

# T4 - Traçado de Raios

In [1]:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from math import sin, cos, sqrt
4 import sys
5
6 TOL = sys.float_info.epsilon
7 print(TOL)
```

2.220446049250313e-16

In [2]:

```
1 def vetor(x, y, z):
2     return np.array([x, y, z], dtype = np.float)
3
4 def dot(u, v):
5     return (u[0] * v[0] + u[1] * v[1] + u[2] * v[2])
6
7 def norma(u):
8     return sqrt(dot(u, u))
9
10 def unitario(v):
11     s = norma(v)
12
13     if (s > TOL):
14         return v/s
15
16     else:
17         return None
18
19 def ang(v1, v2):
20     num = dot(v1, v2)
21     den = norm(v1) * norm(v2)
22
23     return np.arccos(num/den) * 180 / np.pi if den > TOL else 0
24
25 def reflète(v, n):
26     r = 2 * dot(v, n) * n - v
27     return r
28
29 def cross(u, v):
30     return vetor(u[1] * v[2] - u[2] * v[1], u[2] * v[0] - u[0] * v[2], u[0] * v[1] - u[1] * v[0])
```

In [3]:

```
1 class Camera:
2     def __init__(self, fov, w, h, dfocal, eye, at, up):
3         self.fov = fov
4         self.w = w
5         self.h = h
6         self.dfocal = dfocal
7
8         self.a = 2 * dfocal * np.tan(fov * np.pi / 360)
9         self.b = self.a * w / h
10        self.eye = eye
11
12        self.ze = unitario(at - eye)
13        self.xe = unitario(cross(self.ze, up))
14        self.ye = cross(self.ze, self.xe)
15        self.img = np.full(shape = (h, w, 3), fill_value = 0.43, dtype = np.float)
16
17    def ray_to(self, x_im, y_im):
18        dx = self.b * (x_im / self.w - 0.5)
19        dy = self.a * (y_im / self.h - 0.5)
20        dz = self.dfocal
21
22        ray = dx * self.xe + dy * self.ye + dz * self.ze
23
24        return ray
25
26    def get_eye(self):
27        return self.eye
28
29    def pixel (self, x_im, y_im, rgb):
30        self.img[y_im, x_im, :] = rgb
31
32    def get_pixel(self, x, y):
33        return self.img[y, x, :]
34
35    def set_pixel(self, x, y, rgb):
36        self.img[y, x, :] = rgb
37
38    def get_w(self):
39        return self.w
40
41    def get_h(self):
42        return self.h
43
44    def imshow(self):
45        self.img[self.img < 0] = 0
46        self.img[self.img > 1] = 1
47
48        plt.figure(figsize = (8, 8))
49        plt.imshow(self.img)
50        plt.show()
51
52    def show(self):
53        print("CAMERA:")
54        print("fov =", self.fov, "d =", self.dfocal)
55        print("(w, h) = (", self.w, ", ", self.h, ")")
56        print("(b, a) = (", self.b, ", ", self.a, ")")
57        print("xe =", self.xe)
58        print("ye =", self.ye)
59        print("ze =", self.ze)
```



In [4]:

```
1 class Poligono:
2     def __init__(self, pp, pn, material):
3         self.pp = pp
4         self.pn = pn
5         self.ni = None
6         self.material = material
7         self.hitface = None
8         pmin = np.amin(pp, 0)
9         pmax = np.amax(pp, 0)
10        self.size = np.amax(pmax - pmin)
11
12    def show(self):
13        print("POLIGONO:")
14        for i in range(len(self.pp)):
15            print(f'Plano {i}: ponto = {self.pp[i]}, normal = {self.pn[i]}')
16
17        self.material.show()
18        print("\n")
19
20    def intercepta(self, origem, direcao):
21        te = 0
22        ts = np.inf
23        obji = None
24
25        for i in range(len(self.pp)):
26            pn = self.pn[i]
27            pp = self.pp[i]
28            num = dot(pp - origem, pn)
29            den = dot(direcao, pn)
30
31            if den < -TOL:
32                t = num / den
33
34                if t > te:
35                    te = t
36                    obji = self
37                    self.ni = pn
38                    self.hitface = i
39
40            elif den > TOL:
41                t = num / den
42                ts = t if t < ts else ts
43                obji = self
44
45            if ts < te:
46                te = np.inf
47                obji = None
48                self.hitface = None
49                break
50
51        return te, obji
52
53    def get_mat(self):
54        return self.material
55
56    def normal(self, ponto):
57        return self.ni
58
59    def get_texture(self, ponto):
```

