

Trabajo práctico N°2

Integrantes: Casermeiro Maria Silvia - 2013430, Videla Agustin - 1702629 y Vietto Santiago - 1802890

Docente: John Coppens

Institución: UCC

Año: 2022

Desarrollo

Análisis del código

La ejecución del archivo **demo_tracer.py** permite dibujar una figura en una determinada posición, indicando el color de la misma como así también la luz de la escena, que hace que se ilumine la figura, y un rayo de luz, que hace que se genere una intersección con dicha figura.

```
silvia-ubuntu@silviaubuntu-Aspire-E5-573:~/Documentos/2º Parcial/demo_tracer-Agus-cubo/demo_tracer$ /bin/python3 "/home/silvia-ubuntu/Documentos/2º Parcial/demo_tracer-Agus-cubo/demo_tracer/demo_tracer.py"
['#include', '"colors.inc"', ['camera', ['location', <demo_misc.Vec3 object at 0x7fc83f579e20>], ['look at', <demo_misc.Vec3 object at 0x7fc83f51f2b0>], ['angle', 50.0]], ['box', <demo_misc.Vec3 object at 0x7fc83f51f790>, <demo_misc.Vec3 object at 0x7fc83f51f1c0>], ['pigment', ['color', <demo_misc.RGB object at 0x7fc83f5c46a0>]]], ['box', <demo_misc.Vec3 object at 0x7fc83f58cee0>, <demo_misc.Vec3 object at 0x7fc83f58cd00>], ['pigment', ['color', <demo_misc.RGB object at 0x7fc83f5c45e0>]]], ['sphere', <demo_misc.Vec3 object at 0x7fc83f58c160>, 1.0, ['pigment', ['color', <demo_misc.RGB object at 0x7fc83f5c4a60>]]], ['light_source', <demo_misc.Vec3 object at 0x7fc83f51f9d0>, ['color', <demo_misc.RGB object at 0x7fc83f5c4580>]]]
```

Entonces, en el resultado de la terminal, podemos observar como se separa cada término, y si se devuelven todos los archivos significa que el mismo está correcto. De lo contrario va a mostrar el error que encuentra o existe.

El archivo **demo_parser.py** contiene la definición de la figura que elegimos, el cubo, con sus características. Como aclaración, este mismo archivo contiene la definición de la fuente de luz que agrega iluminación a la escena completa mostrando más o menos brillante.

```
demo_parser.py X demo_things.py 2 demo_misc.py demo_tracer.py
demo_parser.py > ...
132         + pigment
133         + curly_close.suppress()
134     ).set_results_name('sphere')
135
136     box = pp.Group(
137         pp.Literal("box")
138         + curly_open.suppress()
139         + vector.set_results_name('lowleft')
140         + comma.suppress()
141         + vector.set_results_name('upright')
142         + pigment
143         + curly_close.suppress()
144     ).set_results_name('box')
145
146     light_source = pp.Group(
147         pp.Literal("light_source")
148         + curly_open.suppress()
```

Luego, continúa parseando la imagen del cubo:

```
demo_parser.py X demo_things.py 2 demo_misc.py demo_tracer.py
demo_parser.py > Scene > make_parser
180     for el in parsed:
181         if isinstance(el, pp.ParseResults):
182             if el[0] == 'sphere':
183                 things.append(('sphere',
184                               Sphere(el.as_dict()['location'],
185                                     el.as_dict()['radius'],
186                                     el.as_dict()['pigment']['color'][1])))
187             elif el[0] == 'box':
188                 things.append(('box',
189                               Box(el.as_dict()['lowleft'],
190                                   el.as_dict()['upright'],
191                                   el.as_dict()['pigment']['color'][1])))
192             elif el[0] == 'light_source':
193                 lights.append(el)
194             elif el[0] == 'camera':
195                 cameras.append(el)
```

Continuando con el archivo **demo_things.py** define la instancia del objeto cubo indicando la posición de la figura en la pantalla:

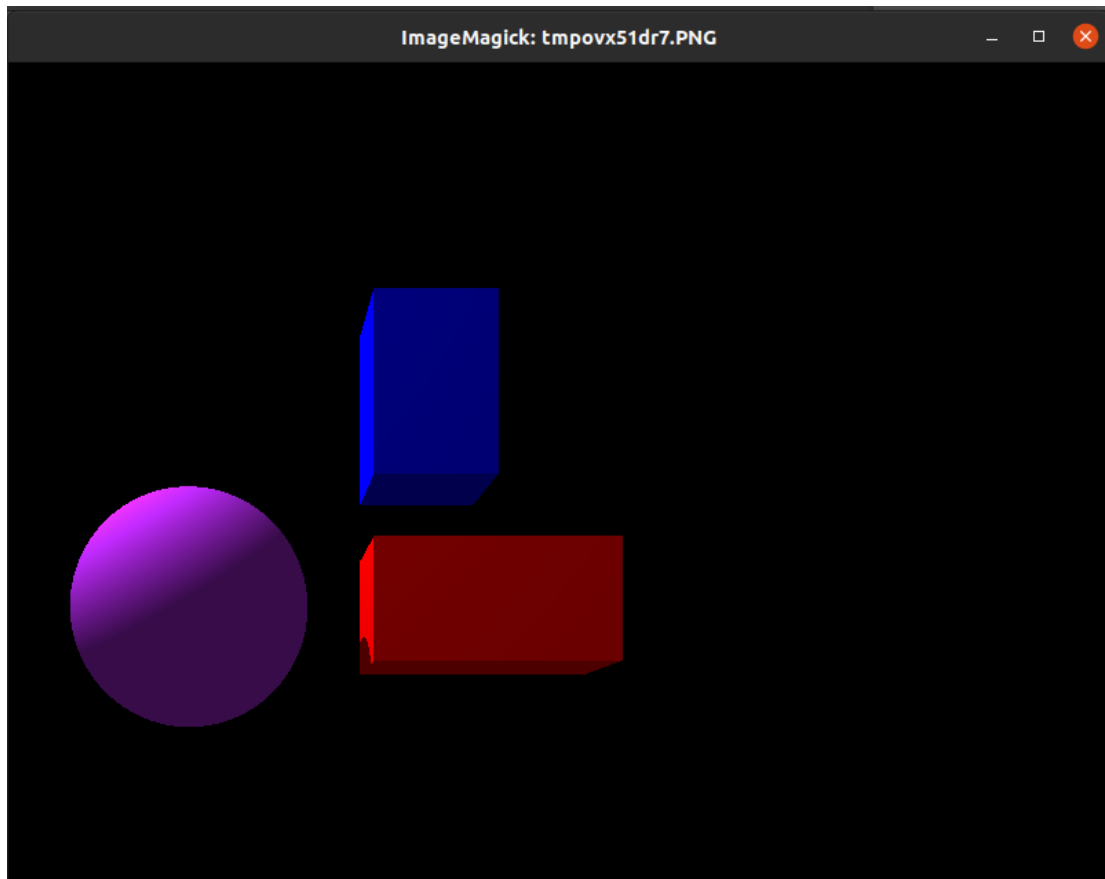
```
demo_parser.py demo_things.py 2 X demo_misc.py demo_tracer.py
demo_things.py > ...
143     return []
144
145     class Box:
146         def __init__(self, lowleft, upright = None, color = RGB.colors['White']):
147             if isinstance(lowleft, dict):
148                 self.lowleft, self.upright = loc['lowleft'], loc['upright']
149                 self.color = lowleft['color']
150             else:
151                 self.lowleft, self.upright, self.color = lowleft, upright, color
152
153
154         def __str__(self):
155             return (f'box: 1st corner {self.lowleft}, '
156                   f'2nd corner: {self.upright}, '
157                   f'color: {self.color}')
158
```

Para determinar si la luz incide sobre la superficie de la figura, hemos seguido las indicaciones del documento *cg_math.pdf* en donde se incluyen las fórmulas que a continuación vamos a explicar:

```
demo_things.py X
demo_things.py > ...
170
171
172 def intersection(self, ray):
173
174     # 6 ecuaciones de los planos
175     planos1 = [ self.lowleft.x, self.lowleft.y, self.lowleft.z ]
176     planos2 = [ self.upright.x, self.upright.y, self.upright.z ]
177     normals = [ Vec3(1,0,0), Vec3(0,1,0), Vec3(0,0,1) ]
178
179     tnear = -np.inf
180     tfar = np.inf
181
182     for p1, p2, rd, ro, n in zip(planos1, planos2, ray.direction, ray.origin, normals):
183         if rd == 0:
184             if ro < p1 or ro > p2:
185                 return []
186             else:
187                 t1 = (p1 - ro) / rd
188                 t2 = (p2 - ro) / rd
189
190                 if t1 > t2:
191                     t1, t2 = t2, t1
192                 if t1 > tnear:
193                     tnear = t1
194                     hit1 = Hit(tnear, n.normalized()*-1, self)
195                 if t2 < tfar:
196                     tfar = t2
197                     hit2 = Hit(tfar, n.normalized(), self)
198                 if tnear > tfar:
199                     return []
200                 if tfar < 0:
201                     return []
202
203     return [ hit1, hit2 ]
204
```

Comenzamos primero en crear los 6 planos necesarios para determinar la posición del cubo y creamos también la normal (las mismas se definen bajo la referencia de “6 ecuaciones de los planos”), para luego calcular los puntos de impacto del rayo. Luego como podemos observar, se definen dos variables (tnear y tfar) que se utilizan para determinar la orientación del plano y la ubicación del cubo y de la cámara, y como se encuentran uno respecto al otro. Aclaremos que las variables rd y ro es en donde termina y comienza un rayo, y las variables t hacen referencia a la distancia. Y de esta forma continuamos desarrollando el algoritmo tomando como guía el mismo que aparece en el documento *cg_math.pdf* (página 13 - 14).

Para poder corroborar el trabajo, procedemos a ejecutar el archivo **demo_tracer.py**, ya que este es desde donde se ejecuta el test de la imagen que definimos. Como resultado, obtuvimos lo siguiente:



Es importante aclarar que en el archivo **test_mini.pov** se definieron la posición y ángulo de la cámara para poder ilustrar dos figuras de tipo cubo (box), en donde para cada figura se definieron sus colores y posiciones respectivamente:

```
demo_parser.py demo_things.py 2 test_mini.pov x demo_misc.py
test_mini.pov
1  #include "colors.inc"
2  camera {
3      location <-1.7, -1.7, -6>
4      look_at <0, 0, 0>
5      angle 50
6  }
7
8  box {
9      <-0.5,1.0,3.5>, <0.5,2.5,4.5>
10     pigment { color Blue }
11 }
12
13 box {
14     <-0.5,-0.5,3.5>, <1.5,0.5,4.5>
15     pigment { color Red }
16 }
17
18 sphere {
19     <-2, 0, 4>, 1
20     pigment { color Med_Purple }
21 }
22
23
24 light_source {
25     <-5, 5, 2.5>, color White
26 }
```