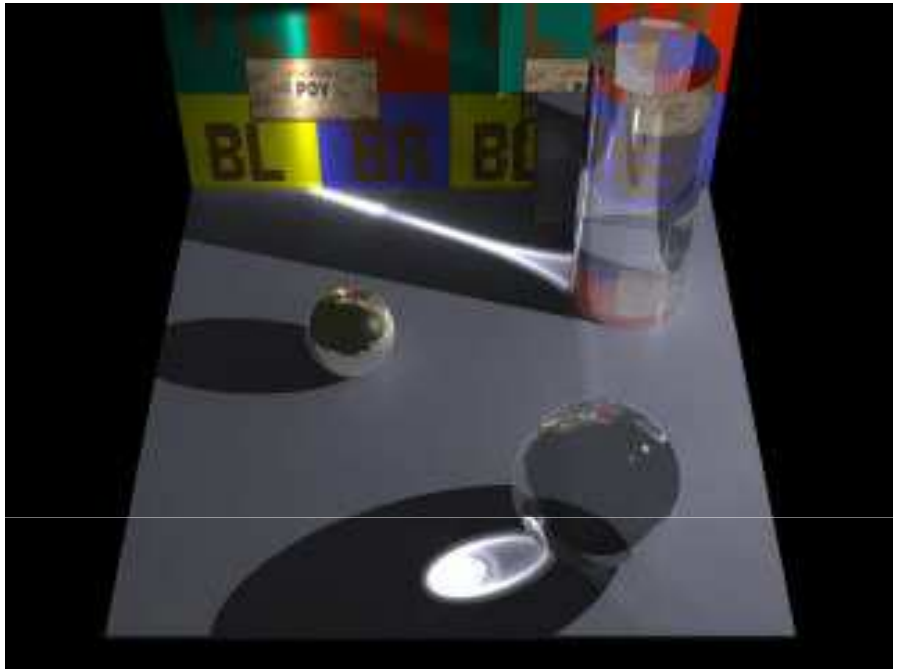
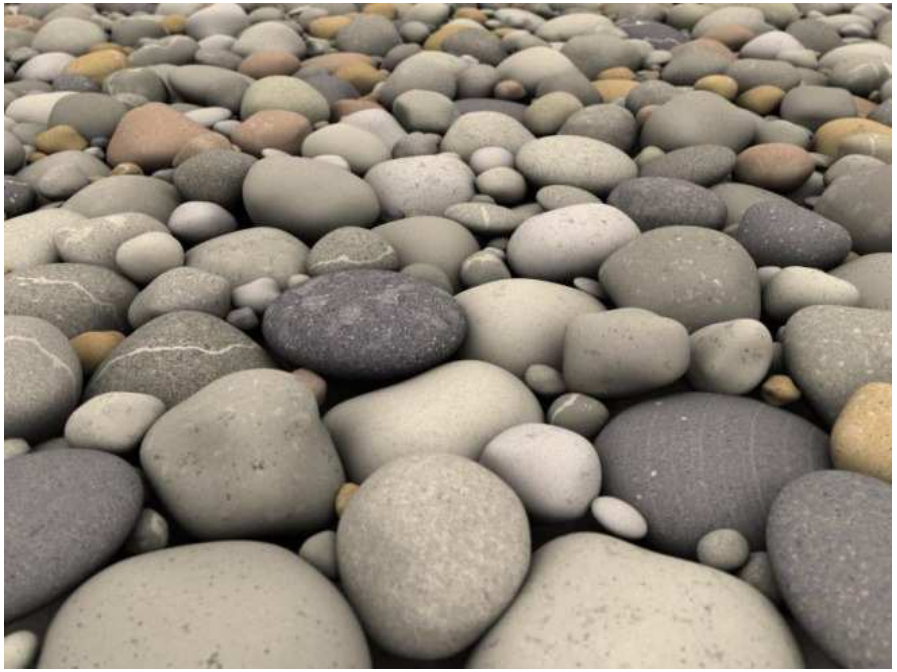
RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG





A Short Introduction to POVRAY

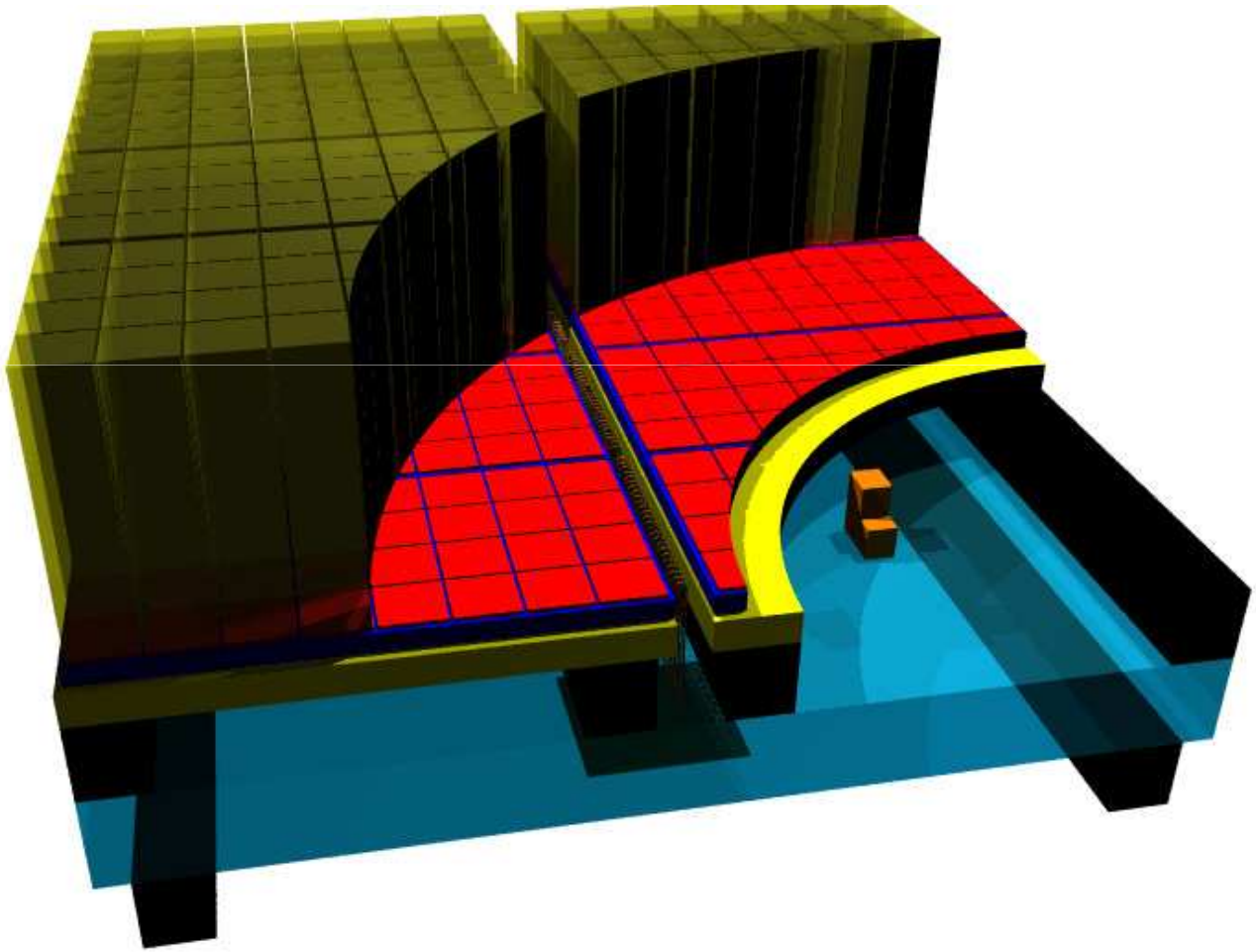
Peter Fischer, ZITI, Uni Heidelberg

What is POVRAY ?

- A software to produce 3D images
- Very easy to use (in my opinion)
  - Simple Concept to generate complex shapes
  - Can define new objects
  - Can do mathematics & calculations & loops & ...
- Can obtain very high quality
  - Based on Ray Tracing
  - Many 3D textures
  - Many illumination schemes
- Open source standard – many examples available

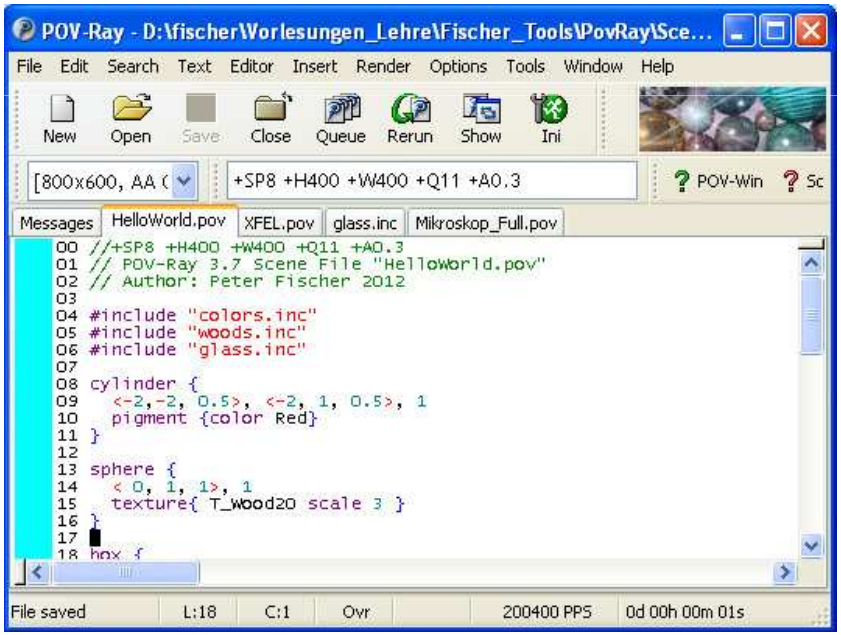
Why Use & Know About POVRAY ?

