Methods of Programming DV2, VT 2009

Project specification

`http://www.it.uu.se/edu/course/homepage/pm2/VT09`

1 Introduction

The purpose of the assignment is to develop a ray tracer in C and make a small movie. The purpose is *not* to teach you everything about ray tracing - rather it is about:

- writing good programs (i.e. programs which are easy to read, well-structured, well-documented, portable, easy to maintain etc.)
- using various UNIX-based development tools
- analysing the resource consumption of programs using profiling tools.
- optimising programs without making them ugly - that is without breaking the principles above
- planning a programming project
- deliver a program on time
- writing shell scripts
- analysing programming problems
- presenting your work in a written form

To solve the problem you will have to read relevant parts of the course literature and possibly also find other information sources.

How to carry out the project

- The assignments are to be done in groups of exactly three persons (except when one or two groups have to have 2 or 4 persons because the number of students taking the course is not a multiple of three or in exceptional circumstances after approval of the teacher responsible for the course).
- The TAs (primarily) and lecturer (secondary) will give you the help you need if you ask them.
- The assignments are to be solved using `make`, `yacc`, `lex`, `awk`, `prof`, `gprof`, `tcov`, `simics`, `gdb-simics`, `sh`, `lint` (or variants thereof) and additional C code.

- Read the *entire* assignment and understand all subassignments before you start. Giving some thought to the later subassignments early on will save you work later. I.e. subassignment 5 requires you to extract some documentation from the program code - you will need at the very beginning to decide on how comments are written to make this feasible.
- The major part of the work is spent doing subassignments 1 through 3. It is up to you how much time you spend on subassignments 4 and 5. It is easy to be very ambitious about optimising and movie making, but note that there are no absolute requirements on either optimisation or the length of the movie (or its artistic value...).
- Deadlines for submitting your work can be found on the course web page. Please note that there is a *final* deadline after which no further assessment of your work will be done.
- Exceptions from the deadlines should be requested from the lecturer (Lars-Henrik Eriksson) and will be granted *only* when there are particular reasons. You should ask for an exception as early as possible.
- Work submitted after the final deadline will *not* be graded. Clarification: "not" meaning "not at all, ever".

2 Reporting

For every subassignment, a written report shall be made, including extensive external documentation. "External documentation" does not **not** refer to explicit documentation in the program file, but rather a separate document written using LaTeX or a similar tool. Note that it is the program documentation that should be extensive – introductory parts of the report such as descriptions of the problem and project setting should be kept short – not more than 2-3 pages.

Reports submitted electronically should be in `pdf` format. **Not** Microsoft Word `.doc` format. Every report should have a cover page with only the authors' names, the number of the subassignment and a statement certifying that the authors have done the reported work themselves (see
`http://user.it.uu.se/~rolandb/front.html` for a sample statement).

Every report should be accompanied by a source code listing. Print the source code using `a2ps -4` or similar command in order to save paper. Make sure every header file (`.h`) is printed immedeately before the corresponding code file (`.c`). The command `a2ps -4 *.h *.c` will *not* give that result.

If the work has to be revised, the original copy of the first report shall be included so that the teaching assistants can read their original comments. It is a good idea to use the tool `diff` on the source code to help the TA compare the old and new code.

A solution to a subassignment is graded with one of the following grades:

U Failed ("underkänd"). The report is not taken as a serious attempt. The entire assignment is failed and there will be no opportunity to revise. This is of course quite serious but very rare in practise.

K Revise ("komplettering"). The work has substantial shortcomings which must be corrected before the work is continued. The revision shall be made a quickly as possible and submitted *before* the report on the next subassignment is submitted. Solutions to following subassignments will not be graded until the revised work has been passed.[1]

GK Passed requiring corrections ("Godkänd med komplettering"). The work has minor shortcomings that must be corrected. The corrections *shall* be made at the latest when the work on the next subassignment is submitted. If they are not, the next report will not be graded. You may - but need not - submit a separate, corrected, solution before submitting the solution to the next subassignment.

G Passed ("godkänd").

- Code shall be written in the spirit of the separately described coding rules.
- The executable program shall be named **tracer**.
- It shall be possible to compile and generate the executable program simply by writing "`make`" in the source code directory.
- The program shall still run and compile correctly even if its directory is changed.
- The program shall run on the department SUN Sparc Unix systems under Solaris 9 (SunOS 5.9).
- You must document and describe the development work itself. Major changes shall be stated and motivated. Bugs shall be described in terms of how they were discovered, how they were corrected and any consequental changes. You must also state the reason there was a bug in the first place.

  One way of doing this is keeping a file `ChangeLog`. This file can be maintained using the Emacs-command `C-x 4 a`. `ChangeLog` is your diary where you describe and date all changes and additions.

  Another way of doing it is using CVS, RCS or SCCS. This has the advantage that you can also access old versions and that you will get a history of changes. (Assuming that you check in your files regularly!)
- The report shall include a pointer to the source code distribution. By "distribution" we mean that every source code is archived and compressed using `tar`. Do "`tar -zcvf tracer-`*version*`.tar.gz *`".
- Any error messages should be writted to `stderr`.
- It shall be possible to build the movie by doing "`make movie`".
- Compiling your own C-files shall always be done with the following directives:
  `-Wall -Werror` (for `gcc`, the SUN C compiler (`/opt/SUNWspro/bin/cc`)

[1] You can - and should - begin working on the next subassignment even after getting a revise grade. You should plan ahead as much as possible. The purpose of this procedure is to prevent shortcomings from dragging along until the end of the course.

does the same thing as `-Wall` by default and `-errwarn` is used instead of `-Werror`). `gcc` (possibly the SUN C compiler as well) can have `-g` enabled together with optimisation.

