Notes for ray trancing

Yue-Wen Fang[1,*]

[1]Key Laboratory of Polar Materials and Devices, Ministry of Education, Department of Electronic Engineering, East China Normal University, Shanghai, 200241, China

[*]fyuewen@gmail.com or fyuewen@protonmail.ch

ABSTRACT

Ray-tracing is a significant method for design. Here, the author presents his notes for leaning the software POV-Ray v3.7 based on his own knowledge and understanding.

Getting started

This part will guide you how to generate your first image by POV-Ray.

1 The first scene

1.1 The coordinate system

The POV-Raysticks with a so called ”left-handed system”, in which we point our **thumb** in the direction of $\boldsymbol{x}$, the **index** finger in the direction of $\boldsymbol{y}$ and the **middle** finger in the positive z direction.

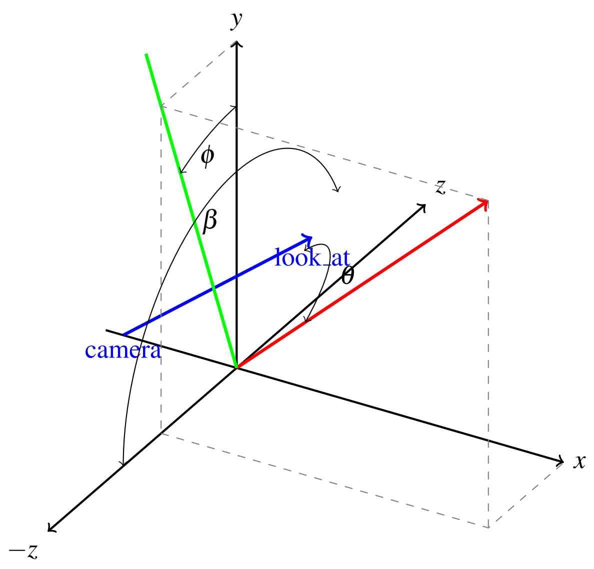


Figure 1. A left-handed coordinates system in POV-Ray.

This coordinate is plotted by a package called **tikz** which supports Latex. Note that in **tikz** package, a right-handed coordinate system is used, so when you are plotting, you have to take care of the conversion between the left-handed and right-handed coordinates.

1.2 Adding standard include files

Create a file called demo.pov. The POV-Ray-v3.7 in windows has a built-in text editor which is easy to use. However, in **Linux**, I prefer using **Vim**.

After creating the demo.pov, type the following text,

```
#include ”colors.inc”   // The include files contain
#include ”stones.inc”   // pre-defined scene elements
```

where '//' is the comment character. Line-1 reads in defections for various useful colors and Line-2 reads in a collection of stone textures (Definition in English: the surface of a material, the structure of a woven fabric). POV-Rayprovides many interesting include files as the following lines show:

```
#include ”texture.inc”   // The include files contain
#include ”shapes.inc”   // pre-defined scene elements
#include ”glass.inc”   // pre-defined scene elements
#include ”metals.inc”   // pre-defined scene elements
#include ”woods.inc”   // pre-defined scene elements
```

it should be noted that **do not include overmany include files because some of them come with POV-Ray are quite large. For saving parsing time and memory, we'd better include what we really need.** In the following examples we just include ”colors.inc” and ”stones.inc”. Sometimes we may need as many include files as possible, some include files may themselves contain include files, but we are limited to declaring includes (nested only ten levels deep). The include files (with suffix .Inc) should be in the current path or the library paths (in Linux, we can set them in .bashrc). Usually in windows, the include files are automatically installed in the ***include*** directory of the installation directory of POV-Ray.

1.3 Adding a camera

The **camera** statement describes where and how the camera sees the scene. It uses x-,y- and z-coordinates to indicate the position of the camera and what part of the scene it is pointing at. The coordinates in the 2D space is denoted by a three-part vector written as $\langle$x, y, z$\rangle$. Here, commas are used to separate the values of x, y and z.

