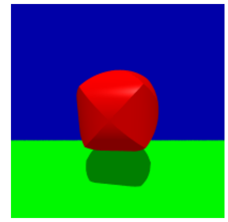
A SCENE. The y axes is up. We place the object (the intersection of three cylinders) at the origin and the camera slightly above on the z axes.

```
camera{ up y right x location <0,2,-5> look_at <0,0,0> }
light_source { <0,300,-100> colour rgb <1,1,1> }
#background { rgb <1,1,1> }

#macro r(c)
  pigment { rgb c }
  finish { phong 1.0 ambient 0.5 diffuse 0.5 }
#end

intersection {
  cylinder { <-1,0,0>,<1,0,0>, 1 }
  cylinder { <0,-1,0>,<0,1,0>, 1 }
  cylinder { <0,0,-1>,<0,0,1>, 1 }
  texture { r(<1,0,0>) }
  rotate <10,10,5>
}

plane {<0,1,0>,-2 texture { r(<0,1,0>) } }
plane {<0,0,1>, 2 texture { r(<0,0,1>) } }
```



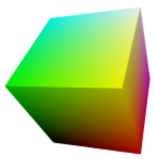
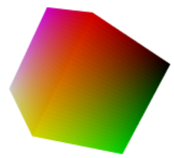
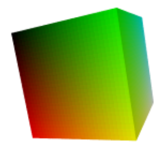
RUNNING POVRAY. If the povray file is called test.pov, run povray as "povray good.ini +i test.pov". The .ini file contains instructions like size of the picture, quality of rendering etc. With "povray anim.ini +i test.pov", a sequence of pictures is rendered. The .ini file contains now instructions how many frames the movie should have.

```
Width=200
Height=200
Quality=11
Antialias_Depth=4
Antialias=On
Antialias_Threshold=0.1
Jitter_Amount=0.5
Jitter=On
```

```
Width=200
Height=200
Quality=11
Antialias_Depth=4
Antialias=On
Antialias_Threshold=0.1
Jitter_Amount=0.5
Jitter=on
Initial_Frame = 1
Final_Frame   = 10
Initial_Clock = 0.0
Final_Clock   = 1.0
```

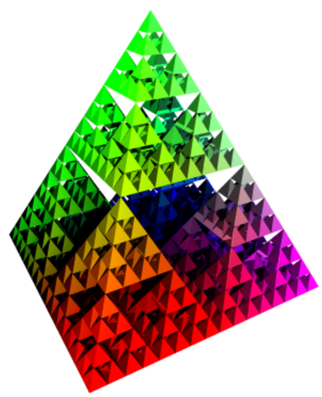
WHAT IS RAYTRACING? The objects, the camera and lights are in place. If photons leave the light source, they will reflect at the objects, change color and intensity and some of them will reach the camera. Adding up the light intensities of all these photons gives the picture. Raytracers work more efficient: instead of shooting photons from the light source and wasting most of the photos because they don't reach the film, it is better to start the light ray at the film and compute backwards.

COLORS. Every color is given by a vector $< r, g, b >$ in the color cube $[0,1] \times [0,1] \times [0,1]$. Color vectors can be added like usual vectors.

