Computer Graphics

P. Healy

CS1-08
Computer Science Bldg.
tel: 202727
patrick.healy@ul.ie

Spring 2021–2022

Outline

Scene Modellers

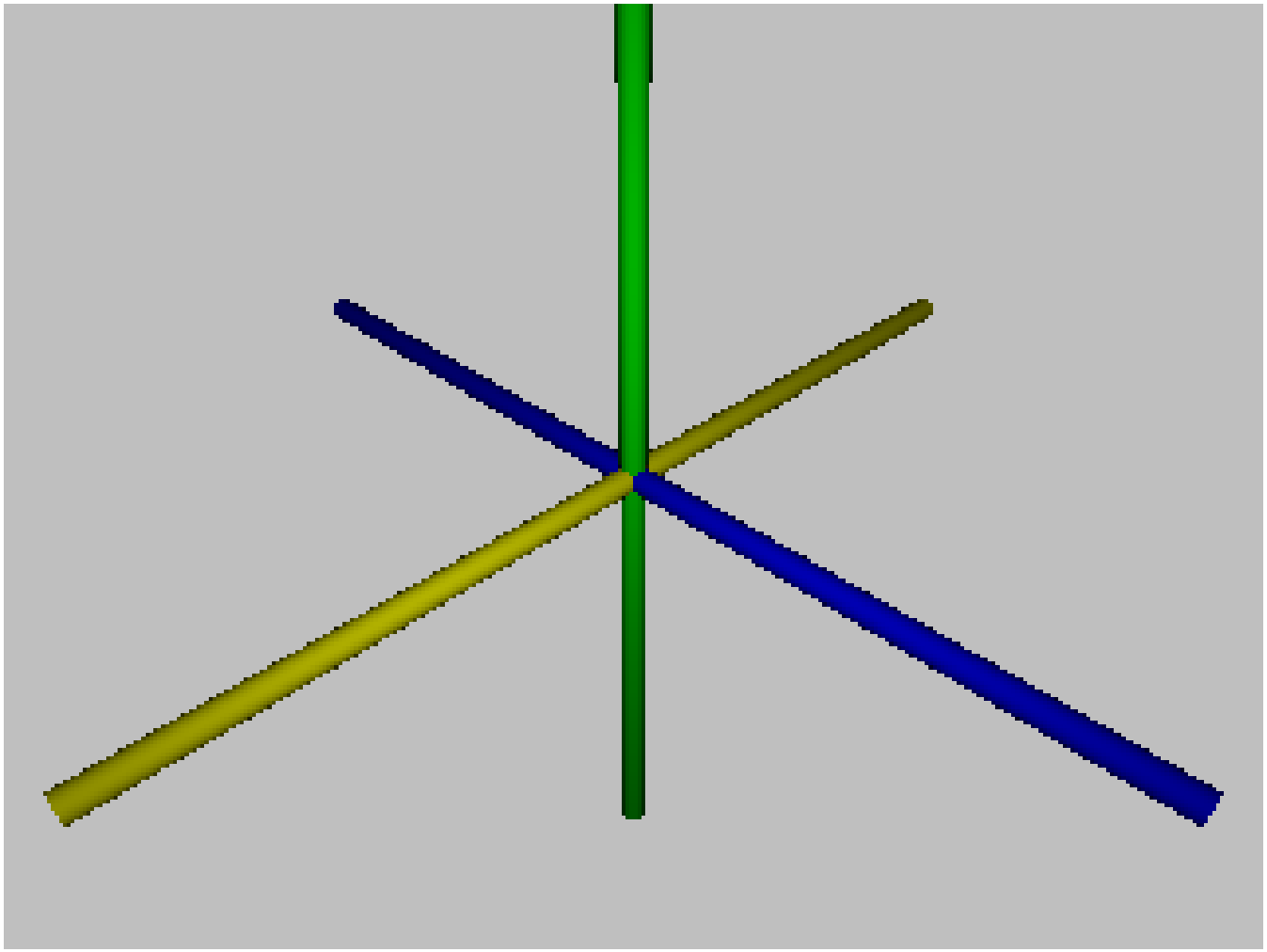
- There are many different scene modelling languages available
- A small sampling
  - Persistence of Vision Raytracer (POV-Ray)
  - Blender
  - OpenSceneGraph Home Page some example shots
  - Coin3D Home Page
  - VTK (Visualization Toolkit) Home Page
- We will take a closer look at the first of these now

Persistence of Vision Raytracer (POV-Ray)

- As name suggests this modelling system permits consideration of lighting effects
- Shadows, multiple light sources, etc. are possible
- Very good documentation
- Do not confuse its *scene description language (SDL)* with Simple DirectMedia Layer (SDL), a cross-platform multimedia library Home Page
- Some high-level modelling tools exist that allow creation of `.pov` files automatically, but...

A simple POV model

Three cylinders meeting at origin



A simple POV model

The cylinder along x-axis, 8 units long

```
cylinder { // x-dirn
  <20, 0, 0>,      // Center of one end
  <-20, 0, 0>,     // Center of other end
  0.5              // Radius
  texture {
    pigment { colour Yellow }
  }
}
```

- 'x', 'y' and 'z' are special variables representing vectors along the three axes; so we could replace '<20,0,0>' with '20*x'
- POV-Ray is a lefthanded coordinate system by default

A simple POV model (contd.)

3 cylinders

```
cylinder { // x-dirn
  20*x,              // Center of one end
 -20*x,              // Center of other end
  0.5               // Radius
  texture { pigment { colour Yellow } }
}
cylinder { // y-dirn
  20*y, -20*y, 0.5 // y=20 to y=-20, dia
  texture { pigment { colour Green } }
}
cylinder { // z-dirn
  20*z, -20*z, 0.5 // z=20 to z=-20, dia
  texture { pigment { colour Blue } }
}
```

A simple POV model (contd.)

Adding lighting and a camera

```
#include "colors.inc"  // The include files contain
#include "stones.inc"  // pre-defined scene element

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

light_source { <9, 9, 9> color White}
background { color Grey }

cylinder { //x-dirn... } // same definition as
cylinder { //y-dirn... } // previous cylinders
cylinder { //z-dirn... }
```

