This document contains some information about light models.

It was written by Nils Schmeisser at Danish Institute of Fundamental Metrology.
You may not redistribute nor resale this document or parts of it.
If you use the document or parts of it you should place the folloing in your index section:

"Light Mode Subroutines for the SPM image processing tool", internal;
Nils Schmeisser, Danish Institute of Fundamental Metrology (DFM),
the SPM program (C) Jan Friis Joergensen, DFM

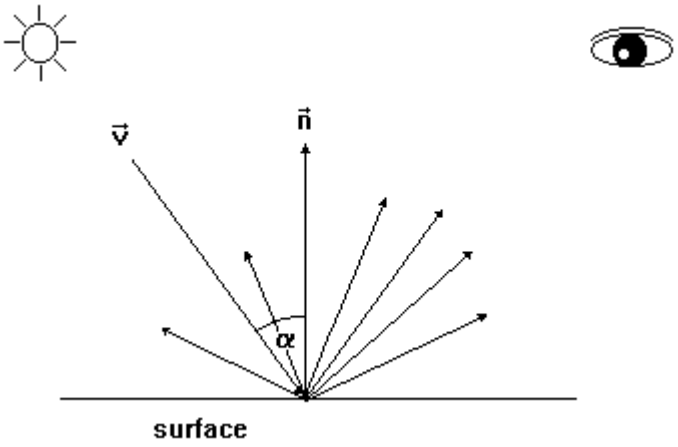
1. LightMode subroutines

1.1. The light models

First the used light models are discussed. There are at least three different types of lightning. The first describes the diffuse ligth, reflected by a "rough" surface (picture 1). To compute the reflected light, a model is used, which is known as the *Rule of Lampert*. It is described by

$$J_d = J_l \, k_l \, cos(\alpha) \qquad (1)$$

where J_l is the luminance of the light source, k_l is the diffuse reflectance constant of the material (i.e. a kind of roughness parameter) and α is the angle between the vector connecting the light source and the considered point and the normal vector of this point of the surface.



(pic. 1)

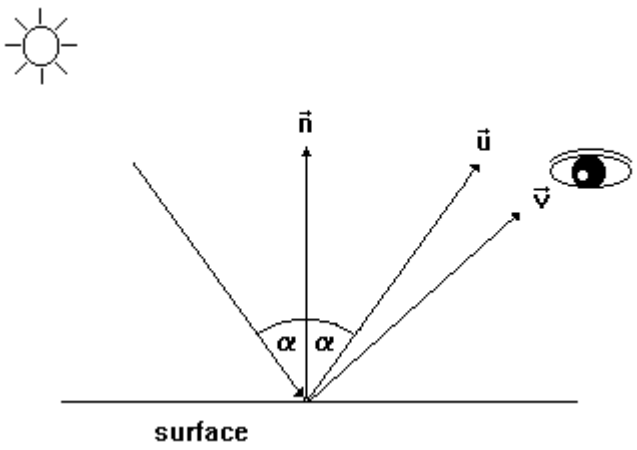
The next step is to think about a more flat surface. If we consider an ideal mirror, the situation is rather sad, because the viewer will only get light, if he looks exactly along a vector $\lambda\underline{v}+\mu\underline{n}$ with $\langle\underline{v},\underline{n}\rangle = \langle\lambda\underline{v}+\mu\underline{n},\underline{n}\rangle$. In this ideal case the reflected light is simply

$$J_r = J_l \qquad (2)$$

Because there are no ideal mirrors, a good approximation for the real surface can be done by the *Phong-Model*. It computes the reflected light in a certain direction. Formula (3) shows how the computations are done.

$$J_s = J_l \, \omega(\underline{v},\underline{u})^S \qquad (3)$$

where $\underline{u}$ is the vector pointing to the viewer and $\omega(\underline{v},\underline{n})$ is normaly a linear form of two vectors. Mostly the scalar product is used. S is again a kind of roughness parameter. The main concepts for *Phong's* model ar shown in picture 2.