Here is a simple example.

```
camera{
location ⟨0, 2, -3⟩
look_at ⟨0, 1, 2⟩
}
```

Briefly, location $\langle 0, 2, -3 \rangle$ places the camera up 2 units and back 3 units from the center of the ray-tracing universe which is at $\langle 0, 0, 0 \rangle$. look_at $\langle 0, 1, 2 \rangle$ **rotates** the camera to point at the coordinates $\langle 0, 1, 2 \rangle$. This make it 5 units in front of and 1 unit lower than the camera which can be seen from Fig. 1 **The look_at point should be the center of attention of our image**

1.4 Describeing an object

Now that the camera is set up, let's place a sphere into the cene by input the following lines:

```
sphere {
 ⟨0, 1, 2⟩, 2      //center of sphere, raidus
 texture {
  pigment {color Yellow}
 }
}
```

The vector specifies the center of sphere. In this example the x coordinate is zero so it is **centered left and right**. It is 1 unit up from the origin and z = 2 shows the center of the sphere is 5 units in front of the camera (z = -3). After the center vector is a comma followed by the **radius**.

1.5 Adding texture to an object

Now we need define the appearance using **texture** which specifies color, bumpiness (Definition in English: uneven, rough) and finish properties (Definition in English: to put a particular surface texture on (wood, cloth, etc)) of an object. In the following example we will speccify the color only. The parameter **pigment** in POV-Ray determines how the color of the object appears.

```
color red 1.0 green 0.8 blue 0.8
```

This example gives a nice shade of pink. Actually, a shortcut notation can be used to remove the superfluous (Definition in English: exceeding what is sufficient or required), as shown in the following. `color rgb ⟨1.0, 0.8, 0.8⟩`

or

```
color ⟨1.0, 0.8, 0.8⟩
```

1.6 Defining a light source

We have created an object with specific texture, so we need a light to illuminate it by adding the line

```
light_source ⟨2, 4, -3⟩ color White
```

to the scene file to get our first complete POV-Rayscene file as shown below.

```
#include ”colors.inc”   // The include files contain
background {color Cyan}   // defining color of background
camera{
location ⟨0, 2, -3⟩
look_at ⟨0, 1, 2⟩
}
sphere {
 ⟨0, 1, 2⟩, 2     //center of sphere, raidus
 texture {
  pigment {color Yellow}
 }
}
light_source {⟨2, 20, -3⟩ color White}
```

The produced picture is shown in Fig. 2. If your hardware such as monitor or GPU is not good, you may cannot get a high color or true color displaying in your monitor, but details of your operations all all written to the image file regardless of the type of display.

Though the scene we just traced is not quite ”state of the art” but we will have to start with basics before we soon get to much more fascinating features and scenes.

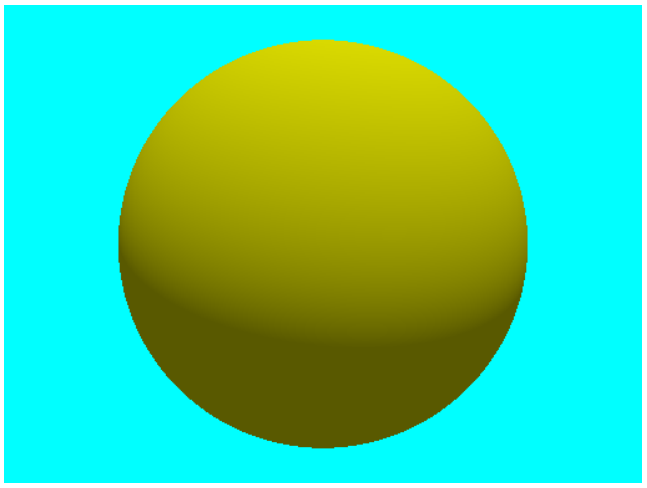


Figure 2. The first basic scene generated by POV-Ray.

2 Basic shapes

So far we have just used the sphere shape. The following sections will describe how to use other simple objects as a replacement for the sphere used above.

