



# miniRT

My first RayTracer with miniLibX

*Summary: This project is an introduction to the beautiful world of Raytracing.  
Once completed you will be able to render simple Computer-Generated-Images and to  
never been afraid again of implement mathematical formulas.*

# Contents

I	Introduction	2
II	Common Instructions	3
III	Mandatory part - miniRT	4
IV	Bonus part	9
V	Examples	11

# Chapter I

## Introduction

When it comes to render 3 dimensions computer generated images there are 2 possible approaches: the “Rasterization”, which is used by almost all graphic engines because of its efficiency and the “Ray Tracing.”

The “Ray Tracing.” method developed for the first time in 1968 (but improved since) is even today more expensive in computation than the “Rasterization” and as a result not well adapted to real time but it produces a really higher degree of visual realism.



Figure I.1: The pictures above are rendered with the ray tracing technique. Impressive isn't it?

Before you can even begin to produce such high quality graphics, you must master the basics: the `miniRT` is your first ray tracer coded in C, normed, humble but functionnal.

The main goal of `miniRT` is to prove yourself that you are able to implement any mathematics or physics formulas without being a mathematician, we will only implement the most basics ray tracing features here so just keep calm, take a deep breath and don't panic! At least you'll then be able to show off nice looking pictures to justify the amount of hours you're spending at school..

# Chapter II

## Common Instructions

- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a 0 if there is a norm error inside.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a 0 during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, and your `Makefile` must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your `Makefile`, which will add all the various headers, libraries or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}`. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated `Makefile`. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

# Chapter III

## Mandatory part - miniRT

<b>Program name</b>	miniRT
<b>Turn in files</b>	All your files
<b>Makefile</b>	all, clean, fclean, re, bonus
<b>Arguments</b>	a scene in format *.rt
<b>External functs.</b>	<ul style="list-style-type: none"><li>• open, read, write, malloc, free, perror, strerror, exit</li><li>• All functions of the math library (-lm man man 3 math)</li><li>• All functions of the MinilibX</li></ul>
<b>Libft authorized</b>	Yes
<b>Description</b>	Your program goal is to generate images according to the Raytracing protocol. Those computer generated images will each represent a scene, as seen from a specific angle and position, defined by simple geometric objects, and each with its own lighting system.

The constraints are as follows:

- You **must** use the `miniLibX`. Either in the version that is available on the operating system, or from its sources. If you choose to work with the sources, you will need to apply the same rules for your `libft` as those written above in **Common Instructions** part.
- The management of your window must remain smooth: passing over of another window, minimization, etc.
- You need at least these 5 simple geometric objects : plane, sphere, cylinder, square and triangle.
- If applicable, all possible intersections and inside of the object must be handled correctly.

- Your program must be able to resize the object unique properties : diameter for a sphere, side size for a square and the width and height for a cylinder.
- Your program must be able to apply translation and rotation transformation to objects, lights and cameras (except for spheres, triangles and lights that cannot be rotated).
- Light management : spot brightness, hard shadows, ambience lighting (objects are never totally in the dark). Colored and multi-spot lights have to be handled correctly.
- In case where **Deepthought** has eyes one day to evaluate your project and if you want to be able to render beautiful desktop wallpapers..  
Your program must save the rendered image in **bmp** format when its second argument is "**--save**".
- if no second argument is supplied, the program display the image in a window and respect the following rules:
  - Pressing **ESC** must close the window and quit the program cleanly.
  - Clicking on the red cross on the window's frame must close the window and quit the program cleanly.
  - if the declared size in the scene is greater than the display's one, the window size will be set depending to the current display's one.
  - If there is more than one camera you must be able to switch between them by pressing the keyboard keys of your choice.
  - The use of **images** of the **minilibX** is strongly recommended.
- Your program must take has first argument a scene description file with the **.rt** extension.
  - It will contain the window/rendered image **size**, which implies your **miniRT** must be able to render in any positive size.
  - Each type of element can be separated by one or more empty line(s).
  - Each type of information from an element can be separated by one or more space(s).
  - Each type of element can be set in any order in the file.
  - Element which are defined by a capital letter can only be declared once in the scene.

