Waves & Optics: Lenses and Mirrors

Understanding the nature of light has been a motivator for the development of much of Physics. Indeed, light (and its larger and smaller wavelength variants) is in most cases the *only* viable means of gaining information about the Cosmos beyond our closest planetary neighbours.

Here we shall build a simulation of how light passes through an **idealized lens**, and is reflected off surfaces. To make things interesting for the latter, we will investigate a *spherically curved* **concave mirror.**

Assumptions

• Light is an **electromagnetic wave**, a disturbance in the electric and magnetic field that exists in space from, respectively, charges and moving charges. A charge in an electric field will feel a *force*. An *additional force* will be experienced *if the charge is moving* in a magnetic field. Unlike a sound wave or water wave, **electromagnetic waves can travel through empty space.**
No 'medium' is required – *it itself moves.*

• **In a vacuum, light travels at speed** $c = 2.998 \times 10^8 \text{ms}^{-1}$
This is the speed limit for the conveyance of information in the Universe. Like other forms of waves, **light will travel in straight lines** *unless the wave speed changes.* If light passes through water or glass, the presence of charges in the atomic structure of these substances will impede its progress, making the path more tortuous, and longer. The overall speed of light will therefore *reduce*.

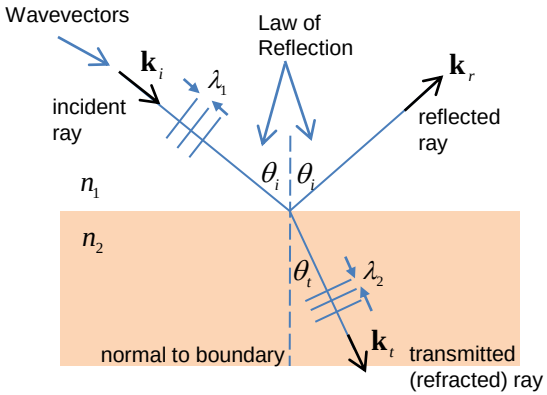
• Light can travel through *any* path between two points, but the *most probable** is the path which takes the *least time.* This is ***Fermat's Principle.*** The result is that light *refracts* (bends) at a boundary of wave speeds (e.g. glass to air), or bends continuously if the wave speed also varies.

• Light might be a thin laser beam, but it might also be a broad illumination from an extended source such as the Sun. In both extremes we can think of the **direction** that light propagates.
We call this the ***wavevector***, or more simply, a ***ray***.

Geometric Optics is in essence the drawing of diagrams that represent the directions light rays take from source to observer. When *multiple rays* can be drawn between source and observer, **the *intersection* of these different paths will predict where an image of a light source will form.**

To construct our simulation we must therefore *determine at least two intersecting ray paths from light source to observer*. In the case of a **converging lens** or a **concave mirror** this will tell us where an **image** will be formed, and in the case of a **magnifying glass** or **convex lens,** where diverging rays *appear to be originating if we assume straight line paths*. We call the latter a **virtual image,** as it is essentially a processing artefact of our minds as we try to make sense of the light that enters our eyes.

We therefore need some *rules* regarding the direction of rays. Firstly some fundamentals of **reflection** and **refraction** at a boundary between regions where light travels at a constant speed.



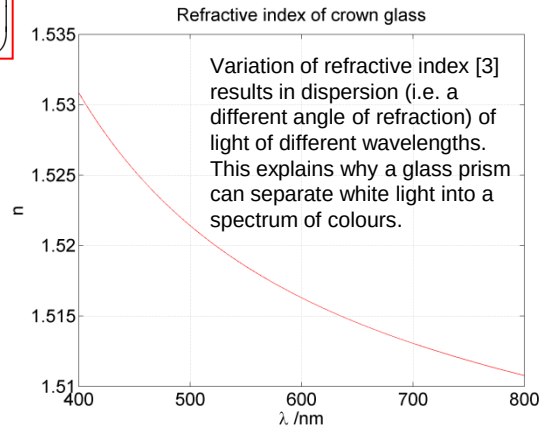
Refractive index *n* defines the ratio between the speed of light in a medium such as glass, to the speed of light in a vacuum. For glass this ratio is about 1.5, although there is variation with wavelength.

Fermat's principle of ray paths being those of least duration results [2] in both the **Law of Reflection** and **Snell's Law of Refraction**, which refers to a boundary between two materials of different refractive index.

Snell's Law

$$n_1 \sin\theta_i = n_2 \sin\theta_t$$
$$\therefore \theta_t = \sin^{-1}\left(\frac{n_1 \sin\theta_i}{n_2}\right)$$

If refractive index across a boundary increases, this means wave speed decreases and hence rays will bend towards the normal



Variation of refractive index [3] results in dispersion (i.e. a different angle of refraction) of light of different wavelengths. This explains why a glass prism can separate white light into a spectrum of colours.

The number of waves passing per second (i.e. the frequency f) must be a *conserved quantity* on both sides of the boundary, otherwise there must be some form of energy, and also information, input. Since the speed of waves is the *frequency multiplied by the wavelength*:

$$\frac{c}{n_1} = f\lambda_1, \quad \frac{c}{n_2} = f\lambda_2 \quad \therefore \frac{\lambda_2}{\lambda_1} = \frac{n_1}{n_2}$$

i.e. wavelength will decrease in the same proportion that wave speed does.

Special cases of Snell's Law

$n_2 \geq n_1$ Rays will always be refracted, i.e. all values of θ_t are *real*

$n_1 > n_2$ Limiting case when refraction angle is 90º. **Total internal reflection** for greater *incident* angles than the **critical angle**.

Critical angle: $n_1 \sin\theta_c = n_2 \sin 90^0 \quad \therefore \theta_c = \sin^{-1}\left(\frac{n_2}{n_1}\right)$

Often we have a glass-air interface so $\theta_c = \sin^{-1}(1/n)$

This effect explains why **optical fibres** can be so useful for communications. Light is orientated such that it is internally reflected within the fibre until it emerges. The lack of transmission at each reflection significantly reduces the overall propagation loss. For a glass-air interface with a refractive index ratio of 1/1.5, the critical angle is 41.8º

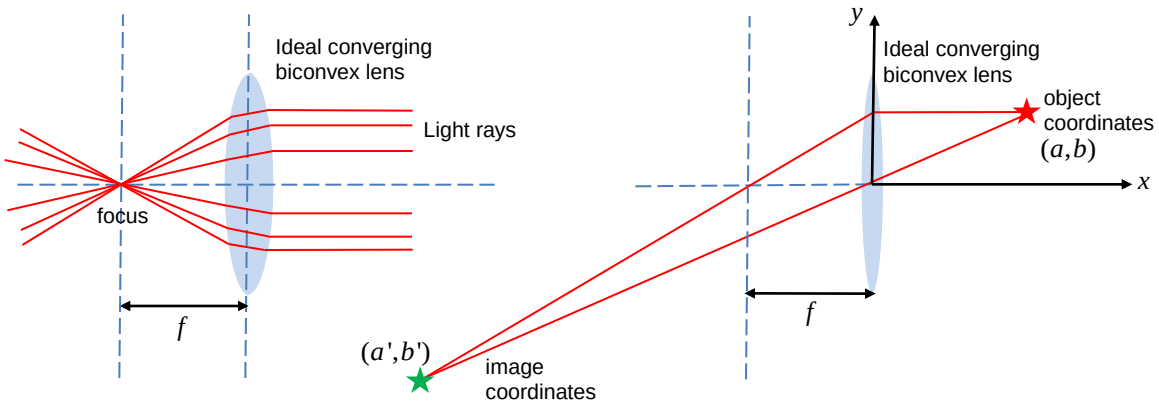
Colour	Wavelength λ /nm
Violet	380-450
Blue	450-495
Green	495-570
Yellow	570-590
Orange	580-620
Red	620-750

*This idea is explored in the theory of *Quantum Electrodynamics* (QED). [1]

Simulation #1: A converging convex lens. This is a model of the glass in spectacles, a projector, and a human eye! Light from a source is *refracted* via the lens such that a **real**, **inverted**, and **possibly scaled**, *image* is formed in space at the opposite side of the lens to the source.

