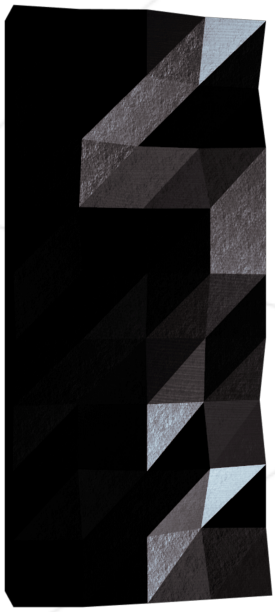

Graphical Project

RT

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

Version: 3.1

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 are 2 possible approaches: “rasterization”, which is used by almost all graphic engines because of its efficiency on accelerated hardware, and the “ray tracing” method that is more extensive, more expensive, not adapted to real time production, but that comes with a high degree of visual realism.

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

You mastered the basics of ray tracing with the `miniRT`. Now, the `RT` project 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). So, relax, take life easy and bound.

Chapter III

General Instructions

- This project will be corrected by humans only. You are allowed to organise and name your files as you want.
- The executable file must be named `rt`.
- You're free to choose between C, C++ or Rust.
- Your program cannot have memory leaks (much more C-oriented directive obviously).
- Within the mandatory part, you are allowed to use the entire libc, or the entire libstdc++, or their Rust equivalents.
- Also you can use all the functions of the math library (`-lm` man `man 3 math`)
- It is allowed to use external native libraries to open well known image formats (libpng, libjpeg ..).
- And you can use all functions of the `MinilibX` or their equivalent in another graphic library (SDL, XCB, ...): open a window, lit a pixel, put an image on a window, and manage events. The graphical rendering can't be created by the GPU (i.e. not with any API involving explicit or implicit shaders loaded onto the GPU, like OpenGL, Metal, Vulkan, DirectX ...).
- You are allowed to use other functions or other librairies to complete the bonus part (not the options) as long as their use is justified during your defense. Be smart!
- Note: heavy calculations can be completed on GPU (but not the rendering). You can use OpenCL / Cuda, it is also possible to use compute shaders with OpenGL, Vulkan, Metal.
- You can ask your questions to other students on the forum, on slack...

Chapter IV

Mandatory part

Your goal is to be able, with your program, to generate images according to the ray tracing technique.

Those computer generated images will each represent a scene, as seen from a specific angle and position of the camera / eye, defined by simple geometric objects and light sources.

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 transformations to objects before displaying them. For example a sphere declared at `(0, 0, 0)` must be able to successfully translate to `(42, 42, 42)`.
- Position and direction of the camera (or eye) can be changed easily.
- Manage to redraw the view or part of the view without recalculating the entire image (ex: with MinilibX, you can manage the expose event properly).
- Light management: different brightness, shadows, multi-spot, shine effect.

Chapter V

Options

The options part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your options part will not be evaluated at all.

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

There are many of them, 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 computers or you can use quadrics or quartics, videos clips created from your own images, a stereoscopic version for VR

headsets etc...

Do not hesitate to be curious, explore various blogs and documentations, it actually exists many more possible options. The evaluation sheet will name various possible options, but on many items of the evaluation, you will have the possibility to give points to unlisted new options.

Note: for those of you that might want to parse some .pov or .3ds files, your ray tracer must to be able to manage correctly the simple objects of the mandatory part from equations and not from vertexes and triangles.

In addition, you have to anticipate that during the defense you will need to do some live manipulations of your scenes, using your own tools (and not existing public tools). We suggest you implement the configuration file option, or allow in-program live configiration and modifications.

In order to facilitate the evaluation of the mandatory part, we strongly advise to reproduce at least the 3 following scenes:

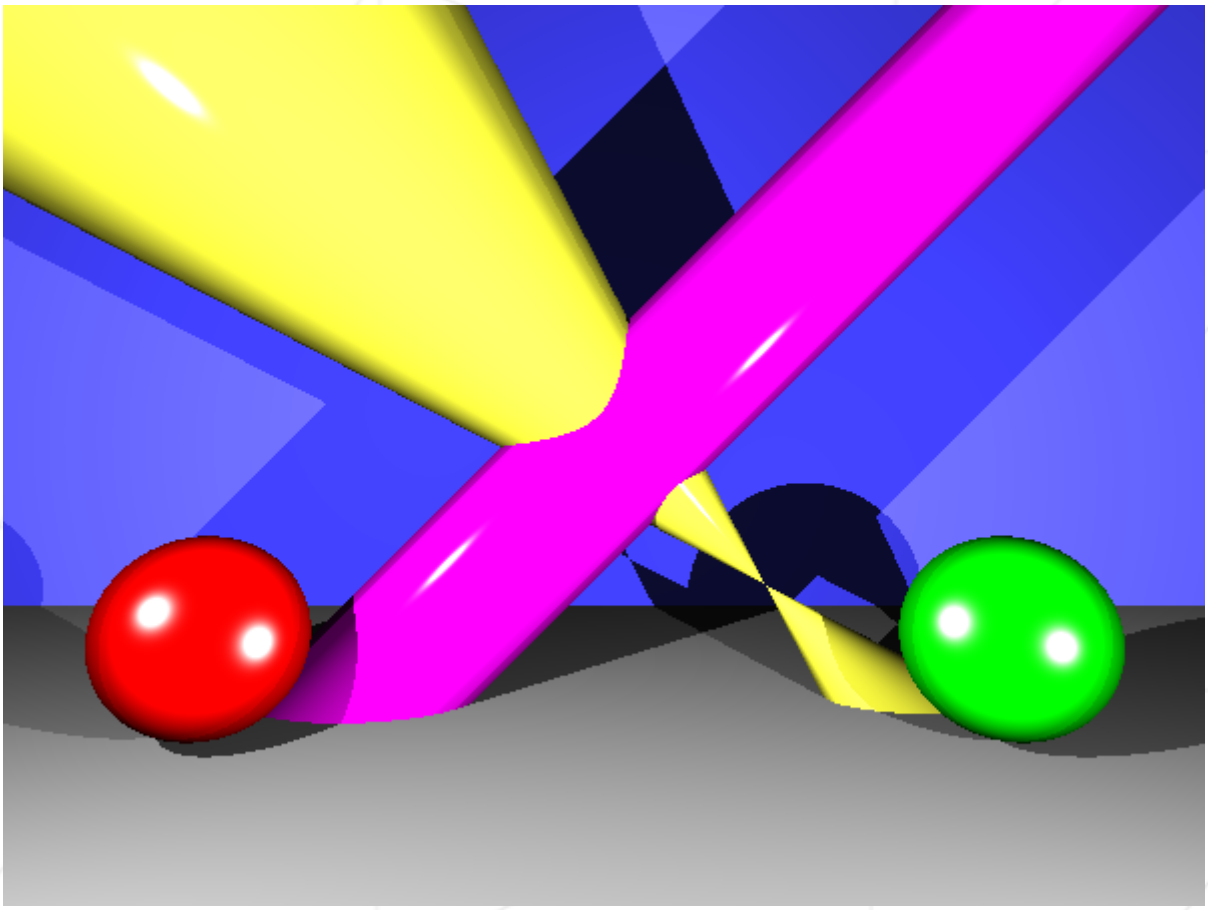

Figure V.1: The 4 basic objects, 2 spots, shadow and shine effect

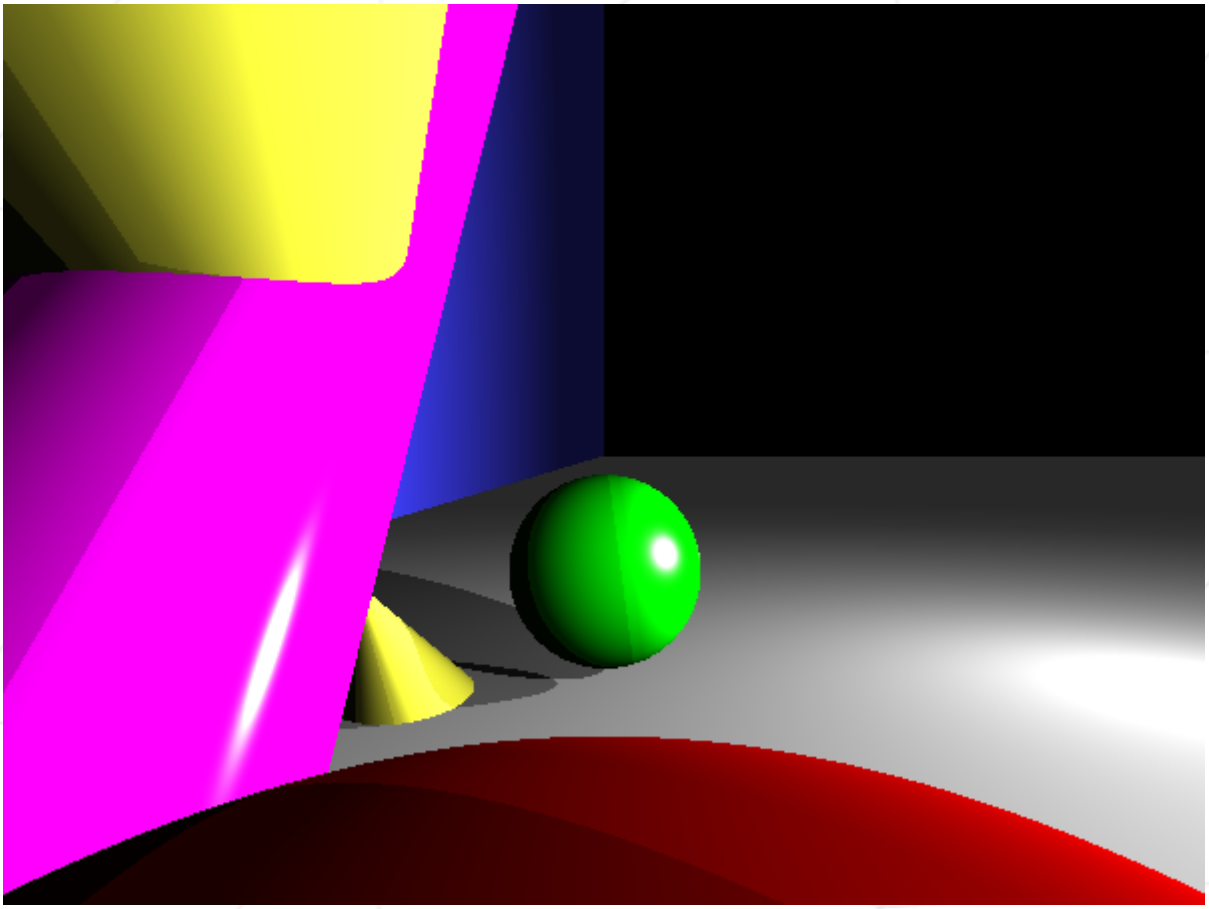

Figure V.2: Same scene from another viewpoint

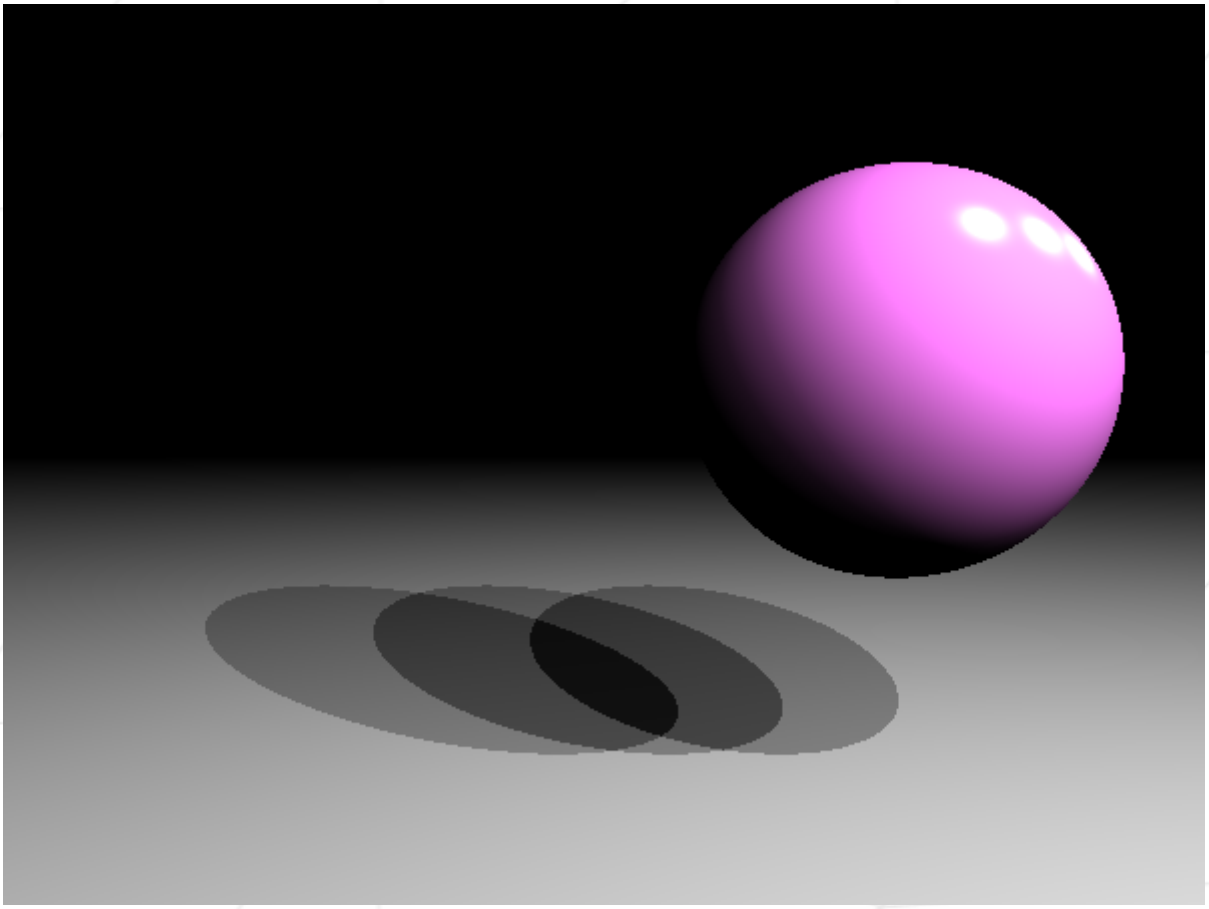

Figure V.3: Shadow mixing

Chapter VI

Bonuses

The evaluation sheet offers bonus points for even more outstanding options, addons, features of your program. The final mark can go up to 125, as usual.

Chapter VII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

You will have to demostrate all your options during the evaluation in order to get the points. Get prepared with multiple configured scenes, ready to be calculated (already create images - jpeg, png ... - are not allowed to prove options).

The validation threshold of the project is low, the more options you create, the more XP you'll get.