- o Each element first's information is the type identifier (composed by one or two character(s)), followed by all specific informations for each objects in a strict order such as :

- \* Resolution:

```
R 1920 1080
```

- identifier: **R**
- x render size
- y render size

- \* Ambient lightning:

```
A 0.2 255,255,255
```

- identifier: **A**
- ambient lightning ratio in range [0.0,1.0]: **0.2**
- R,G,B colors in range [0-255]: **255, 255, 255**

- \* Camera:

```
c -50.0,0,20 0,0,1 70
```

- identifier: **c**
- x,y,z coordinates of the view point: **0.0,0.0,20.6**
- 3d normalized orientation vector. In range [-1,1] for each x,y,z axis: **0.0,0.0,1.0**

- \* Light:

```
l -40.0,50.0,0.0 0.6 10,0,255
```

- identifier: **l**
- x,y,z coordinates of the light point: **0.0,0.0,20.6**
- the light brightness ratio in range [0.0,1.0]: **0.6**
- R,G,B colors in range [0-255]: **10, 0, 255**

- \* Sphere:

```
sp 0.0,0.0,20.6 12.6 10,0,255
```

- identifier: **sp**
- x,y,z coordinates of the sphere center: **0.0,0.0,20.6**
- the sphere diameter: **12.6**
- R,G,B colors in range [0-255]: **10, 0, 255**

- \* Plane:

```
pl  0.0,0.0,-10.0  0.0,1.0,0.0  0,0,225
```

- identifier: **pl**
- x,y,z coordinates: **0.0,0.0,-10.0**
- 3d normalized orientation vector. In range [-1,1] for each x,y,z axis: **0.0,0.0,1.0**
- R,G,B colors in range [0-255]: **0, 0, 255**

- \* Square:

```
sq  0.0,0.0,20.6   1.0,0.0,0.0  12.6  255,0,255
```

- identifier: **sq**
- x,y,z coordinates of the square center: **0.0,0.0,20.6**
- 3d normalized orientation vector. In range [-1,1] for each x,y,z axis: **1.0,0.0,0.0**
- side size: **12.6**
- R,G,B colors in range [0-255]: **255, 0, 255**

- \* Cylinder:

```
cy  50.0,0.0,20.6   0.0,0.0,1.0  10,0,255   14.2  21.42
```

- identifier: **cy**
- x,y,z coordinates: **50.0,0.0,20.6**
- 3d normalized orientation vector. In range [-1,1] for each x,y,z axis: **0.0,0.0,1.0**
- the cylinder diameter: **14.2**
- the cylinder height: **21.42**
- R,G,B colors in range [0,255]: **10, 0, 255**

- \* Triangle:

```
tr  10.0,20.0,10.0  10.0,10.0,20.0  20.0,10.0,10.0  0,0,255
```

- identifier: **tr**
- x,y,z coordinates of the first point: **10.0,20.0,10.0**
- x,y,z coordinates of the second point: **10.0,10.0,20.0**
- x,y,z coordinates of the third point: **20.0,10.0,10.0**
- R,G,B colors in range [0,255]: **0, 255, 255**

- Example for the mandatory part with a minimalist .rt scene:

```
R 1920 1080
A 0.2
c -50,0,20    0,0,0    70
l -40,0,30      0.7
pl 0,0,0        0,1,0,0
sp 0,0,20       20
sq 0,100,40     0,0,1,0   30
cy 50.0,0.0,20.6 0,0,1.0  14.2 21.42 10,0,255
tr 10,20,10     10,10,20  20,10,10  0,0,255
```

- If any misconfiguration of any kind is encountered in the file the program must exit properly and return "Error\n" followed by an explicit error message of your choice
- For the defence, it would be ideal for you to have a whole set of scenes with the focus on what is functional, facilitating that way the control of the elements to create.

# Chapter IV

## Bonus part

Obviously the Ray-Tracing technique could handle many more things like reflection, transparency, refraction, more complex objects, soft shadows, caustics, global illumination, bump mapping, .obj file rendering etc..