- 3D images are good to illustrate presentations or documents



Where to get POVRAY and help ?

- POVRAY is open source: www.povray.org
  - Wiki: http://wiki.povray.org/content/Main_Page
  - Tutorial: http://library.thinkquest.org/3285/tutorial/basics.html
  - Tutorial: http://www.f-lohmueller.de/pov_tut/pov__ger.htm
- Windows version comes with an integrated editor



- Linux version is a command line tool

Command Line Options

- Start for instance with

```
> povray +Ifile.pov +H400 +W400 +SP8 +Q8 +A0.3 +P
```

- Some options are:

`+Ifile.pov`	: input file
`+H400`	: image width in pixels
`+W600`	: image height in pixels
`+SP8`	: generate every 8th pixel first, then every 4th etc.
`+Q8`	: quality: 8=with reflections etc. (slower)
`+A0.3`	: anti-aliasing setting (slower)
`+P`	: pause after rendering (to admire the picture)
`-H`	: show all options

Using a Configuration File

- Options can be put into a **`par.ini`** file.
- It can contain several *sections*. Example:

```
; par.ini
; PovRay configuration file

+SP8  ; start with every 8th pixel
+Q8   ; quality is high
+A0.3 ; anti aliasing
+P    ; pause after rendering

[lo]
+W150 ; lo res image width
+H100 ; lo res image height

[hi]
+W600 ; hi res image width
+H400 ; hi res image height
```

```
> povray par[hi] +Ifile.pov
```

A First Example

Red cylinder

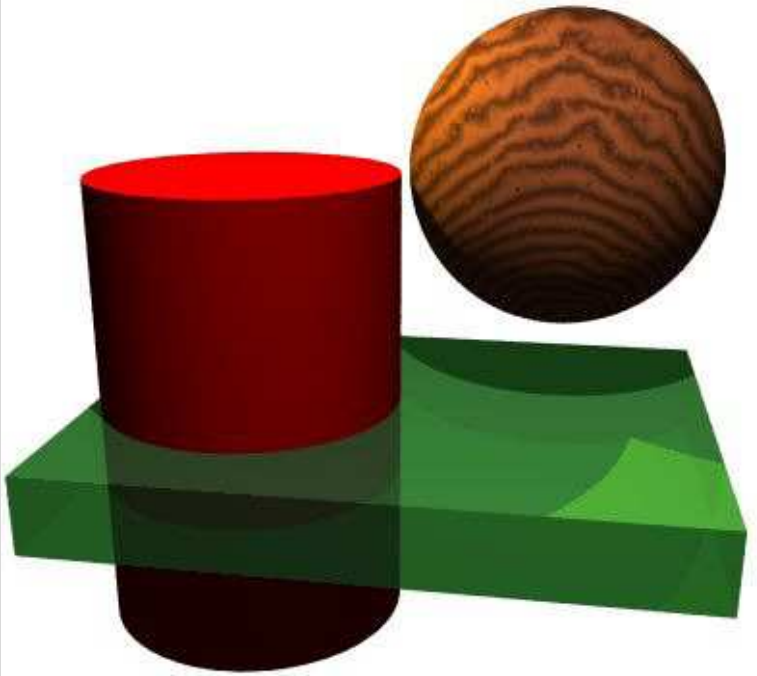
Wooden sphere

transparent box

illumination

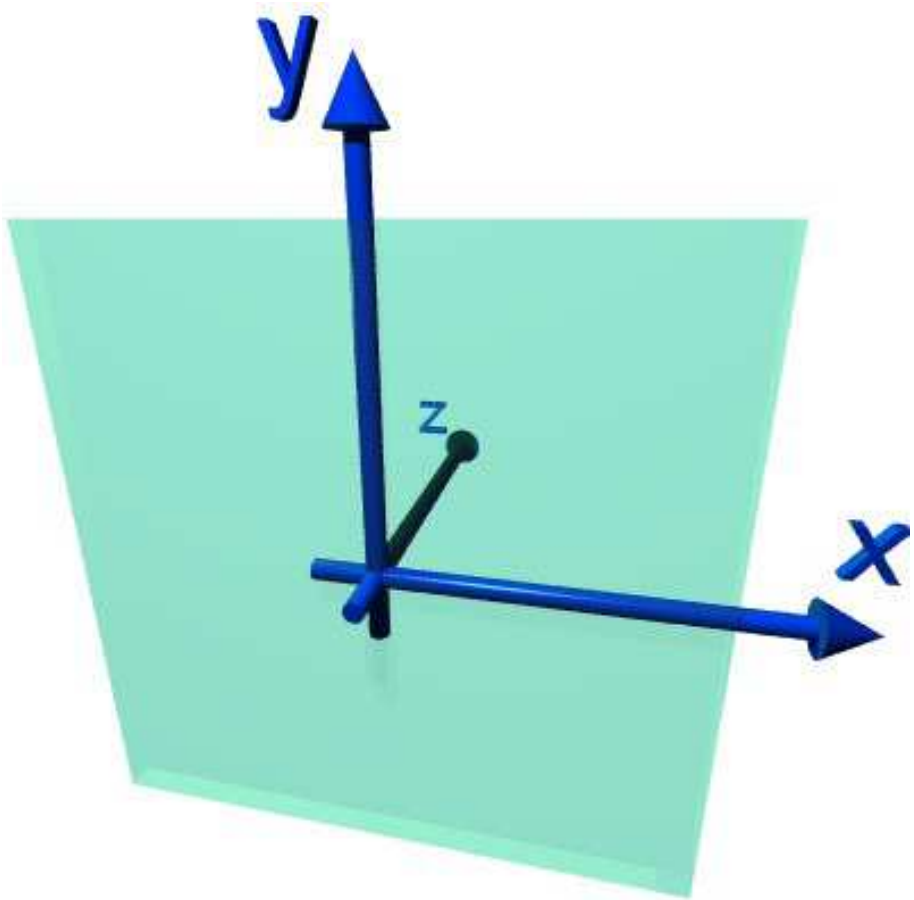
Position of camera

```
#include "colors.inc"
#include "woods.inc"
#include "glass.inc"

cylinder {
  <-2,-2,0.5>, <-2,1,0.5>, 1
  pigment {color Red}
}
sphere {
  <0,1,1>, 1
  texture{ T_Wood20 scale 3 }
}
box {
  <-3,-1,-1>, <1,-0.5, 2>
  pigment {Col_Glass_Winebottle}
}

background   { color White }
light_source {<0, 5, -3>   color White }
light_source {<-2, 2, 0.5> color Yellow }

camera {
  location <0, 2, -6>
  angle 50 right x
  look_at <-1, 0, 0>
}
```



The Coordinate System

- X and Y are like in Mathematics, Z is ‘to the back’
  - more precisely: It is a LEFT-handed coordinate system



Vectors, Directions & Rotations

- Vectors (position / direction) are given by:

```
<cx,cy,cz>
```

- They can be multiplied with numbers (floats):

```
factor * <cx,cy,cz>
```

- Predefined vectors are the 3 axes:

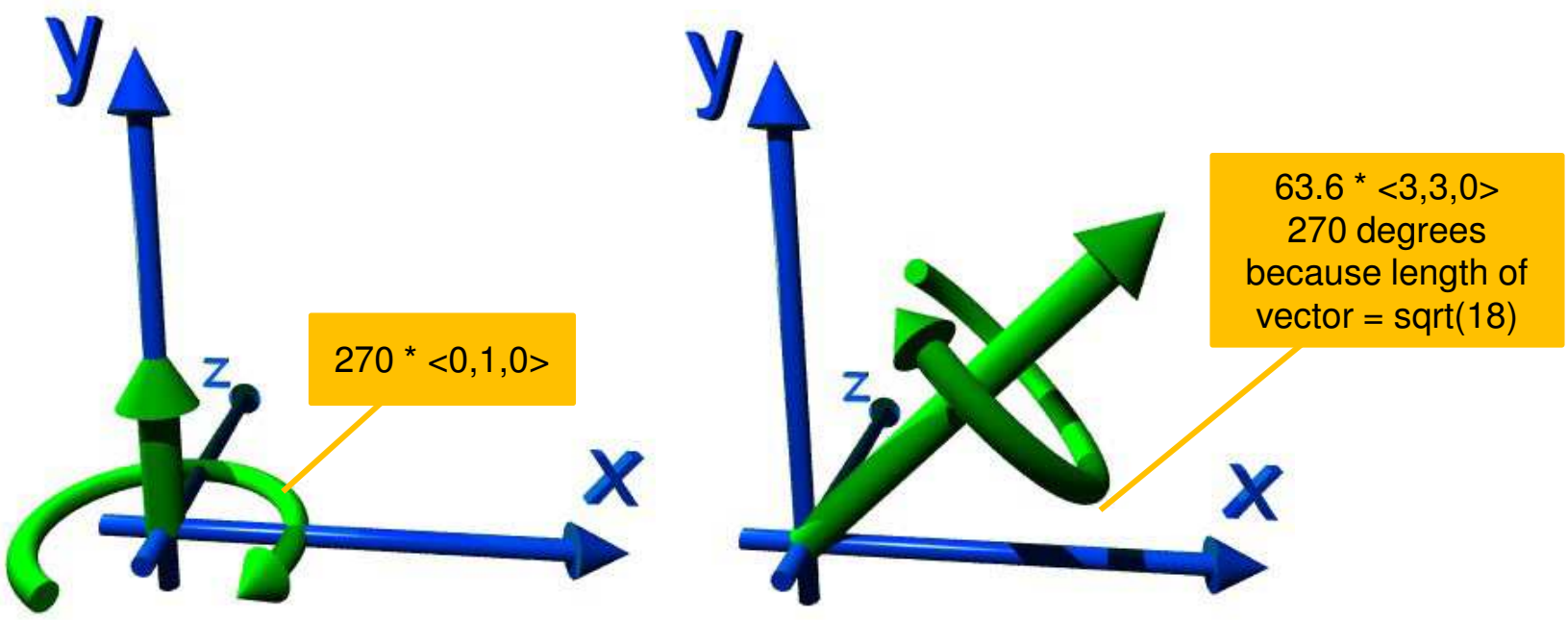
```
x   (= <1,0,0>)
y   (= <0,1,0>)
z   (= <0,0,1>)
```