Optical lenses are typically a structure which uses *refraction* to converge (or diverge) rays of light. A practical *converging* lens might be constructed from thin circular convex sections of glass joined back to back. (i.e. *biconvex*). The design goal of an *ideal lens* is as follows:

- Rays that are incident to the centre of the lens pass through without any refraction.
- The lens is thin i.e. the radial extension is much larger than the width of lens
- Horizontal rays, i.e. those perpendicular to the radial direction of the lens, are refracted such that they always pass through a specific point, called the **focus** of the lens.



Note real lenses do not achieve the third design criteria and are subject to *aberrations*. Correcting for these is a major driver of optical system design!

- ***Chromatic* aberration**: The variation of refractive index with wavelength will vary the focus.
- ***Spherical* aberration**: Parabolic surfaces have the property that horizontal rays are focussed through a single coordinate. This is only approximately true for spherical lenses.

Our **simulation of an ideal lens** is essentially a *mapping of coordinates*. Using the ray intersection of the 'straight through path' and the 'horizontal path, refracted through the focus' we can find out the coordinates of the image formed by light emanating from an object. Using the diagram above we can write down Cartesian equations for these two lines.

$$y = \frac{b}{a}x$$ 'Straight through path'

$$y = \frac{b}{f}x + b$$ 'Horizontal path, refracted through the focus'

The image coordinates are at the intersection of these lines:

$$b' = \frac{b}{a}a', \quad b' = \frac{b}{f}a' + b$$

$$\therefore \frac{b}{a}a' = \frac{b}{f}a' + b$$

$$\therefore \left(\frac{b}{f} - \frac{b}{a}\right)a' = -b$$

$$\therefore a' = -\left(\frac{1}{f} - \frac{1}{a}\right)^{-1}$$

$$\therefore b' = -\frac{b}{a}\left(\frac{1}{f} - \frac{1}{a}\right)^{-1}$$

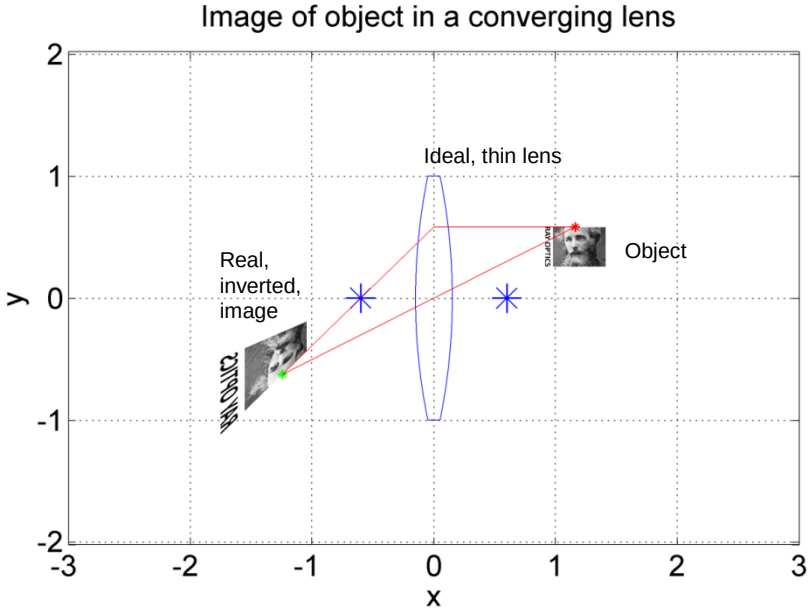
We can clearly see from that the image is inverted, and in general, scaled too.

Dynamic simulation in MATLAB

We can use the transformation idea to not only animate the ray diagram on the left for different object positions, but also investigate how a realistic object (e.g. the portrait of James Clerk Maxwell) is imaged, and distorted.

Below is a screenshot from the simulation `converging_thin_lens.m`
Arrow keys are used to move the object, which includes a red spot on Maxwell's forehead. The red 'straight through' and 'refracted through focus' rays are automatically recalculated and updated and converge on a green spot, the real, inverted image of the red spot. The transformations of the 701 x 457 array of grayscale pixels which constitute the object are also computed, and used to construct the image.

The blue stars represent the focus of the ideal, thin lens.



MATLAB code for `converging_thin_lens.m`

```
%converging_thin_lens
% Determines the image of an object in a thin
% converging lens. (i.e. a bi-convex lens formed by
% two spherical sections. Assume spacing is negligible).

function converging_thin_lens
global d

%Height of lens
d.H = 1;

%Radius of curvature of lenses (merely for visual
%purposes - we don't make use of the Lensmaker's formula
%in this simulation)
Rc = 5;

%Width of lens (merely for visual purposes - we don't
make
%use of the Lensmaker's formula in this simulation)
lw = 0.1;

%Focal length of lens
d.f = 0.6;

%Object centre
x0 = 1;
y0 = 0.0;

%Width of object (this scales the image)
w = 0.5;

%Movement on keypress
d.delta = 0.02;

%Fontsize
fsize = 18;

%

%Import bitmap image of object
d.I = imread('Maxwell.jpg');
d.I = double(flipdim(d.I,1))/255;
dim = size(d.I);
Iwidth = dim(2);
Iheight = dim(1);

%Determine x,y coordinates of pixel locations
d.x = linspace(0,w,Iwidth) - w/2 + x0;
d.y = linspace(0,w*Iheight/Iwidth,Iheight) -
0.5*w*Iheight/Iwidth + y0 ;
[d.x,d.y] = meshgrid(d.x,d.y);

%Detrmine most left x coordinate of object
d.xmin = x0-w/2;

%Determine largest and smallest y coordinates of object
d.ymax = 0.5*w*Iheight/Iwidth + y0;
d.ymin = -0.5*w*Iheight/Iwidth + y0;
```

Define main function. Note % means commentary

Hard code inputs for simulation

Setting up arrays to determine object and image coordinates and pixel colours

```
%Create figure and plot object
d.fig = figure( 'KeyPressFcn',@keypress );
d.Obj = surf(d.x,d.y,-ones(Iheight,Iwidth),d.I);
shading interp
view(2);
hold on;
axis equal
xlabel('x','fontsize',fsize);
ylabel('y','fontsize',fsize);
title('Image of object in a converging
lens','fontsize',fsize);
set(gca,'fontsize',fsize);
xlim([-5*d.f,5*d.f]);
ylim([-d.delta-2*d.H,+d.delta+2*d.H]);
axis manual
box on

%Determine x coordinate of centre of curvature for lenses
xL = sqrt( Rc^2 - d.H^2 ) - lw/2;
xR = - sqrt( Rc^2 - d.H^2 ) + lw/2;

%Determine lens arcs
tmax = atan( d.H/(xL + lw/2) );
t = linspace(-tmax,tmax,500);
xl = xL - Rc*cos(t);
yl = Rc*sin(t);
xr = xR + Rc*cos(t);
yr = Rc*sin(t);
plot([xl,fliplr(xr),xl(1)],[yl,fliplr(yr),yl(1)],'b')

%Plot lens foci
plot(d.f,0,'b*','markersize',fsize);
plot(-d.f,0,'b*','markersize',fsize);

%Define a particular location on image to draw rays from
d.a = x0;
d.b = 0.5*w*Iheight/Iwidth + y0;

%Determine image location due to action of lens
[d.xx,d.yy] = lens( d.x,d.y,d.f );
[d.aa,d.bb] = lens( d.a,d.b,d.f );

%Plot transformed image
d.Imag = surf(d.xx,d.yy,-ones(Iheight,Iwidth),d.I);
shading interp

%Overlay rays from (d.a,d.b) to (d.aa,d.bb) via mirror
d.ray1 = plot( [d.a,0,d.aa],[d.b,d.b,d.bb],'r-' );
d.ray2 = plot( [d.a,d.aa],[d.b,d.bb],'r-' );
d.ab = plot( d.a,d.b,'r*' );
d.aabb = plot( d.aa,d.bb,'g*' );

%%

% Lens image position (assume this lens
% equation 1/x + 1/xx = 1/f and yy/y = (xx - f)/x
function [xx,yy] = lens( x,y,f )
xx = -(( -1./x + 1/f).^-1);
yy = y.*xx ./x;
```