But for the `miniRT` project, we want to stay simple for your first raytracer and your first steps in CGI.

So here is a list of few simple bonuses to implement, if you want to do bigger bonuses we strongly advise you to recode a new ray-tracer later in your developper life and of course once this little one is finished and fully functionnal.



Figure IV.1: a spot, a space skybox and a shiny earth-textured sphere with bump-maping



Bonuses will be evaluated only if your mandatory part is **PERFECT**.  
By **PERFECT** we naturally mean that it needs to be complete, that it cannot fail, even in cases of nasty mistakes like wrong uses etc.  
Basically, it means that if your mandatory part does not obtain ALL the points during the grading, your bonuses will be entirely IGNORED.

Bonus list:

- Normal disruption e.g. using sine which gives a wave effect.
- Color disruption : checkerboard.
- Color disruption : rainbow effect using object's normal.
- Parallel light following a precise direction.
- Composed element : Cube (6 squares).
- Composed element : Pyramid (4 triangles, 1 square).
- Caps on a size-limited cylinder.
- One other 2nd degree object : Cone, Hyperboloid, Paraboloid..
- One color filter : Sepia, R/G/B filters..
- Anti-aliasing.
- Simple stereoscopy (like red/green glasses).
- Multithreaded rendering.
- Sphere texturing : uv mapping.
- Handle bump map textures.
- A beautiful skybox.
- Keyboard interactivity (transaltion/rotation) with camera.
- Keyboard interactivity (translation/rotation) with objects.
- You can move the camera rotation with the mouse.



You are allowed to use other functions to complete the bonus part as long as their use is justified during your evaluation and also to modify the expected file scene file format to fit your needs. Be smart!



To earn all bonus points you need to validate at least 14 of them, so choose wisely but be careful to don't waste your time!

# Chapter V

## Examples

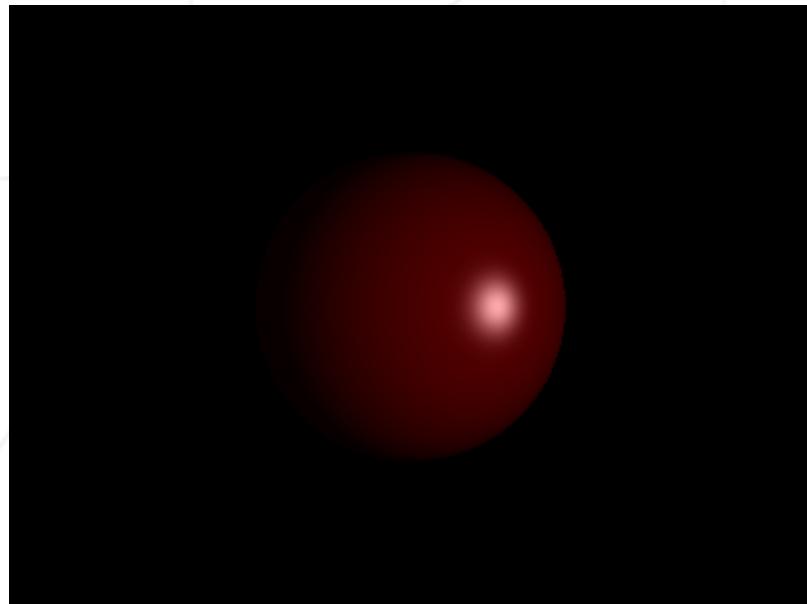


Figure V.1: A sphere, one spot, some shine (optional)