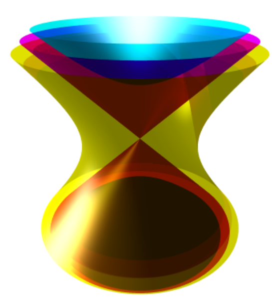
PROGRAMMING. Povray is a programming language. Macros are procedures which can be reused and called within the the macro. This allows recursion like in this rendering of the 3D fractal to the right. You see also how one can play with colors. Identifying them with vectors in space colors the pyramid according to the color cube.

```
camera {location <1,1.2,-1.7> up y right x look_at <0,0,0> }
light_source {< 100, 80,  100> colour rgb <1,1,1> }
light_source {< 100, 80, -100> colour rgb <1,1,1> }
light_source {< 100, 80, -100> colour rgb <1,1,1> }
light_source {<-100, 80, -100> colour rgb <1,1,1> }
background   { rgb <1,1,1> }

#macro te(c)
  texture { pigment { rgb c }
  finish  { phong 1.0 phong_size 10 ambient 0.1 diffuse 0.7 } }
#end

#declare maxrecursion=4;
#macro shirpinski(p,w,n)
   #local hw=w/2;   #local fw=w/4;
   #if (n>=maxrecursion)
      union {
      triangle { p,p+< hw,-w,-hw>,p+< hw,-w, hw> }
      triangle { p,p+< hw,-w, hw>,p+<-hw,-w, hw> }
      triangle { p,p+<-hw,-w, hw>,p+<-hw,-w,-hw> }
      triangle { p,p+<-hw,-w,-hw>,p+< hw,-w,-hw> }
      triangle { p+< hw,-w, hw>,p+<-hw,-w, hw>,p+<-hw,-w,-hw> }
      triangle { p+<-hw,-w,-hw>,p+< hw,-w,-hw>,p+< hw,-w, hw> }
      te(p) }
   #else
      shirpinski(p,hw,n+1)
      shirpinski(p+< fw,-hw,-fw>,hw,n+1)
      shirpinski(p+< fw,-hw, fw>,hw,n+1)
      shirpinski(p+<-fw,-hw, fw>,hw,n+1)
      shirpinski(p+<-fw,-hw,-fw>,hw,n+1)
   #end
#end

union { shirpinski(<0,1,0>,1.3,0)
  translate -y*0.5 rotate y*30 translate y*0.5
}
```



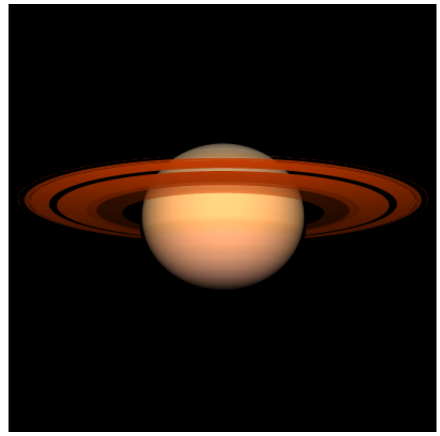
OBJECTS. Many objects like quadrics $ax^2 + bxy + cy^2 + dx + ey + fz + g = 0$ are predefined in Povray. The most basic objects are boxes, spheres, cylinders, lathes (extruded curves). By joining, intersecting or taking differences of such objects, one can build more complicated objects. Additionally, every object can be rotated and scaled.

```
camera         { location  <1,2,-4> up  <0, 1,0> right <1,0,0> look_at <
light_source   { <-100, 30, -10>    color rgb <1,1,1>
light_source   { <0, 2000, 0>  color rgb <1,1,1>  area_light <-6,0,-6>, 
background     { rgb <1.0,1.0,1.0>

#macro te(c) texture { pigment { rgb c} finish { phong 1.0 phong_size 10

poly {2,<1,0,0,0,-1,0,0,1,0,-0.3> scale 1.5  hollow  clipped_by { sphere
poly {2,<1,0,0,0,-1,0,0,1,0, 0.3> scale 1.5  hollow  clipped_by { sphere
poly {2,<1,0,0,0,-1,0,0,1,0, 0.0> scale 1.5  hollow  clipped_by { sphere
```



