Trabajo práctico Nº1

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

Docente: John Coppens

Institución: UCC

Año: 2022

Código

Archivo povparser.py

```
from pyparsing import (Keyword, OneOrMore, Optional, oneOf, Word,
   nums, alphas, alphanums, printables, ZeroOrMore, Combine)

POVFILE = "parser_box.pov"

def parser(which = "scene"):
   integer = Word(nums)
   sign = oneOf("+ -")
   bool = oneOf("on off")

   uns_float = (integer + Optional("." + integer) +
               Optional(oneOf("e E") + Optional(sign) + integer))
   signed_float = Combine(Optional(sign) + uns_float)
   signed_float = signed_float.set_parse_action(lambda toks:
float(toks[0]))
   vector = ("<" + signed_float + ","
                 + signed_float + ","
                 + signed_float + ">")
   vector2d = ("<" + signed_float + ","
                   + signed_float + ">")

   color = Optional("color") + (oneOf("Yellow White Blue Red") |
(Keyword("rgb") + vector))

   # Patterns
   color_pattern = oneOf("checker brick hexagon") + OneOrMore(color +
Optional(","))

   # Object Modifiers
   om_pigment = Keyword("pigment") + "{" + (color_pattern | color) +
"}"
   om_texture = Keyword("texture") + "{" + om_pigment + "}"
   om_hollow = Keyword("hollow") + bool
   om_noshadow = Keyword("no_shadow")
   om_rotate = Keyword("rotate") + vector

   object_modifier = om_pigment | om_texture | om_hollow | om_noshadow
| om_rotate
```

```
    # Camera items
    camera_loc = Keyword("location") + vector
    camera_lookat = Keyword("look_at") + vector
    camera_angle = Keyword("angle") + signed_float

    camera_items = (camera_loc + camera_lookat + Optional(camera_angle))

    # Things
    box = (Keyword("box") + "{" + vector + "," + vector +
        ZeroOrMore(object_modifier) + "}")
    sphere = (Keyword("sphere") + "{" + vector + "," + signed_float +
        ZeroOrMore(object_modifier) + "}")
    triangle = (Keyword("triangle") + "{" + vector + "," + vector + ","
+ vector +
        ZeroOrMore(object_modifier) + "}")
    cone = (Keyword("cone") + "{" + vector + "," + signed_float +
        vector + "," + signed_float + Optional("open") +
        ZeroOrMore(object_modifier) + "}")
    cylinder = (Keyword("cylinder") + "{" + vector + "," + vector +
        "," + signed_float + Optional("open") +
        ZeroOrMore(object_modifier) + "}")
    torus = (Keyword("torus") + "{" + signed_float + "," + signed_float
+
        ZeroOrMore(object_modifier) + "}")
    polygon = (Keyword("polygon") + "{" + signed_float + "," +
            OneOrMore(vector2d + Optional(",")) +
ZeroOrMore(object_modifier) + "}")

    thing = sphere | box | triangle | cone | cylinder | polygon | torus

    camera = Keyword("camera") + "{" + camera_items + "}"
    light = Keyword("light_source") + "{" + vector + "," + color + "}"
    library = Keyword("#include") + '"' + Word(printables,
exclude_chars='"') + '"'

    scene = ZeroOrMore(library) + camera + ZeroOrMore(thing | light)
    return eval(which)

def main(args):
    with open(POVFILE, 'r') as povfile:
        pov = povfile.read()

    pp = parser()
```

```
    print(pp.parse_string(pov))
    return 0

if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
```

Archivo parser_box.pov

```
#include "colors.inc"

camera {
        location <0, 0, -5>
        look_at <0, 0, 0>
        angle 30
}

sphere {
        <0, 1, 0>, 0.5
        pigment {
        color rgb <0.5, 0.5, 0.5>
        }
}


box {
        <0, 0, 0>, <1, 1, 1>
        texture {
        pigment {
        color Yellow
        }
        }
}

triangle {
        <0, 1, -1>, <0, 1, 2>, <1, 0, 1>
        texture {
        pigment {
        color Red
        }
        }
}
```

```
cylinder {
        <0, -2, 0>,
        <0, 2, 0>,
        2
        open
        texture {
        pigment {
        color Yellow
        }
        }
        rotate <-30, 0, 0>
}

polygon {
        30,
        <-0.8, 0.0>, <-0.8, 1.0>,
        <-0.3, 1.0>, <-0.3, 0.5>,
        <-0.7, 0.5>, <-0.7, 0.0>,
        <-0.8, 0.0>,
        <-0.7, 0.6>, <-0.7, 0.9>,
        <-0.4, 0.9>, <-0.4, 0.6>,
        <-0.7, 0.6>
        <-0.25, 0.0>, <-0.25, 1.0>,
        < 0.25, 1.0>, < 0.25, 0.0>,
        <-0.25, 0.0>,
        <-0.15, 0.1>, <-0.15, 0.9>,
        < 0.15, 0.9>, < 0.15, 0.1>,
        <-0.15, 0.1>,
        <0.45, 0.0>, <0.30, 1.0>,
        <0.40, 1.0>, <0.55, 0.1>,
        <0.70, 1.0>, <0.80, 1.0>,
        <0.65, 0.0>,
        <0.45, 0.0>
        pigment { color rgb <1, 0, 0> }
}

cone {
        <0, -2, 0>, 3
        <0, 2, 0>, 1
        texture {
        pigment {
        color Blue
        }
        }
}
```

```
torus {
        4, 1
        texture {
        pigment {
        color Red
        }
        }
        rotate <-30, 0, 0>
}

light_source {
        <2, 3, -3>, White
}
```

