CS3241 Computer Graphics (2020/2021 Semester 1)

Lab Assignment 4

Release Date: 23 October 2020, Friday

Submission Deadline: 15 November 2020, Sunday, 11:59 PM

LEARNING OBJECTIVES

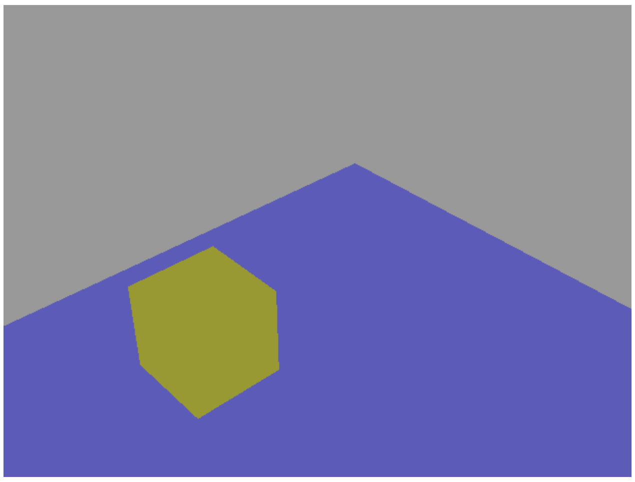
Implementing the Whitted Ray Tracing algorithm. After completing the programming assignment, you should have learned

- how to compute ray intersection with some simple implicit-form surface primitive,
- how to do lighting computation,
- how to shoot shadow rays to generate shadows,
- how to spawn secondary rays,
- how to trace rays recursively, and
- how the Whitted Ray Tracing algorithm works.

TASKS

You are to complete an **incomplete C++ program** that implements the Whitted Ray Tracing algorithm. You have to complete the program according to the following requirements. There are altogether **three tasks** in the assignment.

Please download the ZIP file **`Lab4_todo.zip`** from the **Lab Assignments** folder in LumiNUS Files. If you build and run the given incomplete program, it will produce an image as follows (in file **`out1.png`**). Instructions on how to **build the project** may be found at the end of the document.



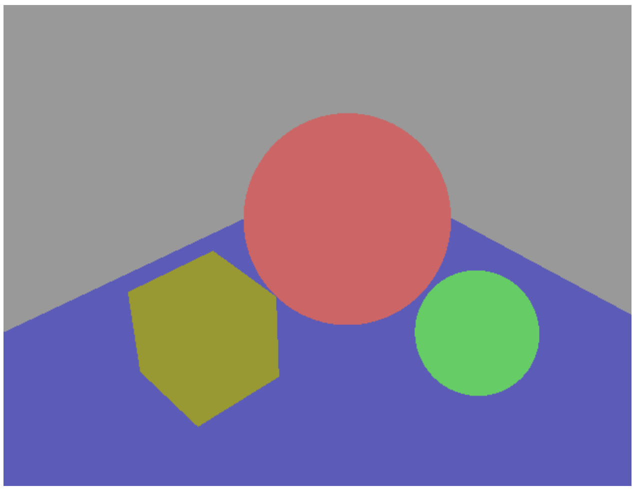
It shows an image of an unlit scene of 3 intersecting planes and a cube made of triangles. Two additional spheres are supposed to be in the image, but the ray-sphere intersection routine has not been implemented yet.

Task 1

You are to complete the **`Sphere::hit()`** and **`Sphere::shadowHit()`** functions in **`Sphere.cpp`** to compute ray-sphere intersection. You must write your code only in places marked "**`WRITE YOUR CODE HERE`**".

You can refer to **`Plane.{h, cpp}`** and **`Triangle.{h, cpp}`** to get some idea how to do it. Other relevant files to study are **`Vector3d.h`**, **`Ray.h`**, **`Surface.h`**, and **`Sphere.h`**.

For this task, you have to **submit** your completed **`Sphere.cpp`**, and the image generated, which should look like the following. You must name your image file **`img_spheres.png`**.


`img_spheres.png`

Task 2

You are to complete the **`Raytrace::TraceRay()`** function in **`Raytrace.cpp`** to perform the recursive ray tracing. You must write your code only in places marked "**`WRITE YOUR CODE HERE`**".

In this implementation, we are assuming that **all objects are opaque**. At each surface point intersected by the ray, the color result is computed using the formula

$$I = I_{local} + k_{rg}\, I_{reflected}$$

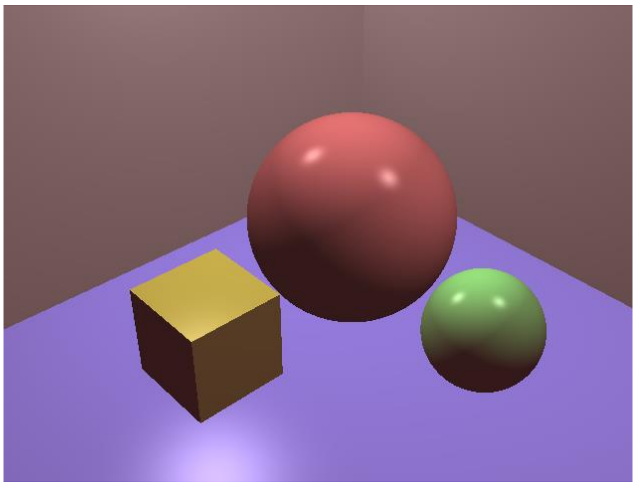
where

$$I_{local} = I_a k_a + \sum_{i=1}^{M} k_{i,shadow}\, I_{i,source}[k_d(\boldsymbol{N} \cdot \boldsymbol{L}_i) + k_r(\boldsymbol{R}_i \cdot \boldsymbol{V})^n]$$