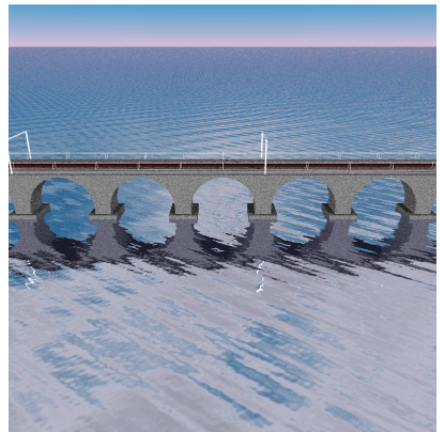
TEXTURES. Textures to planets of the Solar system are available on the web. The texture is mapped onto the sphere using the map $X(u,v) = (\cos(u)\sin(v), \sin(u)\sin(v), \cos(v))$. In the source code, the units are chosen so that 1=1km. Having natural units allows to model things more easily. In the scene an artificial light has been added from the camera in order to see the rings better.

```
camera { location <0,50000,-320000> up z right x look_at 0 }
#declare sun= light_source {
  0 color rgb <1,1,1> looks_like { sphere { 1, 696265
  pigment { color rgb <1,1,0> } finish { ambient 1} }}
  translate -z*142600000
}
light_source {<0,20000,-250000> color rgb <0.3,0.1,0>}
#declare saturn_planet = sphere {0, 60368
  pigment {image_map {gif "saturn.gif" map_type 1 interpolate 2}}
  finish  {ambient 0 diffuse 1}
}
#declare C_ring = disc { <0, 0, 0>  y,       92000, 74500
  pigment {color rgb<0.99,0.88,0.79> filter 0.6}
  finish {ambient 0 diffuse 5}
}
#declare inner_B_ring = disc { <0, 0, 0>  y, 98390, 92000
  pigment {color rgb<0.99,0.88,0.79> filter 0.2}
  finish {ambient 0 diffuse 5}
}
#declare outer_B_ring = disc { <0, 0, 0> y,  117500, 98390
   pigment {color rgb<0.99,0.88,0.79> filter 0.05}
   finish {ambient 0 diffuse 5}
}
#declare inner_A_ring = disc { <0, 0, 0>  y, 133570, 122200
   pigment {color rgb<0.99,0.88,0.79> filter 0.05}
   finish {ambient 0 diffuse 5}
}
#declare outer_A_ring = disc { <0, 0, 0>  y, 136800, 133895
   pigment {color rgb<0.99,0.88,0.79> filter 0.2}
   finish {ambient 0 diffuse 5}
}
#declare F_ring = disc { <0, 0, 0> y,        140460, 140210
   pigment {color rgb<0.99,0.88,0.79> filter 0.05}
   finish {ambient 0 diffuse 5}
}
union {
   object {sun}          object {saturn_planet scale <1,0.9,1>}
   object {C_ring     }  object {inner_B_ring}
   object {outer_B_ring}  object {inner_A_ring}
   object {outer_A_ring}  object {F_ring}
   rotate x*20
}
```



WATER AND SKY. Water and sky are frequenly added, often by "artistic" reasons.

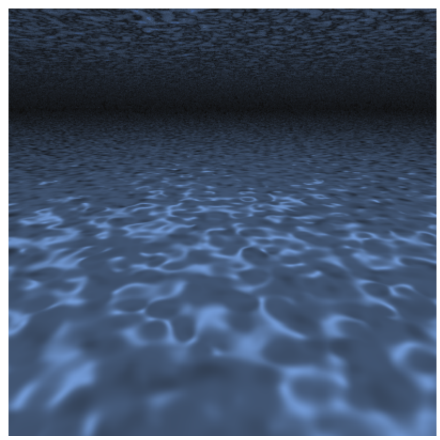
```
#include "colors.inc"
#include "skies.inc"
#include "sbb.inc"

light_source { <-200, 10, -200> color rgb <1,1,1>  }
light_source { < 200,100, -200> color rgb <1,1,1>  }

#declare p=<0,40,-100>;
camera { location p up y right x look_at 0 rotate y*360*clock}

#declare WaterTexture = texture {
    pigment { color rgb<0.39, 0.41, 0.83> }
    finish  { ambient 0.15 brilliance 5 diffuse 0.6 specular 0.80
              roughness 1/100 reflection 0.65 }
}

#declare Water = plane { y, 0.0
   texture { WaterTexture
      normal { waves 0.05 frequency 5000.0 scale 3000.0 }
   }
}

object{ SBBBruecke translate -x*100}
object{ Water }
sky_sphere { S_Cloud2 }
```



REFRACTION. When light passes from one medium to an other, the path of the ray of light is bent because the speed in different media is different and the path choses the path traversed in shortest time. This is called **refraction**. For example air and water have different densities and thus refract differently.