2.1 Box object

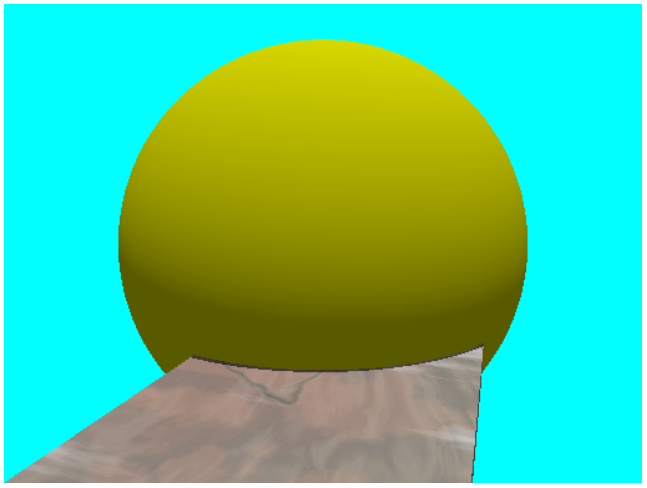


Figure 3. A sphere and a box generated by POV-Ray.

The **box** is one of the common objects used. We try this example in comnination of the previous sphere:

```
box {
 ⟨-1, 0, -1⟩  // Near lower left corner
 ⟨1, 0.5, 3⟩  // Far upper right corner
 texture{
  T_Stone25  //Pre-defined from stones.inc
  scale 4  //Scale by the same amount in all directions
 }
 rotate y*20  //Equivalent to ”rotate ⟨0, 20, 0⟩”
```

}

We see in the above example a box is defined by specifying the 3D coordinates of its opposite coiners (any two opposie corners may be used). We can then rotate them to any angle (e.g. rotate x*100 as shown in Fig. 3 which combines the box with the sphere leaving the source of light and camera no variations).

2.2 Cone object

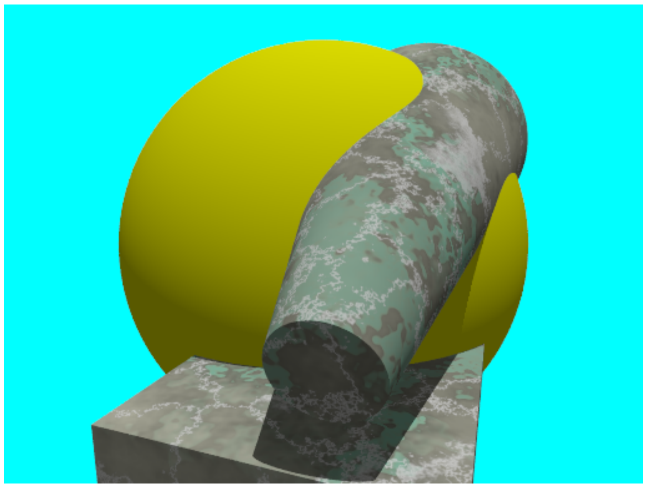


Figure 4. A sphere, a cone and a box generated by POV-Ray.

We can use the following lines to show a **cone**:

```
cone{
⟨0, 1, -1⟩, 0.3
⟨1, 2, 2⟩, 1.0
texture {T_Stone25 scale 4}
}
```

Here, Line-2 defines one center and the radius, and Line-3 defines another center and the corresponding radius. The produced scene is shown in Fig. 4.

{**This part is finished in 26 November, 2015**}

2.3 Cylinder object

cylinder can be defined like this:

```
cylinder {
⟨0, 2, 2⟩
⟨1, 0, -3⟩
0.5
open
texture {T_Stone25 scale 4}
}
```

Line-2 and Line-3 define centers of each side, Line-4 is the radius of the cylinder. **open** meands ”Remove and caps”

Figure 5 including a sphere and a cylinder is produced by the following codes:

```
#include ”colors.inc”
#include ”stones.inc”
background {color Cyan}
  camera{
location ⟨0, 2, -3⟩
look_at ⟨0, 1, 2⟩
}

   sphere {
⟨0, 1, 2⟩, 1
//center of sphere, raidus
texture {
pigment {color Yellow}
}
}

   cylinder{
⟨0, 3, 2⟩
⟨0, -1, 2⟩
0.4
open
texture {T_Stone25 scale 4}
}
light_source {⟨2, 20, -3⟩ color White}
```