- The code you submit shall follow the specification *exactly*. That means it shall do everything the specification says and also that it shall *not* do anything else. There should be nothing in the code to suggest additional functionality. In practise this means that we will not want to see any trace of any extensions that you do yourselves. The course is not about ray tracing but about writing beautiful and efficient programs.

  The reason is twofold: extensions make grading more time-consuming and it also keeps you from spending your time making the code beautiful and efficient. If we find that you have made extensions your solution will straightaway be graded K.

2.1 Interface

The finished ray traces shall read a scene description file from `stdin` and write the result to a file, the name of which is given as an argument. Additionally, it shall accept a number of options. Use the library `getopt` for handling options in a simple way.

The following options shall be supported:

-q Shadowing and reflection should not be done, i.e. the program shall behave as the solution to subassignment 2. This option need only be present in solutions to subassignments 3 through 5.

-r d Set the reflection depth **d**, i.e. the maximal number of reflections a ray can do. The default shall be 5. This option need only be present in solutions to subassigments 3 though 5.

-t The time spent on picture generation shall be written. The time shall be written to `stdout` in the format `Tracetime: 4711 ms`. Note that there *must* be spaces on either side of the number. The reported time shall not include time for parsing and writing out the picture. When this option is used, the program must not write anything to `stdout` beside the time message. This option need only be present in solutions to subassignments 4 and 5..

-z zoom Set the zoom value for the camera (see page 9).

-x xpix Set the number of pixels in the x direction.

-y xpix Set the number of pixels in the y direction.

-w width Set the width of the picture in world coordinate units.

-h height Set the height of the pictire in world coordinate units.

If only three out of the last four options is provided, the program shall compute the missing one under the assumption that $R_x = R_y$ (see page 9). If all four options are given, R_x and R_y will in general be different leading to distortion of the picture.

Part 1

Motivation: Warm-up in C programming and exercise in the writing of beautiful programs. The code shall be neat and abstract. You don't have to give any consideration to efficiency. (This means that efficiency should be a very minor factor when choosing a particular representation or implementation.) The code should be written so that it easy to *later* change the representation or implementation

1. Specify and implement an abstract data type for colours. Presently, it will suffice if you can create a single colour. You will not need to implement operations on colours until they are needed in later subassignments.

2. Specify and implement an abstract data type for bitmaps, i.e. matrices of coloured points (pixels). There shall be operations for reading and writing the bitmap in PPM format (see page 16), as well as manipulating the bitmap (changing the colour of individual pixels).

3. Specify and implement an abstract data type for vectors. Under the heading "Basic vector algebra" (page 8) you will find various useful operations to consider for inclusion in the ADT. The vectors shall be represented as three floating point numbers.

4. [Voluntary] Write a test program for each data type. Note that it is not sufficient to simply call a function - the result must be verified in some way. This can be done using the test program[2] and/or manually.

5. It is not compulsory to use `make` in the solution to this subassignment.

Part 2

Motivation: Using tools – `make` and a XML parser library.
Preparation: Read the introduction to ray tracing in the compendium.

Implement a simple ray tracer which can trace pictures with spheres, planes and triangles, with the camera in an arbitrary position (see page 9). The program need not handle shadowing or reflection. Instead, the colour intensity is made dependant on the incidence angle of the light ray. (see page 15). The result is that we pretend that we have a light source in the same spot as the camera.

Input data to the tracer is given in a file (see page 18) and output data (a picture in PPM format) is written to a file.

Input data (in XML format) shall be read using a XML parser library to be specified separately.

Also write a `Makefile` to

- compile the program (`make` without arguments)
- clean up after a compilation (`make clean`)
- create a `TAGS`-fil (voluntary)

The following items are important:

[2] Which leads to the question of the correctness of the test program.

- Test files can be found through the course web page. At this point, the program does not need to handle light sources, reflection etc. If the file includes such data, it can safely be ignored by the parser.

Some things to remember in order to save disk space:

- Remove `core` files when you do not need them anymore.
- Compress the PPM pictures you want to save using `gzip -9`.

Part 3

Motivation: Extending an existing program, using development support functions and writing portable and correct code using `lint`.

1. Extend the traces so that it can handle
   - light sources at arbitrary positions
   - projection of pictures on spheres and triangles
   - shadowing and reflection
2. The options `-q` and `-r d` shall be supported.
3. Analyse your source code using `lint` and attempt (this is a rather vague requirement) to eliminate all warnings. Warnings that cannot be eliminated shall be commented on in your report.

It is a good idea to extend the traces in steps and test it thoroughly after each step.

As before, test data can be found through the course web page.

Part 4

Motivation: Using profiling tools and practise optimisation on a C program.

1. The option `-t` shall be supported. You will find support routines for this in the directory `/it/kurs/pm2/vt09/timer`.
2. Use a profiling tool of your choise (e.g. `prof, gprof, tcov, lprof, simics`) to determine what must be changed to make the program be more efficient with regard to speed (primarily) and memory (secondarily). Modify the program and evaluate the improvements. Repeat the process a few times so that you obtain a substantial improvement. You should concentrate on minor improvements to the program, not major algorithmic changes.