- Arithmetic expressions can be used everywhere:

```
<rand(1) * 360 * sqrt(2), pi/2, log(3)>
```

Vectors as Rotation Axis

- Vectors are also used for rotations:
  - The *direction* of the vector is the *rotation axis*
  - The *length* of the vector is the *angle* (in degrees)
  - *left hand rotation sense* is used



Technicalities: Comments & Definitions

- POVRAY is *caseSensiTIve*
- Comment *lines* start with //:

```
// comment
```

- Comment *blocks* are done as in C:

```
/* ... (can be multiple lines) ... */
```

- Constants or frequently used expressions are defined by:

```
#declare name = ... ;
```

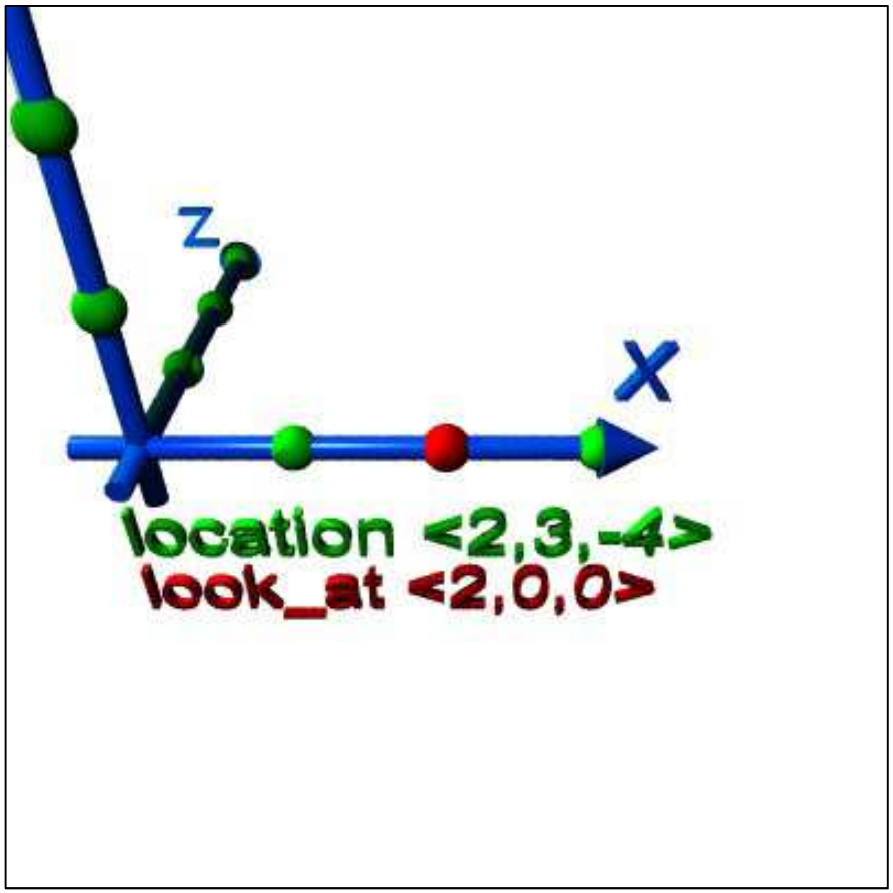
  - Note: #define does *not* work
  - (Semicolon is not needed after `'}'`)

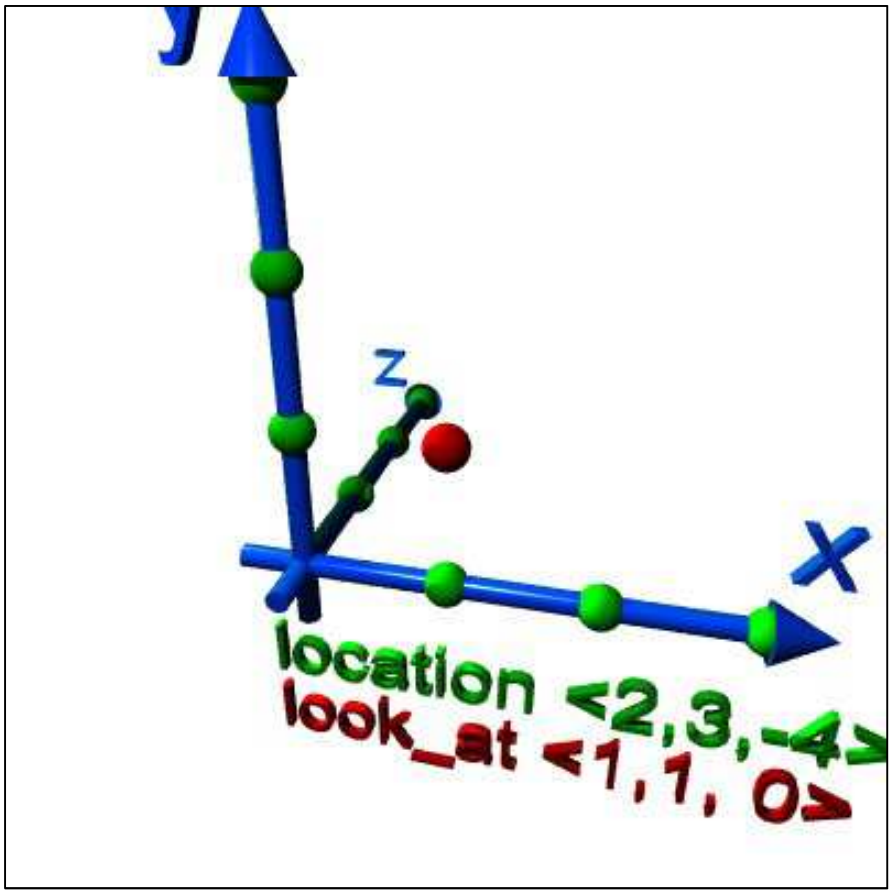
- *Parameters* can be passed by defining a 'macro':

```
#macro name (parameters,..)
  ...
#end
```

The Camera

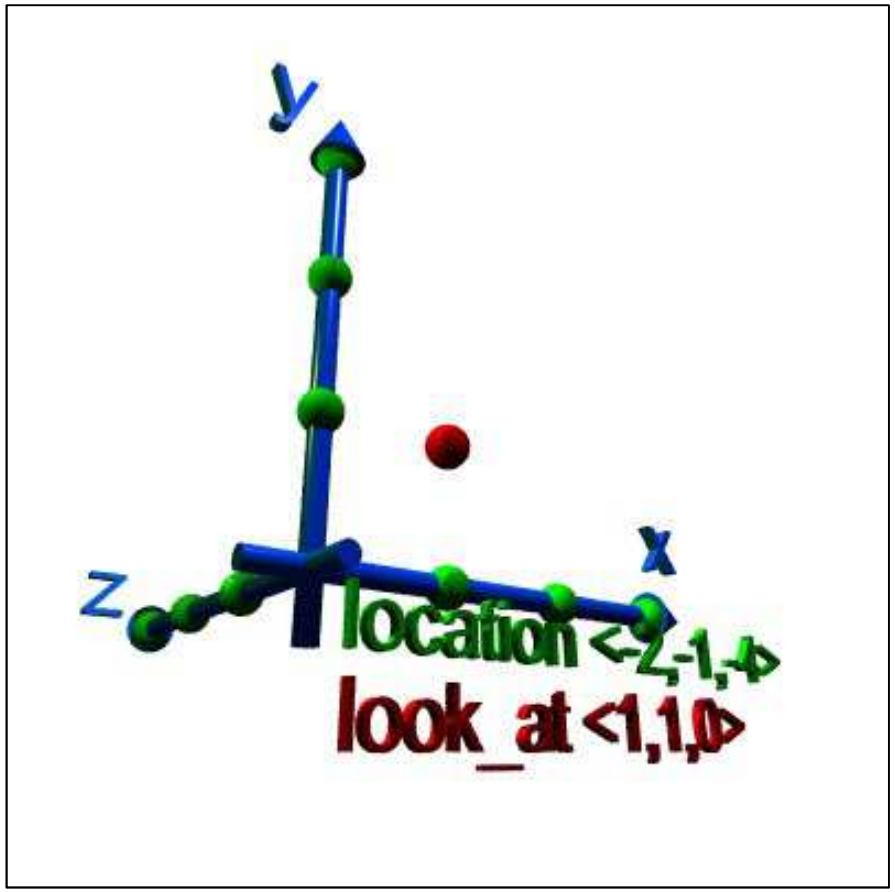
- Position: `location <point>`
- Fix viewing direction by: `look_at <point>`