Setting up plot, which has a *keypress handler*. This is code which executes when a key is pressed

Generate initial plot lines, including items such as the lens which don't move

Sub-function which performs the transformation of object to image coordinates

$$a' = -\left(\frac{1}{f} - \frac{1}{a}\right)^{-1}$$

$$b' = -\frac{b}{a}\left(\frac{1}{f} - \frac{1}{a}\right)^{-1}$$

```
%Function which executes when a key is pressed
function keypress(fig,evnt)
global d

if strcmp(get(fig,'currentkey'),'uparrow')==1
    d.ymax = d.ymax + d.delta;
    %Check that object is not beyond top of lens
    if d.ymax > d.H
        d.ymax = d.ymax - d.delta;
    else
        d.y = d.y + d.delta;
        d.b = d.b + d.delta;
        d.ymin = d.ymin + d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'downarrow')==1
    d.ymin = d.ymin - d.delta;
    %Check that object is not below bottom of lens
    if d.ymin < -d.H
        d.ymin = d.ymin + d.delta;
    else
        d.y = d.y - d.delta;
        d.b = d.b - d.delta;
        d.ymax = d.ymax - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'leftarrow')==1
    d.xmin = d.xmin - d.delta;
    %Check that object is not within focal length
    if d.xmin < d.f
        d.xmin = d.xmin + d.delta;
    else
        d.x = d.x - d.delta;
        d.a = d.a - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'rightarrow')==1
    d.x = d.x + d.delta;
    d.a = d.a + d.delta;
    d.xmin = d.xmin + d.delta;
    update
elseif strcmp(get(fig,'currentkey'),'p')==1
    print( d.fig, 'converging_lens.png','-dpng','-r300' );
end

%%

%Update screen
function update
global d

%Determine transformation in mirror
[d.xx,d.yy] = lens( d.x,d.y,d.f );
[d.aa,d.bb] = lens( d.a,d.b,d.f );

%Update object and image
set( d.Obj,'xdata',d.x,'ydata',d.y );
set( d.Imag,'xdata',d.xx,'ydata',d.yy );
set( d.ray1,'xdata',[d.a,0,d.aa],'ydata',[d.b,d.b,d.bb]);
set(d.ray2,'xdata',[d.a,d.aa],'ydata',[d.b,d.bb]);
set(d.ab,'xdata',d.a,'ydata',d.b);
set(d.aabb,'xdata',d.aa,'ydata',d.bb);
drawnow;

%End of code
```

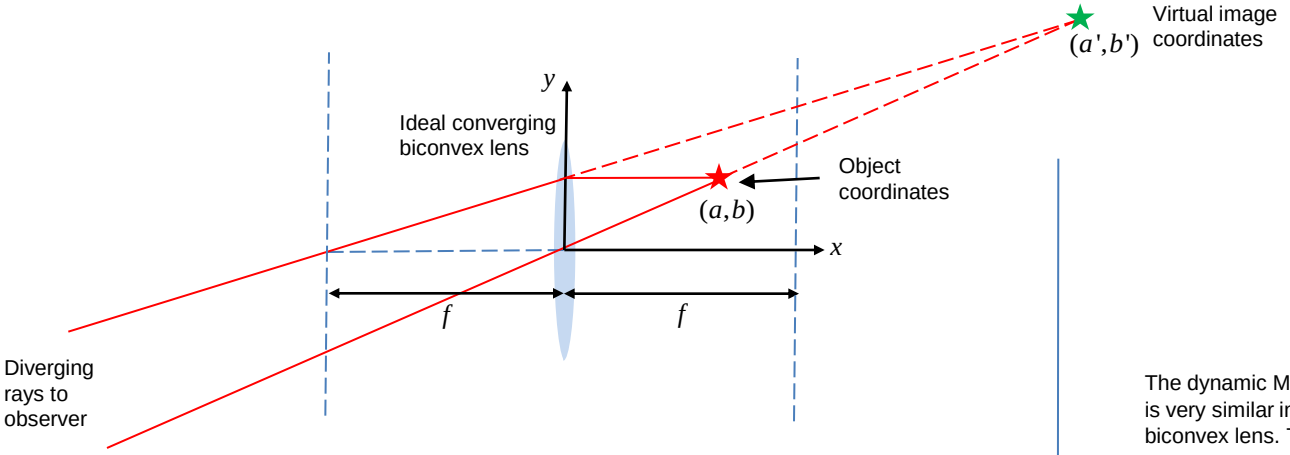
Keypress handler sub-function

Note various limits are built in, which means the object to image transformation will always yield real, and plottable solutions.

The object cannot be beyond the top of the lens or within the focal length of the lens

Screen update sub-function, which redraws the graphic elements which change as the object is moved via key presses. Global structured array `d` is how information is passed around the various code elements of the simulation

Simulation #2: A converging convex lens used as a magnifying glass. In this case the object is placed *within the focal range* of the lens. Rays emanating from the object will diverge as a result of refraction when they pass through the lens. An observer looking at the object through the lens may interpret the origin of these rays by tracing back the diverging paths to a point of confluence. This is the *virtual image*, and effectively what we 'see' in a magnifying glass.



The transformation from object coordinates to virtual image coordinates follows in a very similar fashion to the converging biconvex lens example. In other words we are looking for the coordinates of intersection of two straight lines which represent the 'straight through' and 'refracted through focus' ray paths

$$y = \frac{b}{a}x$$ 'Straight through path'

$$y = \frac{b}{f}x + b$$ 'Horizontal path, refracted through the focus'

The image coordinates are at the intersection of these lines:

$$b' = \frac{b}{a}a', \quad b' = \frac{b}{f}a' + b$$

$$\therefore \frac{b}{a}a' = \frac{b}{f}a' + b$$

$$\therefore \left(\frac{b}{a} - \frac{b}{f}\right)a' = b$$

$$\therefore \boxed{a' = \left(\frac{1}{a} - \frac{1}{f}\right)^{-1}, \quad b' = \frac{b}{a}\left(\frac{1}{a} - \frac{1}{f}\right)^{-1}}$$

The **vertical magnification factor** is:

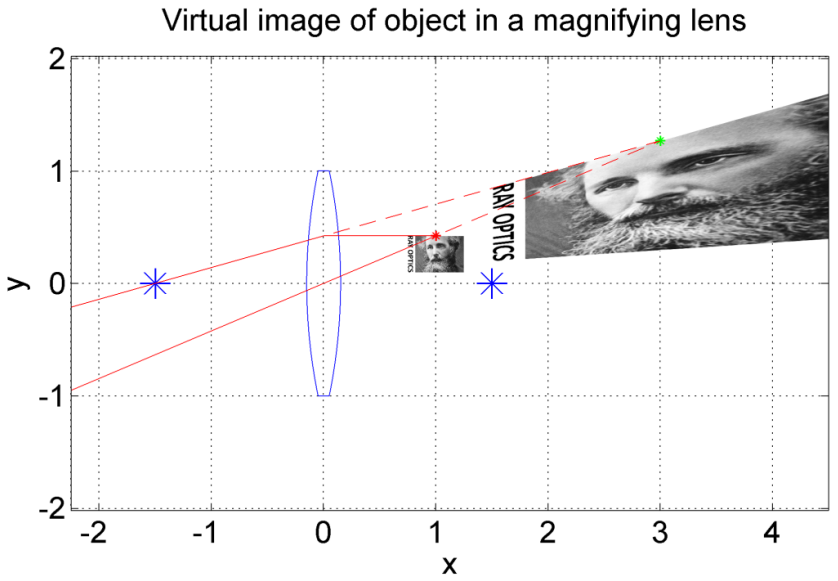
$$M = \frac{b'}{b} = \frac{1}{a}\left(\frac{1}{a} - \frac{1}{f}\right)^{-1}$$

$$M = \frac{1}{a}\frac{af}{f-a}$$

$$\boxed{M = \frac{f}{f-a}}$$ i.e. $M > 1, \quad f > a$

The dynamic MATLAB simulation `magnifying_glass.m` is very similar in structure to that of the converging biconvex lens. The differences are (see code listing on the next page)

- The signs of quantities in the transformation formula
- Addition of dashed lines to indicate the 'extrapolated straight line ray paths' from the diverging rays arriving at the observer to the assumed source, the virtual image.