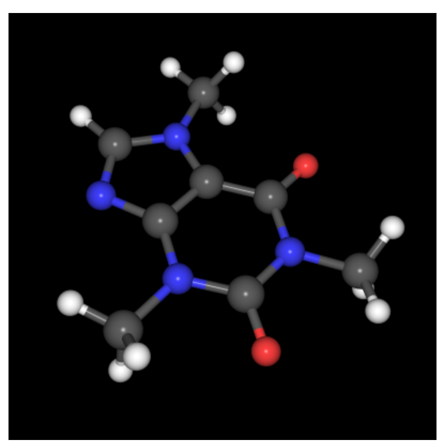
```
light_source { <0, 50, 0> color rgb <1,1,1>              }
camera { direction z location <0,6,-15> look_at <0,2,0> }

plane { y,0
  pigment { rgb <0.6,0.6,0.6> }
  finish { ambient 0.1 diffuse 0.7 }
  normal { bumps 0.8 }
}

plane { y, 12 hollow on
  pigment { rgb <0.5,0.7,1.0> filter 0.93 }
  finish  { reflection 0.7 }
  interior { ior 1.33 caustics 1.0 }
  translate <5, 0, -10>
  normal { bumps 0.8 }
}
```

The index of refraction **ior value** is used to describe the relative density of substances. The default ior value 1.0 (air) gives no refraction. Water has ior 1.33, glass is 1.5 and diamond 2.4. In the picture to the right, the camera is under water, the light source above the water. You see caustics like at the bottom of a swimming pool.

