Using Colour and Light

Colormap

- MatLab using a numerical array with 3 columns to represent colour values. This array is called a **colormap**.Each row in the matrix represent a particular colour using numbers in the range from 0 to 1. These triplets of numbers making up a specific colour.
- An example of a colormap is the following is on the next slide.
- Some predefined colourmaps follow on the slide after that.

Red	Green	Blue	Color
1	0	0	Red
0	1	0	Green
0	0	1	Blue
1	1	0	Yellow
1	0	1	Magenta
0	1	1	Cyan
0	0	0	Black
1	1	1	White
0.5	0.5	0.5	Medium Gray
0.67	0	1	Violet
1	0.4	0	Orange
0.5	0	0	Dark Red
0	0.5	0	Dark Green

Colormap Function	Description
hsv	Hue-saturation-value (HSV) colormap
jet	Variant of hsv that starts with blue and ends with red
hot	Black to red to yellow to white
cool	Shades of cyan and magenta
summer	Shades of red and yellow
autumn	Shades of red and yellow
winter	Shades of blue and green
spring	Shades of magenta and yellow
white	All white
gray	Linear gray value
bone	Gray with a tinge of blue
pink	Pastel shades of pink
copper	Linear copper tome
prism	Alternating red, orange, yellow, green, blue and violet
flag	Alternating red, white, blue and black
lines	Alternating plot line colors
colorcube	Enhance color cube
vga	160-color Windows colormap (16 by 3)

Using Colormaps

- `colormap(M)` installs matrix **M** as the colormap to be used with the current figure window. `colormap(cool)` installs a 64 entry version of the cool colourmap. Most plotting functions, such as `mesh`, `surf`, `contour`, `fill` and `pcolor` and their variants, use the current colormap to determine colour sequences.

- The hot colormap `hot(8)` has the color triplets:

0.33333	0	0
0.66667	0	0
1	0	0
1	0.33333	0
1	0.66667	0
1	1	0
1	1	0.5
1	1	1

hot(8) contents

- The gray colormap `gray(5)` has the color triplets:

0	0	0
0.25	0.25	0.25
0.5	0.5	0.5
0.75	0.75	0.75
1	1	1

gray(5) contents

Pcolor and RGBplot

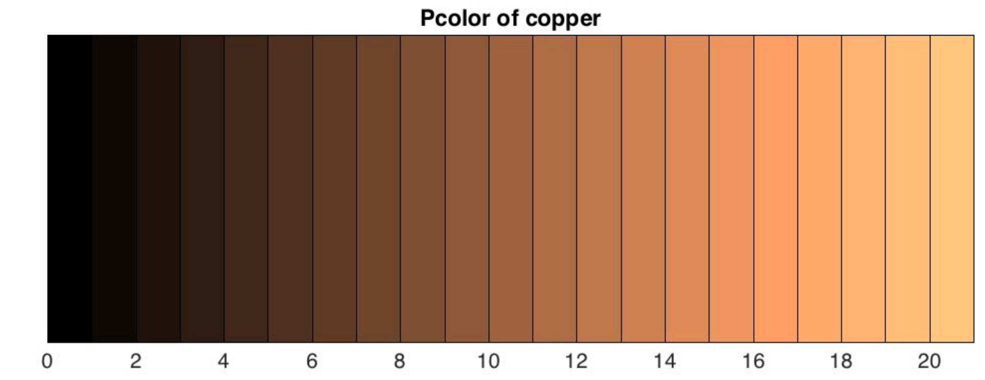
- A colormap is best visualized graphically. We can use the `pcolor` and `rgbplot` functions. **pcolor(C)** makes a pseudocolor plot as a rectangular array of cells with cell colours determined by C. MatLab creates a pseudocolor plot using each set of four adjacent points in C to define a surface rectangle. **rgbplot(map)** plots the three columns of **map**, where **map** is an $m \times 3$ colormap matrix. bf rgbplot draws the first column in red, the second column in green, and the third column in blue.