Desarrollo

Analisis del codigo

La función básicamente del programa en python es tomar un archivo povray, que en nuestro caso es “parser_box.pov” y se va a fijar que la sintaxis de este archivo esté correcta, es decir, que la sintaxis respecto a la librería, la cámara y los objetos esté bien. A esta verificación la va a devolver por la terminal. En nuestro caso al ejecutar el archivo “povparser.py” obtenemos el siguiente resultado:

```
vietto@vietto:~/Documentos/UCC-Santi/CompuGrafica/cmp-graf-master$ ./povparser.py
['#include', '"', 'colors.inc', '"', 'camera', '{', 'location', '<', 0.0, ',', 0.0, ',', -5.0, '>', 'look at', '<', 0.0, ',', 0.0, ','
, 0.0, '>', 'angle', 30.0, '}', 'sphere', '{', '<', 0.0, ',', 1.0, ',', 0.0, '>', ',', 0.5, 'pigment', '{', 'color', 'rgb', '<', 0.5,
',', 0.5, ',', 0.5, '>', '}', '}', 'box', '{', '<', 0.0, ',', 0.0, ',', 0.0, '>', ',', '<', 1.0, ',', 1.0, ',', 1.0, '>', 'texture', '
{', 'pigment', '{', 'color', 'Yellow', '}', '}', '}', 'triangle', '{', '<', 0.0, ',', 1.0, ',', -1.0, '>', ',', '<', 0.0, ',', 1.0, ',
', 2.0, '>', ',', '<', 1.0, ',', 0.0, ',', 1.0, '>', 'texture', '{', 'pigment', '{', 'color', 'Red', '}', '}', '}', 'cylinder', '{', '
<', 0.0, ',', -2.0, ',', 0.0, '>', ',', '<', 0.0, ',', 2.0, ',', 0.0, '>', ',', 2.0, 'open', 'texture', '{', 'pigment', '{', 'color',
'Yellow', '}', '}', 'rotate', '<', -30.0, ',', 0.0, ',', 0.0, '>', '}', 'polygon', '{', 30.0, ',', '<', -0.8, ',', 0.0, '>', ',', '<',
 -0.8, ',', 1.0, '>', ',', '<', -0.3, ',', 1.0, '>', ',', '<', -0.3, ',', 0.5, '>', ',', '<', -0.7, ',', 0.5, '>', ',', '<', -0.7, ',
', 0.0, '>', ',', '<', -0.8, ',', 0.0, '>', ',', '<', -0.7, ',', 0.6, '>', ',', '<', -0.7, ',', 0.9, '>', ',', '<', -0.4, ',', 0.9, '>'
, ',', '<', -0.4, ',', 0.6, '>', ',', '<', -0.7, ',', 0.6, '>', '<', -0.25, ',', 0.0, '>', ',', '<', -0.25, ',', 1.0, '>', ',', '<', 0
.25, ',', 1.0, '>', ',', '<', 0.25, ',', 0.0, '>', ',', '<', -0.25, ',', 0.0, '>', ',', '<', -0.15, ',', 0.1, '>', ',', '<', -0.15, ',
', 0.9, '>', ',', '<', 0.15, ',', 0.9, '>', ',', '<', 0.15, ',', 0.1, '>', ',', '<', -0.15, ',', 0.1, '>', ',', '<', 0.45, ',', 0.0, '
>', ',', '<', 0.3, ',', 1.0, '>', ',', '<', 0.4, ',', 1.0, '>', ',', '<', 0.55, ',', 0.1, '>', ',', '<', 0.7, ',', 1.0, '>', ',', '<',
 0.8, ',', 1.0, '>', ',', '<', 0.65, ',', 0.0, '>', ',', '<', 0.45, ',', 0.0, '>', 'pigment', '{', 'color', 'rgb', '<', 1.0, ',', 0.0,
 ',', 0.0, '>', '}', '}', 'cone', '{', '<', 0.0, ',', -2.0, ',', 0.0, '>', ',', 3.0, '<', 0.0, ',', 2.0, ',', 0.0, '>', ',', 1.0, 'tex
ture', '{', 'pigment', '{', 'color', 'Blue', '}', '}', '}', 'torus', '{', 4.0, ',', 1.0, 'texture', '{', 'pigment', '{', 'color', 'Red
', '}', '}', 'rotate', '<', -30.0, ',', 0.0, ',', 0.0, '>', '}', 'light_source', '{', '<', 2.0, ',', 3.0, ',', -3.0, '>', ',', 'White'
, '}']
```

Entonces, como podemos observar el archivo separa cada término y si devuelve todos los archivos significa que el mismo está correcto. De lo contrario va a mostrar el error que encuentra.