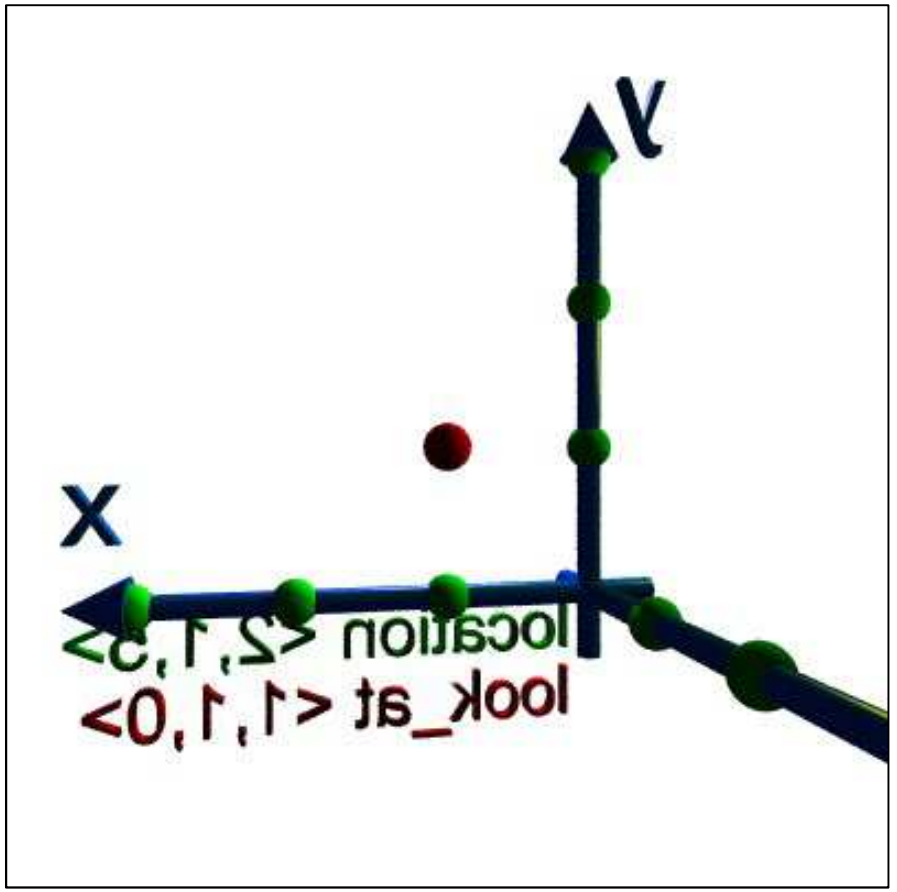




More Camera Positions

- More parameters are, for instance:
  - viewing angle: `view angle` (large number = wide angle)
  - x/y aspect ratio: `right x * W/H` (W,H = image size)





Simple Objects

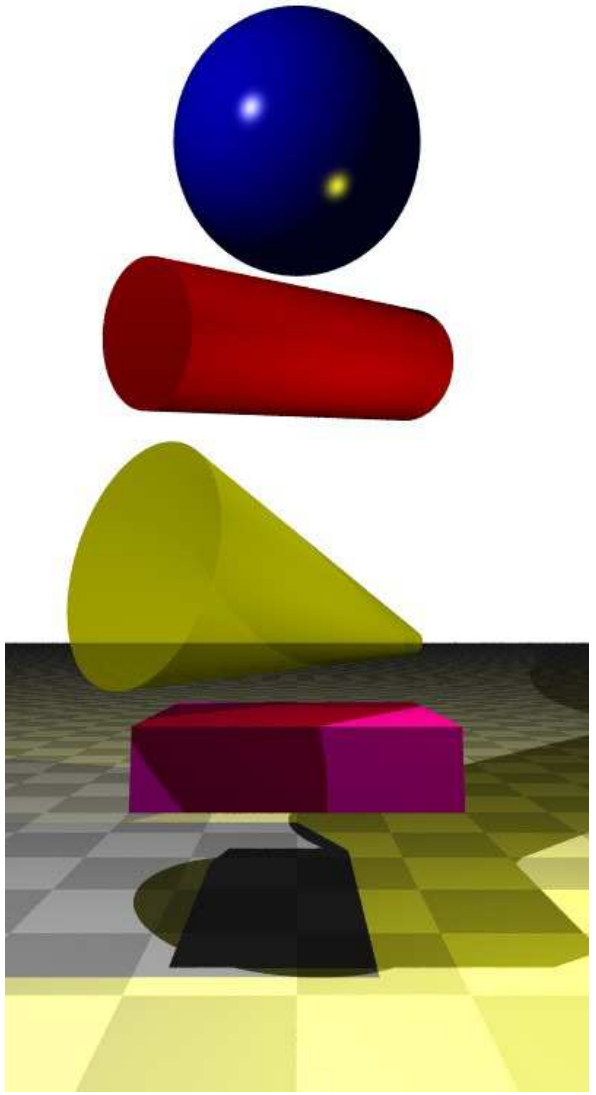
- Simple objects are:
  - `sphere { <location>, radius }`
  - `box { <corner1>, <opposite_corner> }`
  - `cylinder { <p1>, <p2>, radius }`
  - `cone { <p1>, r1, <p2>, r2 }`
  - `plane { <normal>, dist_origin }`

- They can be *coloured* using
  - `pigment {color rgb <r, g, b>}`
    or just
  - `pigment {color <r, g, b>}`

- *Transparency* can be added by a 4th parameter
  - `pigment {color rgbt <r, g, b, t>}`

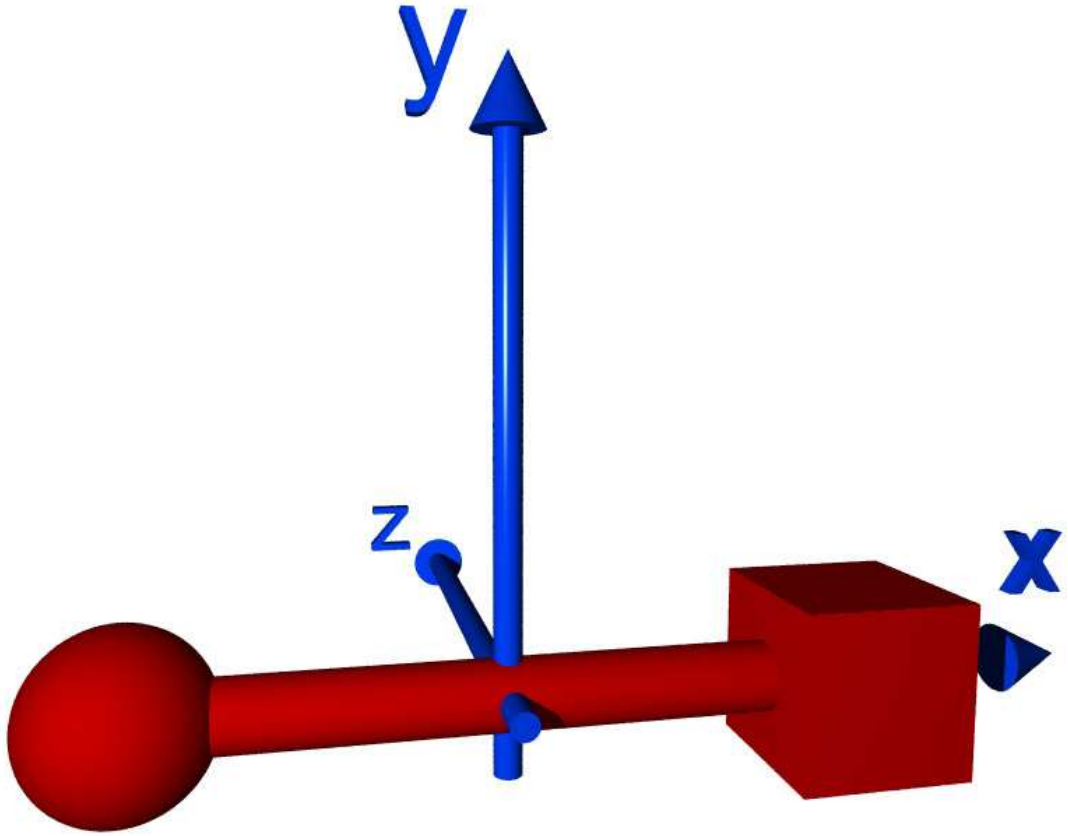
Example

```
#include "colors.inc"

plane { y,-1
  pigment {checker Gray80, White}
}
box { <-1,0,-1>, <1, 0.5, 1>
  pigment {color Magenta}
}
cone {
  <-1, 1.5, -1>, 0.8, <1, 1, 1>,  0.1
  pigment {color rgbt <1,1,0,0.5>}
}
cylinder { <-1, 3, -1>, <1, 3.5, 1>, 0.5
  pigment {color <1,0,0>}
}
sphere { < 0, 5, 0>, 1
  pigment {color Blue}
  finish { phong 0.9 phong_size 60 }
}

background   { color White }
light_source {<-2,5,-3>  color White }
light_source {<2,2,-1.5> color Yellow }

camera {
  location <0, 1, -6>
  angle 40 right x * 400 / 800
  look_at <0 2 0>
}
```



Exercise 1

- Create the following scene (without axes)
  - The centre of the sphere is at <-2,0,0>

