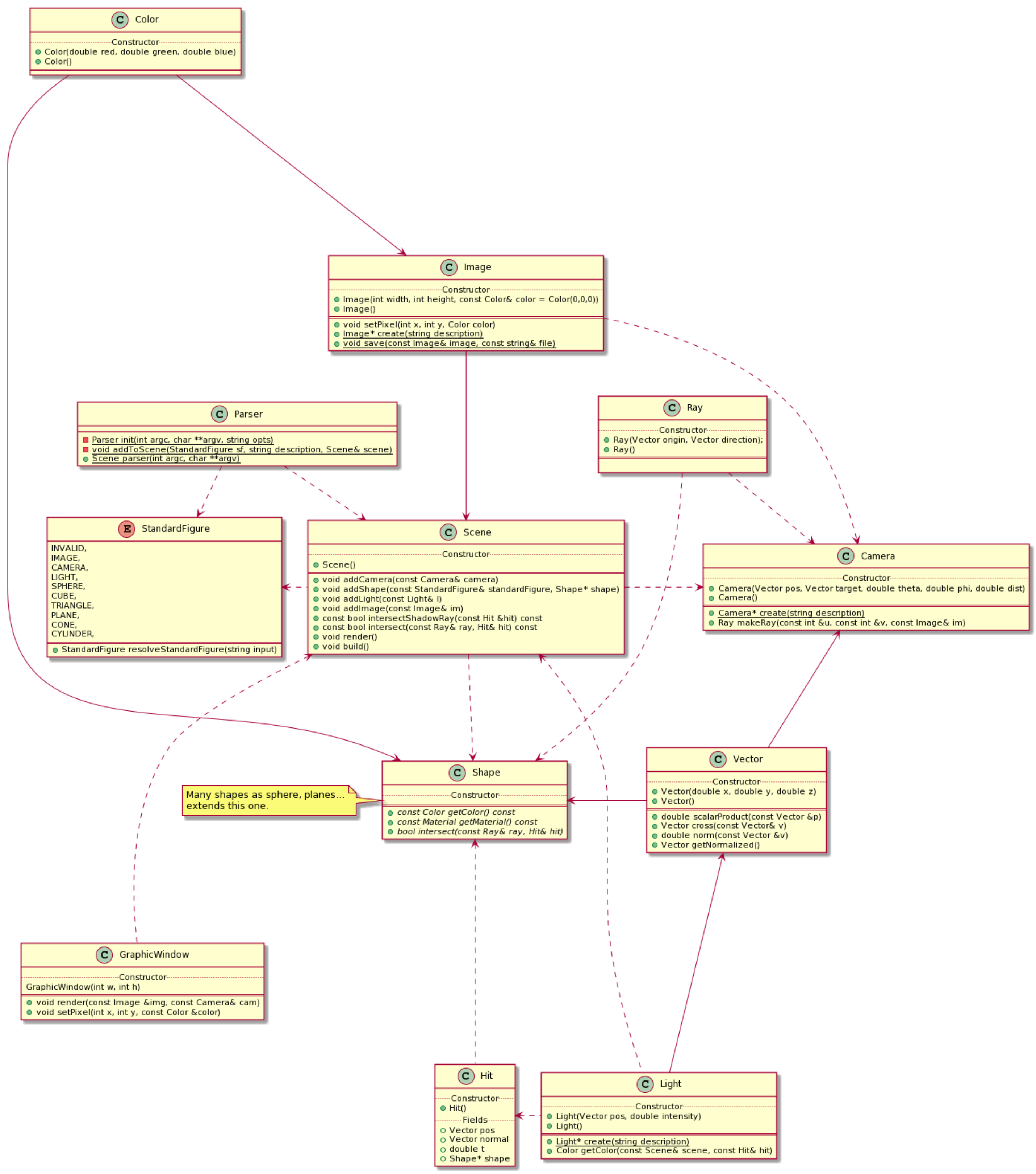
Raytracing : Dev manual

Architercture

This is project uses the principle of Object-oriented programming. It is programmed in C++ and uses sdl library for graphic purposes. The following informations can be found in every .h file.