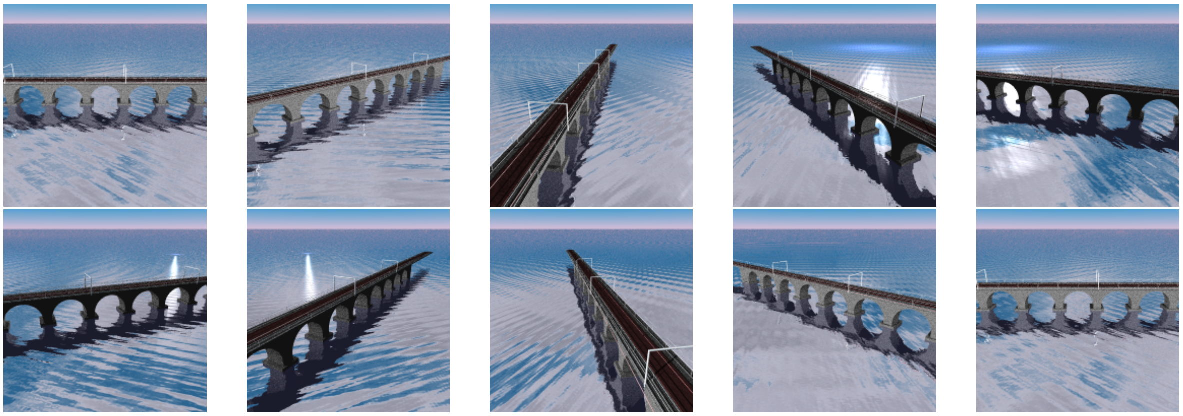


EXTERIOR PROGRAMS. Many organic or an-organic molecules are available on the web in .pdb format. There are viewers which allow to see the molecule and translators, which export a povray file, which can then be rendered. To the right, you see the caffeine molecule.

```
COMPND    caffeine
AUTHOR    Created by Dave Woodcock at Okanagan University College
AUTHOR    email:woodcock@okanagan.bc.ca
AUTHOR    Date revised: Fri Sep 29 14:53:27 2000  GENERATED BY BABEL 1.6
HETATM    1  C           1       0.000   0.000   0.000  1.00  0.00
HETATM    2  C           1       1.392   0.000   0.000  1.00  0.00
HETATM    3  N           1       2.076   1.164   0.000  1.00  0.00
HETATM    4  C           1       1.373   2.321  -0.003  1.00  0.00
HETATM    5  O           1       1.978   3.365  -0.017  1.00  0.00
HETATM    6  N           1       0.017   2.344   0.003  1.00  0.00
HETATM    7  C           1      -0.710   1.202   0.002  1.00  0.00
HETATM    8  O           1      -1.915   1.218  -0.006  1.00  0.00
HETATM    9  N           1      -0.404  -1.287  -0.019  1.00  0.00
HETATM   10  N           1       1.830  -1.279  -0.020  1.00  0.00
HETATM   11  C           1       0.715  -2.048  -0.031  1.00  0.00
HETATM   12  C           1      -1.795  -1.761  -0.044  1.00  0.00
HETATM   13  C           1       3.546   1.178  -0.016  1.00  0.00
HETATM   14  C           1      -0.690   3.634  -0.013  1.00  0.00
HETATM   15  H           1       0.720  -3.138  -0.055  1.00  0.00
HETATM   16  H           1      -1.813  -2.850  -0.090  1.00  0.00
HETATM   17  H           1      -2.307  -1.428   0.860  1.00  0.00
HETATM   18  H           1      -2.302  -1.352  -0.918  1.00  0.00
HETATM   19  H           1       3.894   1.455  -1.011  1.00  0.00
HETATM   20  H           1       3.929   0.190   0.239  1.00  0.00
HETATM   21  H           1       3.911   1.904   0.710  1.00  0.00
HETATM   22  H           1      -1.557   3.583   0.645  1.00  0.00
HETATM   23  H           1      -0.027   4.428   0.329  1.00  0.00
HETATM   24  H           1      -1.020   3.851  -1.029  1.00  0.00
CONECT    1    2    2    7    9
CONECT    2    1    1    3   10
CONECT    3    2    4   13
CONECT    4    3    5    5    6
CONECT    5    4    4
CONECT    6    4    7   14
CONECT    7    1    6    8    8
CONECT    8    7    7
CONECT    9    1   11   12
CONECT   10    2   11   11
CONECT   11    9   10   10   15
CONECT   12    9   16   17   18
CONECT   13    3   19   20   21
CONECT   14    6   22   23   24
CONECT   15   11
CONECT   16   12
CONECT   17   12
CONECT   18   12
CONECT   19   13
CONECT   20   13
CONECT   21   13
CONECT   22   14
CONECT   23   14
CONECT   24   14
MASTER         0    0    0    0    0    0    0    0   24    0   24    0
END
```



ANIMATION. When doing animations, the camera moves along a curve. The velocity and acceleration vector can help to determine, in which direction the camera should look. The frames below were made with a bridge built after the Rheinfall bridge in Switzerland.



MODELS. There are converters to produce povray files from other formats. For example, models created with Poser can be transformed into a form readable by Povray.

```
camera { location <120,11,70> up y right x direction <0,0,7> look_at <30,18,16>}

light_source { <-213, 157, -10> color rgb<1,1,1> }
light_source { <210, 157, -160> color rgb<0.5,0.5,0.5> }
light_source { <-538, 180,481> color rgb<1,1,1> }
light_source { <180, 74, -102> color rgb<1,1,1> }

#include "liberty.inc"
object {statue_of_liberty texture { pigment { color rgb<1,1,1> }
  finish { ambient 0.3 diffuse 0.4 phong 0.7 reflection 0 phong_size 32 } }}
```

The converted file is usually included as a .inc file. The beginning of the file liberty.inc (obtained from the web at http://www.multimania.com/froux/modeles) is displayed blow. There are many models available on the web (good and less good).

```
#declare statue_of_liberty =
union{
mesh{
 smooth_triangle{<1.98,27.1,1.5><9.77,-1.99,-0.8><1.97,27.1,1.02><9.82,-1.3,1.4><1.93,27.34,1.1><9.55,1.41,-2.61>}
 smooth_triangle{<2.1,27.11,1.46><-6.71,1.7,-7.21><2.1,27.49,1.53><-6.33,0.1,-7.74><2.21,27.33,1.36><-7.91,1.6,-5.91>}
 smooth_triangle{<2.02,25.82,1.19><2.3,-4.59,-8.58><2.01,25.82,1.19><-0.5,-4.63,-8.85><2.01,25.86,1.17><0,-5.1,-8.6>}
....
}
```

RAY TRACING COMPETITION. There is a annual ray tracing comptetion going on. Here is an example of an entry from the 1999 competition The authors of the scene are Ian and Ethel MacKay. The source file for this scene is a single povray file.