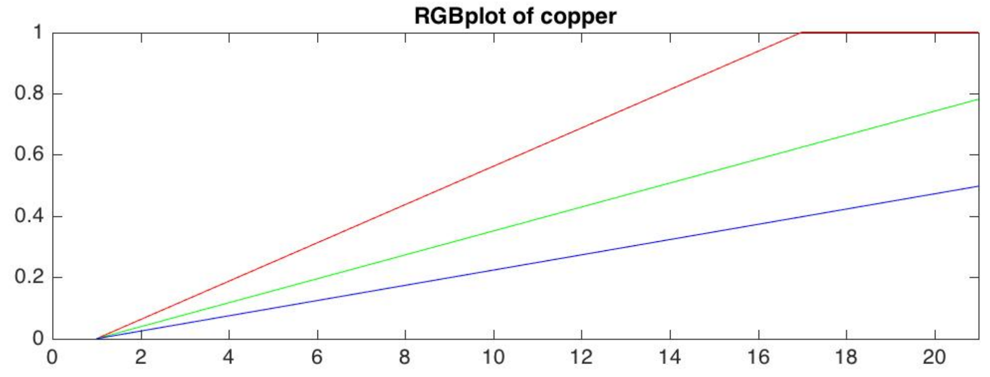
  Consider the following MatLab code (in **L09plot3CL1.m**):

  ```
  n=21;
  map=copper(n);
  colormap(map)
  subplot(2,1,1)
  [xx,yy]=meshgrid(0:n,[0 1]);
  ```

```
c=[1:n+1;1:n+1];
pcolor(xx,yy,c);
set(gca,'Yticklabel','');
title('Pcolor of copper');
subplot(2,1,2)
rgbplot(map);
xlim([0,n]);
title('RGBplot of copper')
print plotCL1.jpg -djpeg
```

This code produces:





Pcolor and RGBplot of copper

- The above figure shows how the copper colormap varies from its 'first row (on the left) to its last row (shown on the right). Each rectangule i has corner coordinates determined $xx(i)$, $xx(i+1)$, $yy(i)$ and $yy(i+1)$, where i varies from 0 to $n-1$.

- Then second figure shows that the `rgbplot` function simply plots the 3 columns of this colormap in red, green and blue, respectively.

3D plot with colorbar

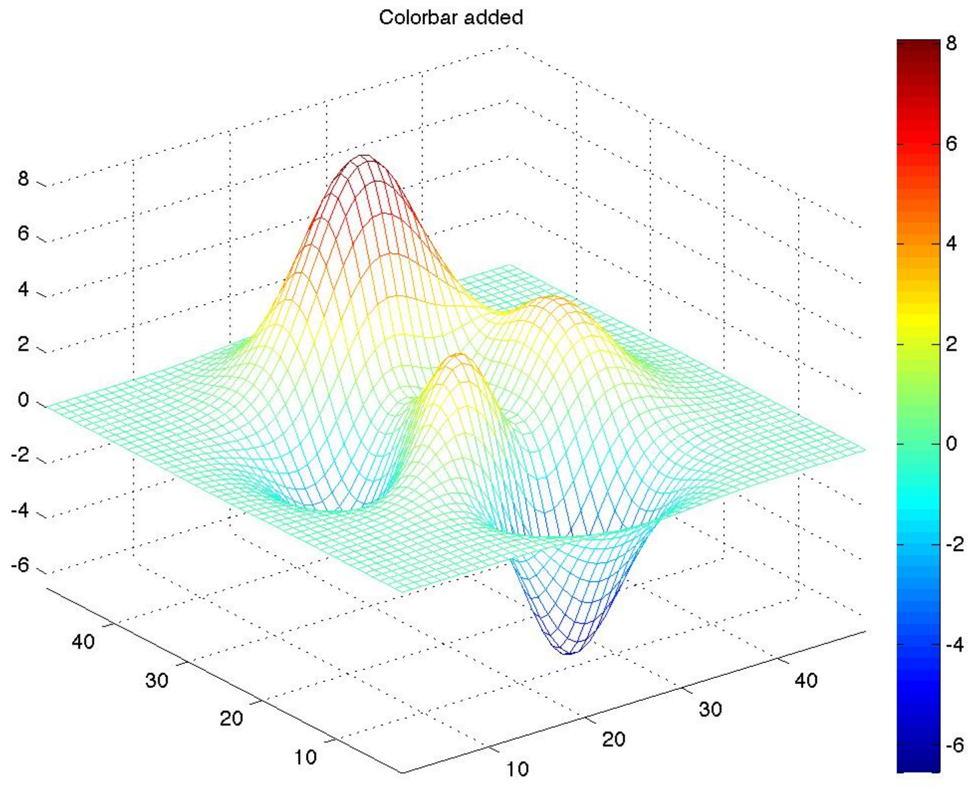
- The colour information in a plot can be displayed as auxiliary information using the `colorbar` function.