The following items are important:

- Note that you shall not profile I/O.

- In your report, you shall present *relevant* results of the profiling together with code *before* and *after* **every** optimisation of the program. Also state - again for every optimisation - how the total runtime changes as a result of your modifications. You must state what aspect of the profiling data motivated each modification. Also note that you should not include the entire source code of each optimisation step, but only the changes made.

  Using `diff` to report on the changes is a good idea. (Assuming that you have kept the old version.) (And so you should.)

- In this context you often have to make a trade-off between elegance and performance. You shall state when this has happened and motivate the particular trade-off you made.

- There is no requirement that your ray tracer shall render a particular test picture within some given time. The purpose of this subassignment is to practise profiling and optimisation.

Part 5

Motivation: Using and writing shell scripts.

- Now the time has come to make a movie! The movie can show e.g. a camera movement in combination with some object or objects moving.

  Write a shell script to generate the movie:

  - The script shall take two arguments. The first argument shall be the full path to the ray tracer program. The second argument shall be a file with the names (one on each line) of the description files making up the frames of the movie. The specified tracer shall be used to generate the picture for each frame.
  - When all pictures have been generated, the total time spent on picture generation shall be written. The total time shall be computed by adding the results of giving the option `-t` to the tracer.
  - On the course web page, you will find sample descriptions which can be used as input to the movie script. You can also make something of your own.
  - To save disc space, it can be a good idea to compress the generated pictures.
  - Make a movie in mpeg format from the picture frames using `mpeg_encode` (a sample parameter file to `mpeg_encode` can be found in `/it/kurs/pm2/vt09/movie/film.param`).

  Watch the movie using e.g. `mpeg_play`.

- Write a shell script to extract interesting documentation from your source code files. "Interesting" documentation include the name of functions exported by a module, information in function headings etc. To make this work you must have decided on a particular format for code and comments so that the script can find the proper information to extract. It is a good idea to use some of the Unix text processing tools such as `awk` and `sed`.

You may format the extracted documentation in any way you want as long as it can be read in a straightforward way. In order words, it suffices with a neatly formatted text file. The industrious student may want to generate HTML, LaTeX, a man page, Emacs info, Postscript, pdf..

3 Basic concepts

3.1 Basic vector algebra

To solve the assignment it suffices to use the following basic vector operations.

$$\begin{aligned}
\mathbf{u} &= [u_1\ u_2\ u_3] \\
\mathbf{v} &= [v_1\ v_2\ v_3] \\
\text{dot product } \equiv \mathbf{u}\cdot\mathbf{v} &= u_1v_1 + u_2v_2 + u_3v_3 && (1)\\
\text{cross product } \equiv \mathbf{u}\times\mathbf{v} &= [u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1] && (2)\\
\mathbf{u}+\mathbf{v} &= [u_1+v_1, u_2+v_2, u_3+v_3] && (3)\\
\mathbf{u}*s &= [u_1*s, u_2*s, u_3*s] && (4)\\
\text{vector length } \equiv \|\mathbf{u}\| &= \sqrt{\mathbf{u}\cdot\mathbf{u}} && (5)\\
\text{normalise}(\mathbf{u}) &= \mathbf{u}*\frac{1}{\|\mathbf{u}\|} && (6)
\end{aligned}$$

A **ray** is described using a point of origin and a direction vector.

$$\begin{aligned}
\mathbf{R}_{\text{origin}} &\equiv \mathbf{R}_0 \equiv [X_0\ Y_0\ Z_0] \\
\mathbf{R}_{\text{direction}} &\equiv \mathbf{R}_\text{d} \equiv [X_\text{d}\ Y_\text{d}\ Z_\text{d}] \\
&\quad \text{där } X_\text{d}^2 + Y_\text{d}^2 + Z_\text{d}^2 = 1 \text{ (i.e. normalised)}
\end{aligned}$$

This defines a ray as:

$$\text{the points on the line } \mathbf{R}(t) = \mathbf{R}_0 + \mathbf{R}_\text{d} * t, \text{ where } t > 0. \quad (7)$$

An **intersection point** is described as a point on the surface hit by the ray ($\mathbf{r}_\text{i}$), a distance from the origin of the ray to the instersection point(t) och a vector normal (at right angles) to the surface. ($\mathbf{r}_\text{n}$).

$$\begin{aligned}
\text{intersection point} &\equiv \mathbf{r}_\text{i} \equiv [x_\text{i}\ y_\text{i}\ z_\text{i}] \\
\text{intersection normal} &\equiv \mathbf{r}_\text{n} \equiv [x_\text{n}\ y_\text{n}\ z_\text{n}] \\
\text{intersection distance} &\equiv t
\end{aligned}$$

3.2 Coordinate system

Note that the coordinate system used is a so-called left-handed system. A convenient and portable physical model of a left-handed system is obtained by spreading the thumb, index finger and middle finger on your left hand such that the angles between them is right (as far as you can without breaking a finger). Next, take a pen and write $\mathcal{X}$ on the thumb, $\mathcal{Y}$ on the index finger and $\mathcal{Z}$ on the middle finger.

A general "finger rule" for cross products is that if you want to compute $u \times v$, let the left hand thumb and index finger represent u and v, respectively. Then make the middle finger point at right angles to the other two fingers. The middle finger will point in the same direction as $u \times v$.