MATLAB code for `magnifying_glass.m`

```
%magnifying_glass
% Determines the virtual image of an object placed
% within the focal length of a thin bi-convex lens.

function magnifying_glass
global d

%Height of lens
d.H = 1;

%Radius of curvature of lenses (merely for visual
%purposes - we don't make use of the Lensmaker's formula
%in this simulation)
Rc = 5;

%Width of lens (merely for visual purposes - we don't
make
%use of the Lensmaker's formula in this simulation)
lw = 0.1;

%Focal length of lens
d.f = 1.5;

%Object centre
x0 = 1;
y0 = 0.0;

%Width of object (this scales the image)
w = 0.5;

%Movement on keypress
d.delta = 0.02;

%Fontsize
fsize = 18;

%

%Import bitmap image of object
d.I = imread('Maxwell.jpg');
d.I = double(flipdim(d.I,1))/255;
dim = size(d.I);
Iwidth = dim(2);
Iheight = dim(1);

%Determine x,y coordinates of pixel locations
d.x = linspace(0,w,Iwidth) - w/2 + x0;
d.y = linspace(0,w*Iheight/Iwidth,Iheight) -
0.5*w*Iheight/Iwidth + y0 ;
[d.x,d.y] = meshgrid(d.x,d.y);

%Detrmine most left x coordinate of object
d.xmin = x0-w/2;

%Determine largest and smallest y coordinates of object
d.ymax = 0.5*w*Iheight/Iwidth + y0;
d.ymin = -0.5*w*Iheight/Iwidth + y0;
```

Define main function. Note % means commentary

Hard code inputs for simulation

Setting up arrays to determine object and image coordinates and pixel colours

Sub-function which performs the transformation of object to image coordinates

```
%Create figure and plot object
d.fig = figure( 'KeyPressFcn',@keypress );
d.Obj = surf(d.x,d.y,-ones(Iheight,Iwidth),d.I);
shading interp
view(2);
hold on;
axis equal
xlabel('x','fontsize',fsize);
ylabel('y','fontsize',fsize);
title('Image of object in a magnifying
glass','fontsize',fsize);
set(gca,'fontsize',fsize);
xlim([-1.5*d.f,1.5*d.f]);
ylim([-d.delta-2*d.H,+d.delta+2*d.H]);
axis manual
box on

%Determine x coordinate of centre of curvature for lenses
xL = sqrt( Rc^2 - d.H^2 ) - lw/2;
xR = - sqrt( Rc^2 - d.H^2 ) + lw/2;

%Determine lens arcs
tmax = atan( d.H/(xL + lw/2) );
t = linspace(-tmax,tmax,500);
xl = xL - Rc*cos(t);
yl = Rc*sin(t);
xr = xR + Rc*cos(t);
yr = Rc*sin(t);
plot([xl,fliplr(xr),xl(1)],[yl,fliplr(yr),yl(1)],'b')

%Plot lens foci
plot(d.f,0,'b*','markersize',fsize);
plot(-d.f,0,'b*','markersize',fsize);

%Define a particular location on image to draw rays from
d.a = x0;
d.b = 0.5*w*Iheight/Iwidth + y0;

%Determine image location due to action of lens
[d.xx,d.yy] = lens( d.x,d.y,d.f );
[d.aa,d.bb] = lens( d.a,d.b,d.f );

%Plot transformed image
d.Imag = surf(d.xx,d.yy,-ones(Iheight,Iwidth),d.I);
shading interp

%Overlay rays from (d.a,d.b) to (d.aa,d.bb) via lens
xlimits = get(gca,'xlim');
d.ray1 = plot( [xlimits(1),0,d.a],[xlimits(1)*d.b/d.f +
d.b,d.b,d.b],'r-' );
d.ray2 = plot( [0,d.aa],[d.b,d.bb],'r--' );
d.ray3 = plot(
[xlimits(1),d.a],[xlimits(1)*d.b/d.a,d.b],'r-' );
d.ray4 = plot( [d.a,d.aa],[d.b,d.bb],'r--' );
d.ab = plot( d.a,d.b,'r*' );
d.aabb = plot( d.aa,d.bb,'g*' );


%%

% Lens virtual image position
function [xx,yy] = lens( x,y,f )
xx = (( 1./x - 1/f).^-1);
yy = y.*xx ./x;
```

Setting up plot, which has a *keypress handler*. This is code which executes when a key is pressed

Generate initial plot lines, including items such as the lens which don't move

$$a' = \left(\frac{1}{a} - \frac{1}{f}\right)^{-1}$$

$$b' = \frac{b}{a}\left(\frac{1}{a} - \frac{1}{f}\right)^{-1}$$

```
%Function which executes when a key is pressed
function keypress(fig,evnt)
global d
if strcmp(get(fig,'currentkey'),'uparrow')==1
    d.ymax = d.ymax + d.delta;
    %Check that object is not beyond top of lens
    if d.ymax > d.H
        d.ymax = d.ymax - d.delta;
    else
        d.y = d.y + d.delta;
        d.b = d.b + d.delta;
        d.ymin = d.ymin + d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'downarrow')==1
    d.ymin = d.ymin - d.delta;
    %Check that object is not below bottom of lens
    if d.ymin < -d.H
        d.ymin = d.ymin + d.delta;
    else
        d.y = d.y - d.delta;
        d.b = d.b - d.delta;
        d.ymax = d.ymax - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'leftarrow')==1
    d.xmin = d.xmin - d.delta;
    %Check that object is not less than lens position
    if d.xmin < 0
        d.xmin = d.xmin + d.delta;
    else
        d.x = d.x - d.delta;
        d.a = d.a - d.delta;
        d.xmax = d.xmax - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'rightarrow')==1
    d.xmax = d.xmax + d.delta;
    %Check that object is not beyond focal length
    if d.xmax > d.f
        d.xmax = d.xmax - d.delta;
    else
        d.x = d.x + d.delta;
        d.a = d.a + d.delta;
        d.xmin = d.xmin + d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'p')==1
    print( d.fig, 'magnifying_glass.png','-dpng','-r300' );
end

%%
```

Keypress handler sub-function

Note various limits are built in, which means the object to image transformation will always yield real, and plottable solutions.

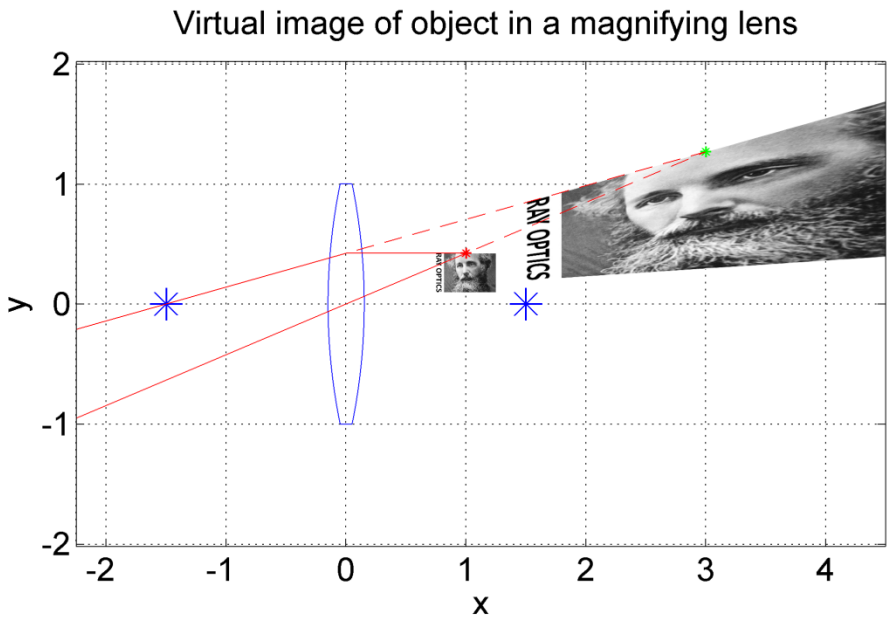
The object cannot be beyond the top of the lens or beyond the focal length of the lens