Figure V.2: A cylinder, one spot

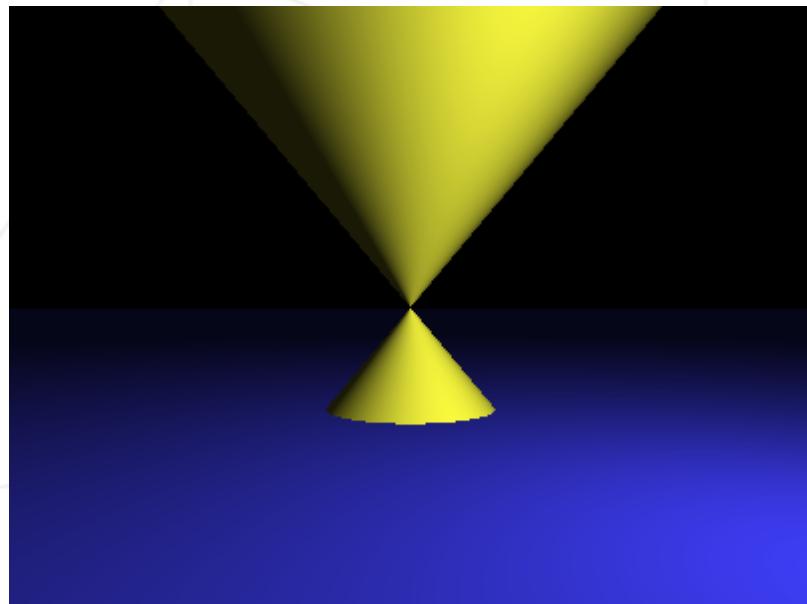


Figure V.3: A cone (optional), a plane, one spot

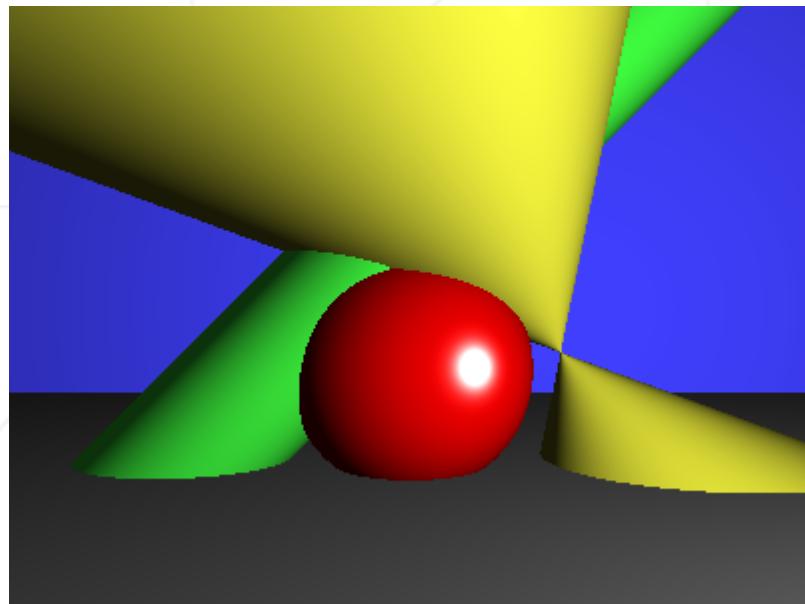


Figure V.4: A bit of everything, including 2 planes

