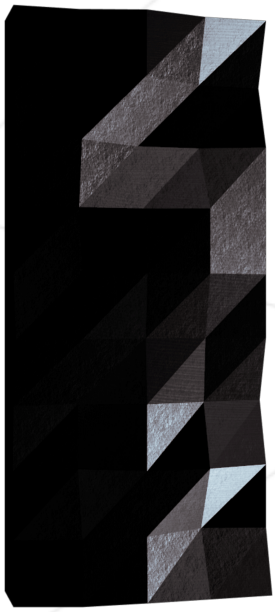

MAJOR Infography project

RT

Summary: This project is nothing but a climax of amazing computer-generated images

Contents

Chapter I

Foreword

I have a little shadow that goes
in and out with me.
And what can be the use of him is
more than I can see.
He is very, very like me from the
heels up to the head;
And I see him jump before me,
When I jump into my bed.

The funniest thing about him is the way
he likes to grow
Not at all like proper children, which is
always very slow;
For he sometimes shoots up taller like an
India-rubber ball,
And he sometimes gets so little that
There's none of him at all.

One morning, very early, before the
sun I was up,
I rose and found the shining dew
on every buttercup;
But my lazy little shadow, like an
arrant sleepyhead.
Had stayed at home behind me and was
Fast asleep in bed.

My shadow
by Robert Louis Stevenson (1850-1894)

Chapter II

Objectives

When it comes to render 3 dimensions computer generated images there is 2 possible approaches: “rasterization”, which is used by almost all graphic engines because of it’s efficiency and “ray tracing.” The ray tracing method is more extensive and as a result not adapted to real time but it produces a high degree of visual realism.

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

Since you mastered the basics with the last project: `RT` opens the door of your second ray tracer coded in `C, C++ or rust`, awesome and functionnal.

And if you are all depressed reading this topic, think of the long way already paved by the lunatics from ILM, Pixar (or any other computer-generated images studio) that all have deployed infinite creativity to get better in that art) relax, take life easy and bound

Chapter III

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- The executable file must be named `RT`.
- Your `Makefile` must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.
- you can use your library for your `RT`. Submit also your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Your program cannot have memory leaks.
- You can use `MacOS` native `MinilibX` library already installed on the imacs, or you can use MinilibX from its sources that you'll need to integrate similarly to libft. Last option, you can use additional graphic libraries (`X11`, `SDL`, etc...). If the library you are using is not installed on the imacs, you will have to submit the sources of this library in your repository, and it will have to be automatically compiled, without doing anything more than compiling your project, exactly like `MinilibX` or like your `libft`. No matter which graphic library you can only use its basic drawings functions similar to `MinilibX`: Open a window, lit a pixel and manage events.
- Within the mandatory part, you are allowed to use only the following libc functions or their C++ and rust equivalent:
    - `open`
    - `read`
    - `write`
    - `close`
    - `malloc`
    - `free`
    - `perror`

  - `strerror`
  - `exit`
  - All functions of the math library (`-lm` man `man 3 math`)
  - All functions of the `MinilibX` or their equivalent in another graphic library.

- You are allowed to use other functions or other librairies to complete the bonus part as long as their use is justified during your defense. Be smart!
- You can ask your questions on the forum, on slack...

Chapter IV

Mandatory part

Your goal is to be able, with the help of your program, to generate images according to 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.

This project will have a mandatory part and many additional options. The mandatory part is worth 0 points and the options will only bring you points IF the mandatory part is 100% complete. The project will only be validated if a substantial volume of options is present when you defend the project.

The elements you need to create are as follows:

- Code in C, C++ or rust. (use the latest version of the language and follow some up-to-date good practice.)
- Implement the Ray-Tracing method to create a computer generated image.
- You need at least 4 simple geometric objects as a base (not composed): plane, sphere, cylinder and cone.
- Your program must be able to apply translation and rotation transformation to objects before displaying them. For example a sphere declared at `(0, 0, 0)` must be able to successfully translate to `(42, 42, 42)`.
- Manage the redraw view without recalculating (basically with MinilibX, you can manage the expose properly): if a part of the window needs to be redrawn, it is best if you don't have to recalculate everything.
- Position and direction of any point of vision and of simple objects.
- Light management: different brightness, shadows. multi-spot, shine effect.