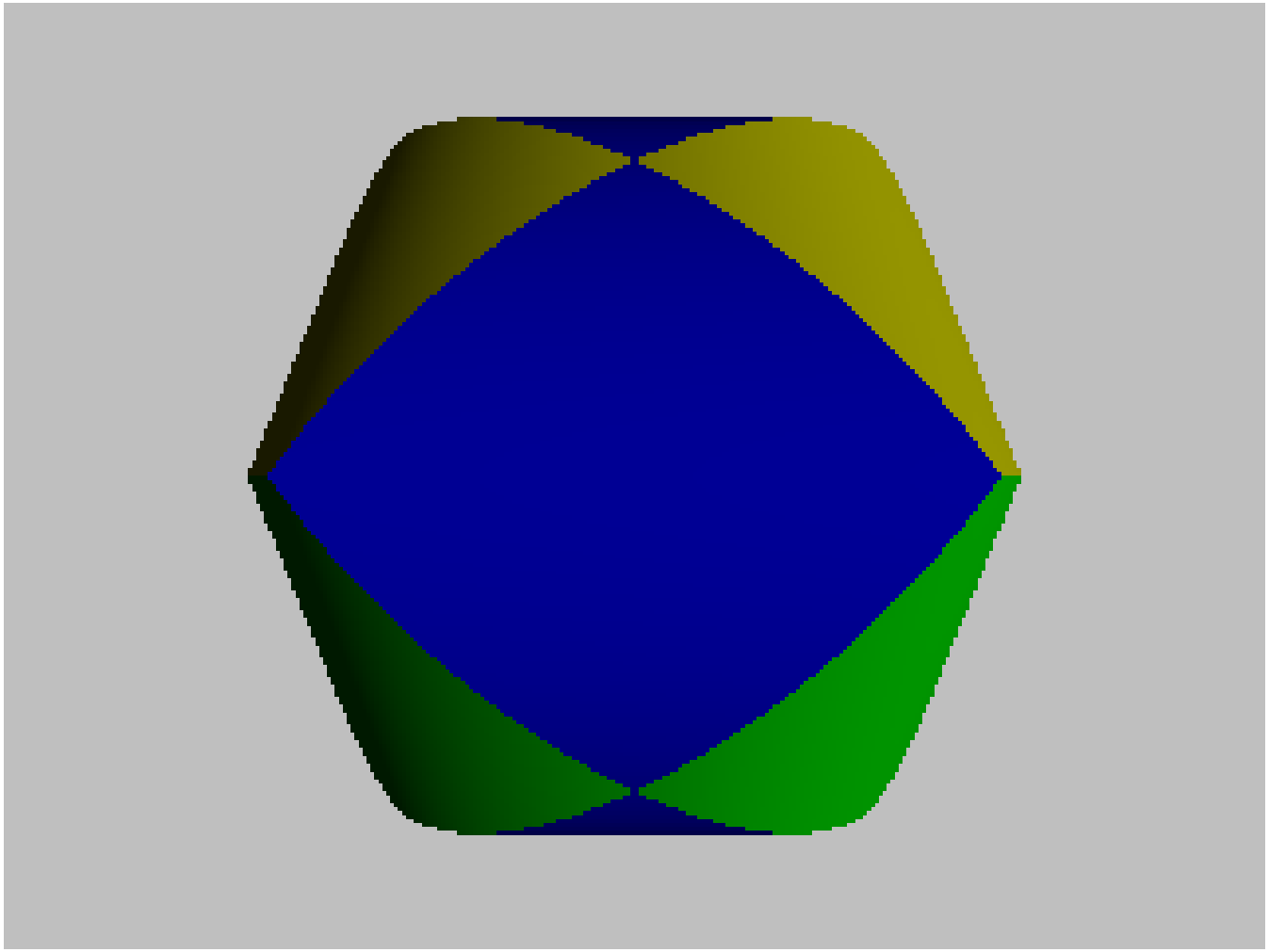
The cylinders' intersection

```
//same lighting, camera, etc.

intersection {
cylinder { // x-dirn (as before) ... }
cylinder { // y-dirn... }
cylinder { // z-dirn... }
}
```