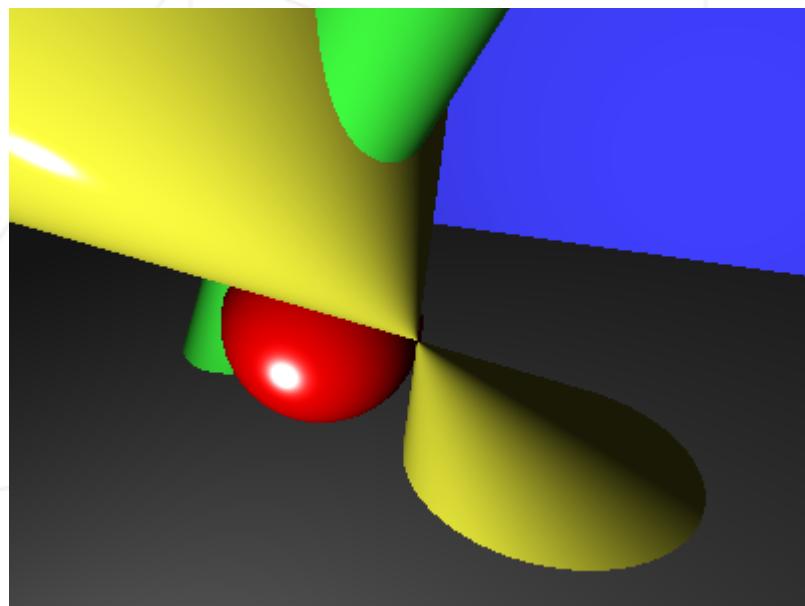


Figure V.5: Same scene different camera

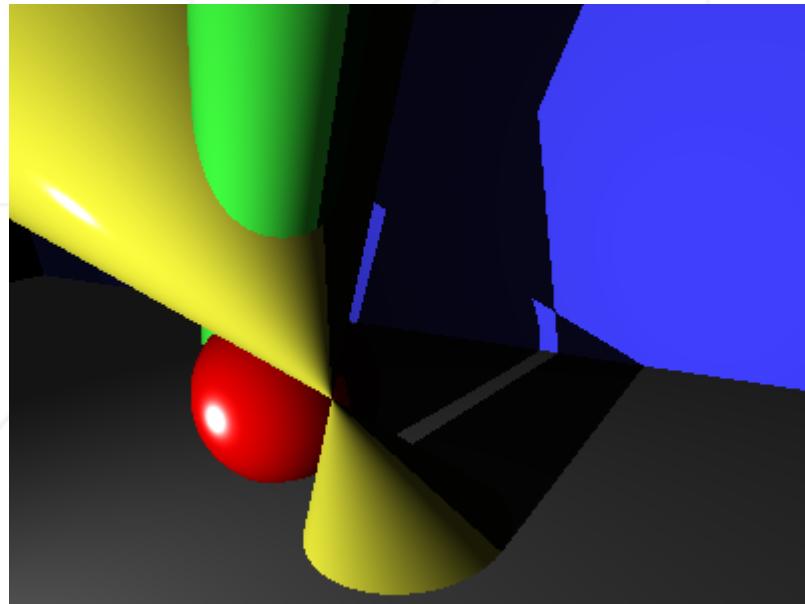


Figure V.6: This time with shadows

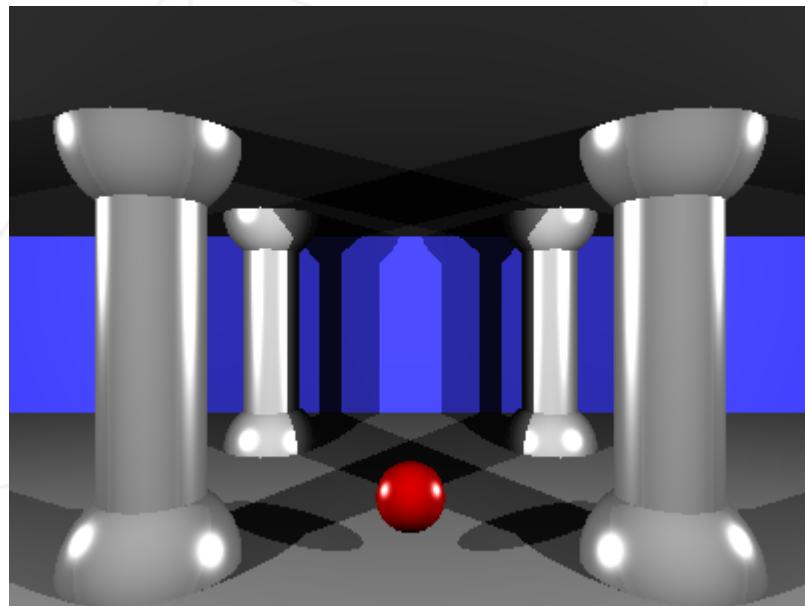


Figure V.7: With multiple spots

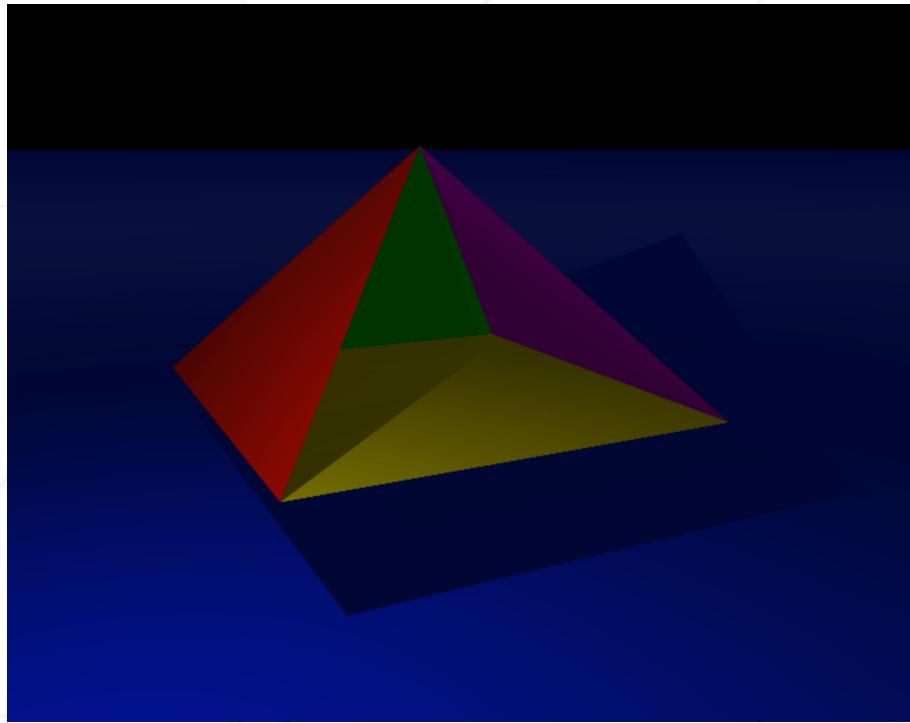