Entrando más en detalle en el código, para que el mismo verifique que la sintaxis del archivo povray es correcta, lo que hacemos es declarar diferentes cosas mediantes expresiones regulares o patrones, en donde a cada objeto le vamos a asignar un patrón que esperamos que cumpla. Entonces, analizamos por ejemplo la cámara como vemos en el codigo, donde esta va a estar definida por la palabra clave

“camara”, seguido de una llave abierta, luego camara_items (que van a ser definidos más adelante) y luego se cierra la llave. A continuación vemos el código de ambos archivos:

```
camera = Keyword("camera") + "{" + camera_items + "}"
```

```
camera {
        location <0, 0, -5>
        look_at <0, 0, 0>
        angle 30
}
```

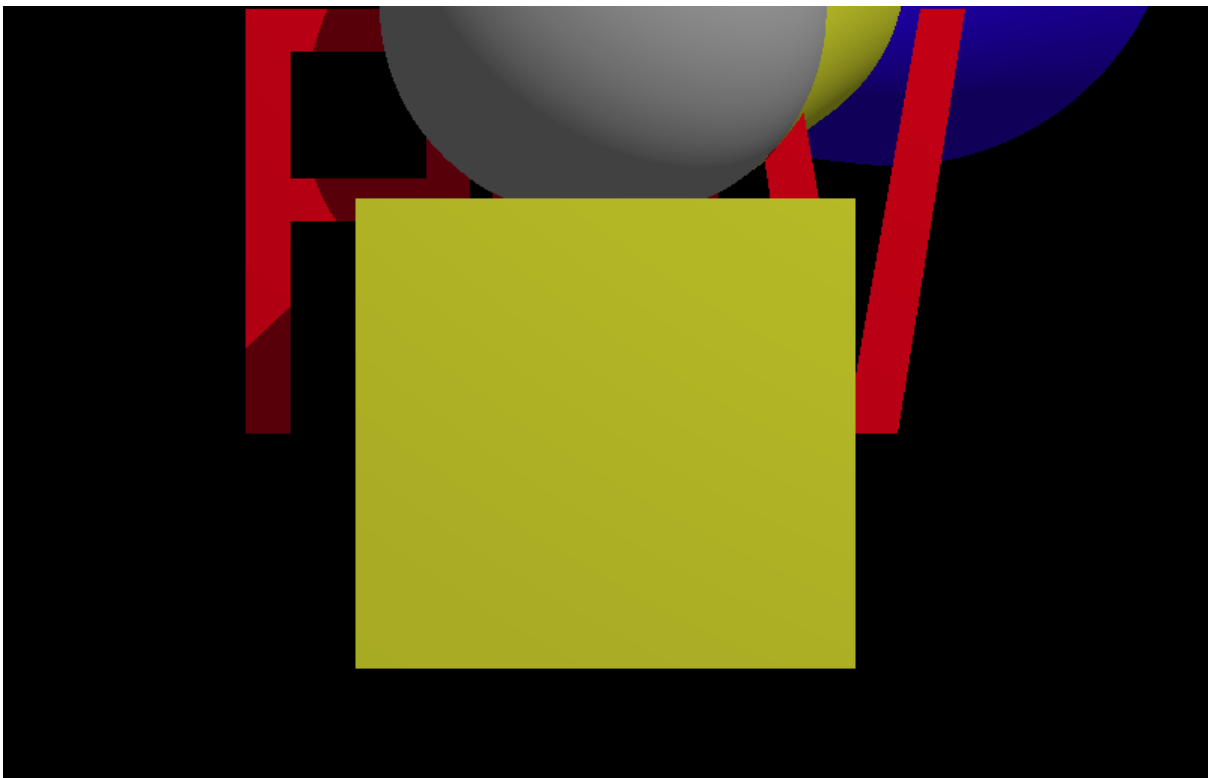
Analizando camara_items vemos que este mismo está compuesto por camera_loc, camera_lookat y opcionalmente un camera_angle:

```
camera_items = (camera_loc + camera_lookat +
       Optional(camera_angle))
```

A su vez, se definen cada uno de estos elementos:

```
camera_loc = Keyword("location") + vector
   camera_lookat = Keyword("look_at") + vector
   camera_angle = Keyword("angle") + signed_float
```

Povray agarra los objetos que se definieron en el archivo y a estos se les puede dar a los objetos un color, una ubicación o una textura. Y como todos los objetos tienen una ubicación también la va a tener la cámara, entonces dependiendo del ángulo de la cámara es como se va a ver el objeto. Entonces al ejecutar el archivo “parser_box.pov” podemos ver como nos muestra el objeto a un cierto ángulo de la cámara, en este caso vemos como ejemplo un box:



Tenemos el caso también de una librería, donde va a empezar con la palabra clave “include”, luego se abre una comilla, seguido tenemos un Word printables que hace referencia a que va a ser una palabra de caracteres imprimibles o ASCII y luego se cierran las comillas. Entonces si hay algo que no comparta con la estructura de la sintaxis, el mensaje de error que salta a la hora de ejecutar el código, va a marcar cual es el error justamente. A continuación vemos el código de ambos archivos:

```
library = Keyword("#include") + '"' + Word(printables,
exclude_chars='"') + '"'
```