Those elements are required in the old RTv1 project. For the RT, you won't have any point (and so no chance to validate the project) if you only do this the whole project is a bonus fest.

Chapter V

Bonus part

Now, let's get to the real deal: the options.

There are many and they don't really have any limits. We can give as examples:

- Ambiance light
- Direct light
- Parallel light
- Limited objects: parallelograms, disks, half-spheres, tubes etc...
- Bump mapping and color disruption
- External files for scene description
- Reflection
- Transparency
- Shadow modification according to transparency of the elements
- Composed elements: cubes, pyramids, tetrahedrons...
- Textures
- Negative elements
- Limit disruption / transparency / reflection, depending on texture
- More native elements: paraboloid, hyperboloid, tablecloth, toroid...
- ...

Those are the basics stuff, naturally you can be more exotic! You can think for example of multiple calculations spread over multiple cumputers or you can use quadrics or quartics, videos clips created from your own images, a stereoscopic version for Oculus Rift etc...

However, for those of you that might want to parse some .pov or .3ds files, your raytracer must to be able to manage correctly the simple objects of the mandatory part from equations and not from facets.

In addition, your defense will require that you do some live manipulations of your scenes. That will need to be done with your tools and not with 3DS (you will to have

your own configuration files basically).

A special bonus will be given to the first group that can create a raytracing image of the Moebius strip (Möbius if you'd rather). From the mathematical equation and not from facets.

You can find in the e-learning section some technical examples to illustrate some options.

For the defense, it would be best if you had at least the 3 following scenes, facilitating the checking of the mandatory part you have to do:

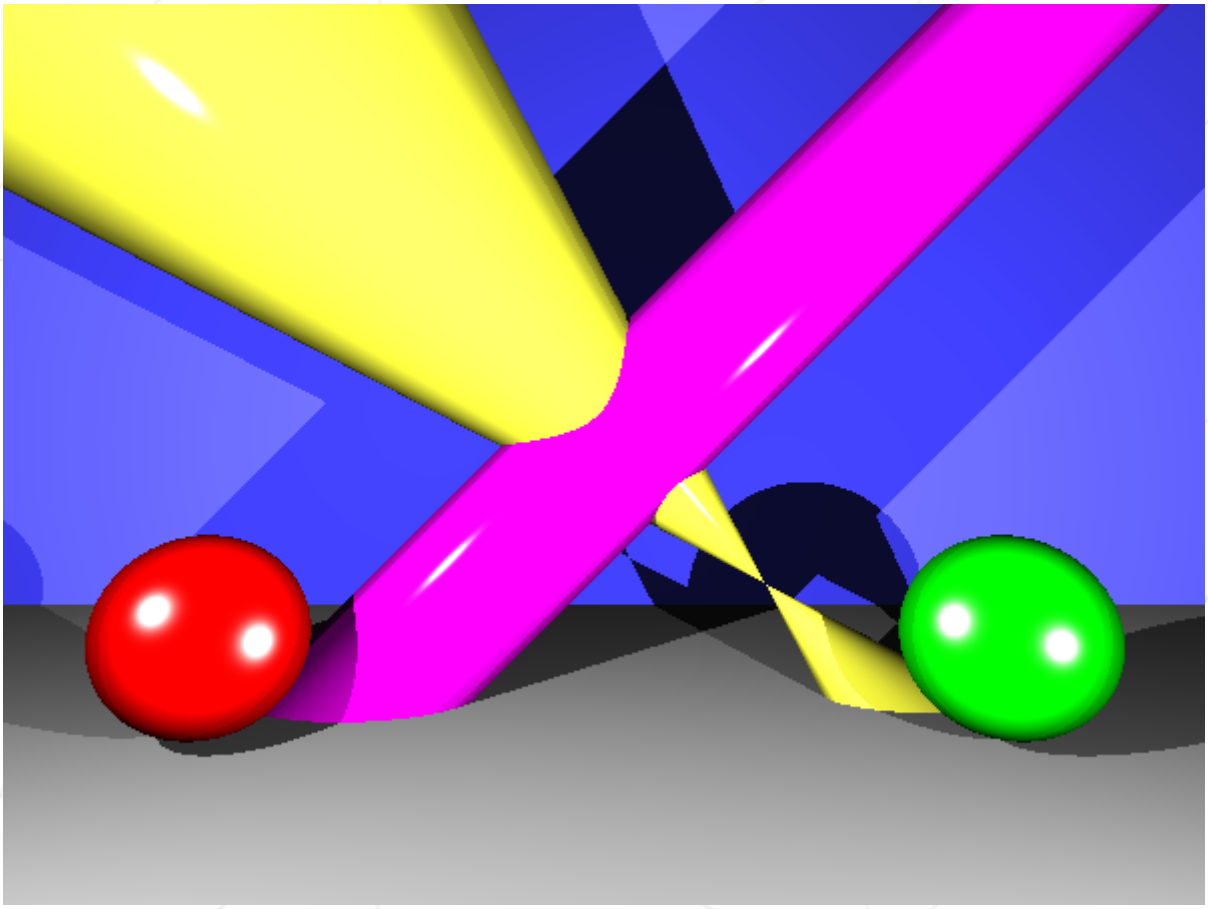

Figure V.1: The 4 basic objects, 2 spots, shadow and shine effect

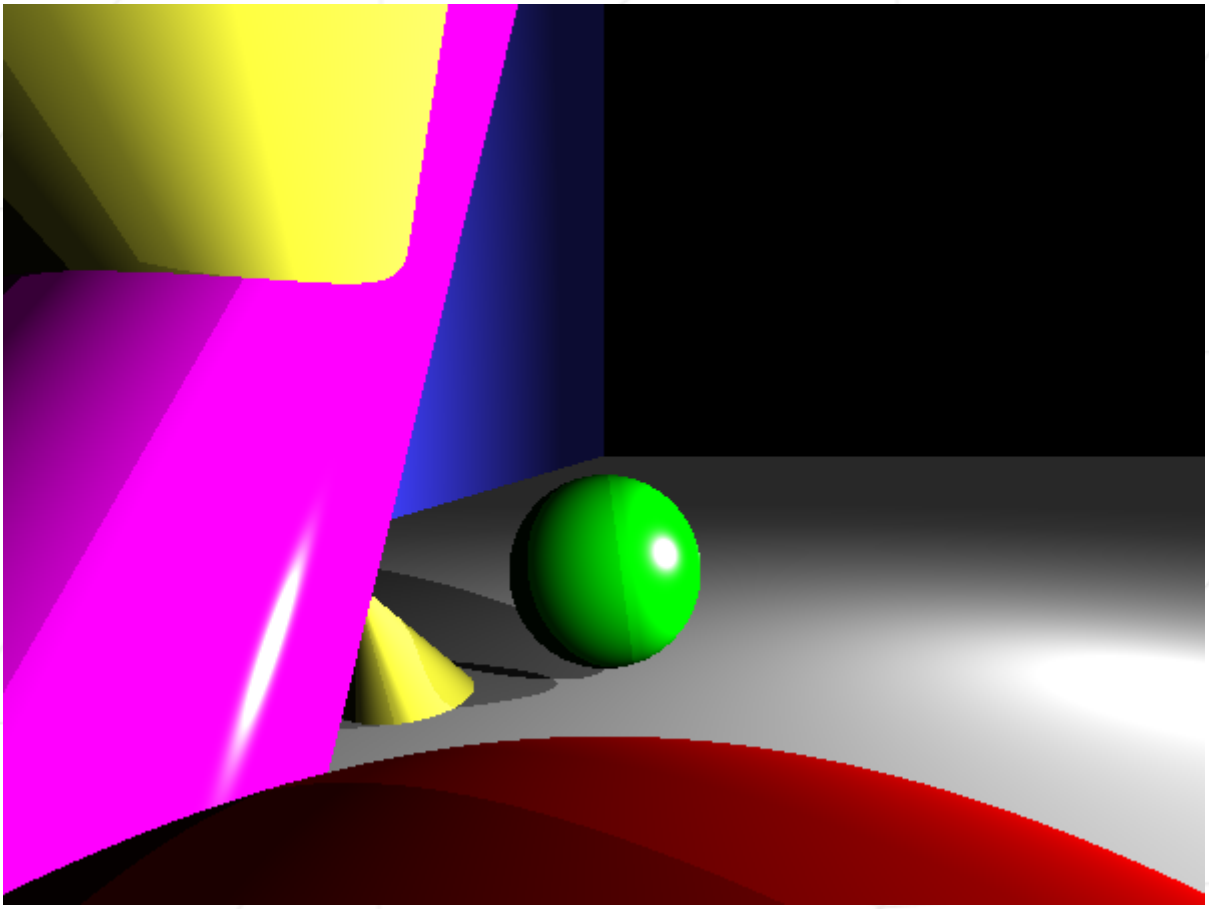

Figure V.2: Same scene from another viewpoint