- The colorbar is associated with the z coordinate values with the colours in the `colormap`.

- Consider the following MatLab code (in **L09plot3CL2.m**)

```
mesh(peaks);
axis tight
colourbar
title('Colorbar added');
print plotCL2.jpg -djpeg
```

The following plot is produced:



Mesh of peaks with a colorbar

Using Colour as a 4^{th} Dimension

- The functions `surf(X,Y,Z)` is equivalent to `surf(X,Y,Z,Z)`, where the second `Z` specifies the colour of the surface according to the `Z` coordinate, which is the default.
- Colouring the surface with the depth ordering specified in `Z` doesn't give any new information as the 3D plot already gives the `Z` information. However, we can use a different colour argument (rather than `Z`) to give additional information.
- The following MatLab code (in **L09plot4D.m**):

```
x=-7.5:0.5:7.5;
```

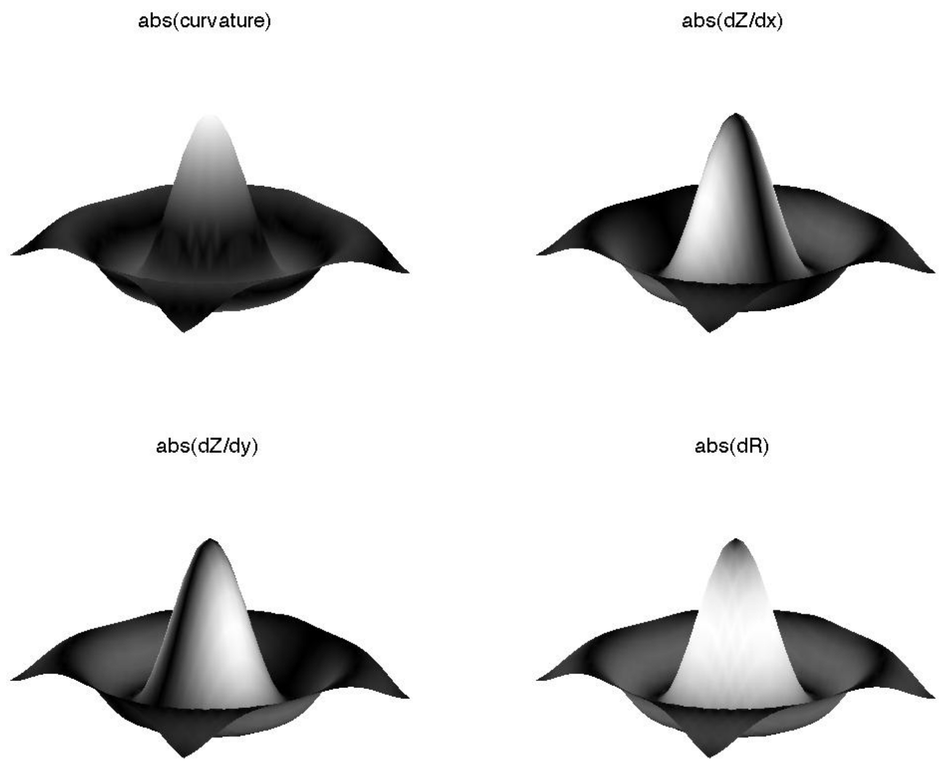
```
[X,Y]=meshgrid(x); % like meshgrid(x,x)
% eps stop R valiues from being exactly 0.0
R=sqrt(X.^2+Y.^2)+eps;
Z=sin(R)./R;

subplot(2,2,1)
% del2 is the discrete Laplacian
% colour is based on the surface curvature
surf(X,Y,Z,abs(del2(Z)))
colormap(gray)
shading interp
axis tight off
```

```
title('abs(curvature)');


subplot(2,2,2)
% Compute the gradient of the surface
% in the x and y dimensions
[dZdx,dZdy]=gradient(Z);
% abs of slope in x
surf(X,Y,Z,abs(dZdx))
colormap(gray)
shading interp
axis tight off
title('abs(dZ/dx)');
```

```
subplot(2,2,3)
% abs of slope in y
surf(X,Y,Z,abs(dZdy))
colormap(gray)
shading interp
axis tight off
title('abs(dZ/dy)');

subplot(2,2,4)
% abs of slope of radius
dR=sqrt(dZdx.^2+dZdy.^2);
```

```
surf(X,Y,Z,abs(dR))
colormap(gray)
shading interp
axis tight off
title('abs(dR)');

print plot4D.jpg -djpeg
```