#include "colors.inc"

Por otro lado vamos a tener las figuras, que también se van a declarar de la misma manera. Las figuras que vemos son box, sphere, triangle, cone, cylinder, torus y polygon. Analizamos por ejemplo cone, y como vemos empieza con la palabra clave “cone”, va a abrir un corchete, dentro va a tener un vector, seguido de una coma, luego un decimal (con signo), le sigue una coma y otro vector, luego otra coma seguido de un decimal (con signo), tenemos un opcional (“open”) que permite dibujar el cono de otra forma y por último va a tener uno o más object_modifier que lo vamos a definir más adelante:

```
cone = (Keyword("cone") + "{" + vector + "," + signed_float + vector +
"," + signed_float + Optional("open") + ZeroOrMore(object_modifier) +
"}")
```

```
cone {
        <0, 1, 0>, 0.3
        <1, 2, 3>, 1.0
        texture {
        pigment {
        color Blue
        }
        }
}
```

Analizando object_modifier, tenemos que está definido por un pigmento, una textura, un hollow y noshadow:

```
 object_modifier = om_pigment | om_texture | om_hollow | om_noshadow
```

Viendo en detalle cada uno tenemos por ejemplo que la textura va a tener una palabra clave “texture”, va a abrir la llave, va a contener un pigmento en específico que es el que definimos antes, y luego cierra la llave:

```
om_texture = Keyword("texture") + "{" + om_pigment + "}"
```

Luego tenemos el pigmento va a tener la palabra clave “pigment”, a continuación se va a abrir un corchete y va a contener un color_pattern o un color, y por último cierra la llave:

```
om_pigment = Keyword("pigment") + "{" + (color_pattern | color) + "}"
```

Tanto color_pattern como color, tienen su propia declaración y estructura como vemos a continuación, en donde color opcionalmente empieza con “color” y va a tener o amarillo, blanco, zul o rojo, o la palabra clave “rgb” lo cual declara un color en forma de vector:

```
color = Optional("color") + (oneOf("Yellow White Blue Red") |
(Keyword("rgb") + vector))

color_pattern = oneOf("checker brick hexagon") + OneOrMore(color +
Optional(","))
```

Luego tenemos los dos componentes restantes:

```
om_hollow = Keyword("hollow") + bool
om_noshadow = Keyword("no_shadow")
```

Podemos observar entonces que la sintaxis en la declaración del cone está correcta en función a la estructura que venimos definiendo. Lo mismo hacemos con las demás figuras:

```
box = (Keyword("box") + "{" + vector + "," + vector +
ZeroOrMore(object_modifier) + "}")

sphere = (Keyword("sphere") + "{" + vector + "," + signed_float +
       ZeroOrMore(object_modifier) + "}")

triangle = (Keyword("triangle") + "{" + vector + "," + vector + "," +
vector + ZeroOrMore(object_modifier) + "}")

cone = (Keyword("cone") + "{" + vector + "," + signed_float + vector +
"," + signed_float + Optional("open") + ZeroOrMore(object_modifier) +
"}")

cylinder = (Keyword("cylinder") + "{" + vector + "," + vector + "," +
signed_float + Optional("open") + ZeroOrMore(object_modifier) + "}")

torus = (Keyword("torus") + "{" + signed_float + "," + signed_float +
```

```
ZeroOrMore(object_modifier) + "}")

polygon = (Keyword("polygon") + "{" + signed_float + "," +
OneOrMore(vector2d + Optional(",")) + ZeroOrMore(object_modifier) +
"}")
```

Por último tenemos la función “parser” del principio del código que toma como parámetro un which, que por defecto si no le ponemos nada, el which va a ser una “scene” (escena) que es un patrón, donde el patrón de escena empieza si o si con una o mas librerias, luego va a seguir si o si con una cámara y va a tener después una o más cosas o luces, y como en este caso tenemos varias figuras, en thing declaramos cada una:

```
parser(which = "scene"):
```

```
scene = ZeroOrMore(library) + camera + ZeroOrMore(thing | light)
return eval(which)
```

```
thing = sphere | box | triangle | cone | cylinder | polygon | torus
```

Entonces se evalúa que esto sea una escena con eval(which), es decir, con su sintaxis, y si es correcto va a imprimir todo. Para imprimir una figura en particular tenemos que modificar el código para que tome solo una figura y no una escena completa como es el caso de nuestro código.

A continuación vemos cómo se representan cada una de las figuras:

