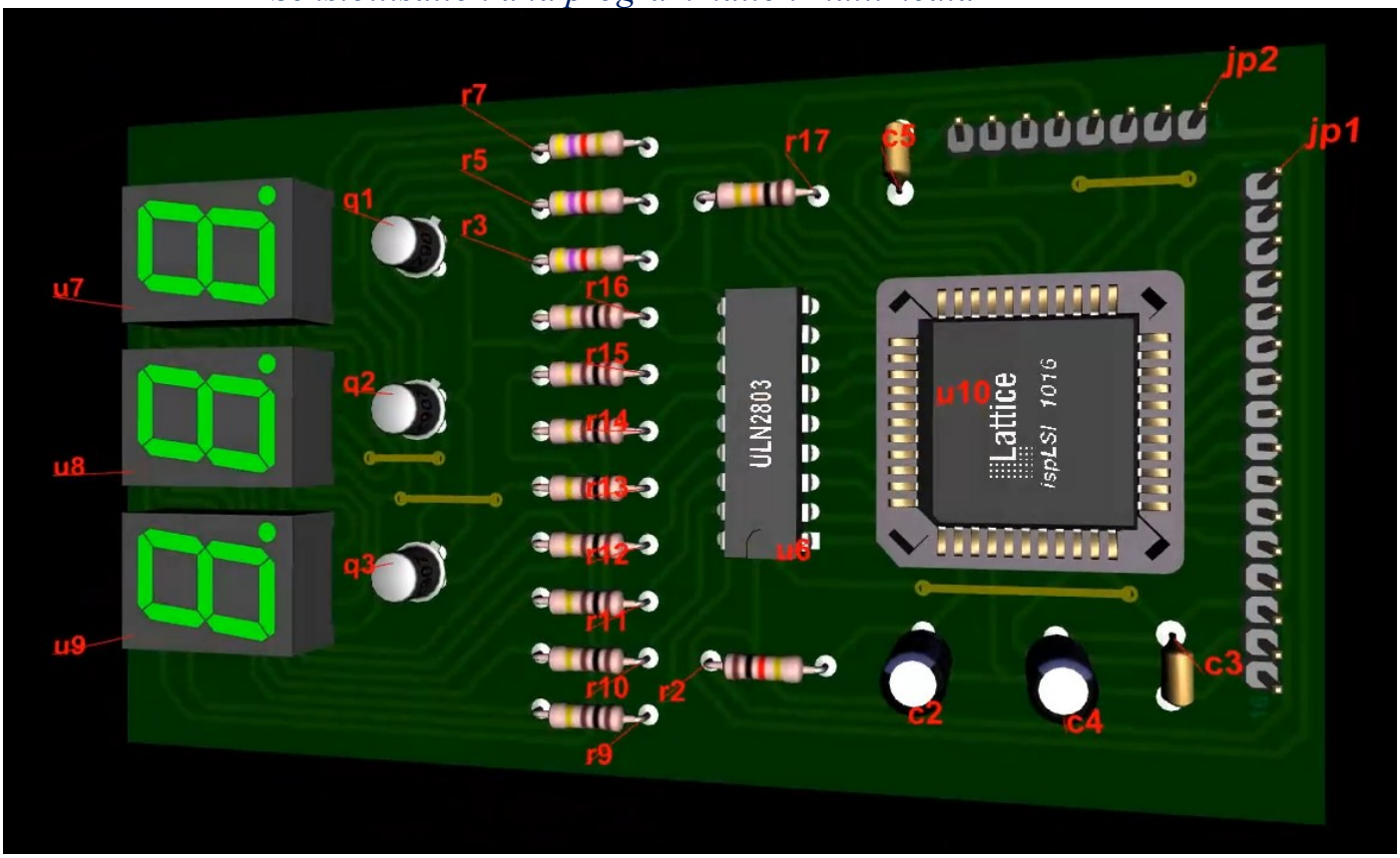
```
    # look_at_z /= look_at_length

    # Set Scene
    Sc = Scene()
    Sc.add_Camera(look_from=vec3(cam_x, cam_y, cam_z), look_at=vec3(look_at_x,
look_at_y, look_at_z),
                  screen_width=width,
                  screen_height=height)

    Sc.add_DirectionalLight(Ldir=vec3(0.0, 0.5, 0.5),
                            color=rgb(0.5, 0.5, 0.5))

    Sc.add(Plane(material=floor,  center=vec3(0, -0.5, 0), width=4.0, height=4.0,
                 u_axis=vec3(1.0, 0, 0), v_axis=vec3(0, 0, -1.0), max_ray_depth=5))

    cb = Cuboid(material=green_glass, center=vec3(-1.5, 0, 0),
                width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb)

    cb1 = Cuboid(material=yellow_glass, center=vec3(1.5, 0, 0),
                 width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb1.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb1)

    cb2 = Cuboid(material=blue_glass, center=vec3(0, 0, 1.5),
                 width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb2.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb2)

    cb3 = Cuboid(material=red_glass, center=vec3(0, 0, -1.5),
                 width=0.5, height=0.5, length=0.5, shadow=False,  max_ray_depth=5)
    # cb3.rotate(θ = 30, u = vec3(0,1,0))
    Sc.add(cb3)

    center = vec3(0, 0, 0)
    material = silver_metal
    max_ray_depth = 3

    p1 = vec3(-0.5, 0, -0.5)
    p2 = vec3(0.5, 0, -0.5)
    p3 = vec3(0.5, 0, 0.5)
    # Define additional points for the other three triangles of the pyramid
    p4 = vec3(-0.5, 0, 0.5)  # Point at the base of the triangle
    p5 = vec3(0, 1, 0)  # Point at the base of the right side of the triangle
    p6 = vec3(0, 0, 0)  # Point at the base of the left side of the triangle

    # Add the other three triangles to form the pyramid
    Sc.add(Triangle(center, material, p1, p2, p5, max_ray_depth, shadow=True))
    Sc.add(Triangle(center, material, p2, p3, p5, max_ray_depth, shadow=True))
    Sc.add(Triangle(center, material, p3, p4, p5, max_ray_depth, shadow=True))