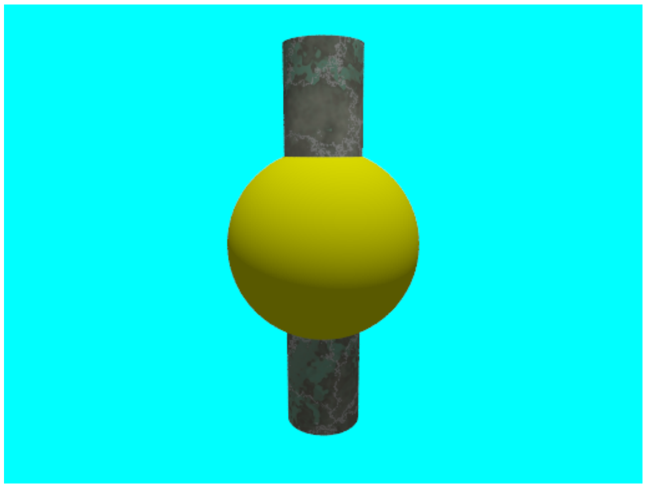


Figure 5. A sphere, a cylinder generated by POV-Ray.

2.4 Plane object

Now let us try to add a "**plane** object to Fig. 5, the new figure is shown in Fig. 6. This figure shows we have built a nice "checkered floor" with only several lines as following:

```
plane { ⟨0, 1, 0⟩, -1
pigment {
checker color Red, color Blue
}
}
```

The object defined here is an infinite plane. ⟨0, 1, 0⟩ defines the surface normal of the plane. The number afterward (-1) is the distance that the plane is displaced along the normal from the origin. We use -1 so that the sphere and the cylinder are on the "floor". Here even though we do not use a **texture** statement there is an implied (Definition in English: not directly expressed) texture here. It is tiresome if we continually (Definition in English: recurring frequently) typing statements that are nested like **texture** {**pigment**}, so POV-Raylet us leave out the **texture** under many circumstances. In general we only need the texture block surrounding a texture identifier (e.g. T_Stone25 in the previous examples), or when creating layered textures which will be covered later.

Besides, in the example, pigment uses the checker color pattern and specifies that the two colors red and blue should be used. Because the vectors ⟨1, 0, 0⟩, ⟨0, 1, 0⟩ and ⟨0, 0, 1⟩ are used frequently thus POV-Rayhas provided us three built-in vector identifiers x, y and z respectively that can be used as a shorthand. Thus the plane could be defines as:

```
plane {
y, -1
pigment{...}
}
```

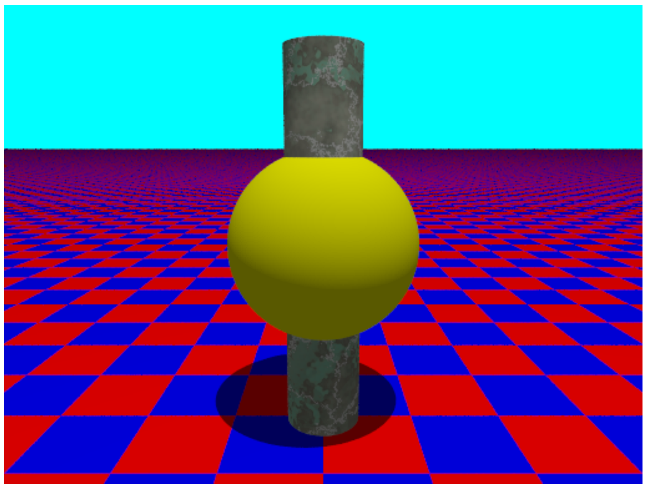


Figure 6. A sphere and a cylinder on a plane.

Look at Fig. 6, a precise, sharp shadow is created by the ray-tracer in POV-Ray. In the real word, penumbral (Definition in

English: the point or area in which light and shade blend) or ”soft” shadows are often seen. Later we will learn how to use extended light sources to soften the shadows.