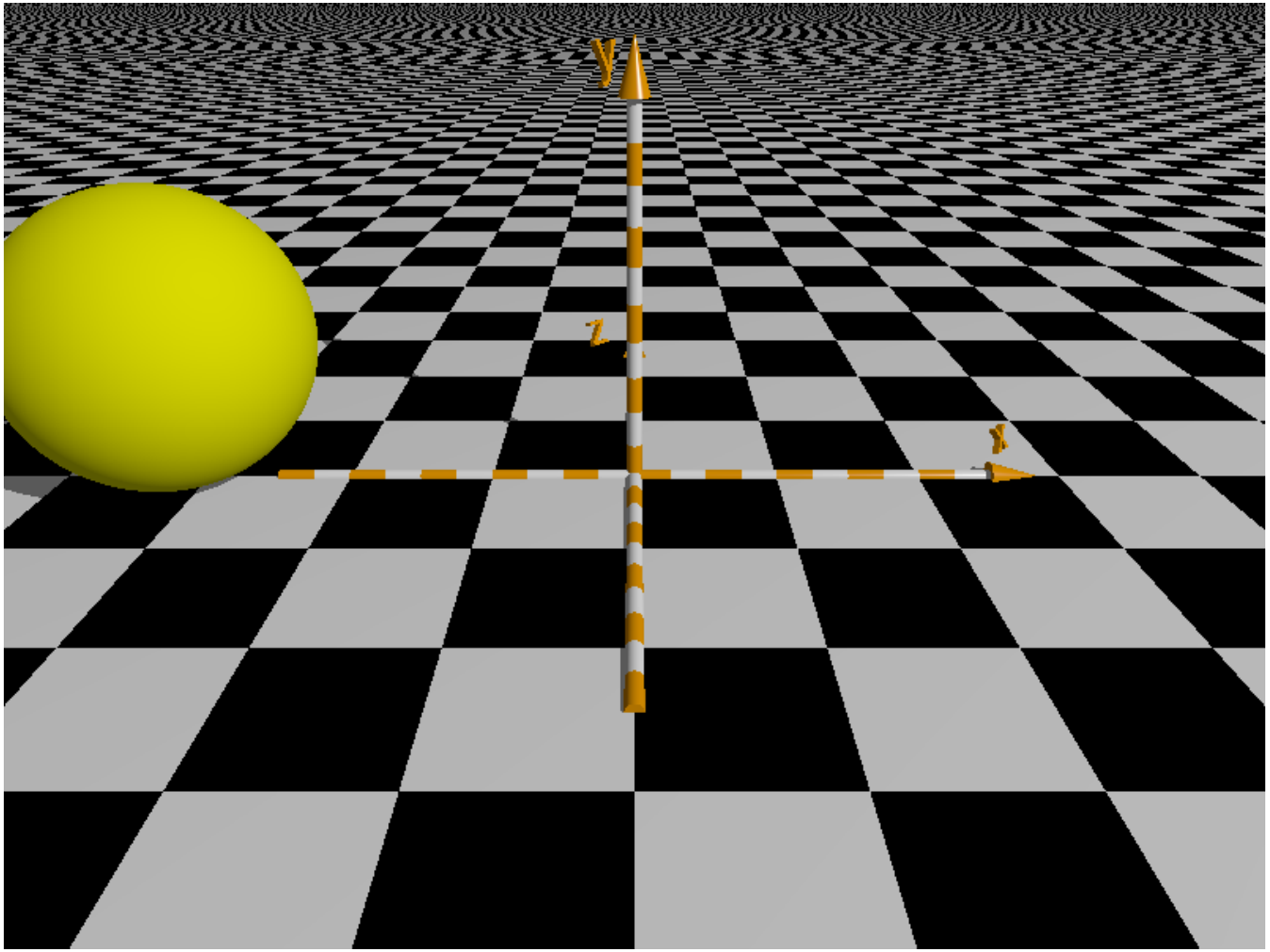
The cylinders' intersection

Intersection of three cylinders meeting at origin



See also Wikipedia's CSG page

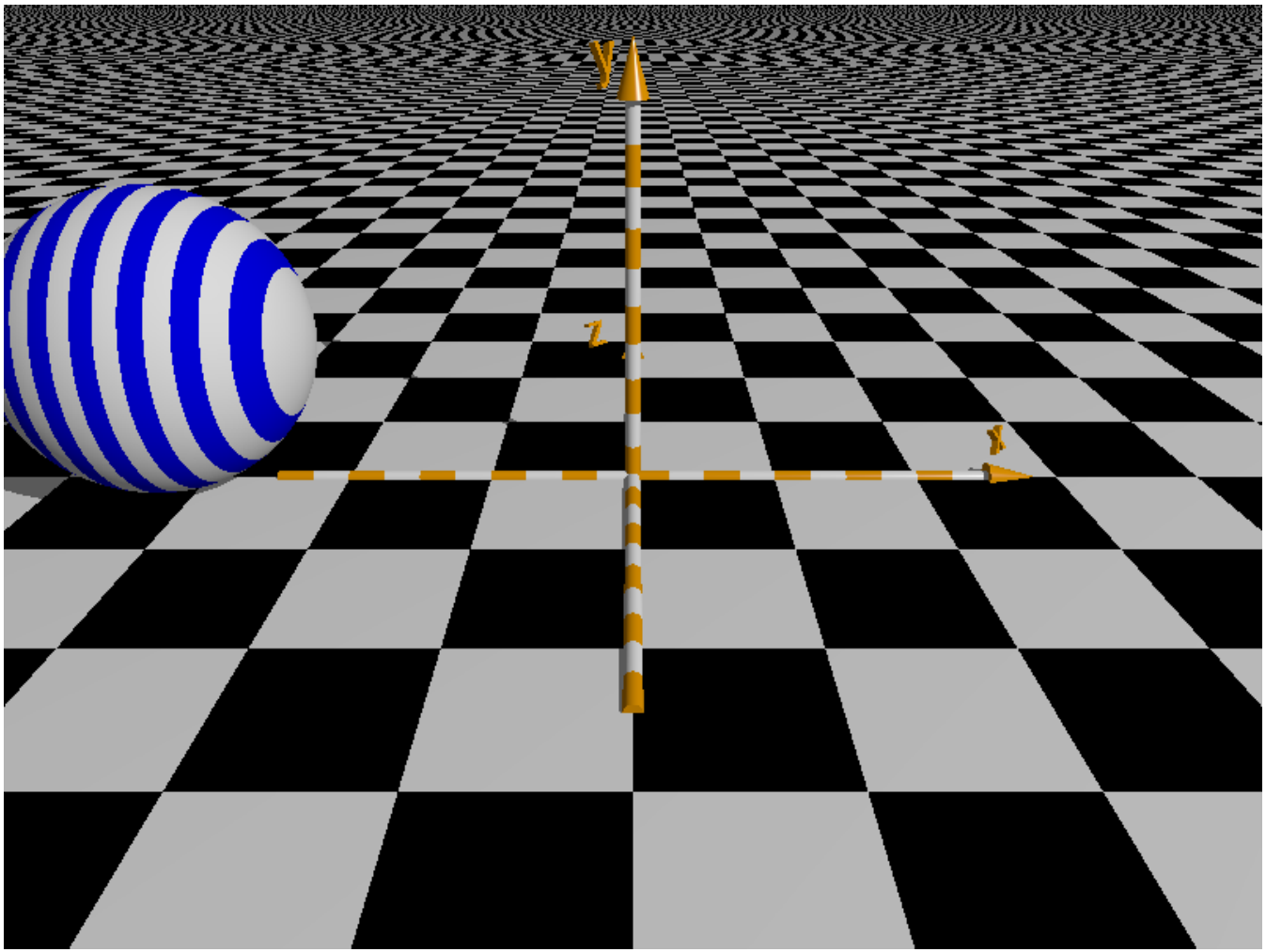
A Resting Sphere



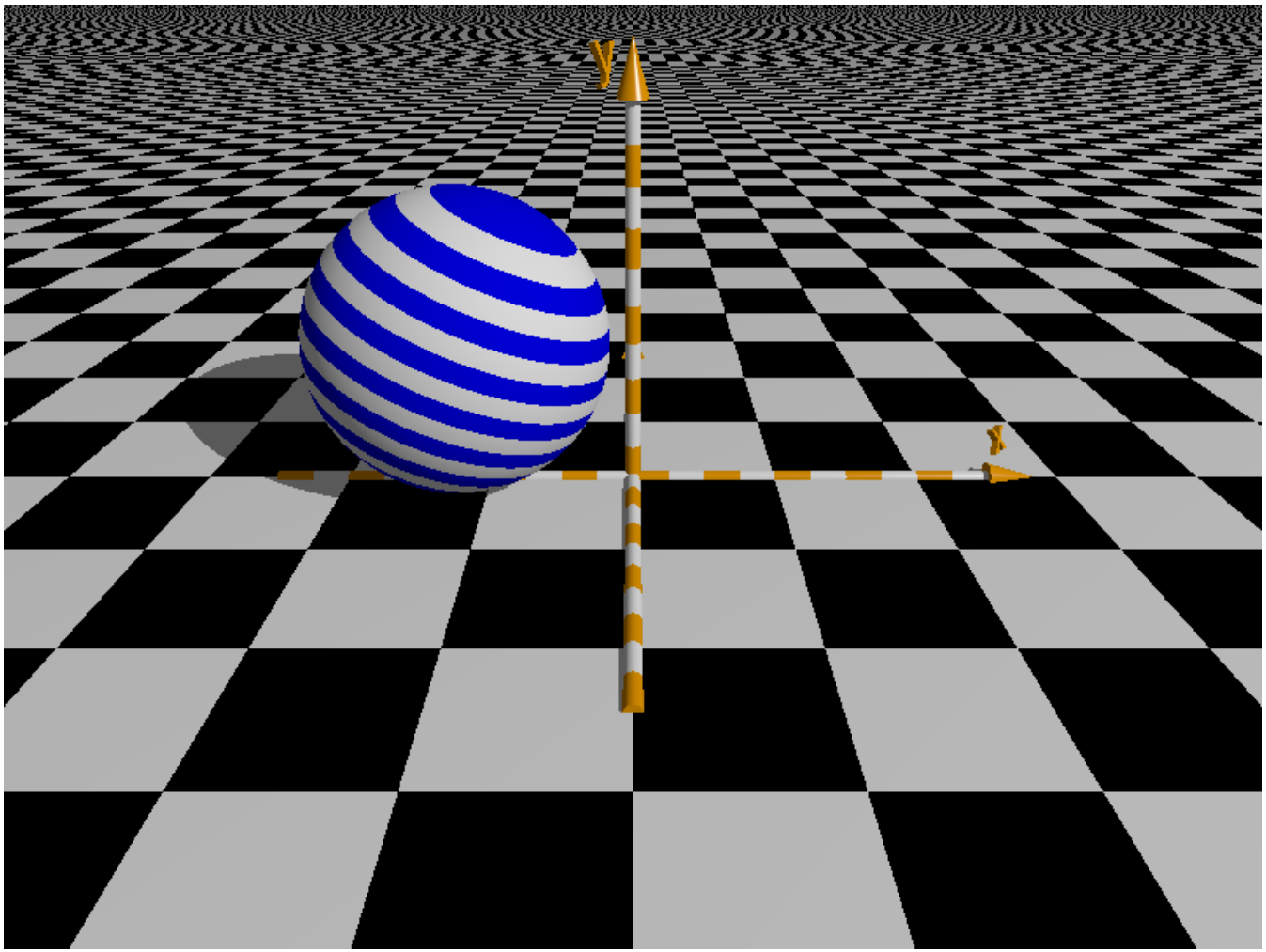
A Resting Sphere

```
#include "colors.inc"
camera {
  location <0, 3, -6>
  look_at <0, 0, 0>
}
light_source { <20, 20, -20> color White }
plane {
  y, 0 // y=<0,1,0> is normal, 0=dist. from origin
  pigment { checker White, Black }
}
sphere {
  <0, 0, 0> , 1
  texture { pigment { colour Yellow } }
  translate <-pi, 1, 0>
}
```