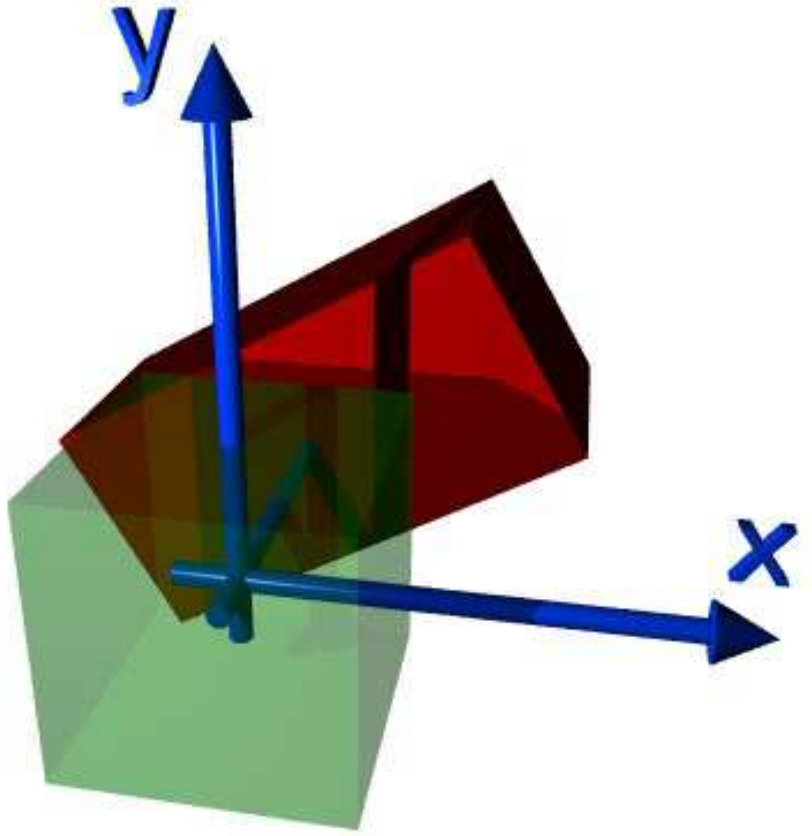


Moving and Rotating Things

- Objects can be moved / scaled & rotated:
  - **translate <shift_vector>**
  - **scale <scalex, scaley, scalez>** (or **scale val**)
  - **rotate <vector>** (vector defines direction & angle)

- Example:

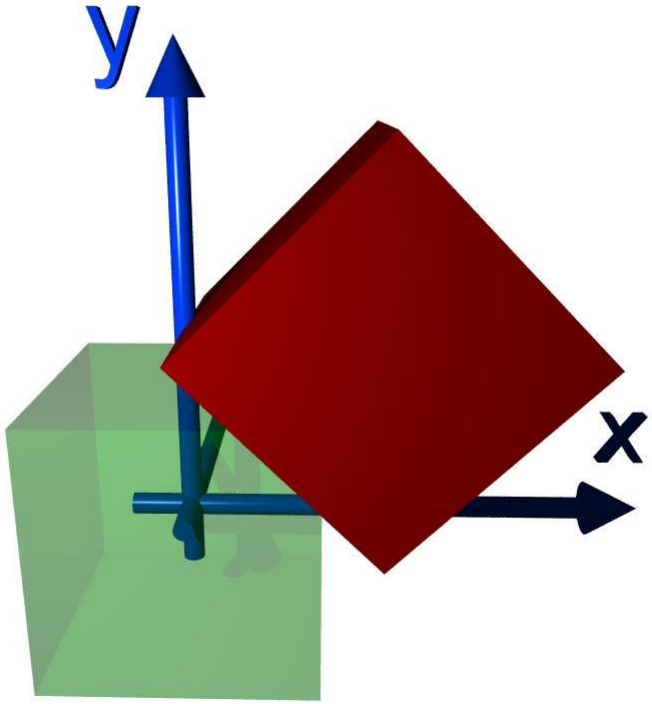
```
box {
  <-1,-1,-1>, <1,1,1>
  pigment {color Red}
  scale <2,1,1>
  rotate 30 * z
  translate 3*z
}
```

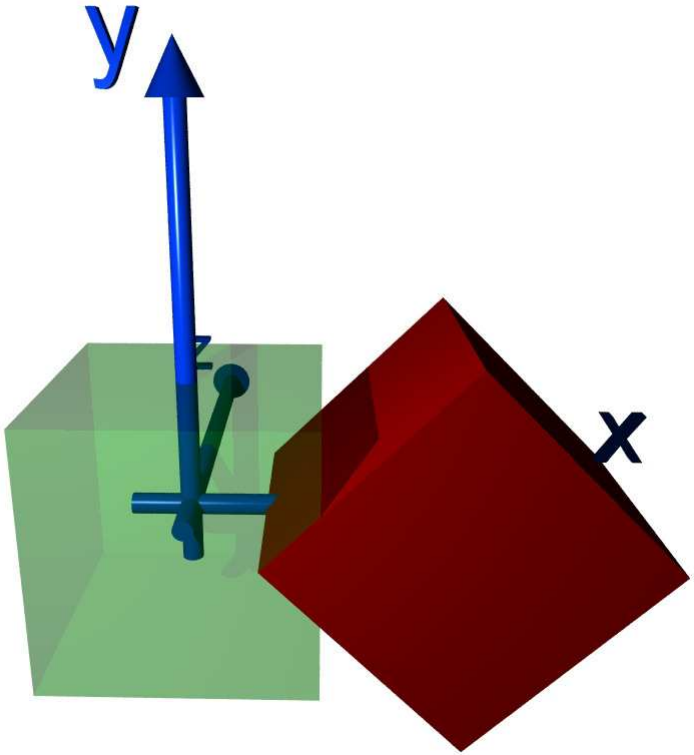


Rotations & Translations are not Commutative!

```
box {
  <-1,-1,-1>, <1,1,1>
  translate 2*x
  rotate   45*z
}
```

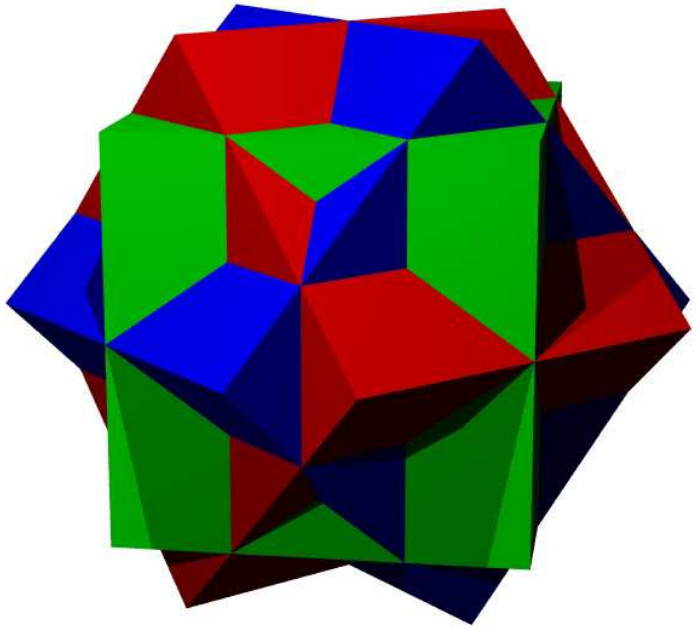
```
box {
  <-1,-1,-1>, <1,1,1>
  rotate   45*z
  translate 2*x
}
```