4 The camera

The camera can be described using the following parameters (the location of the camera, the point you turn the camera towards, a vector giving the "up" direction of the picture, a zoom value, the width and height of the picture in world coordinates and the pixels):

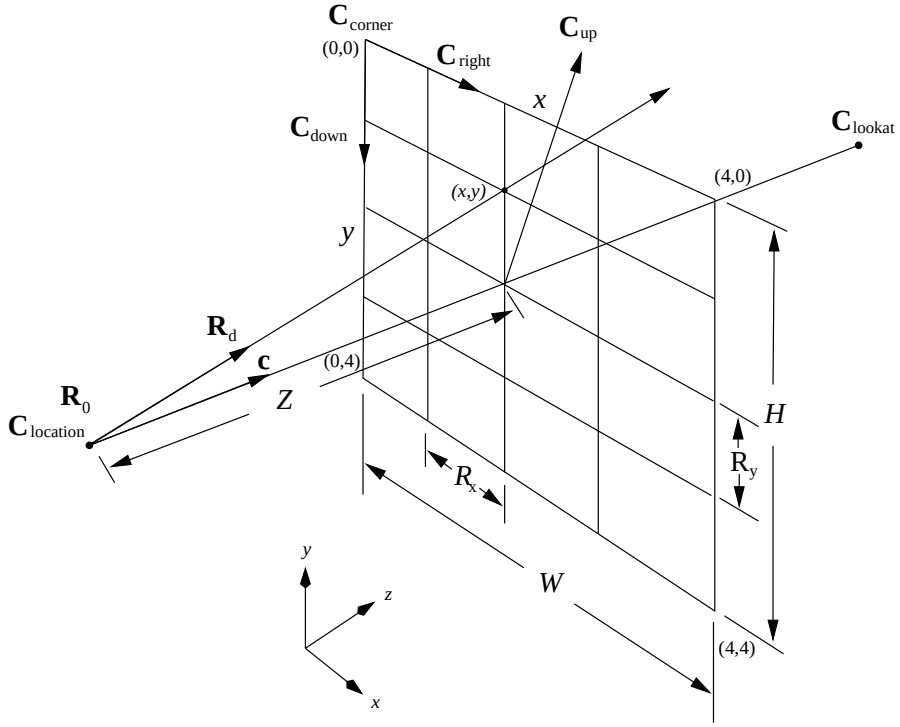
$$\begin{aligned}
\text{location} &\equiv \mathbf{C}_{\text{location}} \equiv \mathbf{C}_{\text{e}} \equiv [X_{\text{e}}\, Y_{\text{e}}\, Z_{\text{e}}] \\
\text{look at} &\equiv \mathbf{C}_{\text{lookat}} \equiv \mathbf{C}_{\text{l}} \equiv [X_{\text{l}}\, Y_{\text{l}}\, Z_{\text{l}}] \\
\text{up vector} &\equiv \mathbf{C}_{\text{up}} \equiv \mathbf{C}_{\text{u}} \equiv [X_{\text{u}}\, Y_{\text{u}}\, Z_{\text{u}}] \\
\text{zoom value} &\equiv Z \\
\text{width} &\equiv W \\
\text{height} &\equiv H \\
\text{pixel width} &\equiv X \\
\text{pixel height} &\equiv Y
\end{aligned}$$

The resolution (which is the link between pixels and world coordinates) is given by:

$$R_x = \frac{W}{X} \tag{8}$$

$$R_y = \frac{H}{Y} \tag{9}$$

The up vector need not be paralell to the picture nor need it be normalised.



Given x and y coordinates in the camera picture, a ray is computed ($\mathbf{R}_0$ and $\mathbf{R}_\mathrm{d}$). The coordinates of the picture are limited as follows:

$$0 \leq x \leq X$$

$$0 \leq y \leq Y$$

x and y can be used to describe pixels, in which case they will be integers. On the other hand, when implementing anti-aliasing you will want x and y to also cover the intervals between pixels.[3]

$$\mathbf{c} = \mathrm{normalise}(\mathbf{C}_\mathrm{l} - \mathbf{C}_\mathrm{e}) \quad (10)$$

$$\mathbf{C}_\mathrm{right} = \mathrm{normalise}(\mathbf{C}_\mathrm{u} \times \mathbf{c}) \quad (11)$$

$$\mathbf{C}_\mathrm{down} = \mathrm{normalise}(\mathbf{C}_\mathrm{right} \times \mathbf{c}) \quad (12)$$

$$\mathbf{C}_\mathrm{corner} = \mathbf{C}_\mathrm{e} + \mathbf{c} * Z - (\mathbf{C}_\mathrm{right} * W/2) - (\mathbf{C}_\mathrm{down} * H/2) \quad (13)$$

$$\mathbf{R}_0 = \mathbf{C}_\mathrm{e} \quad (14)$$

$$\mathbf{R}_\mathrm{d} = \mathrm{normalise}((\mathbf{C}_\mathrm{corner} + R_x * x * \mathbf{C}_\mathrm{right} + R_y * y * \mathbf{C}_\mathrm{down}) - \mathbf{C}_\mathrm{e}) \quad (15)$$

1. Compute the normalised direction vector of the camera $\mathbf{c}$.
2. Compute the normalised right direction vector of the picture. ($\mathbf{C}_\mathrm{right}$).
3. Compute the normalised down direction vector of the picture. ($\mathbf{C}_\mathrm{down}$).
4. Compute the upper left hand corner of the picture ($\mathbf{C}_\mathrm{corner}$).
5. Compute the direction vector of the desired ray $\mathbf{R}_\mathrm{d}$. The origin of the ray is $\mathbf{C}_\mathrm{e}$ ($\mathbf{C}_\mathrm{location}$ in the picture).