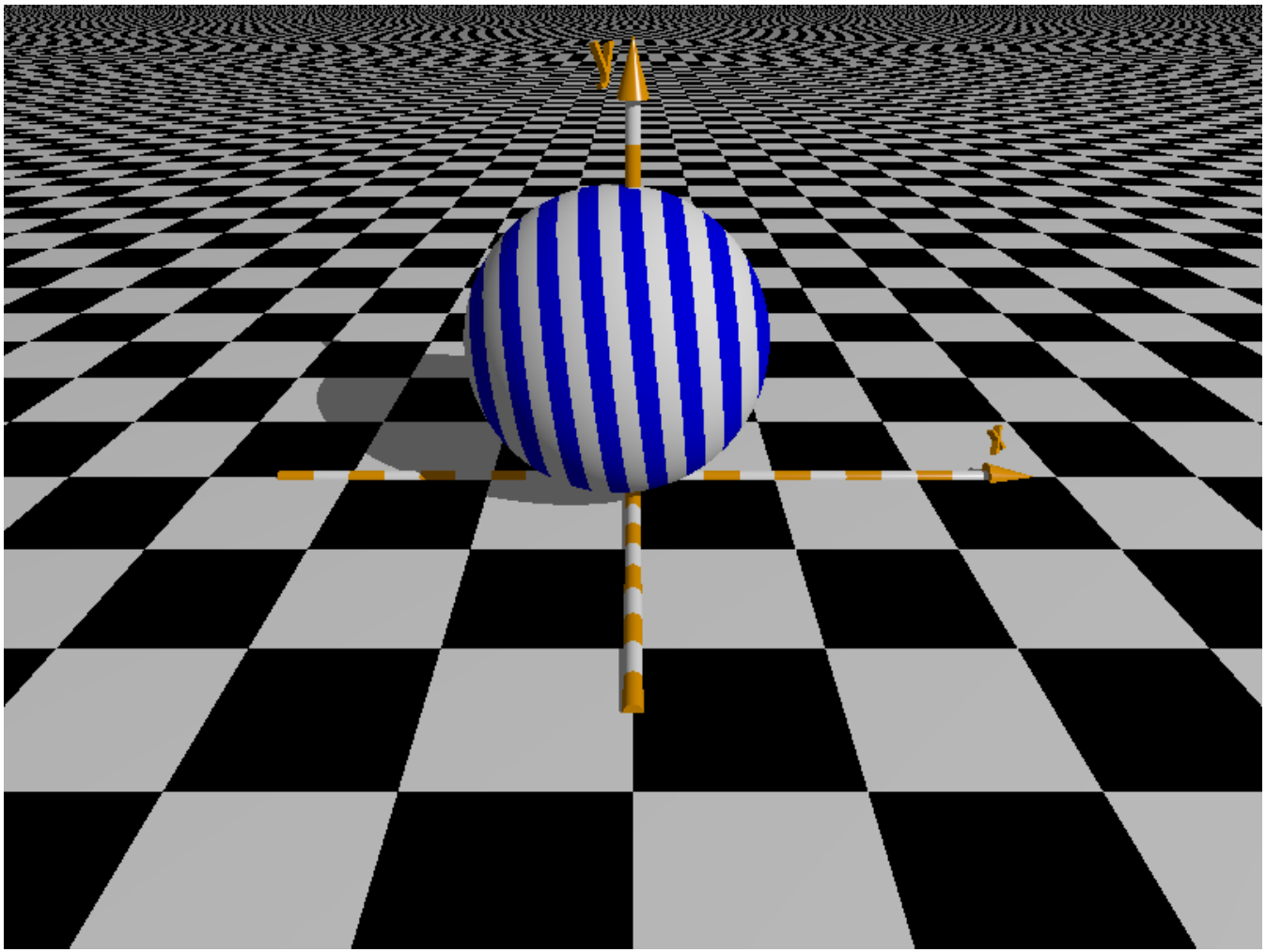
A Rolling Sphere



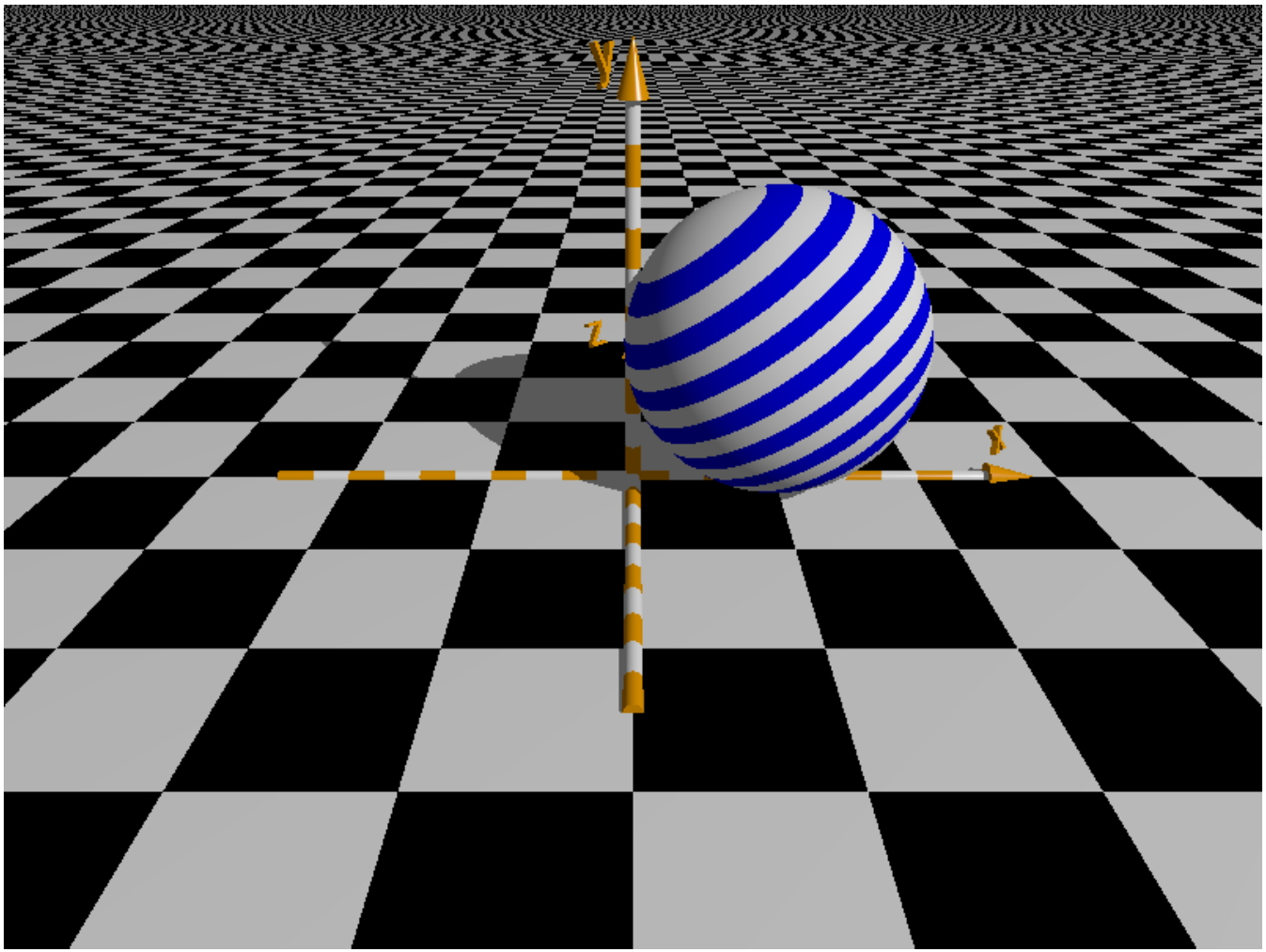
A Rolling Sphere



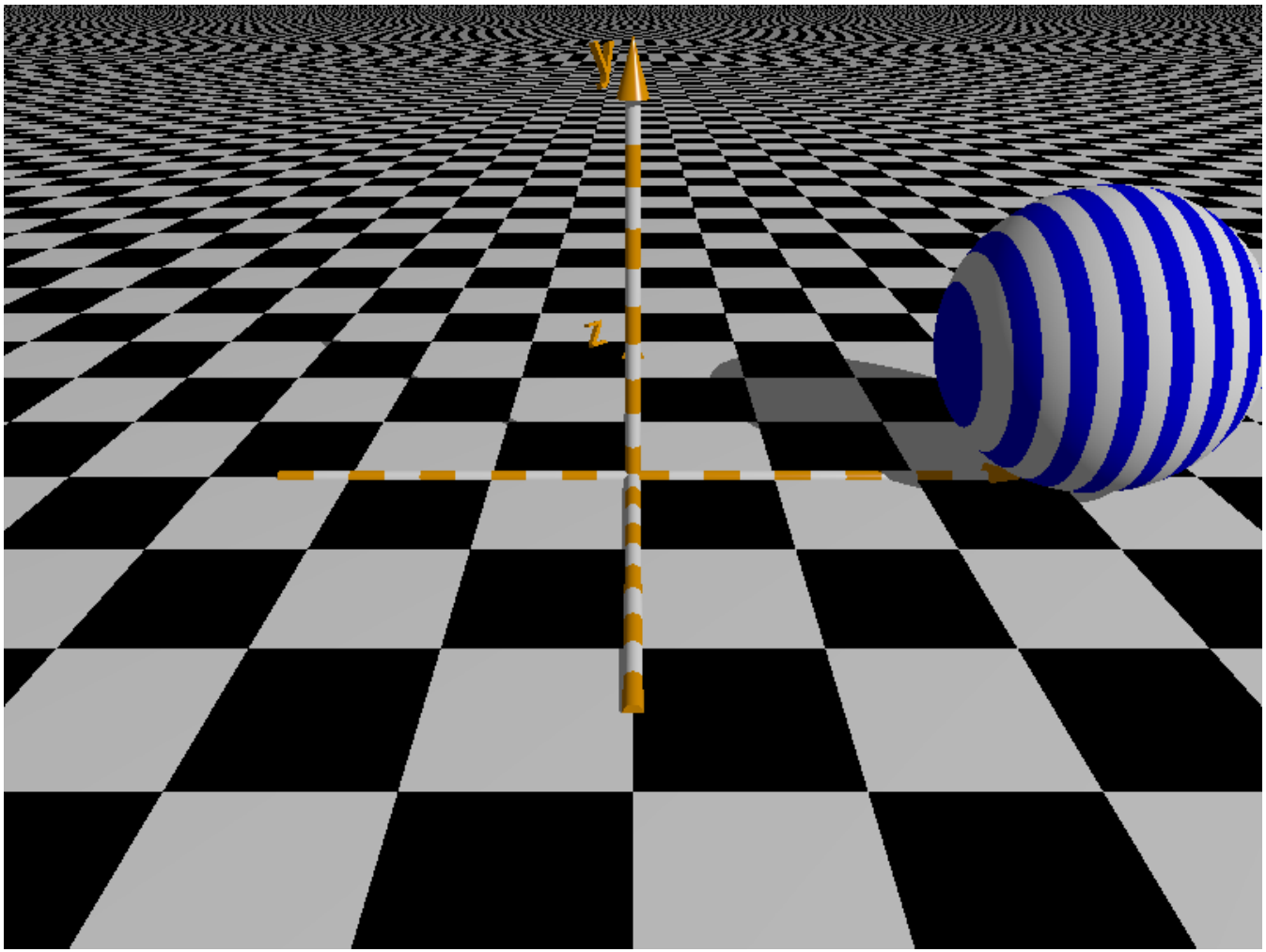
A Rolling Sphere



A Rolling Sphere



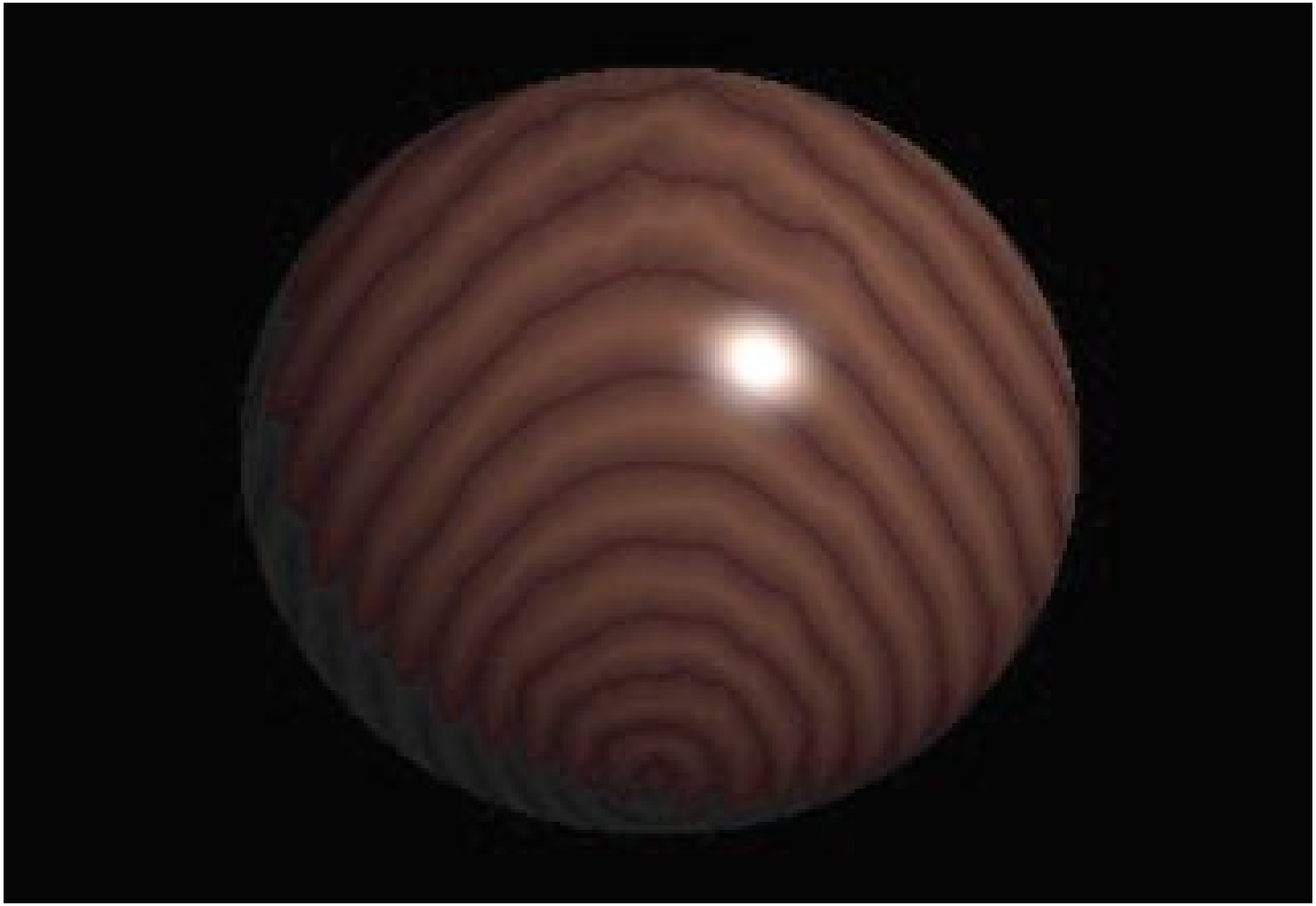
A Rolling Sphere



A Rolling Sphere

```
sphere { //      Modified pov model
  <0, 0, 0> , 1
  pigment {
    gradient x
    color_map {    doc    here
      [0.0 Blue  ]
      [0.5 Blue  ]
      [0.5 White ]
      [1.0 White ]
    }
    scale .25
  }
  translate <-pi, 1, 0>
  rotate <0, 0, -clock*360> //  rotate
  translate <2*pi*clock, 0, 0>
}
```