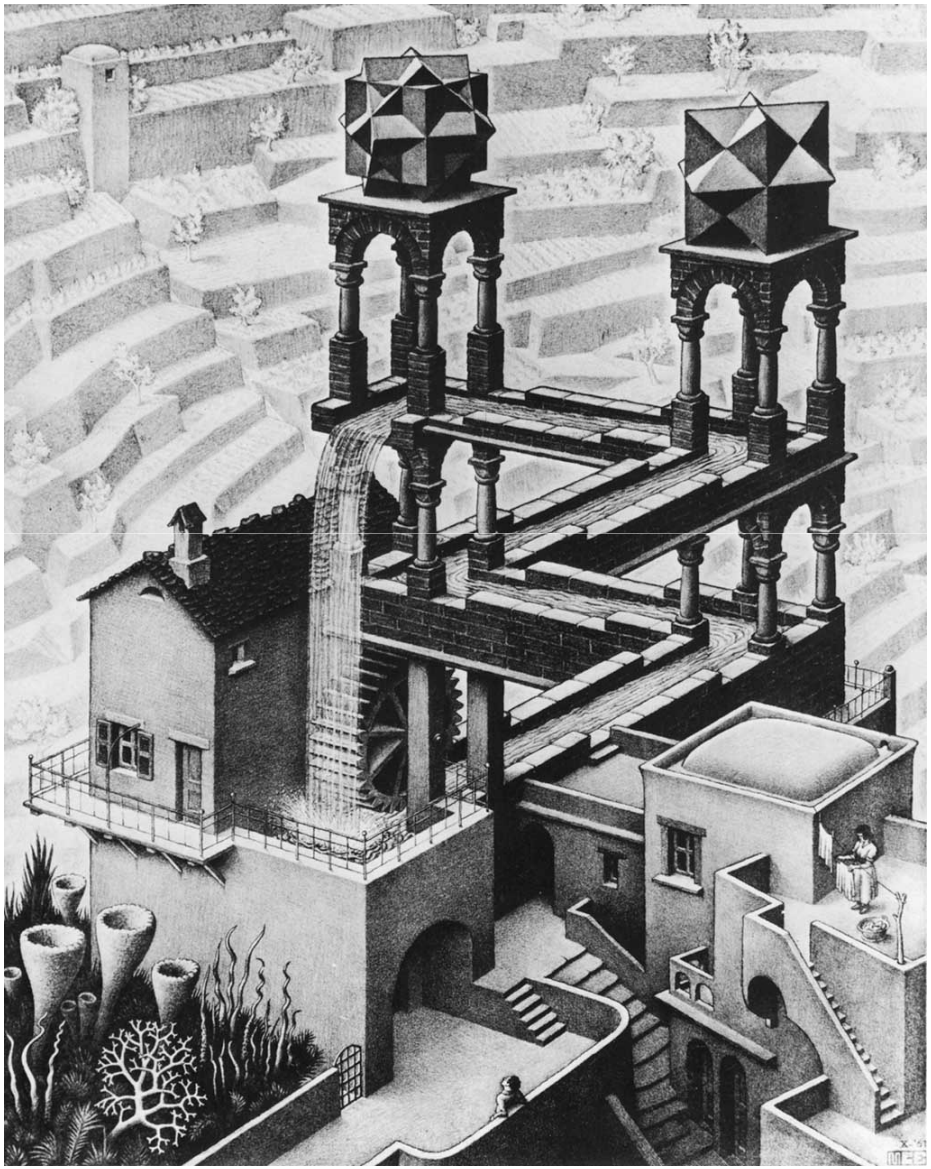




Exercise 2

- The painting ‘Waterfall’ from M.C.Escher contains two geometric figures
- Draw the left one, which consists of 3 rotated cubes!

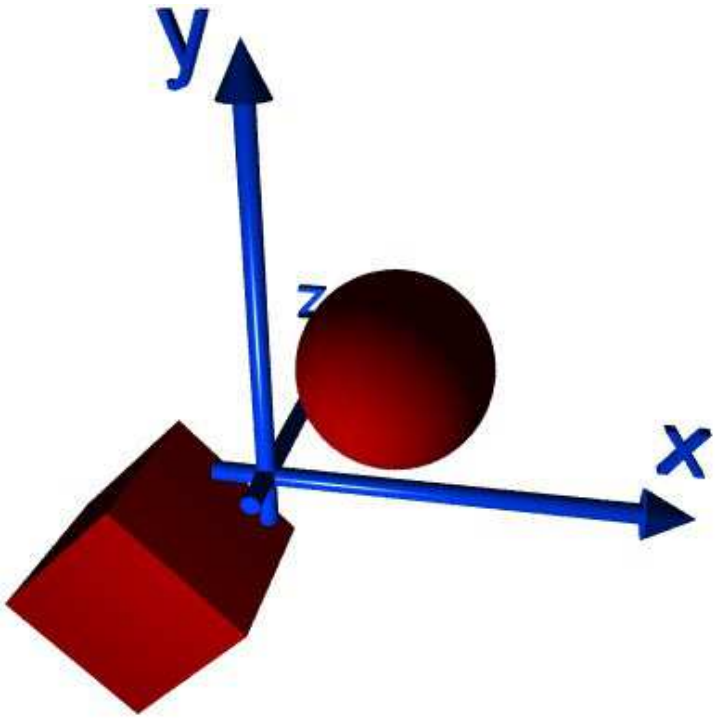




Instantiating and Merging Objects

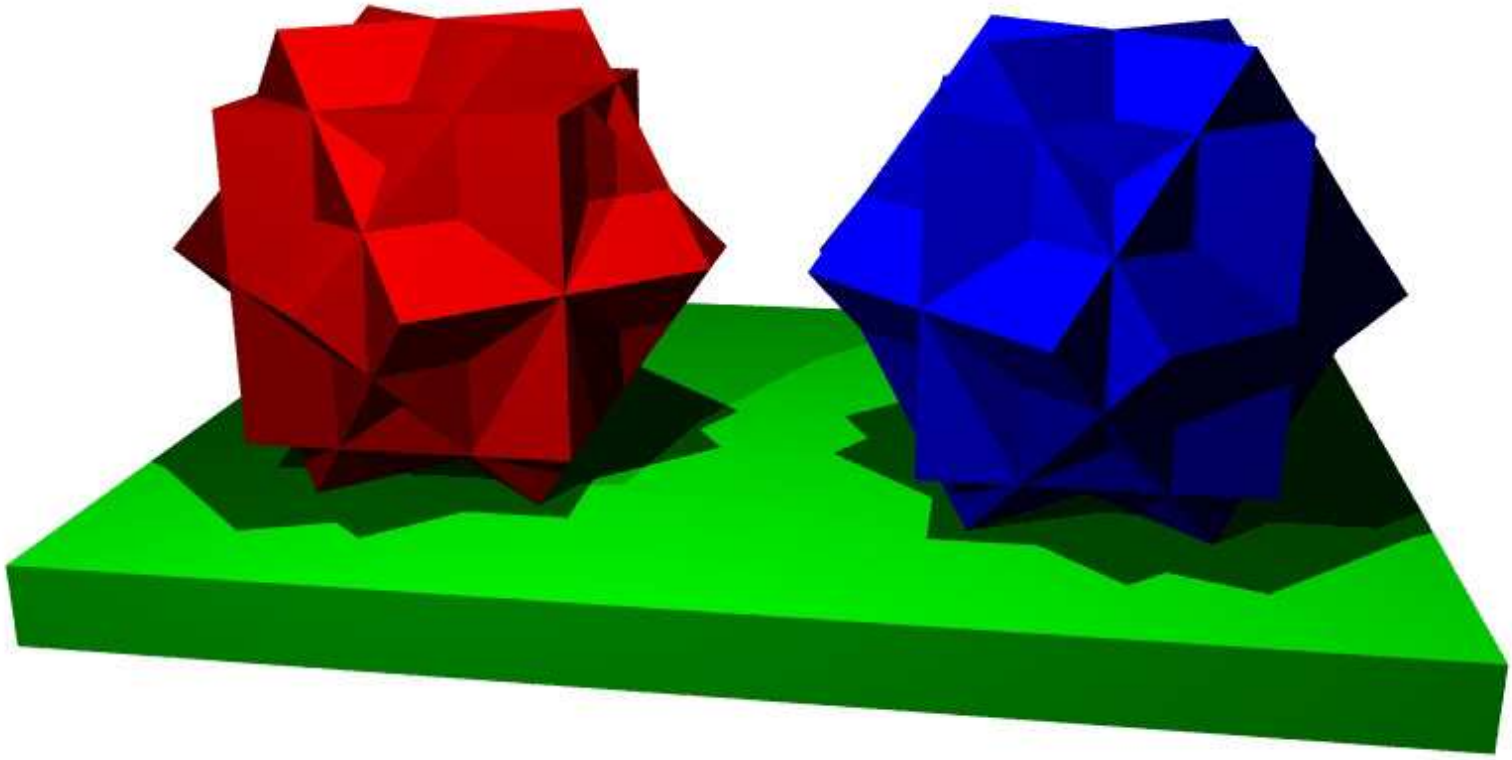
- Several Objects can be grouped with
  - **`union { objects ... transformations ... pigment..}`**
- A predefined object can be instantiated with
  - **`object{ name ... pigment ...}`**
- Example:

```
#declare H = 0.5;
#declare UNIT =
box { <-H,-H,-H>, <H,H,H> }

union {
  object {UNIT translate -x}
  sphere {x, 0.5}
  pigment {color Red}
  rotate 45 * z
  scale 1.5
}
...
```