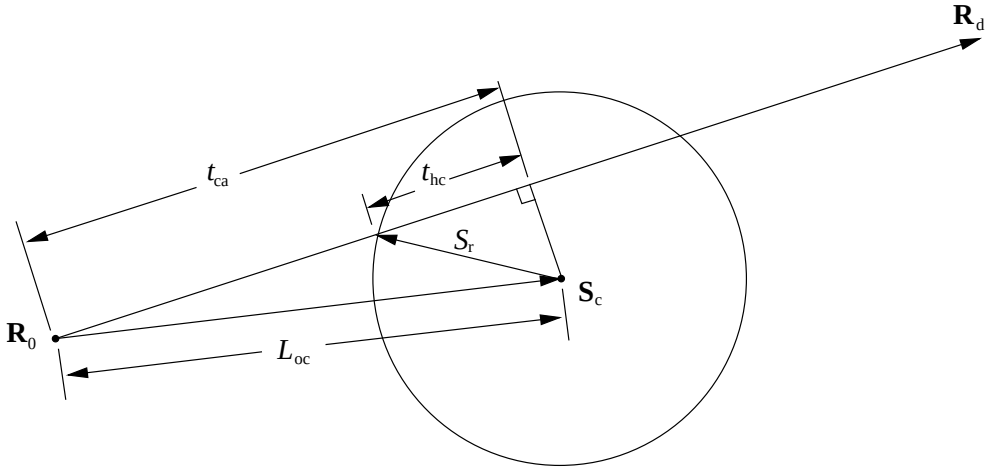
5 Intersection point computations

5.1 Spheres

A sphere is defined by its centre and radius.

$$\begin{aligned} \text{Sphere's center} &\equiv \mathbf{S}_\mathrm{c} \equiv [X_\mathrm{c}\ Y_\mathrm{c}\ Z_\mathrm{c}] \\ \text{Sphere's radius} &\equiv S_\mathrm{r} \end{aligned}$$

[3] Note that implementing anti-aliasing is not part of the basic assignment.



Algorithm

$$\mathbf{OC} = \mathbf{S}_c - \mathbf{R}_0 \tag{16}$$

$$L_{oc}^2 = L_{2oc} = \mathbf{OC} \cdot \mathbf{OC} \tag{17}$$

$$t_{ca} = \mathbf{OC} \cdot \mathbf{R}_d \tag{18}$$

$$t_{hc}^2 = t_{2hc} = S_r^2 - L_{2oc} + t_{ca}^2 \tag{19}$$

$$t = \begin{cases} t_{ca} - \sqrt{t_{2hc}} & \text{for rays outside the sphere} \\ t_{ca} + \sqrt{t_{2hc}} & \text{for rays inside the sphere} \end{cases} \tag{20}$$

$$\mathbf{r}_i = \mathbf{R}_0 + \mathbf{R}_d * t \tag{21}$$

$$\mathbf{r}_n = \begin{cases} (\mathbf{r}_i - \mathbf{S}_c) * 1/S_r & \text{for rays outside the sphere} \\ (\mathbf{r}_i - \mathbf{S}_c) * -1/S_r & \text{for rays inside the sphere} \end{cases} \tag{22}$$

1. Compute the square of the distance from the origin of the ray to the centre of the sphere (L_{2oc}). If $L_{2oc} < S_r^2$, the origin of the sphere will be inside the sphere, if not the origin is outside the sphere and the ray may miss the sphere.

2. Compute the distance from the origin of the ray to that point on the ray closest to the centre of the sphere (t_{ca}). For rays originating outside the sphere, $t_{ca} < 0$ implies that the ray will not strike the sphere.

3. Compute the square of the distance between the point on the ray closest to the centre of the sphere and subtract this distance from the square of the radius of the sphere. If $t_{2hc} < 0$ the ray will not strike the sphere.

4. Compute the intersection distance (t).

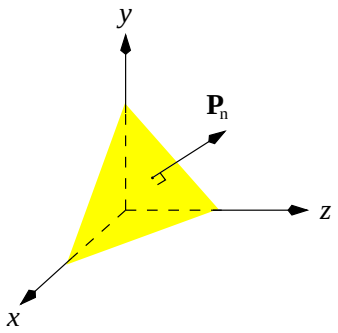
5. Compute the intersection point ($\mathbf{r}_i$).