    Sc.add(Triangle(center, material, p4, p1, p5, max_ray_depth, shadow=True))

    Sc.add(Sphere(material=bluish_metal, center=vec3(1.5, 0.0, 1.5),
```

```
                  radius=.4, shadow=False, max_ray_depth=3))
    Sc.add(Sphere(material=red_metal, center=vec3(-1.5, 0.0, -1.5),
                  radius=.4, shadow=False, max_ray_depth=3))
    Sc.add(Sphere(material=green_metal, center=vec3(-1.5, 0.0, 1.5),
                  radius=.4, shadow=False, max_ray_depth=3))
    Sc.add(Sphere(material=gold_metal, center=vec3(1.5, 0.0, -1.5),
                  radius=.4, shadow=False, max_ray_depth=3))

    # see sightpy/backgrounds
    Sc.add_Background("miramar.jpeg")

    # Render
    img = Sc.render(samples_per_pixel=4)
    # Vérifier si le fichier existe déjà
    if os.path.exists("scene3_JFA"+str(pos_Cam)+".png"):
        os.remove("scene3_JFA"+str(pos_Cam)+".png")
    img.save("scene3_JFA"+str(pos_Cam)+".png")
    # img.show()
    display.Image("scene3_JFA"+str(pos_Cam)+".png")
    image = Image.open("scene3_JFA"+str(pos_Cam)+".png")
    images.append(image)
    print("Image "+str(pos_Cam)+" créée avec succès!")

# Vérifier si le fichier existe déjà
if os.path.exists("animation3_JFA.gif"):
    os.remove("animation3_JFA.gif")
# Créez un nouveau fichier GIF en utilisant la première image comme image de fond
gif_path = 'animation3_JFA.gif'
images[0].save(gif_path, save_all=True, append_images=images[1:],
               optimize=False, duration=200, loop=0)

print("Animation3 GIF créée avec succès!")
```

D. ***Aller plus loin :***

Si vous voulez aller plus en avant dans le Raytracing, je vous conseille le logiciel POV-Ray qui permet de faire de belles images :

- *http://www.povray.org/*
- *https://hof.povray.org/*



Carte réalisée avec POV Ray, JFA 2008

E. Webographie :

- https://excamera.com/sphinx/article-ray.html
- https://github.com/jamesbowman/raytrace
- https://github.com/rafael-fuente/Python-Raytracer
- https://github.com/jrmiranda/raytracer

N'oubliez pas le NETTOYAGE et la reconfiguration à l'état d'origine des machines en fin de séance