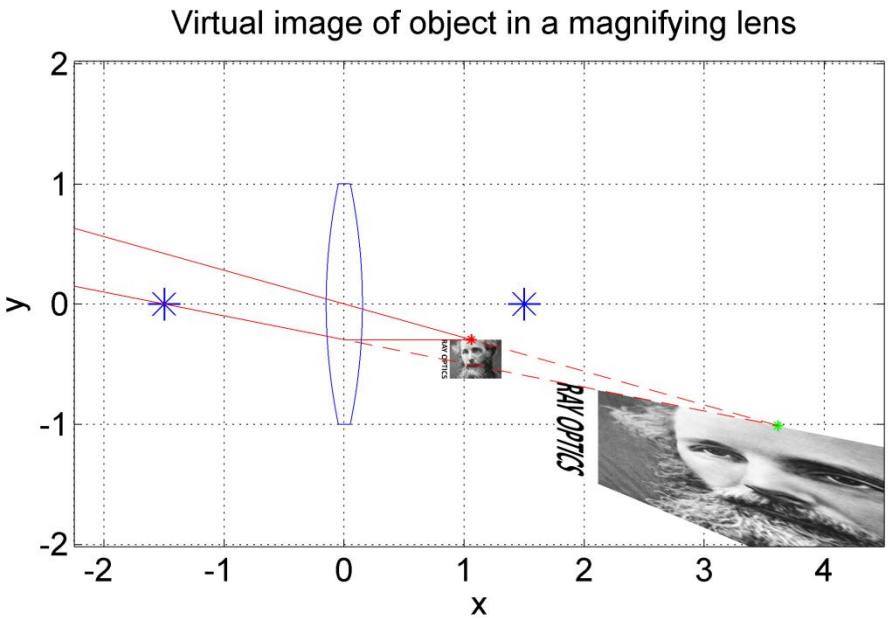
MATLAB code for `magnifying_glass.m` (continued ...)

```
%Update screen
function update
global d

%Determine transformation in mirror
[d.xx,d.yy] = lens( d.x,d.y,d.f );
[d.aa,d.bb] = lens( d.a,d.b,d.f );

%Update object and image
xlimits = get( gca, 'xlim' );
set( d.Obj,'xdata',d.x,'ydata',d.y );
set( d.Imag,'xdata',d.xx,'ydata',d.yy );
set( d.ray1,'xdata',[xlimits(1),0,d.a],'ydata',[xlimits(1)*d.b/d.f + d.b,d.b,d.b] );
set( d.ray2,'xdata',[0,d.aa],'ydata',[d.b,d.bb] );
set( d.ray3,'xdata',[xlimits(1),d.a],'ydata',[xlimits(1)*d.b/d.a,d.b] );
set( d.ray4,'xdata',[d.a,d.aa],'ydata',[d.b,d.bb] );
set(d.ab,'xdata',d.a,'ydata',d.b);
set(d.aabb,'xdata',d.aa,'ydata',d.bb);
drawnow;

%End of code
```

Screen update sub-function, which redraws the graphic elements which change as the object is moved via key presses.

Global structured array `d` is how information is passed around the various code elements of the simulation





Simulation #3: A diverging concave lens.

In this scenario we can create a de-magnified, upright, virtual image of an object placed outside the focal range of a pair of bi-concave lenses.

The coordinate transformation can be determined in a very similar fashion as for the previous two bi-convex lens examples.

$$y = \frac{b}{a}x$$ 'Straight through path'

$$y = -\frac{b}{f}x + b$$ 'Horizontal path, refracted through the focus'

The image coordinates are at the intersection of these lines:

$$b' = \frac{b}{a}a', \quad b' = -\frac{b}{f}a' + b$$

$$\therefore \frac{b}{a}a' = -\frac{b}{f}a' + b$$

$$\therefore \left(\frac{b}{a} + \frac{b}{f}\right)a' = b$$

$$\therefore a' = \left(\frac{1}{a} + \frac{1}{f}\right)^{-1}$$

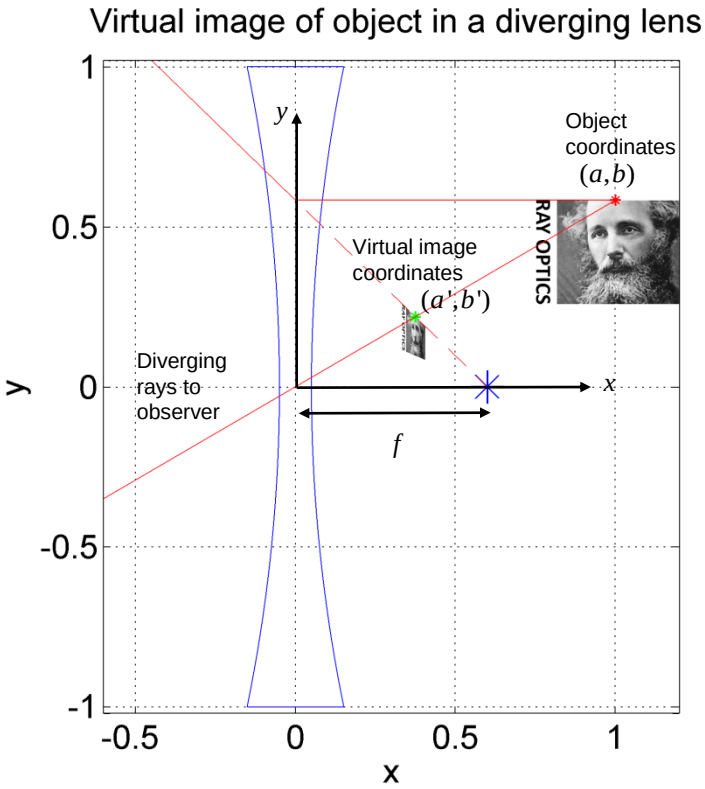
$$\therefore b' = \frac{b}{a}\left(\frac{1}{a} + \frac{1}{f}\right)^{-1}$$

The vertical magnification factor in this case is:

$$M = \frac{b'}{b} = \frac{1}{a}\left(\frac{1}{a} + \frac{1}{f}\right)^{-1}$$

$$M = \frac{1}{a}\frac{af}{f+a}$$

$$M = \frac{f}{f+a}$$ i.e. $M < 1$



MATLAB code for `diverging_thin_lens.m`

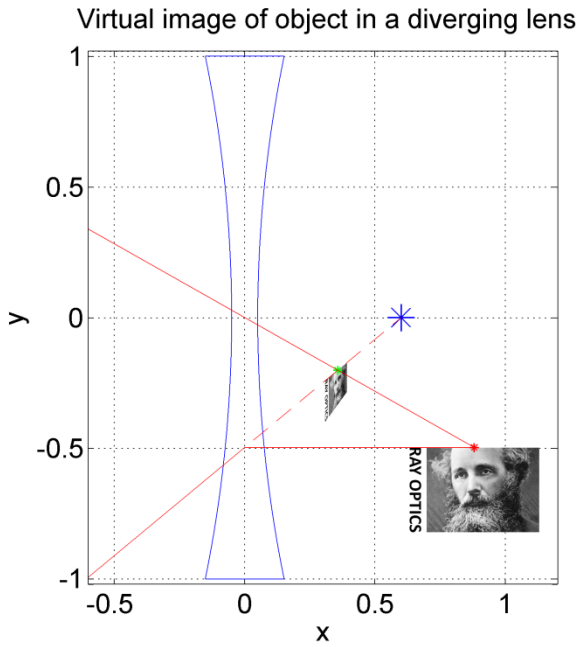
```
%diverging_thin_lens
% Determines the virtual image of an object in a thin diverging lens. (i.e.
% a bi-concave lens formed by two spherical sections. Assume spacing is
% negligible).

function diverging_thin_lens
global d

%Height of lens
d.H = 1;

%Radius of curvature of lenses (merely for visual purposes - we don't make
%use of the Lensmaker's formula in this simulation)
Rc = 5;

%Width of lens (merely for visual purposes - we don't make
%use of the Lensmaker's formula in this simulation)
lw = 0.1;

%Focal length of lens
d.f = 0.6;

%Object centre
x0 = 1;
y0 = 0.0;

%Width of object (this scales the image)
w = 0.5;

%Movement on keypress
d.delta = 0.02;

%Fontsize
fsize = 18;

%

%Import bitmap image of object
d.I = imread('Maxwell.jpg');
d.I = double(flipdim(d.I,1))/255;
dim = size(d.I);
Iwidth = dim(2);
Iheight = dim(1);

%Determine x,y coordinates of pixel locations
d.x = linspace(0,w,Iwidth) - w/2 + x0;
d.y = linspace(0,w*Iheight/Iwidth,Iheight) - 0.5*w*Iheight/Iwidth + y0 ;
[d.x,d.y] = meshgrid(d.x,d.y);

%Detrmine most left x coordinate of object
d.xmin = x0-w/2;

%Determine largest and smallest y coordinates of object
d.ymax = 0.5*w*Iheight/Iwidth + y0;
d.ymin = -0.5*w*Iheight/Iwidth + y0;

%Import bitmap image of object
d.I = imread('Maxwell.jpg');
d.I = double(flipdim(d.I,1))/255;
dim = size(d.I);
Iwidth = dim(2);
Iheight = dim(1);
```