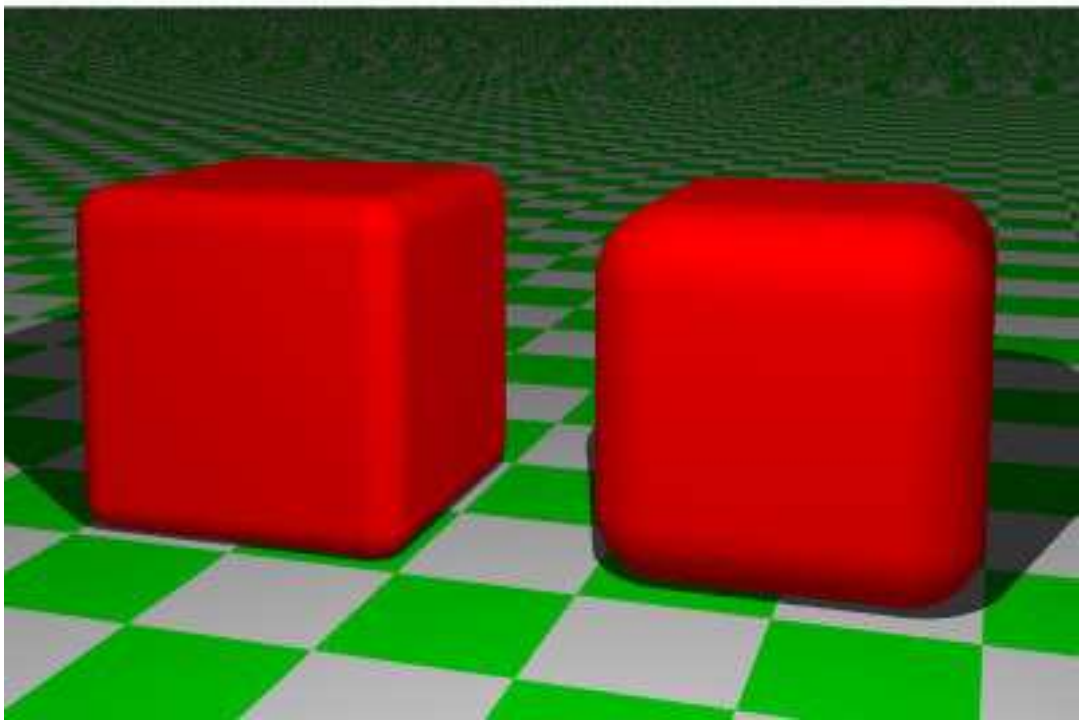
Exercise 3

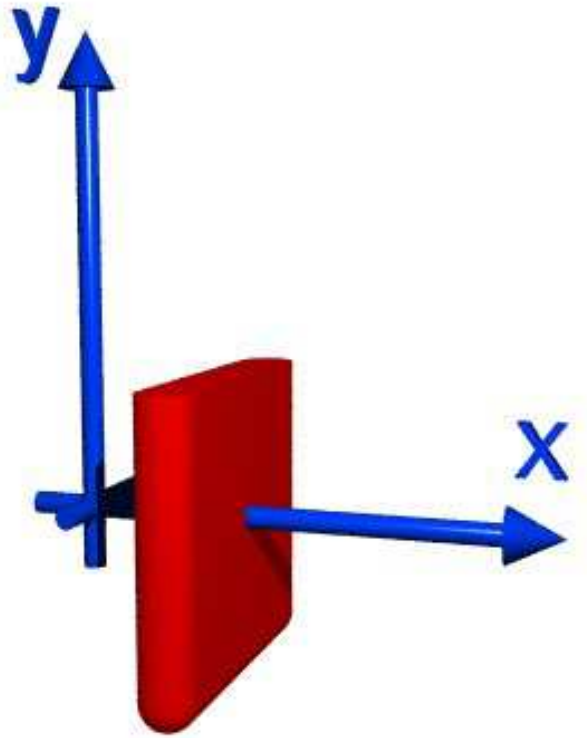
- Draw a Red and a Blue ‘Escher-Cube’ (Exercise 2) side by side on a green table



Exercise 4

- Design a rounded cube
  - Keep *inner cube size* and *corner radius* variable
  - Assemble the cube from 4 faces (as shown on the right) and two box ‘covers’



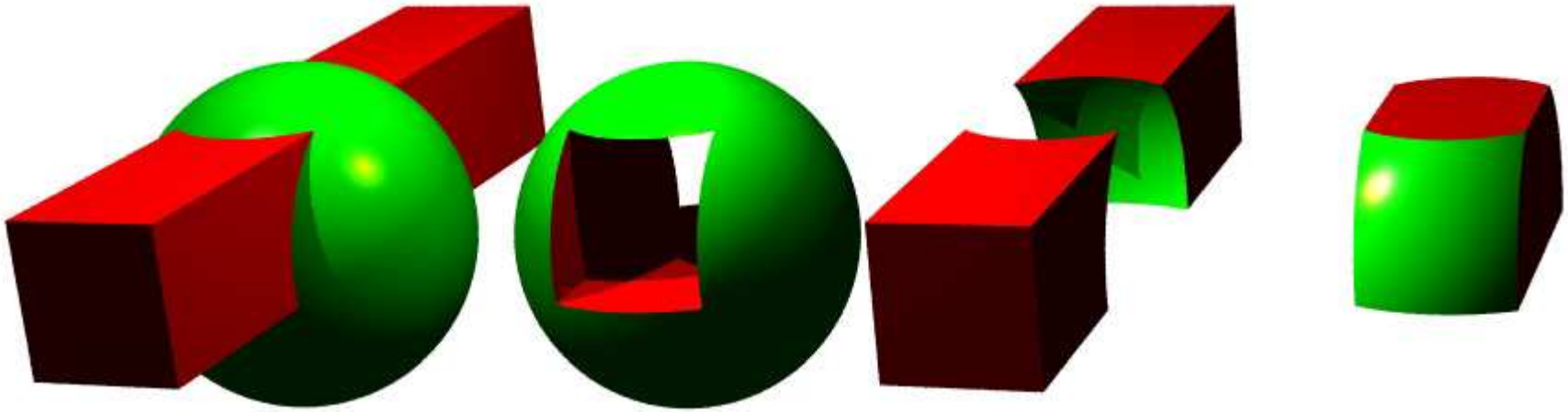


Cutting Things

- *Constructive Solid Geometry* (CSG) implemented in POVRAY allows to construct complex shapes from simple shapes.
- Commands to cut objects are:
- **`Intersection { object1 object2 }`**
  - Volume covered by *both* objects (A and B)
- **`difference { objectA objectsB }`**
  - Volume covered by A and *not* B (A and !B)

- We also have
- **`union { object1 object2 }`**
  - Volume covered by (A or B), inner faces stay
- **`merge { object1 object2 }`**
  - Volume covered by (A or B), inner faces removed

Examples for CSG



```
union {
  object {S}
  object {B}
}
```

```
difference {
  object {S}
  object {B}
}
```

```
difference {
  object {B}
  object {S}
}
```

```
intersection {
  object {B}
  object {S}
}
```

Textures

- Textures define the surface & *volume* appearance
- They consist of, for instance

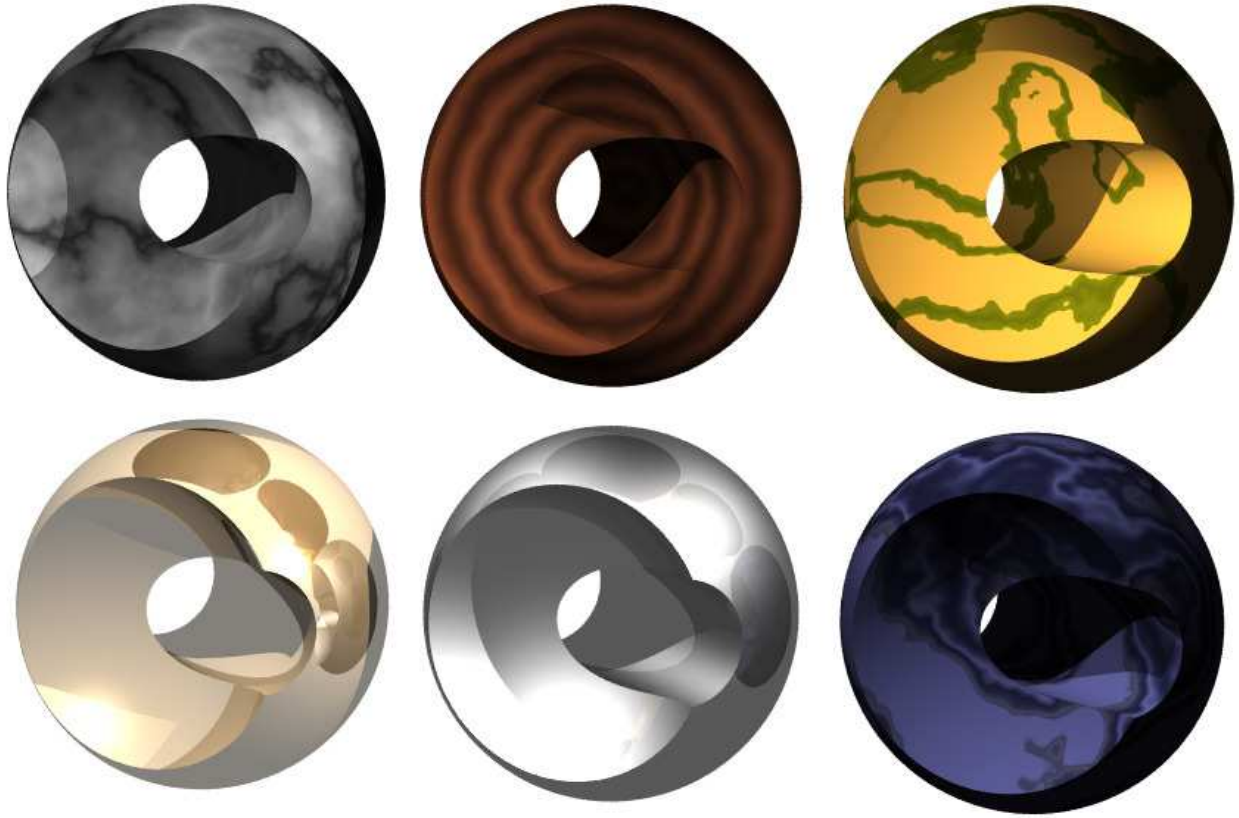
```
texture {
  pigment {color ...}  // color, transparency
  finish {
    ambient 0...1      // emitted light
    diffuse 0...1      // % of light reflected in a diffuse way
    reflection 0...1   // % of light reflected in a specular way
    phong  0...1       // intensity of highlights
    ...                // several more
  }
  normal {bumps 2 scale 0.5}  // surface roughness
}
```