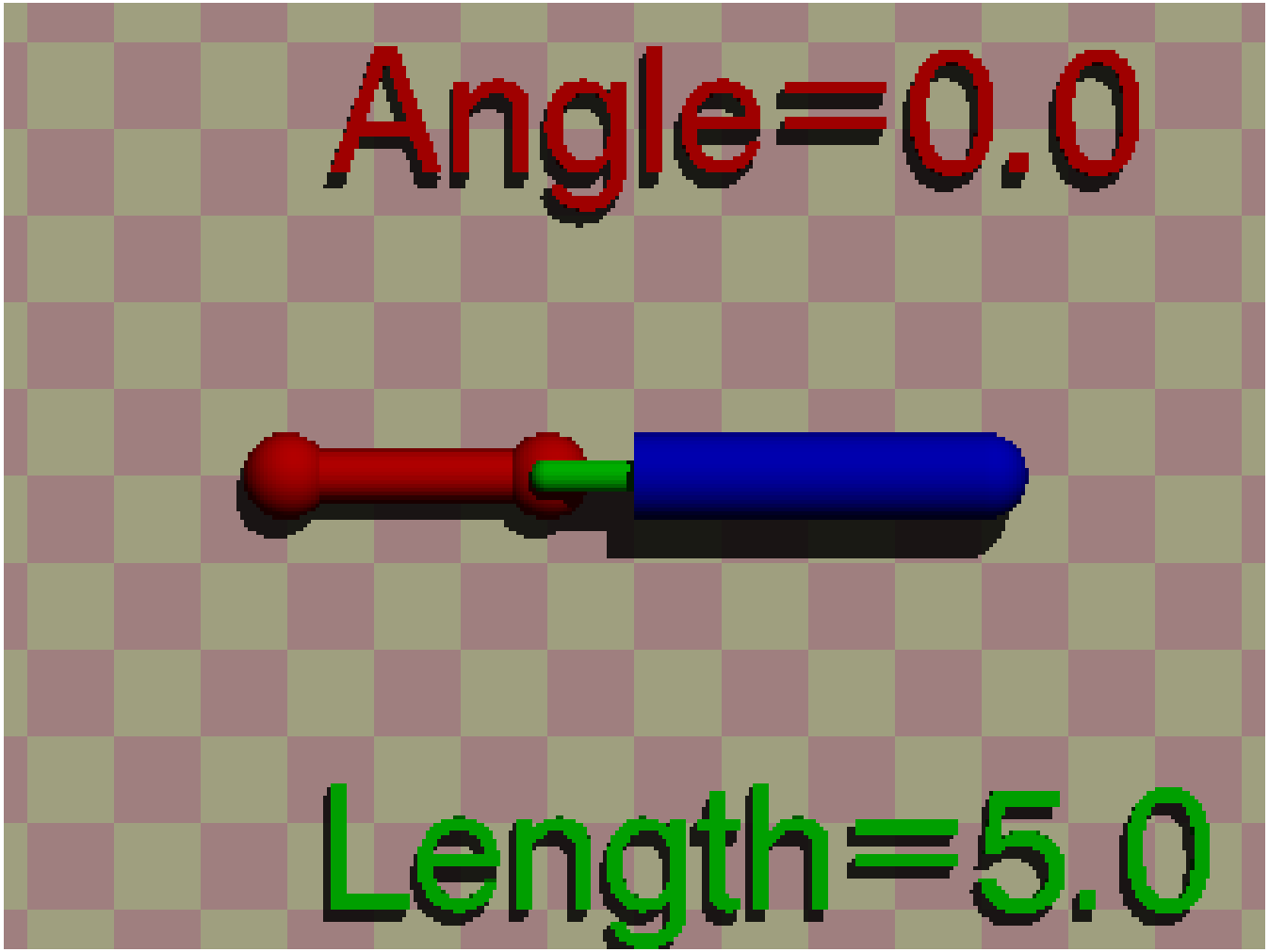
A Textured Sphere



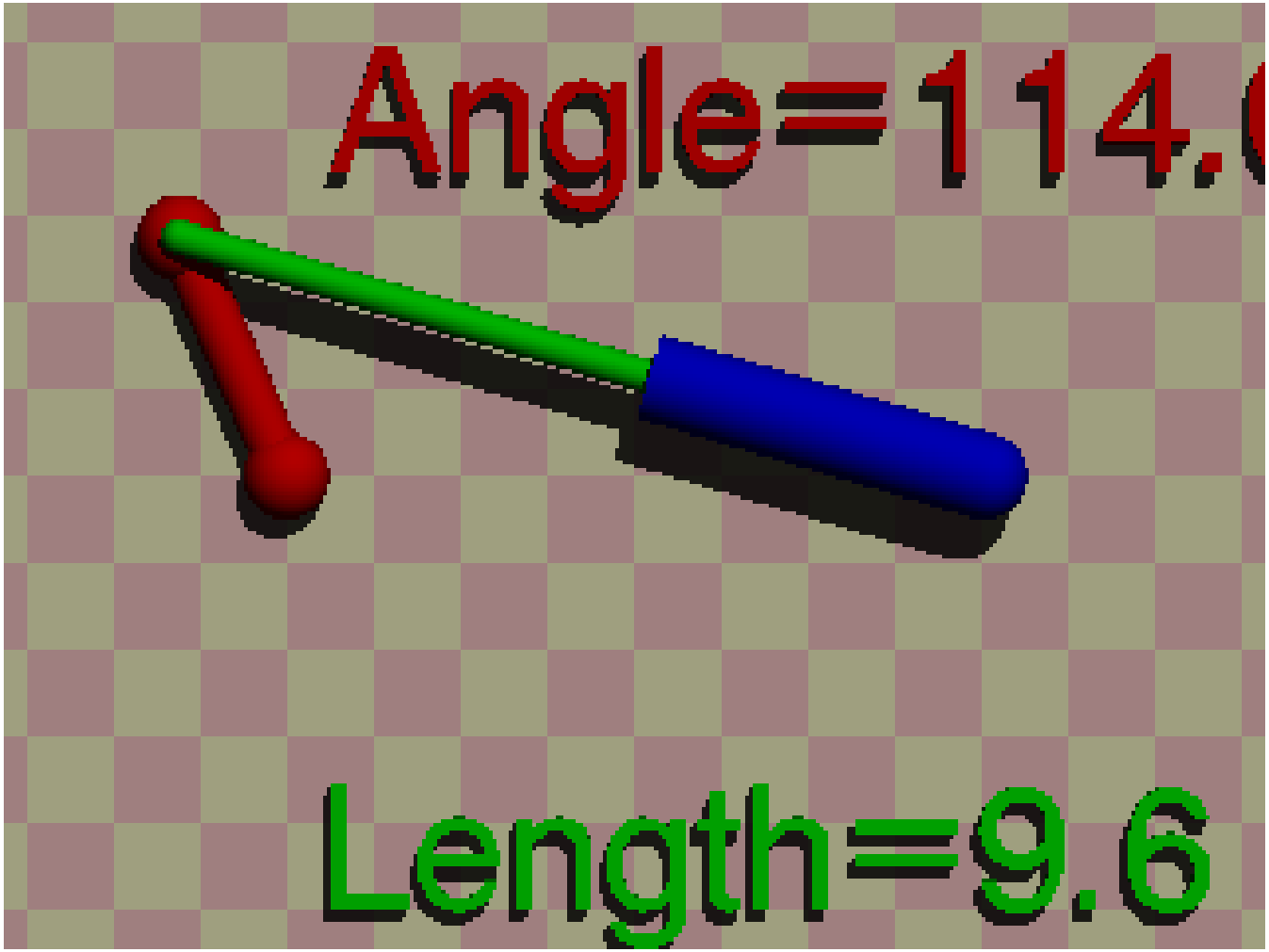
A Textured Sphere

```
sphere {
   <0, 1, 2>, 2
   texture {
     pigment {
       wood
       color_map {
         [0.0 color DarkTan]
         [0.9 color DarkBrown]
         [1.0 color VeryDarkBrown]
       }
       turbulence 0.05
       scale <0.2, 0.3, 1>
     }
  finish {phong 1}
  }
}
```