MATLAB code for `diverging_thin_lens.m` (continued)

```
%Determine x,y coordinates of pixel locations
d.x = linspace(0,w,Iwidth) - w/2 + x0;
d.y = linspace(0,w*Iheight/Iwidth,Iheight) - 0.5*w*Iheight/Iwidth + y0 ;
[d.x,d.y] = meshgrid(d.x,d.y);

%Detrmine most left x coordinate of object
d.xmin = x0-w/2;

%Determine largest and smallest y coordinates of object
d.ymax = 0.5*w*Iheight/Iwidth + y0;
d.ymin = -0.5*w*Iheight/Iwidth + y0;

%Create figure and plot object
d.fig = figure( 'KeyPressFcn',@keypress );
d.Obj = surf(d.x,d.y,-ones(Iheight,Iwidth),d.I);
shading interp
view(2);
hold on;
axis equal
xlabel('x','fontsize',fsize);
ylabel('y','fontsize',fsize);
title('Virtual image of object in a diverging lens','fontsize',fsize);
set(gca,'fontsize',fsize);
xlim([-d.f,2*d.f]);
ylim([-d.delta-d.H,+d.delta+d.H]);
axis manual
box on

%Determine x coordinate of centre of curvature for lenses
xR = Rc + lw/2;
xL = - lw/2 - Rc;

%Determine lens arcs
tmax = asin( d.H/Rc );
t = linspace(-tmax,tmax,500);
xl = xL + Rc*cos(t);
yl = Rc*sin(t);
xr = xR - Rc*cos(t);
yr = Rc*sin(t);
plot([xl,fliplr(xr),xl(1)],[yl,fliplr(yr),yl(1)],'b')

%Plot lens foci
plot(d.f,0,'b*','markersize',fsize);

%Define a particular location on image to draw rays from
d.a = x0;
d.b = 0.5*w*Iheight/Iwidth + y0;

%Determine image location due to action of lens
[d.xx,d.yy] = lens( d.x,d.y,d.f );
[d.aa,d.bb] = lens( d.a,d.b,d.f );

%Plot transformed image
d.Imag = surf(d.xx,d.yy,-ones(Iheight,Iwidth),d.I);
shading interp

%Overlay rays from (d.a,d.b) to (d.aa,d.bb) via lens
xlimits = get(gca,'xlim');
d.ray1 = plot( [d.a,xlimits(1)],[d.b,d.b*xlimits(1)/d.a],'r-' );
d.ray2 = plot( [d.a,0,xlimits(1)],[d.b,d.b,d.b - xlimits(1)*d.b/d.f],'r-' );
d.ray3 = plot( [0,d.f],[d.b,0],'r--' );
d.ab = plot( d.a,d.b,'r*' );
d.aabb = plot( d.aa,d.bb,'g*' );

%%
```

```
%Function which executes when a key is pressed
function keypress(fig,evnt)
global d

if strcmp(get(fig,'currentkey'),'uparrow')==1
    d.ymax = d.ymax + d.delta;
    %Check that object is not beyond top of lens
    if d.ymax > d.H
        d.ymax = d.ymax - d.delta;
    else
        d.y = d.y + d.delta;
        d.b = d.b + d.delta;
        d.ymin = d.ymin + d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'downarrow')==1
    d.ymin = d.ymin - d.delta;
    %Check that object is not below bottom of lens
    if d.ymin < -d.H
        d.ymin = d.ymin + d.delta;
    else
        d.y = d.y - d.delta;
        d.b = d.b - d.delta;
        d.ymax = d.ymax - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'leftarrow')==1
    d.xmin = d.xmin - d.delta;
    %Check that object is not within focal length
    if d.xmin < d.f
        d.xmin = d.xmin + d.delta;
    else
        d.x = d.x - d.delta;
        d.a = d.a - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'rightarrow')==1
    d.x = d.x + d.delta;
    d.a = d.a + d.delta;
    d.xmin = d.xmin + d.delta;
    update
elseif strcmp(get(fig,'currentkey'),'p')==1
    print( d.fig, 'diverging_lens.png','-dpng','-r300' );
end

%%
```

```
%Update screen
function update
global d

%Determine transformation via lens
[d.xx,d.yy] = lens( d.x,d.y,d.f );
[d.aa,d.bb] = lens( d.a,d.b,d.f );

%Update object and image
xlimits = get(gca,'xlim');
set( d.Obj,'xdata',d.x,'ydata',d.y );
set( d.Imag,'xdata',d.xx,'ydata',d.yy );
set( d.ray1,'xdata',[d.a,xlimits(1)],'ydata',[d.b,d.b*xlimits(1)/d.a]);
set( d.ray2,'xdata',[d.a,0,xlimits(1)],'ydata',[d.b,d.b,d.b -...
 xlimits(1)*d.b/d.f]);
set( d.ray3,'xdata',[0,d.f],'ydata',[d.b,0]);
set(d.ab,'xdata',d.a,'ydata',d.b);
set(d.aabb,'xdata',d.aa,'ydata',d.bb);
drawnow;

%%


% Lens virtual image position
function [xx,yy] = lens( x,y,f )
xx = ( ( 1./x + 1/f).^-1 );
yy = y.*xx ./x;

%End of code
```



Simulation #4: A concave spherical mirror. This is a different type of scenario, using reflection in a mirror rather than refraction via a lens system. As indicated by the curious 'flying cow' demonstration (see photograph below) a real, inverted image is manifested between the object and the mirror, giving the appearance of the image floating in space like a strange spectral form!

As in the previous examples, we can determine a transformation of object to image coordinates by finding the intersection of two rays. In this case it is the ray which passes through the circle centre and reflects straight back (since all circle radials are normal to the circle), and a horizontal ray which is reflected off the circular surface at angle θ to the normal

$$y = \frac{b}{a}x$$

'Straight through path'

$$y = -mx + c$$

'Horizontal path, reflected off the circle

Let circle have radius R and be centred on the origin

$$R^2 = x^2 + y^2$$

Hence coordinates of reflection point C are:

$$\left(-\sqrt{R^2 - b^2}, b\right)$$

Therefore reflection angle from circle normal is:

$$\theta = \tan^{-1}\left(\frac{b}{\sqrt{R^2 - b^2}}\right)$$

The reflected ray has gradient magnitude

$$m = \tan 2\theta$$

and can therefore be readily computed from input parameters to the simulation.