(pic. 2)

An effect which should be taken into account is the border-angle of total reflectance, where for all $\alpha>\alpha_0$ the outgoing beam is parallel to the surface, which means, that for the case of total reflectance there is no outgoing light from the point.
Considering that we have a hot surface, we will figure out that the surface itself will emit light. So we come to the third kind of light, the *Emission*. There is no special model for it because the emission will be normaly homogenous all over the hot surface. Therefore the only formula we need is

$$J_e=J_{Emission} \tag{4}$$

Normal if we have a look into a scene we will find out that we will never have a real dark-room. This means that there will always be some ligth, lighning **all** points of the scene with the same luminance. This kind of light is called the *Ambient-Light*. It is again very simple to give a description for those kinds of light by

$$J_a=J_{Ambient}\,k_a \tag{5}$$

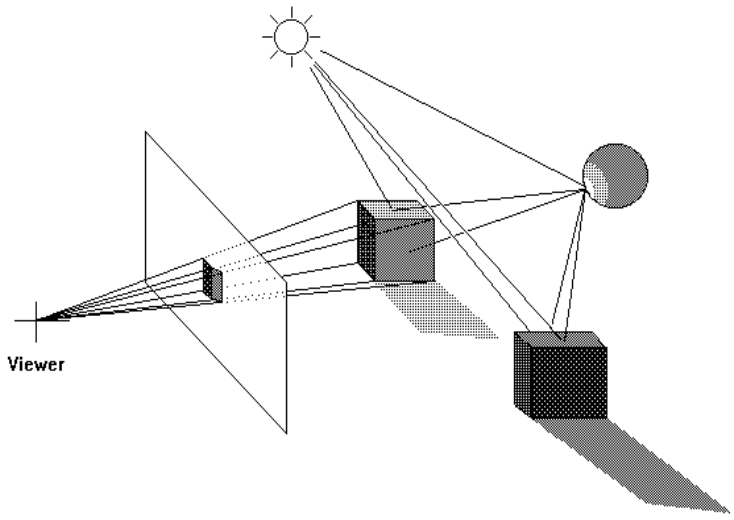
where $J_{Ambient}$ is the ambient light in the scene and k_a is a surface parameter.
The Luminance of a certain point can now be computed by adding all these *J's*. So we come to

$$J=J_l\,k_l\,cos(\alpha)+J_l\,\omega(\underline{v},\underline{u})^S+J_{Emission}+J_{Ambient}\,k_a \tag{6}$$

This formula is valid only for monochromatic light. For multiple coloured ligtht we simply have to chose a well fitting *Model of Light* like the *RGB-Model*. Then we have to do the computation for a point with both the *R, G* and *B* value. Assuming that we have a *True-Color* display it is very easy to display a point simply by writing the computed RGB-value to the screen.

1.2. The Raytracing algorithm

These computations won't work properly if we can not determine wether a light sources has effect on a point or not. Now an algorithm called *"Raytracing"* has to start his job. First we want to point to a situation shown in picture 3.



(pic. 3)

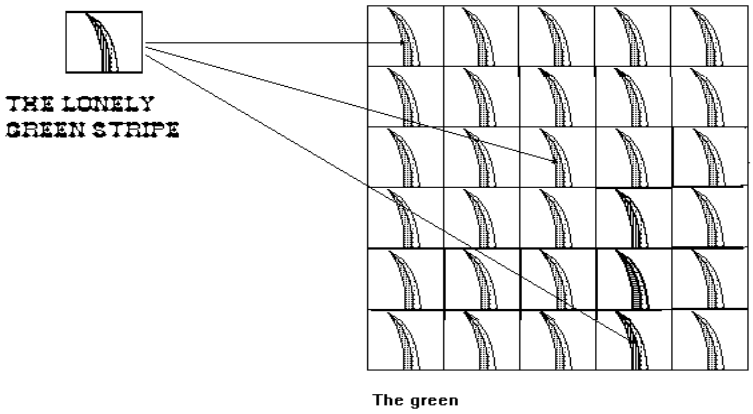
The raytracing simply works following these steps:
1. choose a point from the screen ($point_{2D}$)
2. check wether the beam (viewer,$point_{2D}$) targets a $point_{3D}$ of the scene's objects
3. if there is no $point_{3D}$ then set the $point_{2D}$ to the ambient color (light) and stop
4. check for each light source if $point_{3D}$ is visible from the light source
5. for each light source compute the J value and sum all of them
6. add the ambient light's J
7. set the $point_{3D}$ *the virtual viewers (VVP)* point
8. check rekursiv if there is any object in the scene where the *VVP*could get light from
9. for all *virtual light sources* compute the J's and add them to the already computed J

(e.g. compare the left cube in picture 3 and consider a point on the shadowed side of the cube as the *VVP* then both the ball an one side of the second cube will become a *virtual ligth source* for the *VVP*. The last steps are rather complicated so that these algorithm is very hard to implement for a common scene. Easy examples like balls over a chessboard are well known.)