```
60     axis = [[1,2],[2,0],[0,1],[1,2],[2,0],[0,1]]
61     i = axis[self.hitface][0]
62     j = axis[self.hitface][1]
63     u = abs(ponto[i] - self.pp[self.hitface][i]) / self.size
64     v = abs(ponto[j] - self.pp[self.hitface][j]) / self.size
65     return u, v
```

In [5]:

```
1 class Caixa(Poligono):
2     def __init__(self, pmin, pmax, material):
3         super().__init__([pmin, pmin, pmin, pmax, pmax, pmax],
4                           [vetor(-1, 0, 0), vetor(0, -1, 0), vetor(0, 0, -1),
5                             vetor(1, 0, 0), vetor(0, 1, 0), vetor(0, 0, 1)], material)
```

In [6]:

```
1 class Esfera:
2     def __init__(self, raio, centro, material):
3         self.raio = raio
4         self.centro = centro
5         self.material = material
6
7     def show(self):
8         print("ESFERA:")
9         print(f'raio = {self.raio}')
10        print(f'centro = {self.centro}')
11        self.material.show()
12        print("\n")
13
14    def intercepta(self, origem, direcao):
15        a = dot(direcao, direcao)
16        b = 2 * dot(direcao, origem - self.centro)
17        c = dot(origem - self.centro, origem - self.centro) - self.raio**2
18        delta = b**2 - 4 * a * c
19
20        if delta > TOL:
21            raiz = sqrt(delta)
22            t1 = (-b - raiz) / (2 * a)
23            t2 = (-b + raiz) / (2 * a)
24            t = t1 if t1 < t2 else t2
25
26            if t > 0:
27                return t, self
28            else:
29                return np.inf, None
30        else:
31            return np.inf, None
32
33    def normal(self, ponto):
34        return unitario(ponto - self.centro)
35
36    def get_mat(self):
37        return self.material
38
39    def get_texture(self, ponto):
40        p = ponto - self.centro
41        phi = np.arctan2(p[1], p[0])
42        theta = np.arctan2(sqrt(p[0]**2 + p[1]**2), p[2])
43        u = 0.5 * (1 + phi / np.pi)
44        v = theta / np.pi
45        return u, v
```

In [7]:

```
1 class Material:
2     def __init__(self, kd, ks, ns, espelhamento, opacidade, textura):
3         self.kd = kd
4         self.ks = ks
5         self.ns = ns
6         self.espelhamento = espelhamento
7         self.opacidade = opacidade
8         if textura:
9             img = plt.imread(textura)
10            img = img / np.amax(img)
11            img = img[:, :, :3]
12            self.textura = img
13        else:
14            self.textura = None
15
16    def show(self):
17        print("MATERIAL:")
18        print("Cor Difusa = ", self.kd)
19        print("Cor Especular = ", self.ks)
20        print("Coeficiente Especular = ", self.ns)
21        print("Espelhamento = ", self.espelhamento)
22        print("Opacidade = ", self.opacidade)
23        print("\n")
24
25    def get_phong(self, u, v):
26        if self.textura is None:
27            return self.kd, self.ks, self.ns
28        else:
29            u = u - int(u)
30            v = v - int(v)
31            ht, wt = self.textura.shape[:2]
32            x = int(u * wt)
33            y = int(v * ht)
34            kd = self.textura[y, x, :]
35            return kd, self.ks, self.ns
36
37    def get_espelhamento(self):
38        return self.espelhamento
39
40    def get_opacidade(self):
41        return self.opacidade
```

In [8]:

```
1 class Luz:
2     def __init__(self, posicao, intensidade):
3         self.posicao = posicao
4         self.intensidade = intensidade
5
6     def show(self):
7         print("LUZ:")
8         print(f'Posicao = {self.posicao}')
9         print(f'Intensidade = {self.intensidade}')
10        print("\n")
11
12    def get_pos(self):
13        return self.posicao
14
15    def get_int(self):
16        return self.intensidade
```



In [9]:

```
1 class Cena:
2     def __init__(self, camera, objetos, luzes, ambiente):
3         self.camera = camera
4         self.objetos = objetos
5         self.materiais = materiais
6         self.luzes = luzes
7         self.ambiente = ambiente
8
9     def show(self):
10        print("CENA:")
11
12        self.camera.show()
13
14        for obj in self.objetos:
15            obj.show()
16
17        for mat in self.materiais:
18            mat.show()
19
20        for luz in self.luzes:
21            luz.show()
22
23        print("Ambiente = ", self.ambiente)
24        print("\n")
25
26    def sombra(self,objeto, origem, direcao):
27        ti = np.inf
28        obji = None
29
30        for obj in self.objetos:
31            if obj is not objeto:
32                t, objx = obj.intercepta(origem, direcao)
33
34                if t > TOL and t < 1:
35                    return True
36
37        return False
38
39    def shade(self, objeto, origem, ponto, nrec):
40        u, v = objeto.get_texture(ponto)
41        kd, ks, ns = objeto.get_mat().get_phong(u, v)
42        cor = self.ambiente * kd
43        material = objeto.get_mat()
44        normal = objeto.normal(ponto)
45        toEye = unitario(origem - ponto)
46
47        for luz in self.luzes:
48            toLuz = luz.get_pos() - ponto
49            naSombra = self.sombra(objeto, ponto, toLuz)
50            if naSombra is False:
51                vluz = unitario(toLuz)
52                cosseno1 = dot(vluz, normal)
53
54                if cosseno1 > TOL:
55                    # Difusa
56                    cor += luz.get_int() * kd * cosseno1
57
58                    # Especular
59                    cosseno2 = dot(reflete(vluz, normal), toEye)
```

```

60
61         if cosseño2 > TOL:
62             cor += luz.get_int() * ks * (cosseño2 ** ns)
63
64     # Recursão nos raios
65     espelhamento = objeto.get_mat().get_espelhamento()
66     if espelhamento > 0 and nrec < 5:
67         dir_espelhada = reflete(toEye, normal)
68         rgb_espelhado, obji = self.trace(objeto, ponto, dir_espelhada, nrec + 1)
69         if obji:
70             cor = (1 - espelhamento) * cor + espelhamento * rgb_espelhado
71         else:
72             cor = (1 - espelhamento) * cor
73
74     return cor
75
76 def trace(self, objeto, origem, direcao, nrec):
77     ti = np.inf
78     obji = None
79
80     for obj in self.objetos:
81         if obj is not objeto:
82             t, objx = obj.intercepta(origem, direcao)
83             if t < ti:
84                 ti = t
85                 obji = objx
86
87     if obji and ti > TOL:
88         ponto = origem + ti * direcao
89         normal = obji.normal(ponto)
90
91         rgb = self.shade(obji, origem, ponto, nrec)
92         return rgb, obji
93
94     else:
95         return None, None
96
97 def render(self):
98     w = self.camera.get_w()
99     h = self.camera.get_h()
100     origem = self.camera.get_eye()
101
102     for y in range(h):
103         for x in range(w):
104             direcao = self.camera.ray_to(x, y)
105             rgb, obji = self.trace(None, origem, direcao, 0)
106
107             if obji:
108                 cor = self.camera.get_pixel(x, y)
109                 self.camera.set_pixel(x, y, cor + rgb)

```

In [10]:

```

1 eye = vetor(100, 40, 40)
2 at = vetor(0, 0, 0)
3 up = vetor(0, 1, 0)
4 camera = Camera(90, 300, 300, 30, eye, at, up)
5 #camera.show()

```

In [11]:

```
1 luz1 = Luz(vetor(60, 120, 40), vetor(0.5, 0.5, 0.5))
2 luz2 = Luz(vetor(100, 40, 40), vetor(0.5, 0.5, 0.5))
3 #Luz.show()
```

In [12]:

```
1 azul_metalico = Material(vetor(0, 0, 1), vetor(1, 1, 1), 30, 0, 1, "images\marmore.bmp")
2 amarelo_fosco = Material(vetor(0.7, 0.7, 0), vetor(0, 0, 0), 1, 0, 1, "images\mahogany.jpg")
3 amarelo_espelho = Material(vetor(0.7, 0.7, 0), vetor(0, 0, 0), 1, 1, 1, "images\mahogany.jpg")
4 #azul_metalico.show()
5 #amarelo_fosco.show()
6 #amarelo_espelho.show()
```

In [13]:

```
1 esfera = Esfera(25, vetor(0, 20, 0), azul_metalico)
2 #esfera.show()
```

In [14]:

```
1 piso = Caixa(vetor(-80, -50, -50), vetor(50, -45, 50), amarelo_fosco)
2 parede = Caixa(vetor(-80, -50, -60), vetor(50, 50, -50), amarelo_espelho)
3 #piso.show()
4 #parede.show()
```

In [15]:

```
1 objetos = [esfera, piso, parede]
2 materiais = [azul_metalico, amarelo_fosco, amarelo_espelho]
3 luzes = [luz1, luz2]
4 ambiente = vetor(0.1, 0.1, 0.1)
5 cena = Cena(camera, objetos, luzes, ambiente)
6 cena.show()
```

CENA:

CAMERA:

```
fov = 90 d = 30
(w, h) = ( 300 , 300 )
(b, a) = ( 59.999999999999986 , 59.99999999999999 )
xe = [ 0.37139068  0.          -0.92847669]
ye = [ 0.32325409 -0.93743687  0.12930164]
ze = [-0.87038828 -0.34815531 -0.34815531]
```

ESFERA:

```
raio = 25
centro = [ 0. 20.  0.]
MATERIAL:
Cor Difusa = [0. 0. 1.]
Cor Especular = [1. 1. 1.]
Coeficiente Especular = 30
Espelhamento = 0
Opacidade = 1
```

POLIGONO:

```
Plano 0: ponto = [-80. -50. -50.], normal = [-1.  0.  0.]
Plano 1: ponto = [-80. -50. -50.], normal = [ 0. -1.  0.]
Plano 2: ponto = [-80. -50. -50.], normal = [ 0.  0. -1.]
Plano 3: ponto = [ 50. -45.  50.], normal = [1.  0.  0.]
Plano 4: ponto = [ 50. -45.  50.], normal = [0.  1.  0.]
Plano 5: ponto = [ 50. -45.  50.], normal = [0.  0.  1.]
MATERIAL:
Cor Difusa = [0.7 0.7 0. ]
Cor Especular = [0. 0. 0.]
Coeficiente Especular = 1
Espelhamento = 0
Opacidade = 1
```

POLIGONO:

```
Plano 0: ponto = [-80. -50. -60.], normal = [-1.  0.  0.]
Plano 1: ponto = [-80. -50. -60.], normal = [ 0. -1.  0.]
Plano 2: ponto = [-80. -50. -60.], normal = [ 0.  0. -1.]
Plano 3: ponto = [ 50.  50. -50.], normal = [1.  0.  0.]
Plano 4: ponto = [ 50.  50. -50.], normal = [0.  1.  0.]
Plano 5: ponto = [ 50.  50. -50.], normal = [0.  0.  1.]
MATERIAL:
Cor Difusa = [0.7 0.7 0. ]
Cor Especular = [0. 0. 0.]
Coeficiente Especular = 1
Espelhamento = 1
```