2.5 Torus object

A torus is a ring-shaped surface that can be thought as a donut or an inner-tube. It is a kind of useful Constructive Solid Geometry (CSG) so POV-Rayhas adopted this 4th order quartic polynominal as a primitive shape. In POV-Ray, the major and minor radius is defines as shown in Fig. 7a. The syntax for a torus is so simple that it makes it a very easy shape to work with once we learn what the two float values mean. Now, let’s produce it!

We create a file called **tordemo.pov** and edit it as follows:

```
{#}include ”colors.inc”
camera {
location ⟨0, .1, -25⟩
look_at 0
angle 30
}
background {color Gray50}   // to make the torus easy to see
light_source {⟨300, 300, -1000⟩ White}
torus {
4, 1   //major and minor radius
rotate -90*x   // so we can see it from the top
pigment { Green }
}
```

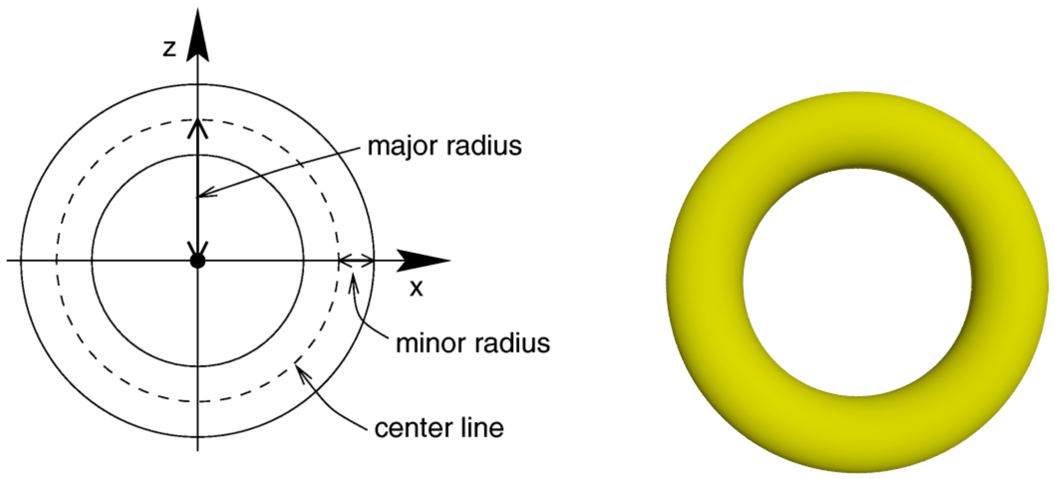
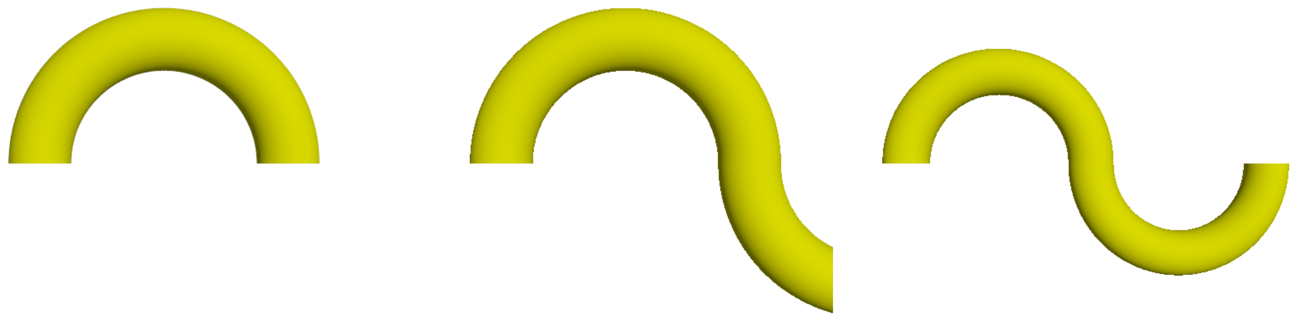


Figure 7. a. The definition of major and minor raidus. b. A donut-like torus. To make the same background of a and b, I used a white background instead of Gray50

You can see a nice-looking ”donut” in Fig. 7b, changing radius can turn the ”donut” into ”hula-loop”-like (major 4,minor 0.3) torus. However, using such simple syntax we cannot do much else. Let us see...