6. Compute the vector normal (at right angles) to the surface at the intersection point ($\mathbf{r}_n$).

5.2 Planes

A plane is defined using two parameters; a normal (right angle) vector ($\mathbf{P}_\mathrm{n}$) and a point on the plane ($\mathbf{P}_0$).

$$\begin{aligned} \mathbf{P}_\mathrm{normal} &\equiv \mathbf{P}_\mathrm{n} \equiv [A\ B\ C] \quad (\mathbf{P}_\mathrm{n} \text{ is normalised}) \\ \mathbf{P}_\mathrm{point} &\equiv \mathbf{P}_0 \equiv [X_0\ Y_0\ Z_0] \end{aligned}$$



Algorithm

$$D = \mathbf{P}_0 \cdot \mathbf{P}_\mathrm{n} \tag{23}$$

$$v_\mathrm{d} = \mathbf{P}_\mathrm{n} \cdot \mathbf{R}_\mathrm{d} \tag{24}$$

$$v_0 = D - \mathbf{P}_\mathrm{n} \cdot \mathbf{R}_0 \tag{25}$$

$$t = v_0 / v_\mathrm{d} \tag{26}$$

$$\mathbf{r}_\mathrm{i} = \mathbf{R}_0 + \mathbf{R}_\mathrm{d} * t \tag{27}$$

$$\mathbf{r}_\mathrm{n} = \begin{cases} \mathbf{P}_\mathrm{n} & \text{if } v_\mathrm{d} < 0 \\ -\mathbf{P}_\mathrm{n} & \text{if } v_\mathrm{d} \geq 0 \end{cases} \tag{28}$$

1. Compute the distance from the origin of the coordinate system $(0, 0, 0)$ to the plane (D). You don't need to do this if the plane is specified according to the POVrays format.

2. Compute v_d and compare with zero. If v_d is equal to zero, the ray is paralell to the plane and will not strike it.

3. Compute v_0 and t (the distance) and compare t to zero. If $t < 0$ the ray will not strike the plane.

4. Compute the intersection point $\mathbf{r}_\mathrm{i}$ and normal vector $\mathbf{r}_\mathrm{n}$.

5.3 Triangles

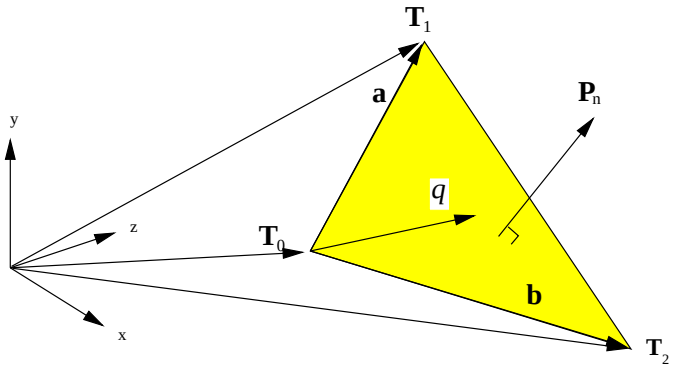
A triangle is defined using three parameters – an origin and two vectors giving two of the legs of the triangle. Alternatively a triangle can be defined by three points.

$$\text{triangle origin (corner 0)} \equiv \mathbf{T}_0 \equiv [X_0\ Y_0\ Z_0]$$

$$
\begin{aligned}
\text{triangle corner 1} &\equiv \mathbf{T}_1 \equiv [X_1\ Y_1\ Z_1] \\
\text{triangle corner 2} &\equiv \mathbf{T}_2 \equiv [X_2\ Y_2\ Z_2] \\
\text{triangle vector a} &\equiv \mathbf{a} \equiv [X_\mathrm{a}\ Y_\mathrm{a}\ Z_\mathrm{a}] \\
\text{triangle vector b} &\equiv \mathbf{b} \equiv [X_\mathrm{b}\ Y_\mathrm{b}\ Z_\mathrm{b}]
\end{aligned}
$$

It is straight-forward to convert between the two representations:

$$
\begin{aligned}
\mathbf{T}_1 &= \mathbf{T}_0 + \mathbf{a} \\
\mathbf{T}_2 &= \mathbf{T}_0 + \mathbf{b}
\end{aligned}
$$



Algorithm

$$
\begin{aligned}
\mathbf{P}_\mathrm{n} &= \text{normalise}(\mathbf{a} \times \mathbf{b}) && (29) \\
D &= \mathbf{T}_0 \cdot \mathbf{P}_\mathrm{n} && (30) \\
q &= \mathbf{r}_\mathrm{i} - \mathbf{T}_0 && (31) \\
u &= \frac{(\mathbf{b}\cdot\mathbf{b})(q\cdot\mathbf{a}) - (\mathbf{a}\cdot\mathbf{b})(q\cdot\mathbf{b})}{(\mathbf{a}\cdot\mathbf{a})(\mathbf{b}\cdot\mathbf{b}) - (\mathbf{a}\cdot\mathbf{b})^2} && (32) \\
v &= \frac{(q\cdot\mathbf{b}) - u(\mathbf{a}\cdot\mathbf{b})}{\mathbf{b}\cdot\mathbf{b}} && (33)
\end{aligned}
$$

The points q in the triangle can be described by $q = u\mathbf{a} + v\mathbf{b}$ and three conditions.

$$
\begin{aligned}
0 \le u \le 1 \\
0 \le v \le 1 \\
u + v \le 1
\end{aligned}
$$

1. Compute the normal vector to the triangle's plane ($\mathbf{P}_\mathrm{n}$).
2. Compute the distance from the origin of the coordinate system $(0, 0, 0)$ to the plane (D).
3. Check that the ray strikes the plane using the algorithm for planes. If the ray does strike the plane, you will obtain $\mathbf{r}_\mathrm{i}$, $\mathbf{r}_\mathrm{n}$ och t. It remains to determine if the intersection point is inside the triangle.