Figure V.8: A plane, 3 triangles, 1 square and one low brightness spot at the left

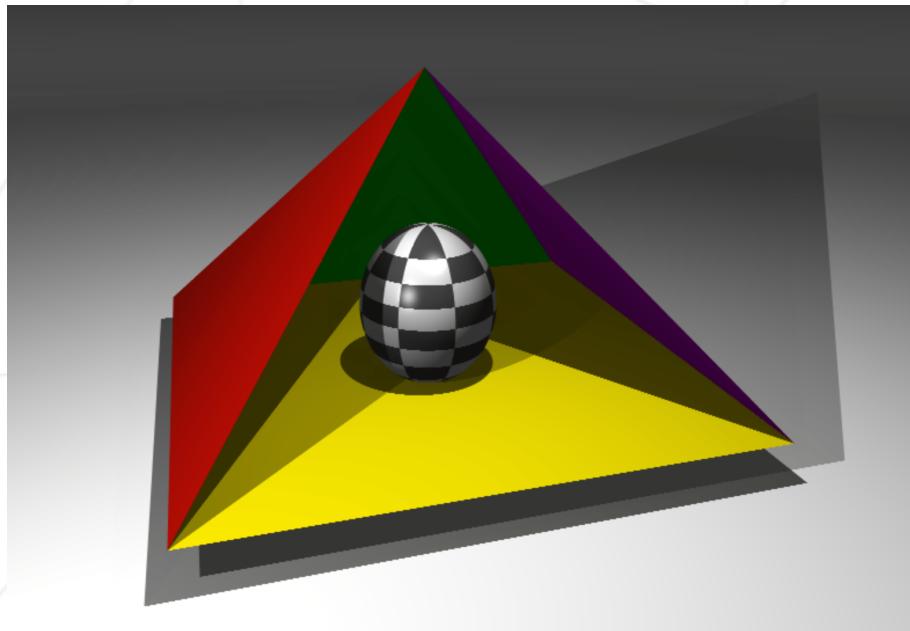


Figure V.9: And finally with multiple spots and a shiny checkered (optional) sphere in the middle