Note that `colormap(gray)` colours the images with grayvalues. The code produces the following figure:



Colour as a 4th dimension

Transparency

- We can also use object transparency to show additional information in 3D visualization: for example, we can make object surfaces transparent or semi-transparent to show information normally hidden (similar to the `hidden` command for mesh plots we saw earlier).
- Transparency is controlled by an **alpha** value between 0 and 1. 0 means totally transparent and 1 means totally opaque.
- `alpha(x)`, where `0 <= x <= 1` sets the transparency to `x`.
- `alpha('str')`, where `str` is one of `'clear'`, `'opaque'`, `'flat'`, `'interp'` or `'texture'` sets appropriate alpha values for these situations (as MatLab sees fit).

- `alpha(M)`, where `M` is a mtrix the same size as the colour data, sets the alpha value for each element of the object.

- `alpha('alstr')`, where `alstr` is one of `'x'`, `'y'`, `'z'`, `'color'` or `'rand'`, sets the alpha values to be the same as the x data, the y data, the z data, the colour data or random values.

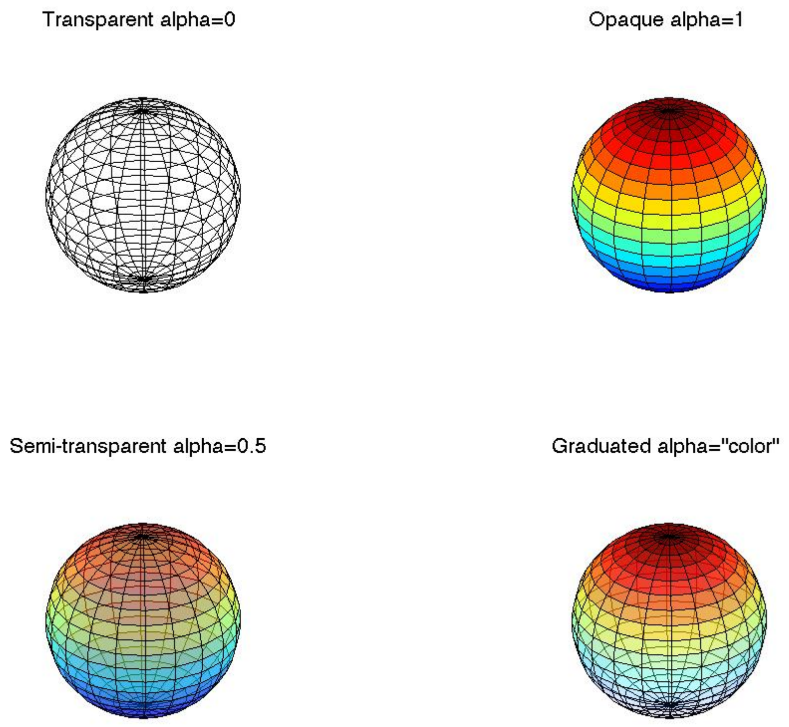
- The following MatLab code (in **L09plotTO**.m'):

```
subplot(2,2,1)
sphere
axis square off
alpha(0)
title('Transparent alpha=0')
```

```
subplot(2,2,2)

sphere

axis square off

alpha(1)

title('Opaque alpha=1')


subplot(2,2,3)

sphere

axis square off

alpha(0.5)

title('Semi-transparent alpha=0.5')
```

```
subplot(2,2,4)

sphere

axis square off

alpha('color')

title('Graduated alpha="color"')


print plotTO.jpg -djpeg
```

produces the plot:



Transparent, opaque, semi-transparent and graduated tranpsrency for a sphere

The alphamap function

- The `alphamap` function sets the alpha map for a figure.
- The following table shows some possibilities.