4. Compute the vector q between the origin of the triangle and the intersection point.

5. Compute the **a**-coordinate of the intersection point (u). The ray doesn't strike unless
$0 < u < 1$.

6. Compute the **b**-coordinate of the intersection point (v). The ray doesn't strike unless
$0 < v < 1$ and $u + v \leq 1$.

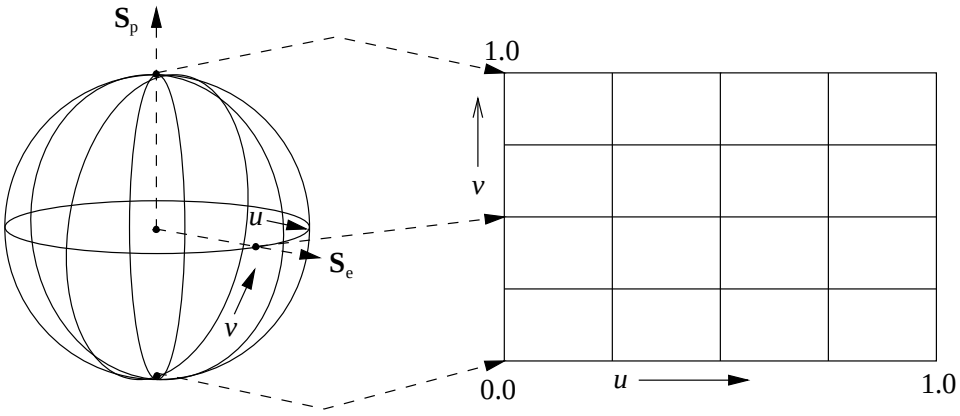
6 Inverse mappings

6.1 Spheres

Instead of choosing the colour of the sphere, a bitmap can be mapped onto the sphere, the colour of each point of the bitmap determining the colour of the corresponding point of the sphere.

To describe the mapping, we need the north pole and equator of the sphere:

$$\begin{aligned} \mathbf{S}_{\text{pole}} &\equiv \mathbf{S}_{\text{p}} \equiv [X_{\text{p}}\, Y_{\text{p}}\, Z_{\text{p}}] \\ \mathbf{S}_{\text{equator}} &\equiv \mathbf{S}_{\text{e}} \equiv [X_{\text{e}}\, Y_{\text{e}}\, Z_{\text{e}}] \end{aligned}$$



$\mathbf{S}_{\text{p}}$ and $\mathbf{S}_{\text{e}}$ shall be normalised and at right angles to each other (i.e. $\mathbf{S}_{\text{p}} \cdot \mathbf{S}_{\text{e}} = 0$).

$$\phi = \arccos(-\mathbf{r}_{\text{n}} \cdot \mathbf{S}_{\text{p}}) \tag{34}$$

$$v = \phi/\pi \tag{35}$$

If v is equal to one or zero, u is defined to be zero, otherwise u is computed as follows:

$$\theta = \frac{\arccos((\mathbf{S}_{\text{e}} \cdot \mathbf{r}_{\text{n}})/\sin(\phi))}{2\pi} \tag{36}$$

$$u = \begin{cases} \theta & \text{if } ((\mathbf{S}_{\text{e}} \times \mathbf{S}_{\text{p}}) \cdot \mathbf{r}_{\text{n}}) > 0 \\ 1 - \theta & \text{if } ((\mathbf{S}_{\text{e}} \times \mathbf{S}_{\text{p}}) \cdot \mathbf{r}_{\text{n}}) \leq 0 \end{cases} \tag{37}$$

The coordinates u and v vary between 0 och 1 (that is, they describe a fraction of the width and height of the bitmap). Not that the origin of a bitmap is usually in the upper left-hand corner, so you might want to convert the coordinate system unless you want the picture upside-down.

Also note that the argument to arccos in equation 36 may have an absolute value larger than 1 due to roundoff errors. If you don't take this into consideration, the inverse mapping will be incorrect in some cases.[4]

6.2 Triangles

Instead of choosing the colour of the triangle, a bitmap can be mapped onto the triangle, in the same way as on a sphere. The computed coordinates v and u can be used as fractional coordinates of a bitmap.

7 Shading

To determine if an intersection point is lighted or in shade, you trace a ray from the intersection point to every light source ($\mathbf{S}_0$ och $\mathbf{S}_\mathrm{d}$). Om the ray doesn't strike any object between the intersection point and the light, the point is lighted. The amount of light ($\cos\phi$) is computed as shown below. The model assumes diffuse reflection of the light, i.e. equal amounts of light is reflected in all directions.

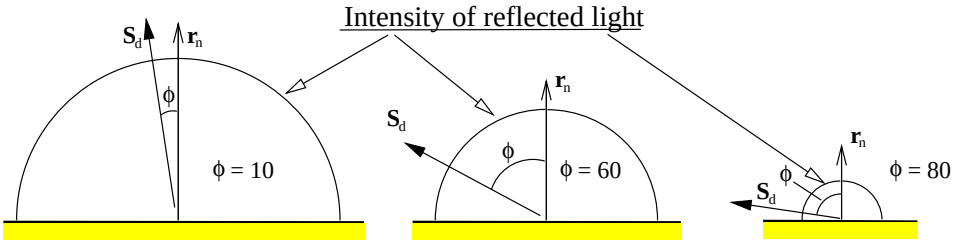
To prevent roundoff errors from putting the intersection point on the wrong side of the intersected surface, the intersection point is offset slightly in the direction away from (at right angles to) the surface. (e.g. $\varepsilon = 0.0001$ units).