<u>Sphere:</u>

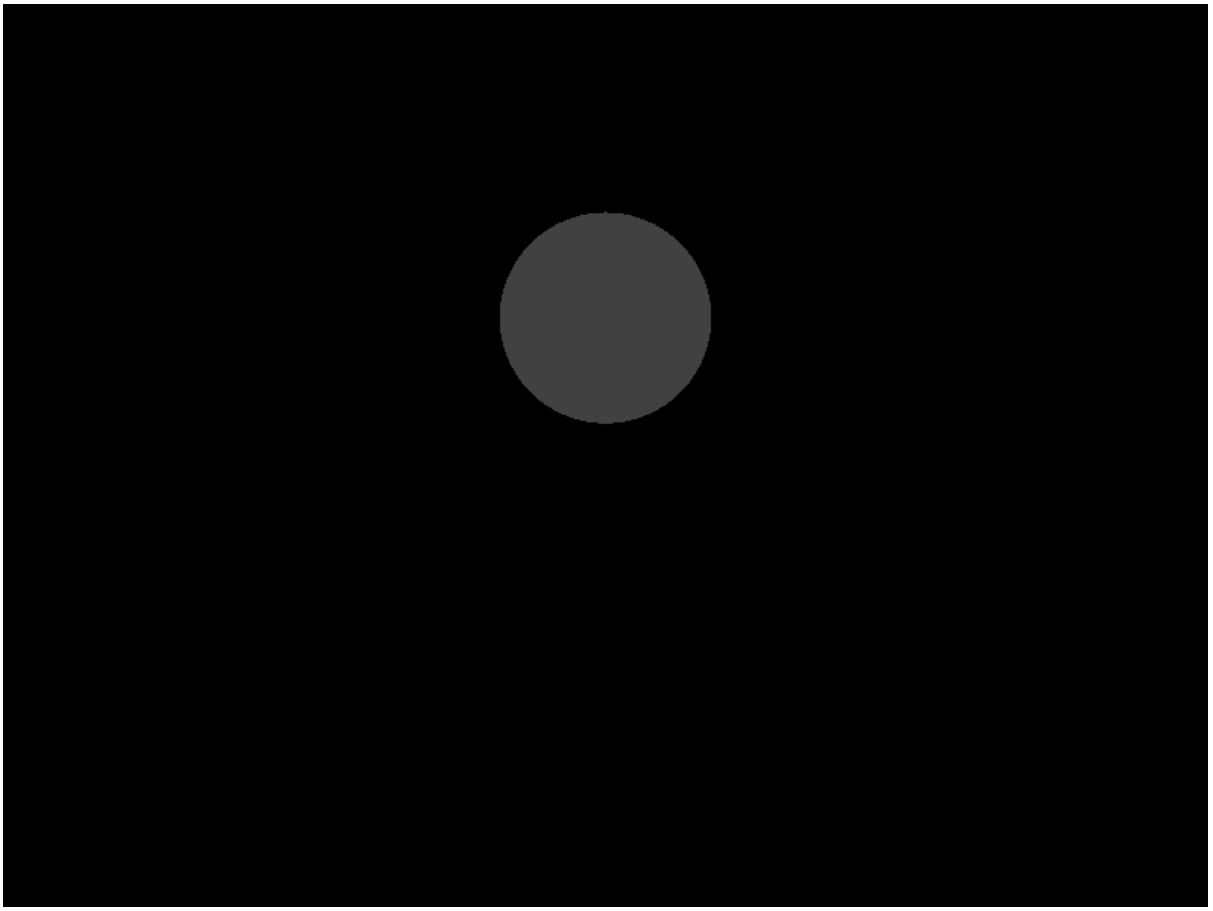
```
pp = parser(“sphere”)
```

```
#include "colors.inc"

camera {
        location <25, 50, 35>
        look_at <0, 0, 0>
        angle 5
}

sphere {
        <0, 1, 0>, 0.5
        pigment {
        color rgb <0.5, 0.5, 0.5>
        }
```

}



Box:

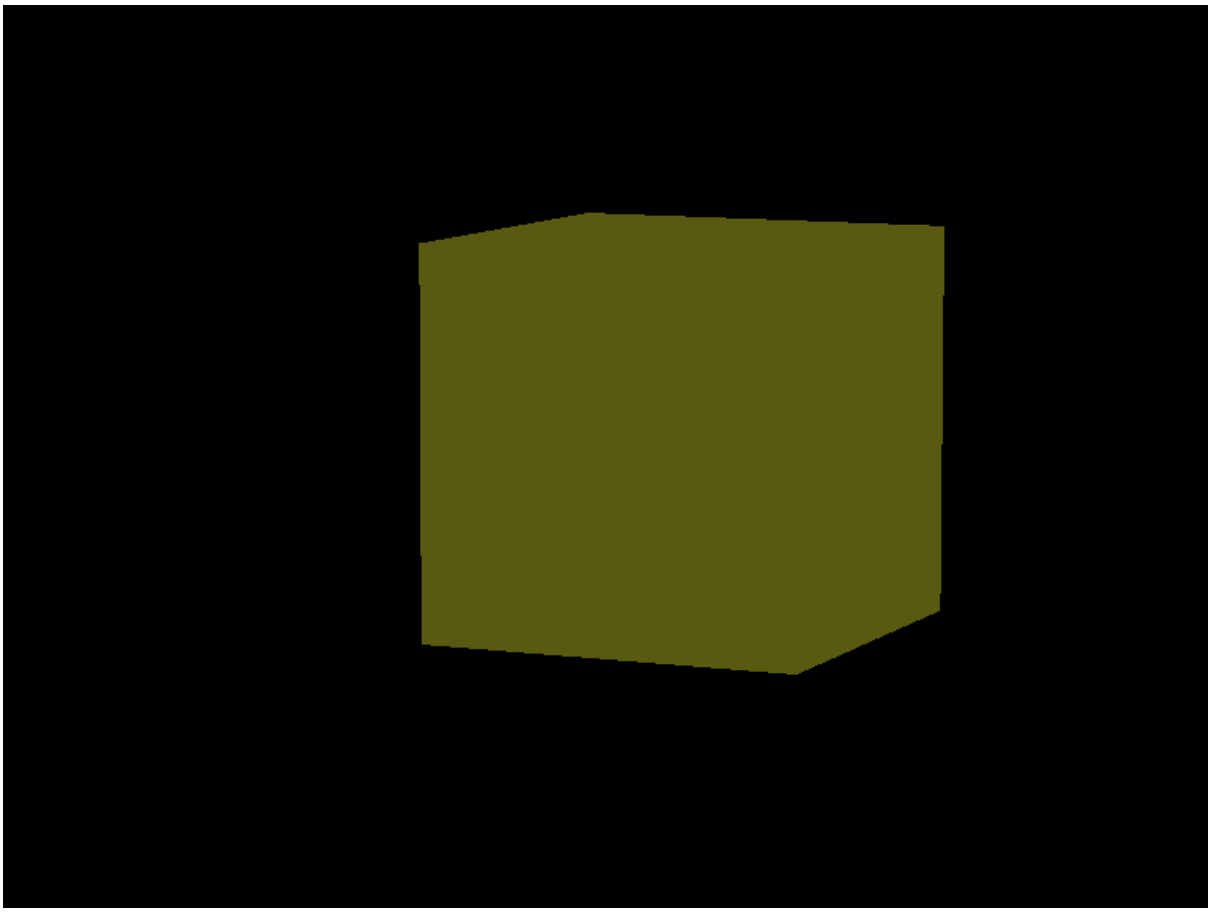
```
pp = parser("box")
```

```
#include "colors.inc"

camera {
        location <5, 2, -10>
        look_at <0.5, 0.5, 0>
        angle 15
}

box {
        <0, 0, 0>, <1, 1, 1>
        texture {
        pigment {
        color Yellow
        }
        }
```

}



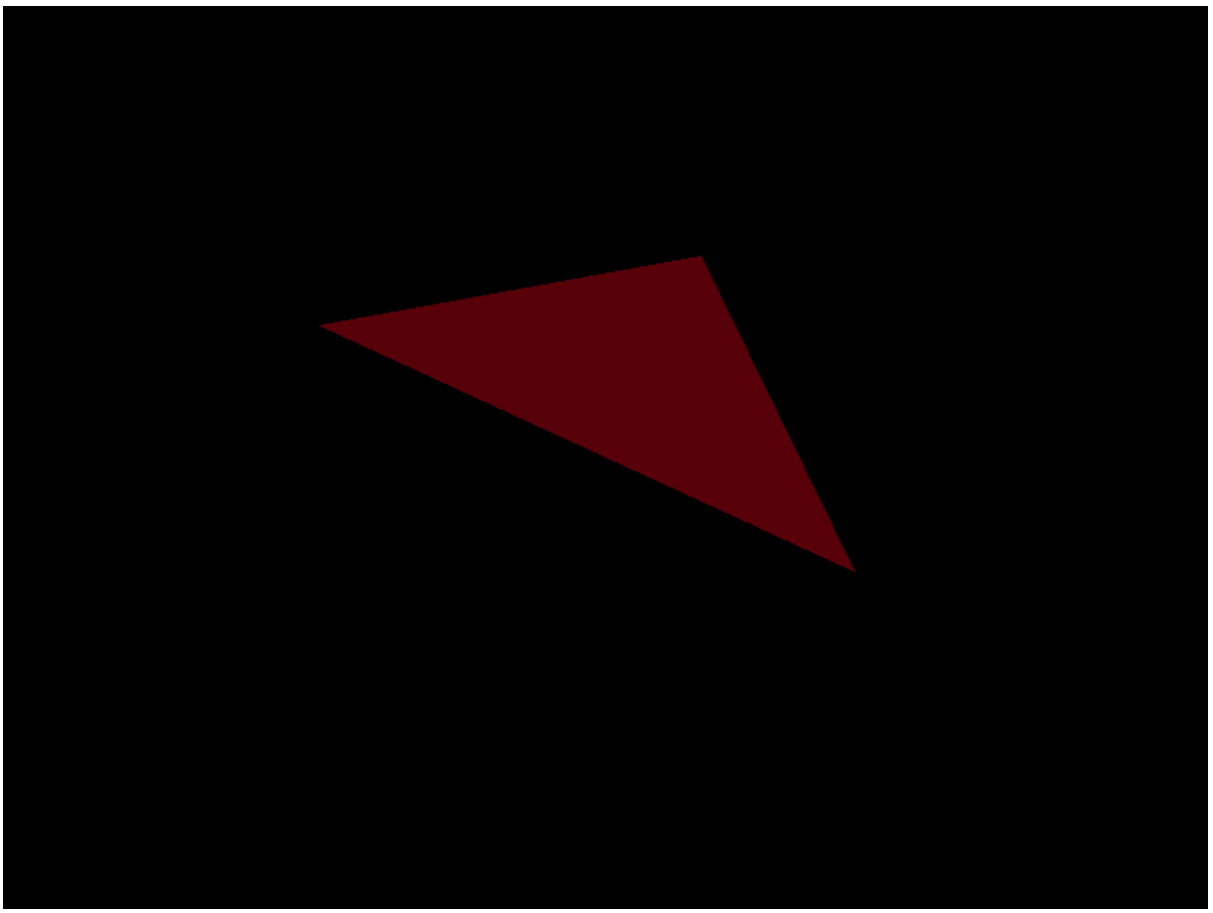
Triangle:

```
pp = parser("triangle")
```

```
#include "colors.inc"

camera {
        location <5, 2, -10>
        look_at <0.5, 0.5, 0>
        angle 20
}

triangle {
        <0, 1, -1>, <0, 1, 2>, <1, 0, 1>
        texture {
        pigment {
        color Red
        }
        }
}
```