(a) A half-torus **(b)** Two half-tori **(c)** Same to b but with a optimized camera

Figure 8. Torus object

Tori are very useful CSB, and we can try a little experiment with it. We make a **difference** of a torus and a box:

```
difference {  torus {
4, 1
rotate -90*x  // so we can see it from the top
}
box {⟨-5, -5, -1⟩, ⟨5, 0, 1⟩}
pigment {Yellow}
}
```

Interesting... a half-torus as shown in Fig. 8a.

{**This part is finished in 27 November, 2015.**}

In the previous part, we have produced a half-torus, now we add another one flipped the other way. Only, let's declare the original half-torus and the necessary transformations so we can use them again:

```
#declare Half_Torus = difference {
torus {
4, 1
rotate -90*x  // so we can see it from the top
}
box {⟨-5, -5, -1⟩, ⟨5, 0, 1⟩}
pigment { Yellow }
}
#declare Flip_It_Over = 180*x;
#declear Torus_Translate = 8;  // twice the major radius
```

Fig. 8b shows two connected half-tori. Note that we do not modify the parameters of background, camera and light source. One might find that we cannot see the whole scene of Fig. 8b, it requires us to make a minor fix of the **location** and **camera**, for example, look_at ⟨3.8, 0, 0⟩ and location ⟨0, .1, -34⟩ can display more, as shown in Fig. 8c.

union is very useful as we have used just now, we can use it to make more torus like this:

```
{#}include ”colors.inc”
camera {
location ⟨0, .1, -34⟩
look_at ⟨3., 0, 0⟩
angle 30
}
background {color White}   // to make the torus easy to see
light_source {⟨300, 300, -1000⟩ White}

#declare Half_Torus = difference {
torus {
4, 1
rotate -90*x   // so we can see it from the top
}
box {⟨-5, -5, -1⟩, ⟨5, 0, 1⟩}
pigment { Yellow }
}
#declare Flip_It_Over = 180*x;
#declear Torus_Translate = 8;  // twice the major radius
union{
object {Half_Torus}
object {Half_Torus
rotate Flip_It_Over
translate x*Torus_Translate
}
object {Half_Torus
translate x*Torus_Translate*2
}
object {Half_Torus
rotate Flip_It_Over
translate x*Torus_Translate*3
}
object {Half_Torus
rotate Flip_It_Over
translate -x*Torus_Translate
}
object {Half_Torus
translate -x*Torus_Translate*2
}
object {Half_Torus
rotate Flip_It_Over
translate -x*Torus_Translate*3
}
object {Half_Torus
translate -x*Torus_Translate*4
}
rotate y*65
translate z*30
}
```

The new scene is plotted in Fig. 9. We can see a cool, undulating, snake-like something-or-other. Neato (Definition in English: wonderful).

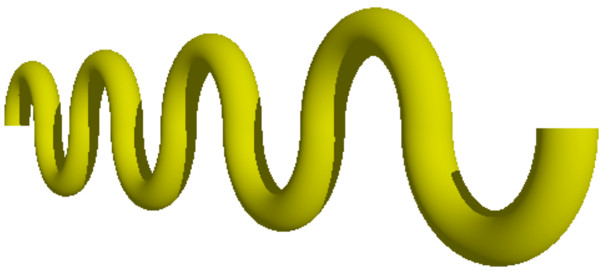


Figure 9. A undulating, snake-like object

Actually we can model something useful like a chain. Thinking about it for a moment, we realize that a single link of chain is composed of two half tori and two cylinders. We create a new file chain.pov. We can use the same camera, background, light source and declared objects and transformations as we used in tordemo.pov:

```
{#}include ”colors.inc”
camera {
location ⟨0, .1, -34⟩
look_at ⟨3., 0, 0⟩
angle 30
}
background {color White}   // to make the torus easy to see
light_source {⟨300, 300, -1000⟩ White}

#declare Half_Torus = difference {
torus {
4, 1
rotate -90*x   // so we can see it from the top
}
box {⟨-5, -5, -1⟩, ⟨5, 0, 1⟩}
pigment { Yellow }
}
#declare Flip_It_Over = 180*x;
#declear Torus_Translate = 8;  // twice the major radius
```