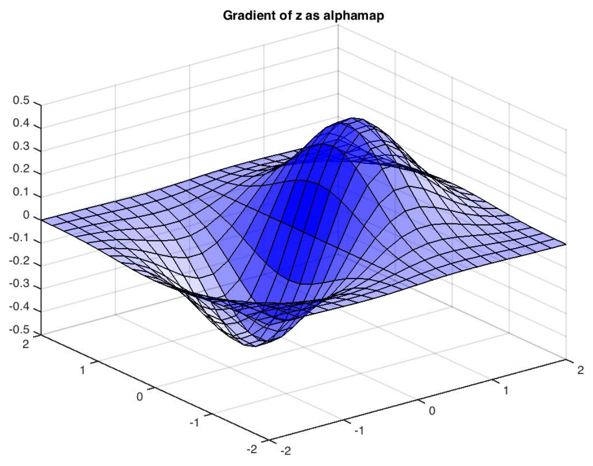
Alphamap function	**Description**
alphamap('default')	sets the alphamap to the default
alphamap('rampup')	linear alphamap with increasing opacity
alphamap('rampdown')	linear alphamap with decreasing opacity
alphamap('vup')	center transpareny, incrasing opacity towards each end
alphamap('vdown')	center opacity, decreassing opacity towards each end
alphamap('increase')	increases opacity by 0.1 across the map
alphamap('decrease')	increases transparency by 0.1 across the map
alpjamap(M)	sets the alphamap to m*1 array M contents
amap=alphamap	return the current alphamap to amap

Some available alphamap options

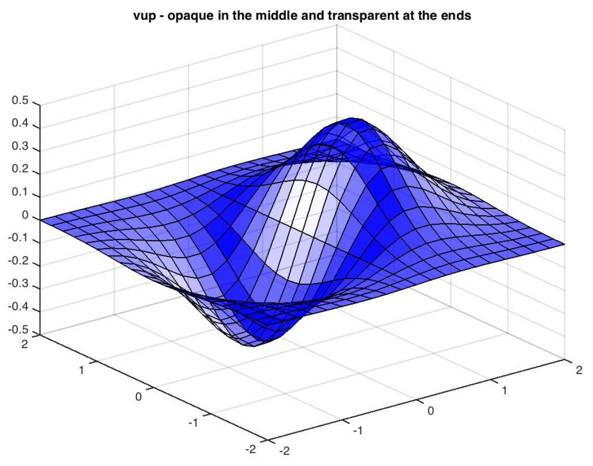
- Condier the following MatLab code (in **L09alphamap.m**):

```
[x,y] = meshgrid([-2:.2:2]);
z = x.*exp(-x.^2-y.^2);
figure
surf(x,y,z+.001,'FaceAlpha','flat',...
    'AlphaDataMapping','scaled',...
    'AlphaData',gradient(z),...
    'FaceColor','blue')
title('Gradient of z as alphamap');
print L09blue1.jpg -djpeg

alphamap('vup');
title('vup - opaque in the middle and transparent at the ends');
print L09blue2.jpg -djpeg

alphamap('vdown');
title('vup - transparent in the middle and transparent at the ends');
print L09blue3.jpg -djpeg
```

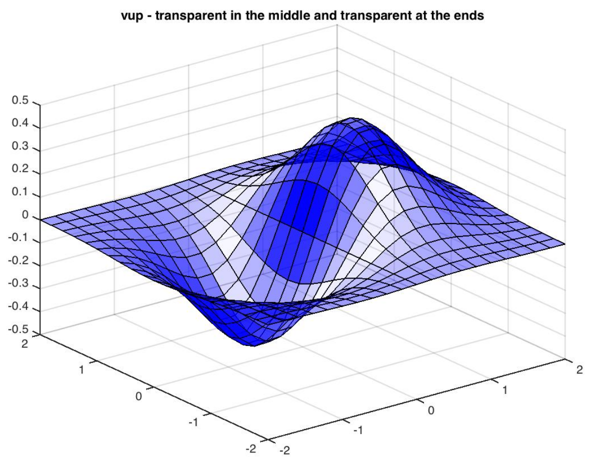
The following figures are plotted:



(a)



(b)



(c)

(a) Plot of a surface, using the gradient of z as the alphamap, (b) plot of the surface with varying alpha values so that the middle values are opaque (1'ish) and the end values are transparent (0'ish) and (c) plot of the surface with varying alpha values so that the middle values are transparent (0'ish) and the end values are opaque (1'ish).

Lighting

- Functions like `mesh` and `surf` render objects that appear well lit from all sides by diffuse light. Although the data can be visualized quite clearly, using lighting effects can **enhance** or **diminish** (!!!) the realism of the scene.