```
Opacidade = 1
```

```
MATERIAL:
```

```
Cor Difusa = [0. 0. 1.]  
Cor Especular = [1. 1. 1.]  
Coeficiente Especular = 30  
Espelhamento = 0  
Opacidade = 1
```

```
MATERIAL:
```

```
Cor Difusa = [0.7 0.7 0. ]  
Cor Especular = [0. 0. 0.]  
Coeficiente Especular = 1  
Espelhamento = 0  
Opacidade = 1
```

```
MATERIAL:
```

```
Cor Difusa = [0.7 0.7 0. ]  
Cor Especular = [0. 0. 0.]  
Coeficiente Especular = 1  
Espelhamento = 1  
Opacidade = 1
```

```
LUZ:
```

```
Posicao = [ 60. 120. 40.]  
Intensidade = [0.5 0.5 0.5]
```

```
LUZ:
```

```
Posicao = [100. 40. 40.]  
Intensidade = [0.5 0.5 0.5]
```

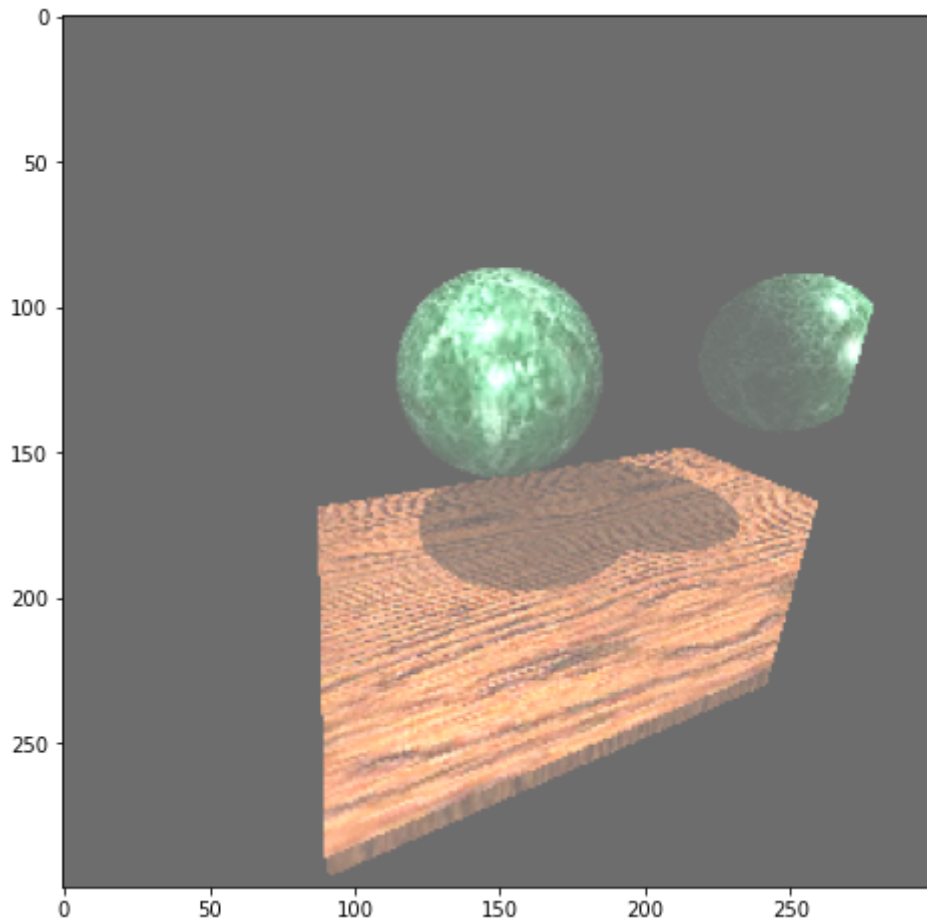
```
Ambiente = [0.1 0.1 0.1]
```

```
In [16]:
```

```
1 cena.render()
```

In [17]:

```
1 camera.imshow()
```



In [ ]:

```
1
```