Now using the results above for the reflected ray:

$$b = m\sqrt{R^2 - b^2} + c$$

$$\therefore c = b - m\sqrt{R^2 - b^2}$$

$$\therefore c = -\left(m\sqrt{R^2 - b^2} - b\right)$$

The image coordinates can be found from the solution of:

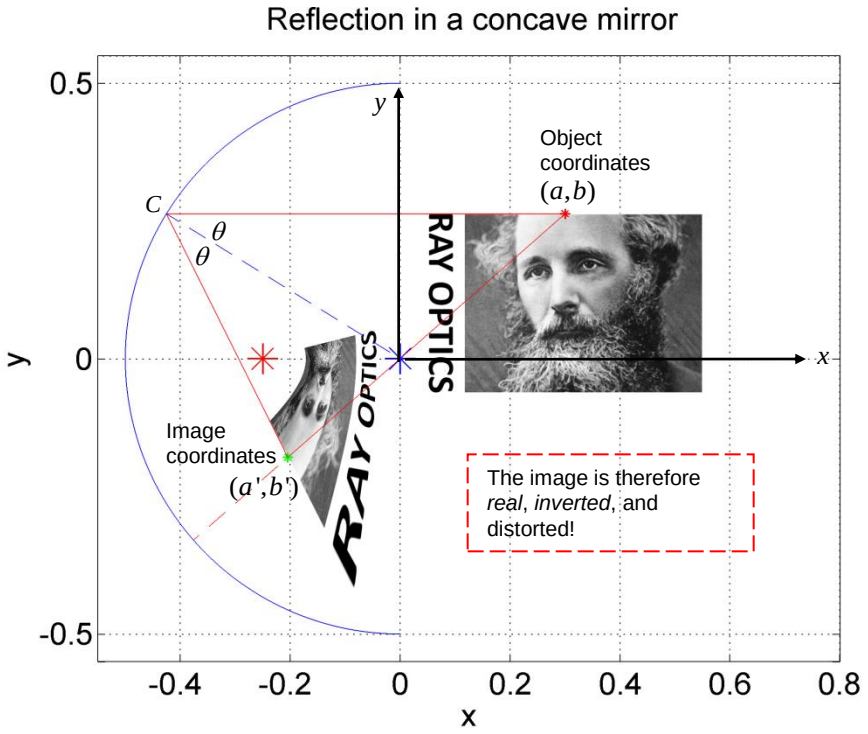
$$b' = \frac{b}{a}a', \quad b' = -ma' + c$$

$$\therefore \frac{b}{a}a' = -ma' + c$$

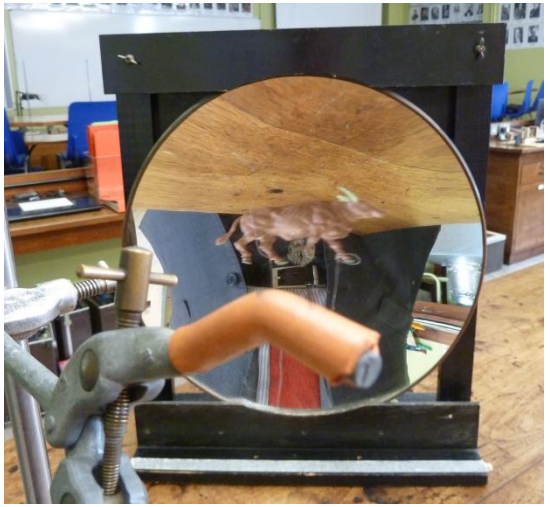
$$\therefore \left(\frac{b}{a} + m\right)a' = c$$

$$\therefore a' = -\frac{m\sqrt{R^2 - b^2} - b}{m + \frac{b}{a}} \qquad \theta = \tan^{-1}\left(\frac{b}{\sqrt{R^2 - b^2}}\right)$$

$$\therefore b' = -\frac{b}{a}\frac{m\sqrt{R^2 - b^2} - b}{m + \frac{b}{a}} \qquad m = \tan 2\theta$$

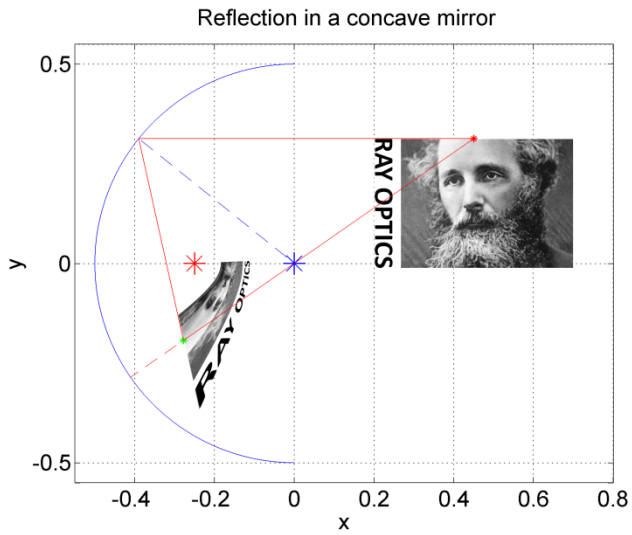


"Flying Cow" demonstration



MATLAB code for `concave_mirror.m`

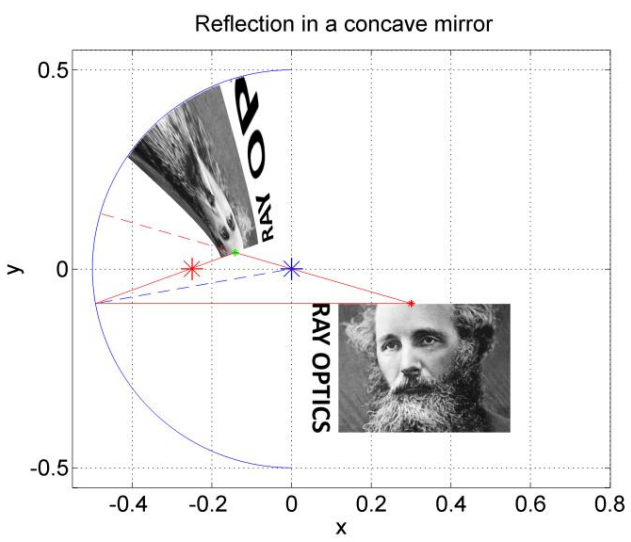
```
%concave_mirror
% Determines the image of an object in a spherical concave mirror.

function concave_mirror
global d

%Radius of mirror /m
d.R = 0.5;

%Object centre
x0 = 0.3;
y0 = 0.0;

%Width of object (this scales the image)
w = 0.5;

%Movement on keypress
d.delta = 0.05;

%Fontsize
fsize = 18;

%

%Import bitmap image of object
d.I = imread('Maxwell.jpg');
d.I = double(flipdim(d.I,1))/255;
dim = size(d.I);
Iwidth = dim(2);
Iheight = dim(1);

%Determine x,y coordinates of pixel locations
d.x = linspace(0,w,Iwidth) - w/2 + x0;
d.y = linspace(0,w*Iheight/Iwidth,Iheight) - 0.5*w*Iheight/Iwidth + y0 ;
[d.x,d.y] = meshgrid(d.x,d.y);

%Detrmine most left x coordinate of object
d.xmin = x0-w/2;

%Determine largest and smallest y coordinates of object
d.ymax = 0.5*w*Iheight/Iwidth + y0;
d.ymin = -0.5*w*Iheight/Iwidth + y0;

%Create figure and plot object
d.fig = figure( 'KeyPressFcn',@keypress );
d.Obj = surf(d.x,d.y,-ones(Iheight,Iwidth),d.I);
shading interp
view(2);
hold on;
axis equal
xlabel('x','fontsize',fsize);
ylabel('y','fontsize',fsize);
title('Reflection in a concave mirror','fontsize',fsize);
set(gca,'fontsize',fsize);
xlim([-d.delta-d.R,x0+w]);
ylim([-d.delta-d.R,+d.delta+d.R]);
axis manual
box on

%Determine mirror
t = linspace(0,pi,500);
xm = -d.R*sin(t);
ym = d.R*cos(t);
plot(xm,ym,'b')

%Plot mirror centre of curvature and approximate focus
plot(0,0,'b*','markersize',fsize);
plot(-d.R/2,0,'r*','markersize',fsize);

%Define a particular location on image to draw rays from
d.a = x0;
d.b = 0.5*w*Iheight/Iwidth + y0;

%Find intersection of horizontal ray with mirror x coordinate
d.xm = -sqrt( d.R^2 - d.b^2 );

%Determine transformation in mirror
[d.xx,d.yy] = mirror( d.x,d.y,d.R );
[d.aa,d.bb] = mirror( d.a,d.b,d.R );

%Plot transformed image
d.Imag = surf(d.xx,d.yy,-ones(Iheight,Iwidth),d.I);
shading interp

%Overlay rays from (d.a,d.b) to (d.aa,d.bb) via mirror
d.ray1 = plot( [d.a,d.xm,d.aa],[d.b,d.b,d.bb],'r-' );
d.ray2 = plot( [d.a,d.aa],[d.b,d.bb],'r-' );
d.ray3 = plot( [d.aa,-d.R/sqrt(1+(d.b/d.a)^2)],[d.bb,-(d.b/d.a)*d.R/sqrt(1+(d.b/d.a)^2)],'r--' );
d.ray4 = plot( [0,d.xm],[0,d.b],'b--' );
d.ab = plot( d.a,d.b,'r*' );
d.aabb = plot( d.aa,d.bb,'g*' );

%%
```