- A directional light source (direction `[1 0 1]`) can be created using `light`. A lighting calculation using a lighting model can then be done for surface points using this light source, a point's surface normal and knowledge of where the viewer is. Recall from an earlier lecture that a surface is tesselated into triangular or quadrilateral facets (polygons). MatLab has four lighting models:

1. `none` ignores any light source

2. `flat` shading is the default lighting when a light source is created. The colour of a surface facet is determined by a lighting calculation using the surface normal of its first vertex, then this colour value is used for all points within that surface.

3. `gouraud` shading or smooth shading determines the lighting calculation at each vertex of a facet and interpolated the colour elsewhere in the facet using bilinear interpolation.

4. `phong` shading computes the surface normals at each vertex of the facet, bilinearly interpolates these values everywhere in the facet and then does a full lighting calculation at each facet point using these surface normals. This is the most expensive shading calcula-

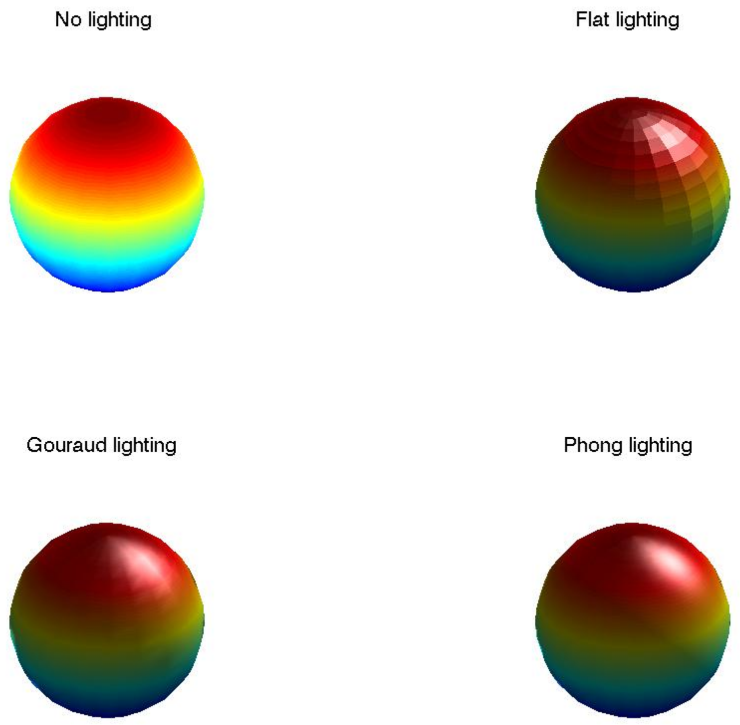
tion but gives the best results, especially around specularities. Flat or gouraud shading can completely eliminate a specularity if it occurs inside a facet or greatly exaggerate it if it occurs at a vertex of the facet but Phone shading will get these specularity correct. **Fast Phong** shading speeds things up considerably with the same quality results.

- The following MatLab code (in **L09plotShade.m**):

```
subplot(2,2,1)
sphere
light
shading interp
axis square off
lighting none
title('No lighting')

subplot(2,2,2)
sphere
light
shading interp
```

```
axis square off
lighting flat
title('Flat lighting')

subplot(2,2,3)
sphere
light
shading interp
axis square off
lighting gouraud
title('Gouraud lighting')

subplot(2,2,4)
sphere
light
shading interp
axis square off
lighting phong
title('Phong lighting')

print plotShade.jpg -djpeg
```

produces the following plot:



No, flat, gouraud and phone shading for a solid sphere.

- Matlab also has a material function that lets you shiny, dull, metal surfaces. You can set the amount of ambient light, diffuse and specular coeffecients of reflection, etc. but this is **not** a Computer Graphics course so full details are left to private investigation.
- One last comment: `light` is a **Hangle Graphics** object creation function (we'll cover Handle Graphics soon). For example:

  ```
  H1=light('Position',[x,y,z],'Color',[r,g,b],'Style','local')
  ```

  creates a light source at position `[x,y,z]` using light colour `[r,g,b]` and specifies the light in in the local 3D scene. It saves a handle (pointer) of the light object in `H1`. Then:

  ```
  set(H1,'Position',[1 0 1],'Color',[1 1 1],'Style','imfinite')
  ```

  sets the light sourse with handle `H1` to its default characteristics (direction `[1 0 1]`, white colour and a point light source (like the sun which is assumed to be ∞ away).