$$\text{light origin} \equiv \mathbf{L}_0 \tag{38}$$

$$\mathbf{S}_0 = \mathbf{r}_\mathrm{i} + \varepsilon * \mathbf{r}_\mathrm{n} \tag{39}$$

$$\mathbf{S}_\mathrm{d} = \text{normalise}(\mathbf{L}_0 - \mathbf{S}_0) \tag{40}$$

$$\cos\phi = \text{abs}(\mathbf{r}_\mathrm{n} \cdot \mathbf{S}_\mathrm{d}) \tag{41}$$



The colour of the object is multiplied with $\cos\phi$ and the coefficient for diffuse reflection (given by **diffuse** in the grammar). If the intersection point is lighted by more than one light source, the different colour components are added together.

[4] When the point you are looking from is on the plane determined by the pole and equator vectors.

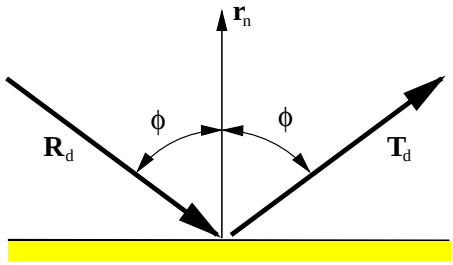
8 Reflection

If a surface is reflecting, the reflected ray is traced ($\mathbf{T}_0$ and $\mathbf{T}_\mathrm{d}$). The resulting colour value, multiplied with the reflection coefficient (given by **reflection** in the grammar), is combined with the colour of the intersection point. The direction of the reflected ray is computed as shown below.

The intersection point is offset in the same way as for shading.

$$\text{reflected ray's origin} \equiv \mathbf{T}_0 = \mathbf{r}_\mathrm{i} + \varepsilon * \mathbf{r}_\mathrm{n} \quad (42)$$

$$\text{reflected ray's direction} \equiv \mathbf{T}_\mathrm{d} = \mathbf{R}_\mathrm{d} - 2 * (\mathbf{r}_\mathrm{n} \cdot \mathbf{R}_\mathrm{d}) * \mathbf{r}_\mathrm{n} \quad (43)$$



9 Colour

Every point of a picture can be described using three values, (r, g, b). These values signify the amount of red, green and blue in the point. The maximal colour value for each colour is usually 255. A completely black point has colour $(0, 0, 0)$, a completely white point has colour $(255, 255, 255)$ a clear green point $(0, 255, 0)$ etc. In the file `/it/kurs/pm2/vt09/color/rgb.txt` you will find sample rgb-colours and their names.

Another representation is letting the components vary between 0 and 1 (inclusive). This will make some operations simpler, but demands more storage space.

The colour of a certain point is a combination of the colours resulting from the point being lighted (by zero or more light sources) and any reflected light.

A simple way of combining colours is adding their respective r, g, and b values and limit the sum to the maximal value. This makes a strongly lighted point white. It is slightly better to adjust the other components if any one component becomes too large. You keep the hue, but it becomes weaker.[5]

You can adjust (scale) colours in the same way, by adjusting every component separately.

10 PPM

The rendered picture should be written using the PPM format. There are binary and ASCII variants of the format. Your program shall be able to read and write both variants. The format is:

[5] ..and you should properly also adjust all other points in the picture. The drawback is that the picture will be very dark if there are a few strongly lightened points. Colour is difficult to do well.

- A magic number. The ASCII variant has P3 and the binary has P6.
- Whitespace (one or more of blankspace, TAB, CR or LF).
- The width of the picture (in pixels) as a decimal number in ASCII.
- Whitespace.
- The height of the picture (in pixels) as a decimal number in ASCII.
- Whitespace.
- The maximal colour component value (you need only handle 255) as a decimal number in ASCII.
- Whitespace. In the binary format, only a single whitespace character.
- The RGB-components of the picture. In the ASCII format, each colour component is written as a decimal number in ASCII with whitespace between the components. No single line may be more than 70 characters long. In the binary format, every colour component is stored as a single byte. In that case, there is no whitespace.

Comments begin with `#` and ends at the end of the line. In the binary format, there must not be any comments among the RGB component data. Another description can be found in the file `/it/kurs/pm2/vt09/man/ppm.5`. You can use `man -M /it/kurs/pm2/vt09/man ppm` to read the file.

Sample binary PPM file

In this particular example, all bytes are printable. In general, a binary file will include non-printable characters.

```
P6
# CREATOR: ray tracer
4 4
255
Zx2FFFFFFFFFZx2Zx2FFFFFFZx2Zx2Zx2FFFFFFFFFFFFFFF
```

Sample ASCII PPM file

```
P3
# CREATOR: ray tracer
4 4
255
90 120 50 70 70 70 70 70 70 70 70 70
90 120 50 90 120 50 70 70 70 70 70 70
90 120 50 90 120 50 90 120 50 70 70 70
70 70 70 70 70 70 70 70 70 70 70 70
```

11 Input data — world descriptions

The parameters determining how the picture is going to be traced are stored in a file. The file has information about all objects in the world (spheres, planes, triangles and light sources) as well as the position of the camera.

The format of the file is XML and an exact description will be found on the course web page.