MATLAB code for `concave_mirror.m` (continued)

```
%Function which executes when a key is pressed
function keypress(fig,evnt)
global d

if strcmp(get(fig,'currentkey'),'uparrow')==1
    d.ymax = d.ymax + d.delta;
    if ( ( (d.xmin^2 + d.ymax^2) > d.R^2 ) || ( (d.xmin^2 + d.ymin^2) > d.R^2 ) ) && ( d.xmin < 0 )
        d.ymax =  d.ymax - d.delta;
    else
        d.y = d.y + d.delta;
        d.b = d.b + d.delta;
        d.ymin = d.ymin + d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'downarrow')==1
    d.ymin = d.ymin - d.delta;
    if ( ( (d.xmin^2 + d.ymax^2) > d.R^2 ) || ( (d.xmin^2 + d.ymin^2) > d.R^2 ) ) && ( d.xmin < 0 )
        d.ymin = d.ymin + d.delta;
    else
        d.y = d.y - d.delta;
        d.b = d.b - d.delta;
        d.ymax = d.ymax - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'leftarrow')==1
    d.xmin = d.xmin - d.delta;
    if ( ( (d.xmin^2 + d.ymax^2) > d.R^2 ) || ( (d.xmin^2 + d.ymin^2) > d.R^2 ) ) && ( d.xmin < 0 )
        d.xmin = d.xmin + d.delta;
    else
        d.x = d.x - d.delta;
        d.a = d.a - d.delta;
        update
    end
elseif strcmp(get(fig,'currentkey'),'rightarrow')==1
    d.x = d.x + d.delta;
    d.a = d.a + d.delta;
    d.xmin = d.xmin + d.delta;
    update
elseif strcmp(get(fig,'currentkey'),'p')==1
    print( d.fig, 'mirror.png','-dpng','-r300' );
end

%%
```

```
%Update screen
function update
global d

%Determine transformation in mirror
[d.xx,d.yy] = mirror( d.x,d.y,d.R );
[d.aa,d.bb] = mirror( d.a,d.b,d.R );

%Update object and image
set( d.Obj,'xdata',d.x,'ydata',d.y );
set( d.Imag,'xdata',d.xx,'ydata',d.yy );
d.xm = -sqrt( d.R^2 - d.b^2 );
set( d.ray1,'xdata',[d.a,d.xm,d.aa],'ydata',[d.b,d.b,d.bb]);
set(d.ray2,'xdata',[d.a,d.aa],'ydata',[d.b,d.bb]);
set(d.ray3,'xdata',[d.aa,-d.R/sqrt(1+(d.b/d.a)^2)],...
    'ydata',[d.bb,-(d.b/d.a)*d.R/sqrt(1+(d.b/d.a)^2)] );
set(d.ray4,'xdata',[0,d.xm],'ydata',[0,d.b]);
set(d.ab,'xdata',d.a,'ydata',d.b);
set(d.aabb,'xdata',d.aa,'ydata',d.bb);
drawnow;

%%

%Transformation in mirror
function [xx,yy] = mirror( x,y,R )
theta = atan( y./sqrt( -y.^2 + R^2 ) );
m = tan(2*theta);
c = y - sqrt( -y.^2 + R^2 ).*m;
xx = c./(y./x + m);
yy = y.*xx./x;
bad = ( ( yy.^2 + xx.^2 ) > R^2 ) |  ( ( -y.^2 + R^2 ) < 0 ) |  ( x < 0 );
xx(bad) = NaN;
yy(bad) = NaN;

%End of code
```



$$a' = -\frac{m\sqrt{R^2 - b^2} - b}{m + \frac{b}{a}}$$

$$b' = -\frac{b}{a}\frac{m\sqrt{R^2 - b^2} - b}{m + \frac{b}{a}}$$

$$\theta = \tan^{-1}\left(\frac{b}{\sqrt{R^2 - b^2}}\right)$$

$$m = \tan 2\theta$$

References

[1] Idea that *Fermat's principle* – ray paths correspond to those of *least time* – represents the *most probable* of all possible trajectories.
• FEYNMAN, R.P., *QED: The Strange Theory of Light and Matter*. Penguin Books. 1985.

[2] Derivation of the Law of Reflection and Snell's Law of Refraction using Fermat's Principle
• HECHT, E., *Optics*. Third Edition. Addison Wesley Longman, Inc. 1998 pp105.
• Reflection & Refraction Eclecticon notes.
http://www.eclecticon.info/physics_notes_waves.htm

[3] *Sellmeier* empirical equations for refractive index of glass.
https://refractiveindex.info

IGCSE (UK 16 year old examination) geometric Optics references

• ENGLAND, N., *Physics Matters*. Hodder & Stoughton. 3rd Edition 2001. pp199-2016
• POPLE, S., *Complete Physics for Cambridge IGCSE*. Oxford University Press. pp14-168.

Pre-U (UK 18 year old examination, pre-University Optics references

• ALLDAY, J., ADAMS, S., *Advanced Physics*. Oxford University Press. 2000. pp274.
• KIRK, T., *Physics for the IB Diploma*. Oxford University Press. 2nd Edition. 2007. pp157-163.
• MUNCASTER, R., *A-Level Physics*. Nelson Thornes. 4th Edition. 1993. pp361-419

University level general reference on Optics

• HECHT, E., *Optics*. Third Edition. Addison Wesley Longman, Inc. 1998
• WOAN, G., *The Cambridge Handbook of Physics Formulas*. Cambridge University Press. 2003. pp161-175.
• HyperPhysics website. Georgia State University. http://hyperphysics.phy-astr.gsu.edu

$$n^2 - 1 = \sum_{i=1}^{3} \frac{a_i \lambda^2}{\lambda^2 - b_i}$$

wavelength λ in microns

For BK7 Crown Glass:

$$a_1 = 1.03961212$$
$$a_2 = 0.231792344$$
$$a_3 = 1.01146945$$
$$b_1 = 0.00600069867$$
$$b_2 = 0.0200179144$$
$$b_3 = 103.560653$$

MATLAB code to plot the refractive index of crown glass vs wavelength for optical frequencies

```
%refractive_index
% Plots the refractive index over the visible range of light wavelengths,
% using the dispersion formula given in https://refractiveindex.info

function refractive_index

%Define range of wavelengths /nm
lambda_min = 400;
lambda_max = 800;
N = 1000;
lambda = linspace(lambda_min,lambda_max,N);

%Define refractive indices
n = crown_glass(lambda);

%Plot refractive index vs wavelength
figure('color',[1 1 1],'name','Refractive index')
fsize = 18;
plot(lambda,n,'r')
set(gca,'fontsize',fsize);
xlabel('\lambda /nm','fontsize',fsize);
ylabel('n','fontsize',fsize);
title('Refractive index of crown glass','fontsize',fsize);
grid on;
print(gcf,'crown glass.png','-dpng','-r300');

%%

%Dispersion formula for BK7 Crown Glass
% https://refractiveindex.info/?shelf=3d&book=glass&page=BK7
% wavelength lambda is in nm. n is the refractive index
function n = crown_glass(lambda)

%Convert to microns
x = lambda/1000;

%Sellmeier coefficients
a = [1.03961212, 0.231792344, 1.01146945];
b = [0.00600069867, 0.0200179144, 103.560653];

%Build up formula for refractive index
y = zeros(size(x));
for n=1:length(a)
    y = y + ( a(n)*x.^2 )./( x.^2 - b(n) );
end
n = sqrt( 1 + y );

%End of code
```

Colour	Wavelength λ /nm
Violet	380-450
Blue	450-495
Green	495-570
Yellow	570-590
Orange	580-620
Red	620-750