Predefined Texture

- Many textures are available in the include files

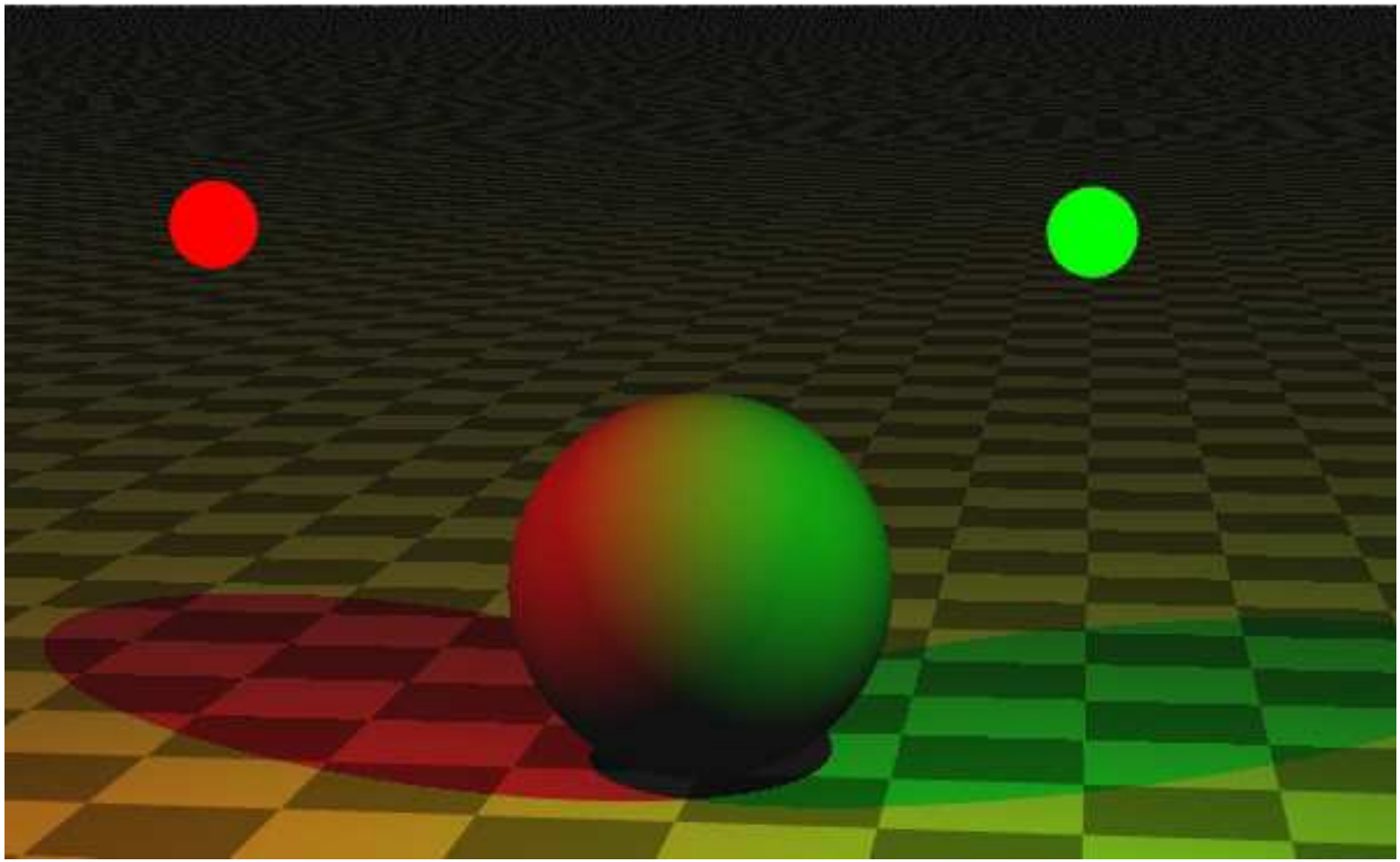
```
#include "stones.inc"
#include "woods.inc"   etc.
```

- For an overview, see for instance http://texlib.povray.org



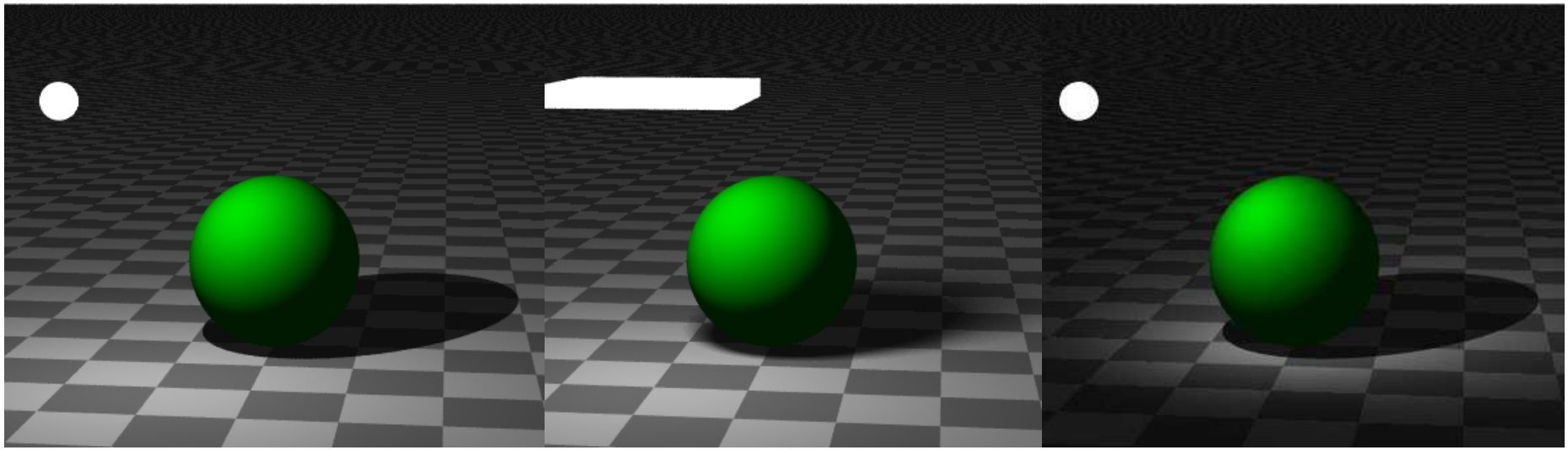
Light Sources

- Light Sources can have color



Light Sources

- They can be *point sources*, *area sources* or *spotlights*



```
Light_source {
  <position>
  color ...
}
```

```
Light_source {
  ...
  area_light
  ...
}
```

```
Light_source {
  ...
  spotlight
  ...
}
```

Programming

- 2 examples:

- Conditional blocks:

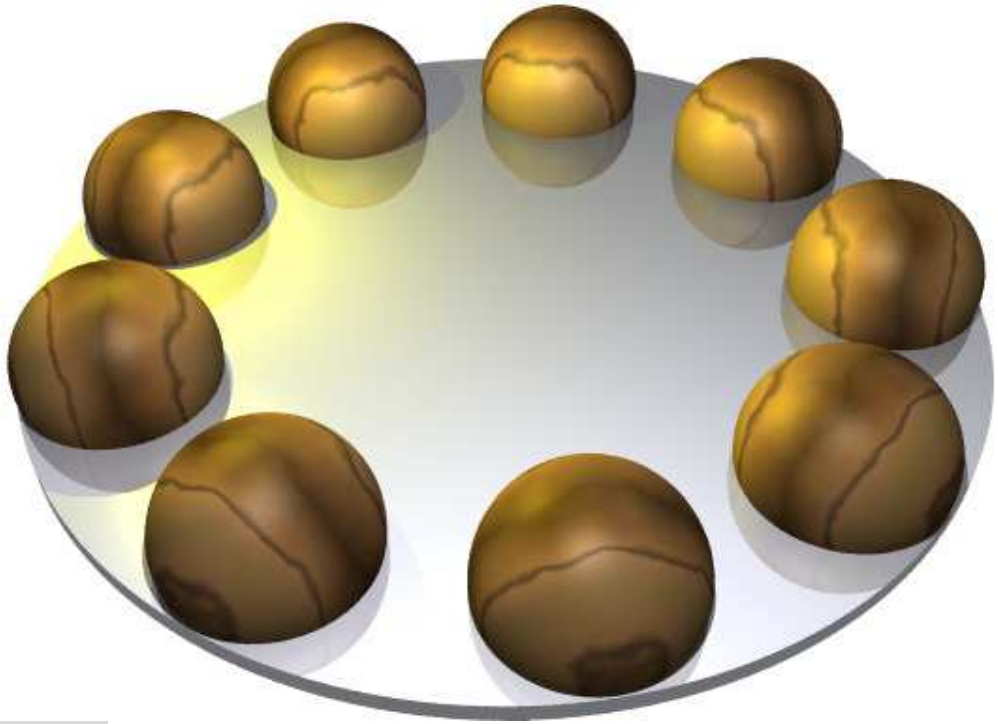
```
#if (VERSION=1)  // only one '='
  ...
#end
```

- Loops

```
#declare angle = 0;
#while(angle < 360)
  ...
  #declare angle = angle + 60;
#end
```

Programming Example

```
#include "textures.inc"

cylinder {-0.2*y, -0.1*y, 2.5
  texture {Silver_Metal}
}
#declare S = sphere {2*x, 0.5
  texture {EMBWood1}
}

#declare phi = 0;
#while(phi < 360)
  object { S rotate phi * y}
  #declare phi = phi + 40;
#end

background   { color White }
light_source {<0, 5, -3> color White }
light_source {<-2,2,0.5> color Yellow }

camera {
  location <0, 4, -5>
  look_at  <0,-0.5,0>
  angle 50 right 6/4*x
}
```



Animations

- Several renderings can be done in a batch job
- The value of `clock` is incremented in each frame from a start to an end value in predefined steps
- The sequence of images can be merges to a movie

- See Demo, or for instance
  - http://www.alzinger.de/cms5/robert/raytracing/raytracing-video.html
  - http://www.alzinger.de/cms5/robert/raytracing/marble-machine-in-povray.html

VRML

- VRML = Virtual Reality Modeling Language

- Is a 3D Scene description language similar to POVRAY
- Can be rendered *in real time* using browser plugins
  - for instance Cortona3D Viewer
- User interaction is possible

- Much less powerful, but interactive

- VRML has not really become a standard.
  Plugins installation still unusual