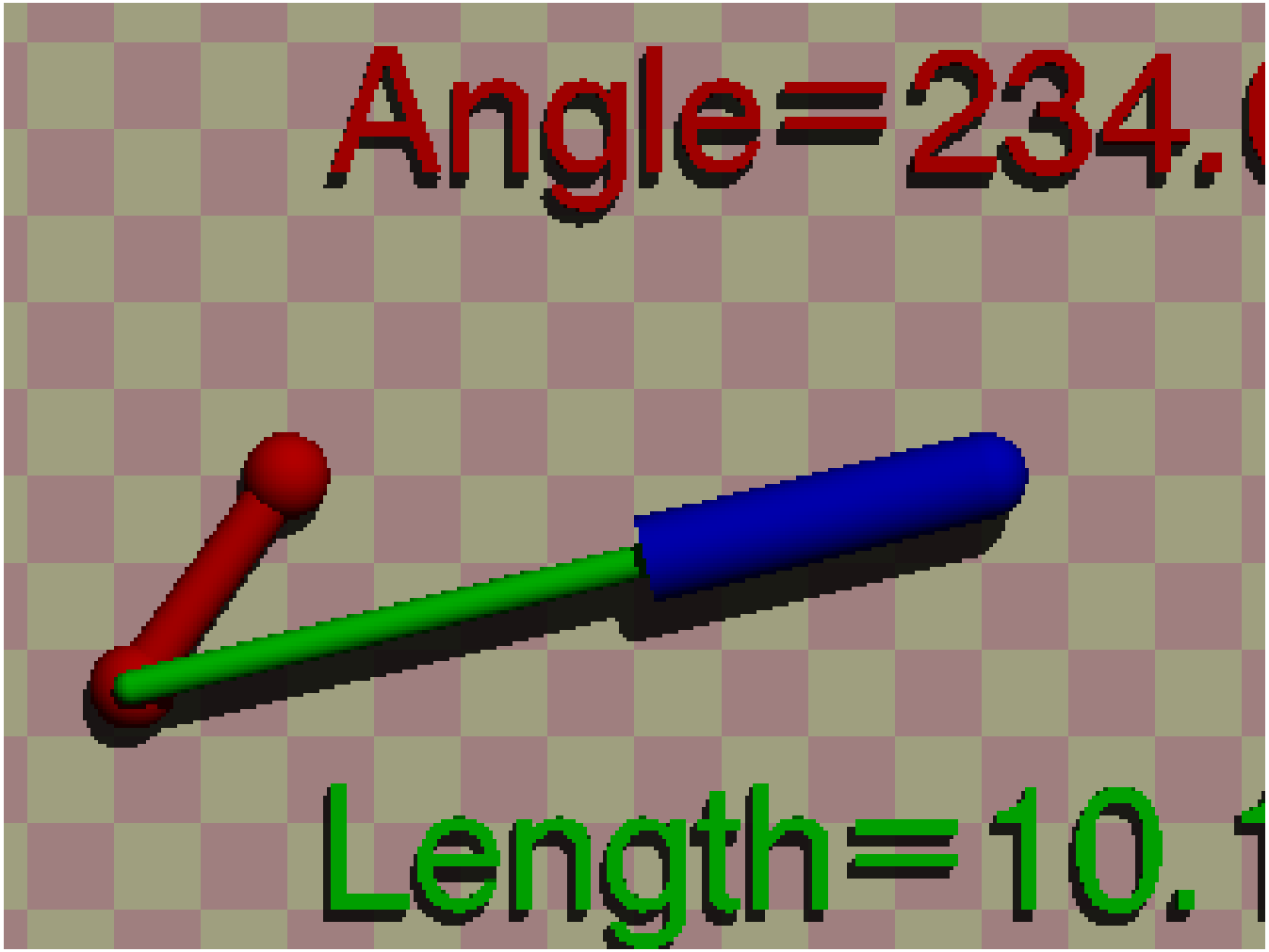
A More Complex Example



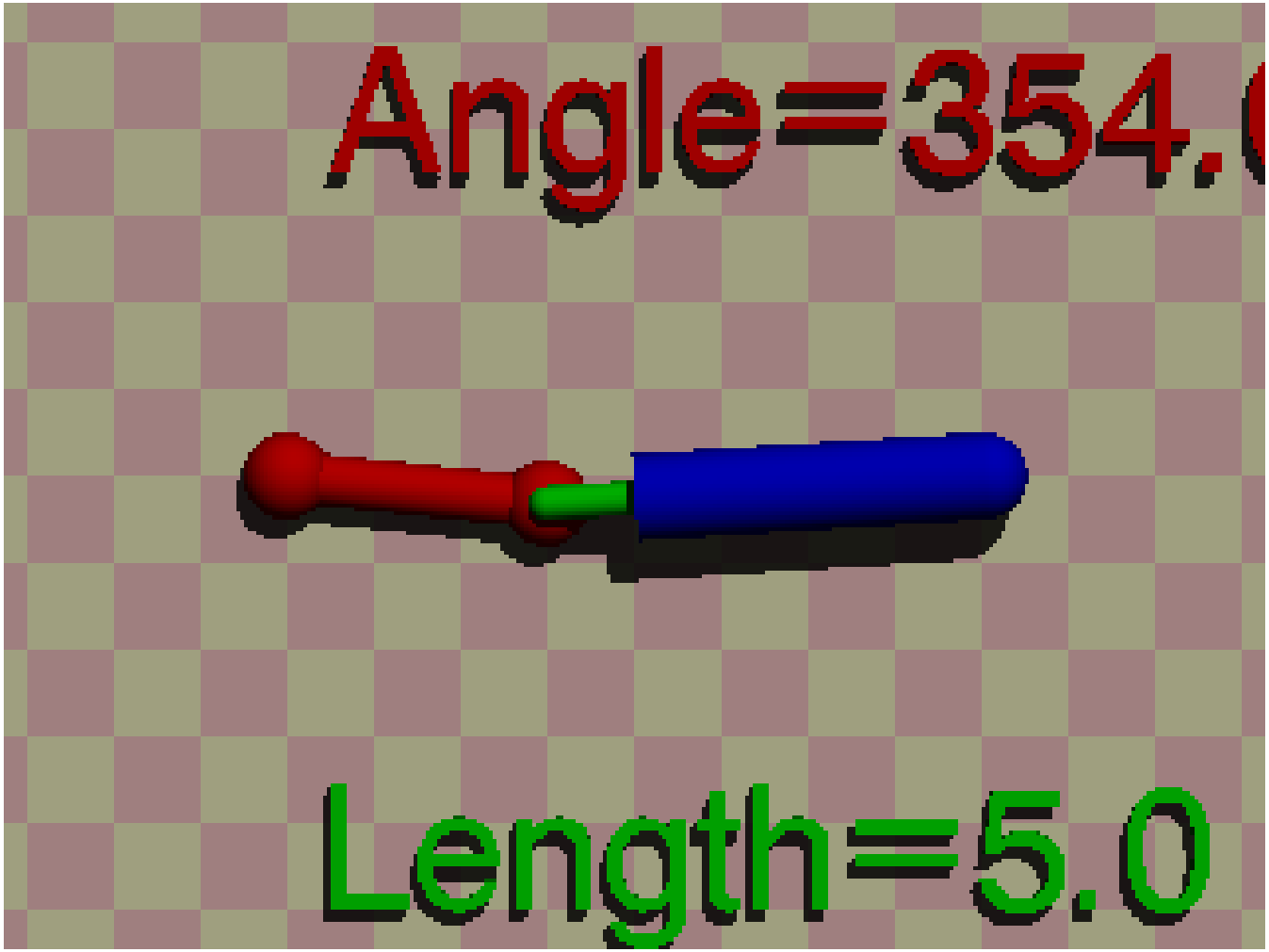
A More Complex Example



A More Complex Example



A More Complex Example



A More Complex Example

A `.ini` file to control generation of animation

```
    :
Test_Abort_Count=100

Input_File_Name=vect1.pov

Initial_Frame=1
Final_Frame=60
Initial_Clock=0
Final_Clock=1

Cyclic_Animation=on
Pause_when_Done=off
```

A More Complex Example

And now the `vect1.pov` file

```
global_settings { assumed_gamma 2.2 }

#include "colors.inc"

#declare Font="cyrvetic.ttf"

// Basic clock runs from 0.0 to 1.0 but we want
// different units than that.  So, define
//   a scaled version.

#declare Clock360 = 360*clock;
#declare ClockRot = Clock360*z;
```

A More Complex Example

```
text{ttf Font concat("Angle=",str(Clock360,1,1)),
  0.1,0 scale 2 pigment{Red} translate <-3.5,3.5,0>
text{ttf Font concat("Length=",str(Long_Length,1,1)
  0.1,0 scale 2 pigment{Green} translate <-3.5,-5,0

camera {
   location  <0, 0, -120>
   direction <0, 0.5,  11>
   look_at   <0, 0,   0>
}

light_source { <5000, 10000, -20000> color White}
plane { -z, -1/3
        pigment {checker color rgb <1,.8,.8>
                         color rgb <1,1,.8>}
}
```

A More Complex Example

```
// An object that rotates one full circle of 360 de
#declare Arm =
 union{
   cylinder{0,3*x,.3}
   sphere{0,.5}
   sphere{3*x,.5}
   pigment{Red}
   rotate ClockRot
 }

// A point on the object that is rotating
#declare Attach_Point=vrotate(x*3,ClockRot);

// A point where we will anchor the push rod
#declare Fixed_Point =x*8;
```

A More Complex Example

```
// This rod runs from Attach_Point to Fixed_Point.
// It varies in length as the Arm rotates.
#declare Long_Rod=
 union{
   sphere{Attach_Point,.2}
   cylinder {Attach_Point,Fixed_Point,0.2 }
   pigment{Green}
 }

// Use the vlength function to compute the length.
#declare Long_Length= \
      vlength(Attach_Point - Fixed_Point);
```

A More Complex Example

```
// We want a fixed length short, fat rod that
// follows the same angle as the long rod.  Compute
// a unit vector that is parallel to the long rod.
#declare Normalized_Point = \
   vnormalize(Attach_Point-Fixed_Point);

#declare Short_Length=4;
#declare Short_Rod=
  union{
    sphere{0,.5}
    cylinder {0,Short_Length*Normalized_Point,0.5}
    translate Fixed_Point  // move into place
    pigment{Blue}
  }
```

A More Complex Example

```
union {
  object{Arm}
  union {
    object{Long_Rod}
    object{Short_Rod}
    translate -z/2
  }
  translate -x*4
}
```

Using the Torus

How to Create a Chain

Have a look at torus documentation for uses