Based on this, we can add a complete torus of two half tori:

```
union{
object {Half_Torus}
object {Half_Torus
rotate Flip_It_Over
}
```

This may seem like a wasteful way to make a complete torus, but we are really going to move each half apart to make room for the cylinders. First we add the declared cylinder **before** the union:

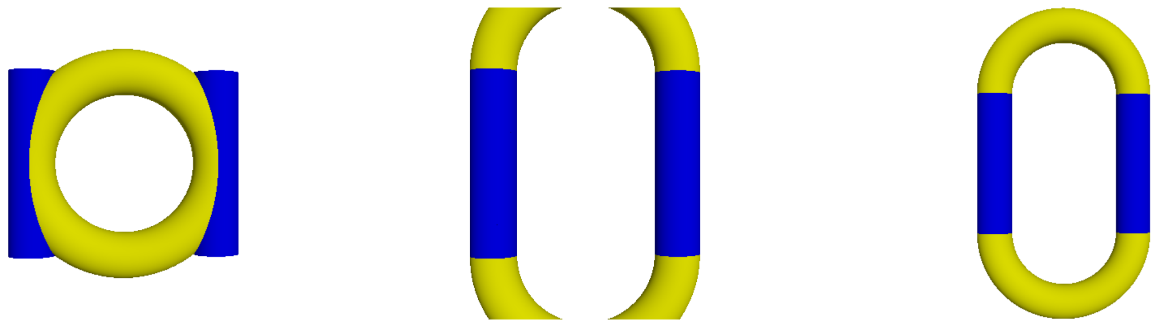
```
#declare Chain_Segment = cylinder {
⟨0, 4, 0⟩, ⟨0, -4, 0⟩, 1
pigment {Blue}
}
```

We then add two **Chain_Segments** to the union and translate them so that they line up with the minor radius of the torus on each side.

```
union{
object {Half_Torus}
object {Half_Torus rotate Flip_It_Over}
object {Chain_Segments translate x*Torus_Translate/2}
object {Chain_Segments translate -x*Torus_Translate/2}
}
```

Before pushing forward, we can produce this scene in Fig. 10a first. Then we can translate the two half tori along +y and -y so that the clipped (Definition in English: neatly cut) ends of the cylinders. This distance is equal to half of the previously declared Torus_Translate:

```
union{
object{
Half_Torus
translate y*Torus_Translate/2
}
object{
Half_Torus rotate Flip_It_Over
translate -y*Torus_Translate/2
```



(a) A torus and two cylinders **(b)** A single link **(c)** A single link seen from a farther camera

Figure 10. Torus Construction a link from torus and cylinders.

```
}
object{
Chain_Segment
translate x*Torus_Translate/2
}
object{
Chain_Segment
translate -x*Torus_Translate/2
}
}
```

Yew, Fig. 10b is what we just generate using POV-Ray. It's cool. In order to get whole scene, I move the camera location to farther (location ⟨0, .1, -45⟩) and voila, the updated scene is displayed in Fig. 10c

Yes, we have got a single link. But we are not done yet! I used two colors in the link to let you have a deep insight of how POV-Rayworks with these objects. Now we need use a nice metallic color instead.
{**This part is finished in 29 November, 2015**} First,

Torus object

Example text under a subsection. Bulleted lists may be used where appropriate, e.g.

- First item
- Second item

Discussion

The Discussion should be succinct and must not contain subheadings.

Methods

First-principles calculations.

Acknowledgements

Author contributions

Additional information

Supplementary Information accompanies this paper at http://www.nature.com/srep

Competing financial interests: The authors declare no competing financial interests.

Condition	n	p
A	5	0.1
B	10	0.01

Table 1. Legend (350 words max). Example legend text.

Figures and tables can be referenced in LaTeX using the ref command, e.g. Table 1.