RENDERING. Raytracing is a CPU time intensive task. The software has to compute the light ray paths bouncing around in a virtual world, compute reflections or refractions. Tracing an image can take from a few seconds to days. A single frame in movies like "Toy Story" takes several hours to render. To get the large number of frames needed for a movie companies like "Pixar" (recently again visible with Monsters inc.) use "computer farms" a huge number of workstations now mostly running linux.

THE ROLE OF MATH 21a. Many topics you learned appear in the area of ray tracing. For example, to compute the normal vector to a polygon, one uses an area formula, which we have proven in a homework using Green's theorem (see below). Normal vectors play an important role because they are needed to compute the reflection of rays. Just look inside a book or article on 3D graphics and many now familiar objects will pop up.

EXAMPLE 1). The method of computing normals to a surface by just looking at three points is error prone. It is often better to consider a polygon $P_i = (x_i, y_i, z_i)$ on the surface. What is the normal to such a polygon? Note that the points P_i are not necessarily on a plane. Here is what people do in 3D graphics: consider the xy projection of the polygon. This gives a polygon (x_i, y_i) in the plane. Now compute the area of that projected polygon with the formula $1/2 \sum_k (x_{k+1} + x_k)(y_{k+1} - y_k)$ (see page 275 in your book). This formula was proven using Green's theorem but can be seen geometrically. This area will be the third component c of the normal vector. Analoguously, compute the other components.

The normal vector to a not necessarily planar polygon $P_i = (x_i, y_i, z_i)$ in space is defined as

$$n = \begin{bmatrix} 1/2 \sum_k (y_{k+1} + y_k)(z_{k+1} - z_k) \\ 1/2 \sum_k (z_{k+1} + z_k)(x_{k+1} - x_k) \\ 1/2 \sum_k (x_{k+1} + x_k)(y_{k+1} - y_k) \end{bmatrix} .$$

EXAMPLE 2) Snells law of refraction is the problem to determine the fastest path between two points, if the path crosses a border of two media and the media have different indices of refraction. This is a Lagrange multiplier problem:

A light ray travles from $A = (-1, 1)$ to the point $B = (1, -1)$ crossing a boundary between two media (air and water). In air (y¿0) the speed of the ray is $v_1 = 1$ (in units of speed of light). In the second medium (y¡0) the speed of light is $v_2 = 0.9$. The light ray travels on a straight line from A to a point $P = (x, 0)$ on the boundary and on a straight line from P to B. Verify Snell's law of refraction $\sin(\theta_1)/\sin(\theta_2) = v_1/v_2$. where θ_1 is the angle the ray makes in air with the y axes and where θ_2 is the angle, the ray makes in water with the y axes.

Solution: Minimize

$$F(x, y) = \sqrt{(-1-x)^2 + y^2}/v1 + \sqrt{(1-x)^2 + y^2}/v2 = l_1/v1 + l_2/v2$$

under the constraint $G(x, y) = y = 0$. The Lagrange equations show that $F_x(x, y) = 0$. This is already Snells law $F_x = v_1 2(x + 1)/(2l_1) + v_2 2(1 - x)/(2l_2) = 0$ means $v_1/v_2 = \sin(\theta_1)/\sin(\theta_2)$.

URL'S		
	http://www.povray.org	official povray site
	http://www.povworld.org	collection of objects

OTHER RAYTRACING PROGRAMS OR MODELERS. Note that Povray provides the source code to the software in contrast to commercial software which can be very expensive.

http://www.blender.nl	Blender (free 3D modeling)
http://www.aliaswavefront.com	Alias Wave Front (Maya)
http://www.lightwave.com	Lightwave 3D
http://www.corel.com	Bryce
http://www.curiouslabs.com	Poser (3D figure design)
http://www.eovia.com	Carrara (Ray Dream studio)
http://www.artifice.com	Radiance (architecture)
http://www.3dlinks.com	Links