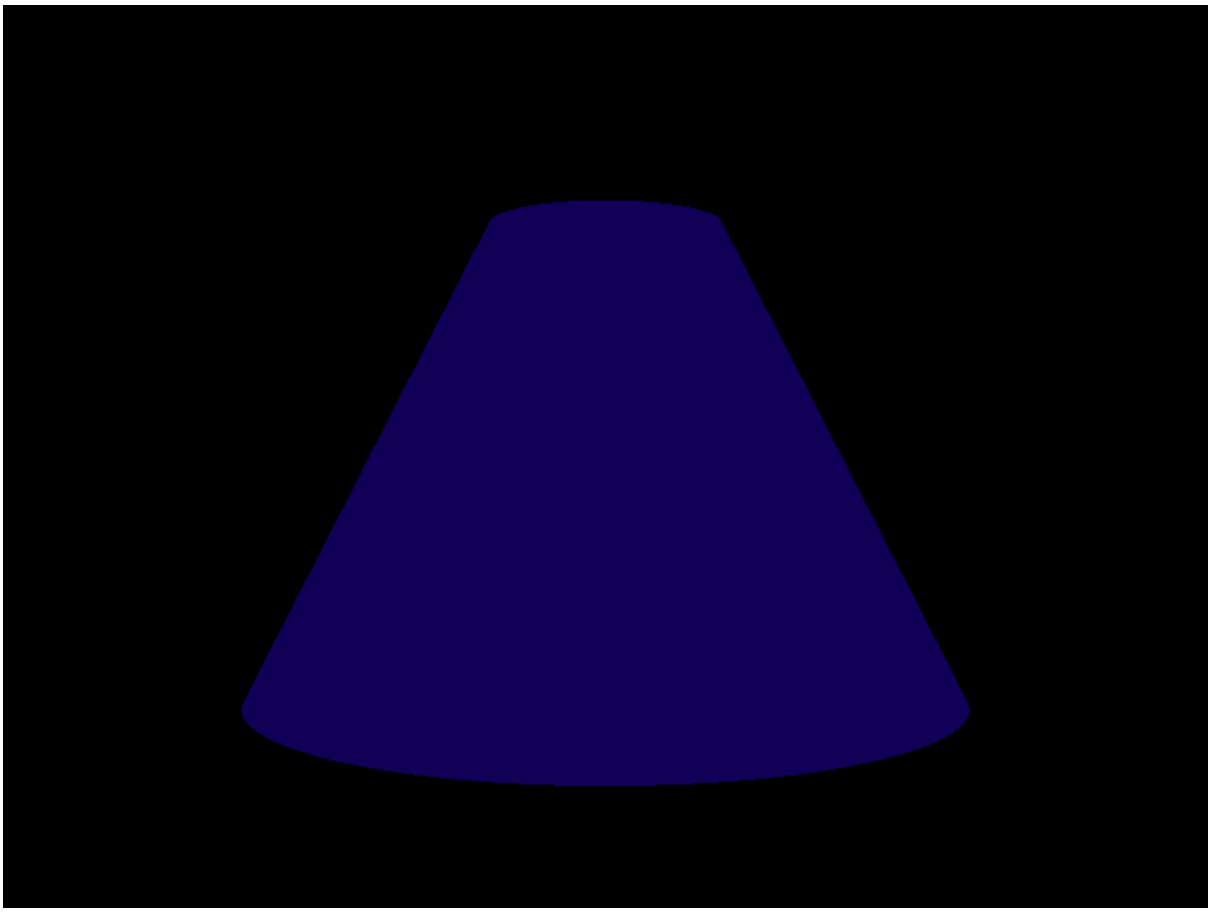
Cone:

```
pp = parser("cone")
```

```
#include "colors.inc"

camera {
        location <0, 0, -10>
        look_at <0, 0, 0>
        angle 55
}

cone {
        <0, -2, 0>, 3
        <0, 2, 0>, 1
        texture {
        pigment {
        color Blue
        }
        }
}
```



Cylinder:

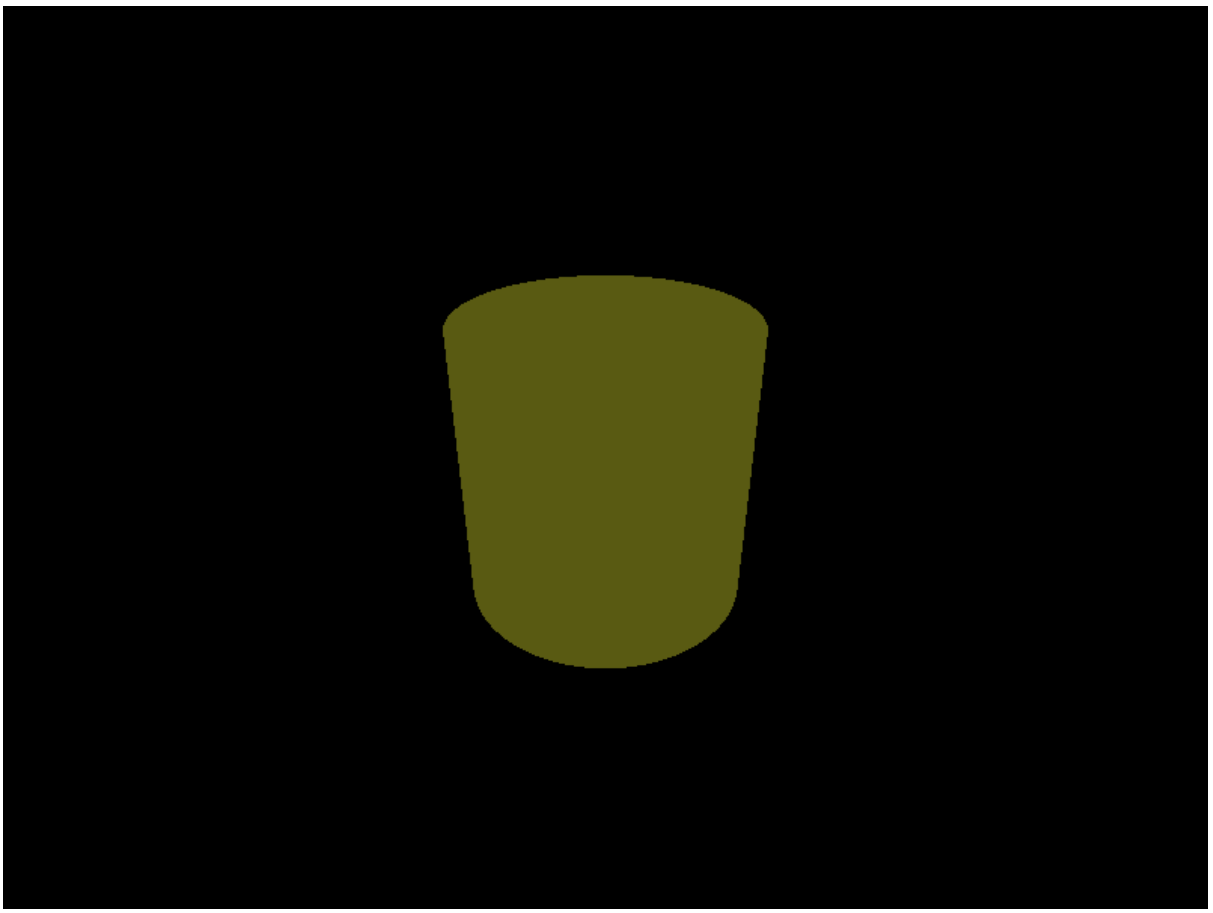
```
pp = parser(“cylinder”)
```

```
#include "colors.inc"

camera {
        location <0, 0, 55>
        look_at <0, 0, 0>
        angle 10
}

cylinder {
        <0, -2, 0>,
        <0, 2, 0>,
        2
        open
        texture {
        pigment {
        color Yellow
        }
```

```
        rotate <-30, 0, 0>
        }
}
```



Polygon:

```
pp = parser("polygon")
```

```
#include "colors.inc"

camera {
        location <0, 0, -10>
        look_at <0, 0.5, 0>
        angle 20
}

polygon {
        30,
        <-0.8, 0.0>, <-0.8, 1.0>,
        <-0.3, 1.0>, <-0.3, 0.5>,
        <-0.7, 0.5>, <-0.7, 0.0>,
        <-0.8, 0.0>,
```

```
        <-0.7, 0.6>, <-0.7, 0.9>,
        <-0.4, 0.9>, <-0.4, 0.6>,
        <-0.7, 0.6>
        <-0.25, 0.0>, <-0.25, 1.0>,
        < 0.25, 1.0>, < 0.25, 0.0>,
        <-0.25, 0.0>,
        <-0.15, 0.1>, <-0.15, 0.9>,
        < 0.15, 0.9>, < 0.15, 0.1>,
        <-0.15, 0.1>,
        <0.45, 0.0>, <0.30, 1.0>,
        <0.40, 1.0>, <0.55, 0.1>,
        <0.70, 1.0>, <0.80, 1.0>,
        <0.65, 0.0>,
        <0.45, 0.0>
        pigment { color rgb <1, 0, 0> }
}
```

Torus:

```
pp = parser(“torus”)
```

```
#include "colors.inc"

camera {
        location <0, 0, -10>
        look_at <0, 0, 0>
        angle 80
}

torus {
        4, 1
        texture {
        pigment {
        color Red
        }
        }
        rotate <-30, 0, 0>
}
```