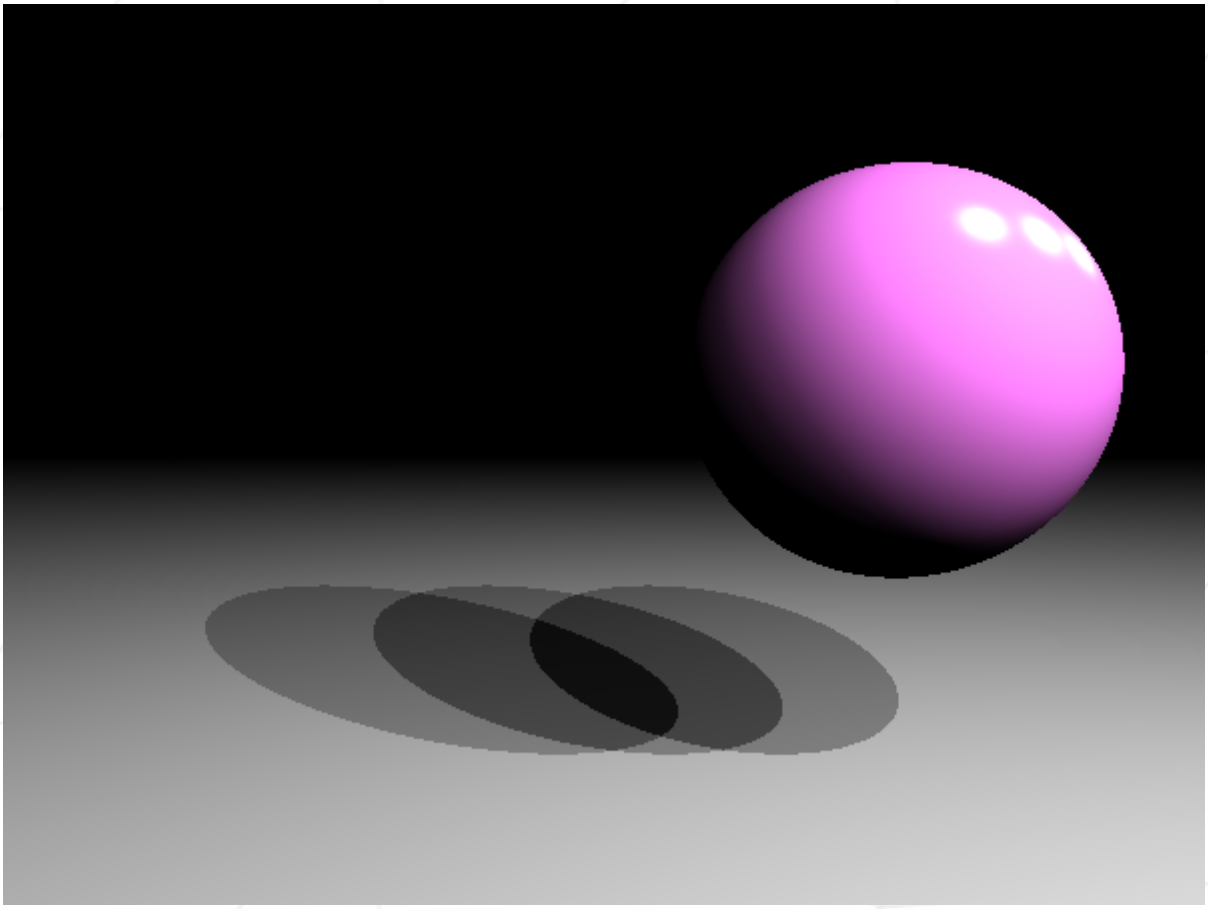

Figure V.3: Shadow mixing

Chapter VI

Submission and peer correction

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.

Good luck to all !