class	purpose	important marks
Color	The color must have three factors *red*, *green* and *blue*. They must be double values between 0 and 1.	This class redefines many basic operator and each color has a getter.
Vector	A vector will be represented by 3 coordinates *x*, *y* and *z* that are doubles. It can calculate the scalar product, cross and normalize the Vector.	The method **scalarProduct** and **cross** that will calculate the scalar product and cross of two vectors and **getNormalized**, **normalize** that will normalize a vector.
Material	Has the information to define the texture of the shape but also the way it is perceived in the scene.	Has the *ambient diffuse specular* color and the *exponent* has the phong lighting algorithm. Also has two boolean *mirror* and *transparency*. All the fields are public. And by default the ambience, diffuse and specular will be white the exponent equal to 0 and mirror and transperecy will be false.
Camera	Creates the camera that will be place on the scene and generate the rays that will hit the shapes of the scene.	The static method ***create*** that will create a Camera for its description (look at USER for the format). The method **makeRay** that will generate a ray for every i and j the coordinates of a pixel but also will take the image as a parameter.
Ray	Will create a Ray with it's *origin* and *direction*. This class will be use to see if a ray intersect a shape.	All the fields have a getter.
Hit	Will keep the intersect informations as the *point*, the *normal*, *t* and the *shape*.	All the fields are public.
Light	Will create a Light to illuminate the scene. The method **getColor** will return the right color according to the scene, camera and hit information.	The static method ***create*** that will create a Light for its description (look at USER for the format). The method **calcIntensity** will calculate the light intensity with a norm given as a parameter. The method **getColor** will use this method but also the fields of the shape material and it's color to calculate the right color.
Image	Will generate an Image by using the protocol to generate a ppm. The method save will construct the file ppm. Will need to use the **setPixel** to set a pixel color.	The static method create that will ***create*** an Image for its description (look at USER for the format). Every field has a getter. IT is also possible to set and get a pixel. There's also the method **toIntArray** that will return the int array version the image, this method is only used for the graphic window.
GraphicWindow	Creates a graphic window that will represent the image in real time and will allow us to navigate the scene with the camera.	The method **render** will update the graphic window to create the scenario from the image with the current viewpoint of the camera.
StandardFigure	Will create an enumeration to regroup every element that can be added to the scene.	The method **resolveStandardFigure** will verify the input is well written. Refer to the USER to see the format of the text.
Parser	Parser the terminal line as according to the demand and return a Scene.	The static method ***parser*** will parse the terminal line to create a Scene. Refer to USER to see how to write the terminal line.
Scene	Will create a scene with its camera, lights and shapes. Will build an image after it traces its rays to determine the color of the pixels.	The method **render** will save the scene in the image. The method **build** will build the image file, the graphic window or both.
Utilis	Has globals methods that can be used in other classes	The methods **decode** will decode the line as one of the classes.
Shape	Will create an abstract class Shape that will regroup every shape. Every shape as it's surface color and material. This class will use the method intersect to check if there's a ray intersection.	Every method is a virtual method. There's a getter of the color and material but most importantly the method **intersect** will check if the ray intersect our shape and keep the information from the intersection in Hit.
Cone	Will generate a Cone that implements a Shape.	The static method ***create*** that will create a Cone for its description (look at USER for the format).
Cube	Will generate a Cube that implements a Shape.	The static method ***create*** that will create a Cube for its description (look at USER for the format).
Cylinder	Will generate a Cylinder that implements a Shape.	The static method ***create*** that will create a Cylinder for its description (look at USER for the format).
Sphere	Will generate a Sphere that implements a Shape.	The static method ***create*** that will create a Sphere for its description (look at USER for the format).
Plane	Will generate a Plane that implements a Shape.	The static method ***create*** that will create a Plane for its description (look at USER for the format).

UML


Color
Constructor
Color(double red, double green, double blue)
Color()
Image
Constructor
Image(int width, int height, const Color& color = Color(0,0,0))
Image()
void setPixel(int x, int y, Color color)
Image* create(string description)
void save(const Image& image, const string& file)
Parser
Parser init(int argc, char **argv, string opts)
void addToScene(StandardFigure sf, string description, Scene& scene)
Scene parser(int argc, char **argv)
Ray
Constructor
Ray(Vector origin, Vector direction);
Ray()
StandardFigure
INVALID,
IMAGE,
CAMERA,
LIGHT,
SPHERE,
CUBE,
TRIANGLE,
PLANE,
CONE,
CYLINDER,
StandardFigure resolveStandardFigure(string input)
Scene
Constructor
Scene()
void addCamera(const Camera& camera)
void addShape(const StandardFigure& standardFigure, Shape* shape)
void addLight(const Light& l)
void addImage(const Image& im)
const bool intersectShadowRay(const Hit &hit) const
const bool intersect(const Ray& ray, Hit& hit) const
void render()
void build()
Camera
Constructor
Camera(Vector pos, Vector target, double theta, double phi, double dist)
Camera()
Camera* create(string description)
Ray makeRay(const int &u, const int &v, const Image& im)
Shape
Constructor
const Color getColor() const
const Material getMaterial() const
bool intersect(const Ray& ray, Hit& hit)
Many shapes as sphere, planes...
extends this one.
Vector
Constructor
Vector(double x, double y, double z)
Vector()
double scalarProduct(const Vector &p)
Vector cross(const Vector& v)
double norm(const Vector &v)
Vector getNormalized()
GraphicWindow
Constructor
GraphicWindow(int w, int h)
void render(const Image &img, const Camera& cam)
void setPixel(int x, int y, const Color &color)
Hit
Constructor
Hit()
Fields
Vector pos
Vector normal
double t
Shape* shape
Light
Constructor
Light(Vector pos, double intensity)
Light()
Light* create(string description)
Color getColor(const Scene& scene, const Hit& hit)

Bugs

The transparency for the cubes does not work for the cubes. The shadows are not perfect. If the command line is not written properly the program will not work correctly. The graphic window since it will reder the image every time it can be very slow.

Conclusion

Since we did not have many lessons we had to do our own research, and we follow many tutorials and inspire ourselves from many GitHub projects.
It's also the first time we did this type of project, so we did not have the basics it was very complicated, but we were able to deliver a small program.