1.3. Textures

For some applications it might be good to have a photo realistic output of the scene.
Textures give direct information about the surface facilities like roughness better then all other kinds of displaying images, e.g. if we think of a cup of tea the surface of the tea will never be flat and homogenous but always a little bit "stormy".
How does it work? Very simple, just make the surface *k-constants* a function of the surface, this means make the *k's* dependend on the position on the surface.

To give an example we want to describe how we could display a green of a soccer-field.
First of all we have to know something about the surface. Then we can try to detect wether there is some systematic in the surface. If we make a zoom in to the green so that we can see a lonely stripe we will find out that nearly all stripes look like this one. So we now map the *k-data* of this lonely strip to a matrix. If we now want to display just this lonely strip we only have to look into the matrix for getting the *k's*. But if we have more than one stripe? We just try to project the matrix over the surface. The result looks like picture 4.

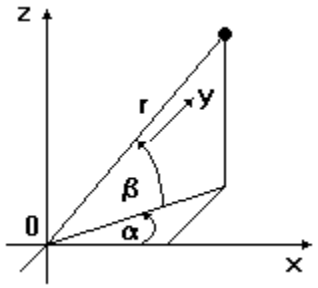


(pic. 4)

2. User interface of STM program

2.1. The coordinate system

The STM program will use homogenous polar coordinates, that means, that a position is defined by two angles and a distance to the center. If the distance is set to *0* then it is assumed that the point is infinite away from the center.

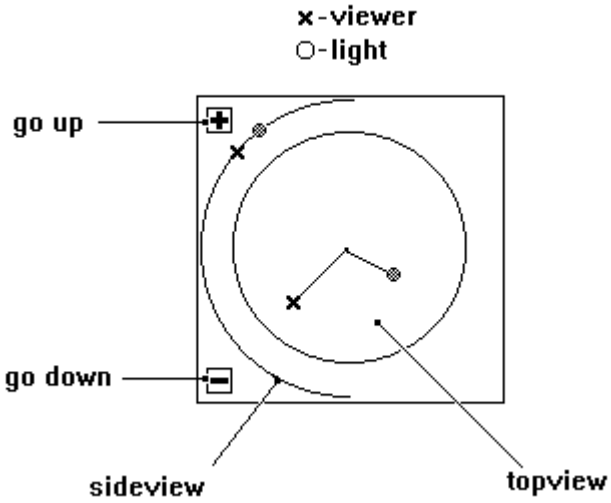


(pic. 5)

The angles are meassured by mathematical positive sense, this means positive in the direction of the arrows. α is also called the *azimuth* and can get values between -180.0 degree up to 180.0 degree. β is called *incidence* and can get values from -90.0 degree up to 90.0 degree. The *radius* or *distance r* can get values from 0 to ..., but the user should be aware that if he is too far away nothing is to be seen. Otherwise if *r=0* the viewer is infinitly far away, but another projection is used to display the image.

2.2. The interface

When clicking on the LightMode Button a view is poped up where several settings can be done.
There are two groups of values, the *position* and the *luminance* values.
The values of the first group can be edited in two different ways, simply by selecting the right input line and writing the value or clicking the *Mouse-Button* and the select and move an object. The user should try to imagine that the objects are moved on a sphere (see picture 6).



(pic. 6)

Bevor any displaying is done, the values for the material should be updated if necessary. That means the user can specify *k-constants* for *emission, diffusion* and *reflectance* (last is called *specularity*). If the material gots some reflectance properties the *shininess* which is simply the same as the *S* from the *Phong*-model should be set to a value bigger than 0.
Default values for the material are, given as *RGB-tripel*,

emission=(0,0,0),
diffusion=(1,1,1),
specularity=(0,0,0) and
shininess=0.

The next step is to define the ambient light in the scene (default is *(0,0,0)*).
If this is done, the lights themself should be checked. By pressing the selector button a light can be activated. Because of some programming reasons always the next light (from top) will be turned on or off. Giving the position in one of the above mentioned ways and setting the *RGB*-values the light is defined. If the *"White"* button is pressed then the *G* and the *B* value are set to the *R* value.
Default values are *RGB=(1,1,1).*
The values for the *k* constants, the *ambient* and the *RGB* for the lights have a range of *0.0 ... 1.0*. The *shininess* can get values from *0 ... 128*.
Now the *"Show"* button can be pressed and the result appears in the *Image Window*.

The *"Animation"* button creates a simple *turn around* for the image. Pressing the *"ColorMode"* button brings the user back to *ColorMode*. The editing areas for the lights and the material properties disappear from the *Light Edit Window* but the viewer position remains. The *"ColorMode"* button changes to a *"LightMode"* button.