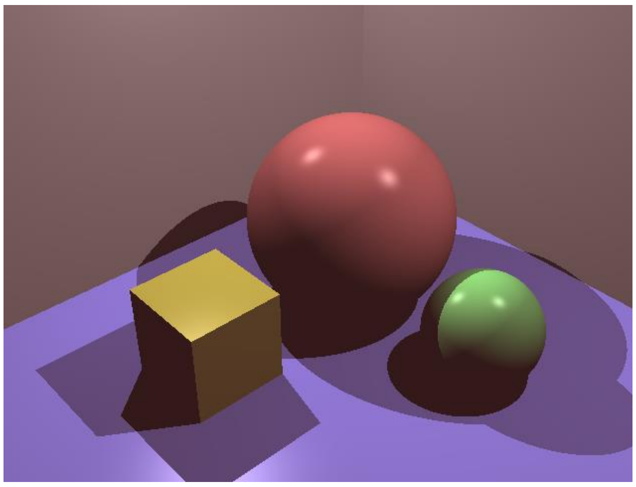
and *M* is the number of point light sources in the scene.

The relevant files to study first are **`Vector3d.h`**, **`Color.h`**, **`Ray.h`**, **`Material.h`**, **`Light.h`**, **`Surface.h`**, **`Scene.h`** and **`Raytrace.h`**.

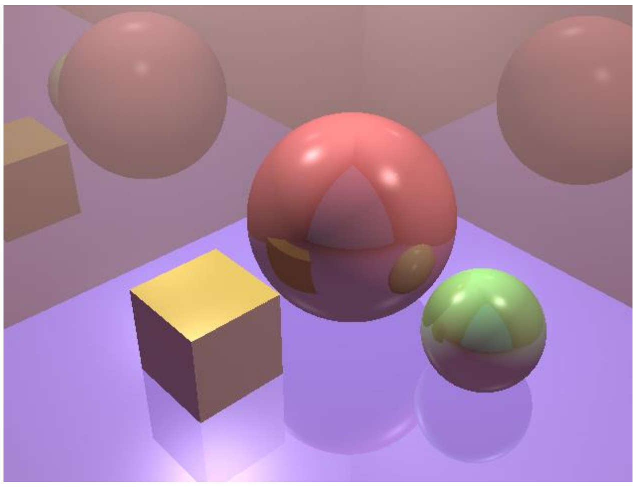
For this task, you have to **submit** your completed **`Raytrace.cpp`**, and the images generated by the program, which should look like the followings. There are **6 images** you need to submit, and you must name them as shown at the bottom of each image shown below.



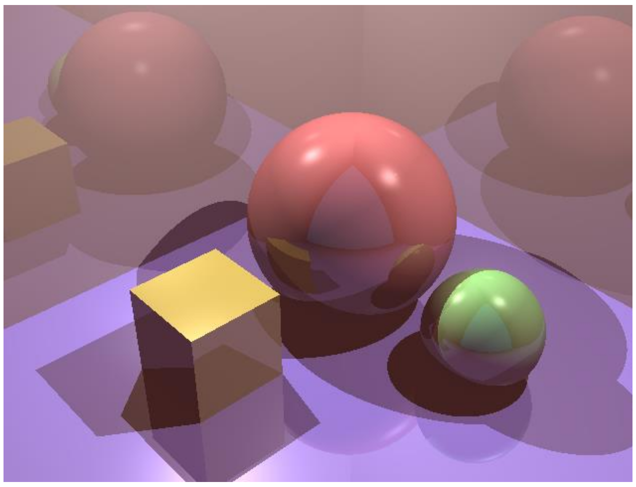
img_r0.png
reflectLevels = 0
hasShadow = false



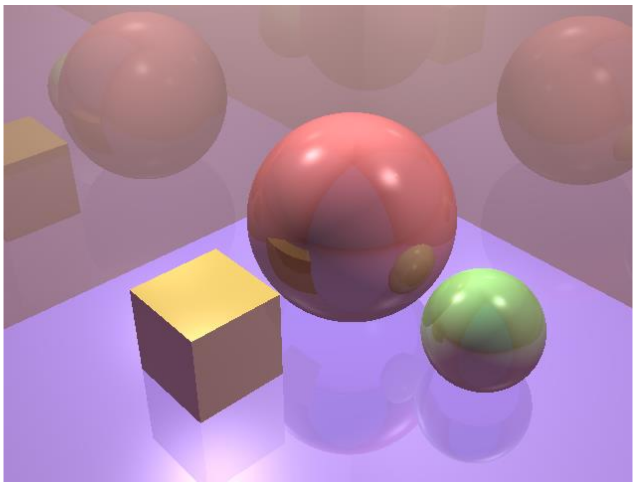
img_r0s.png
reflectLevels = 0
hasShadow = true



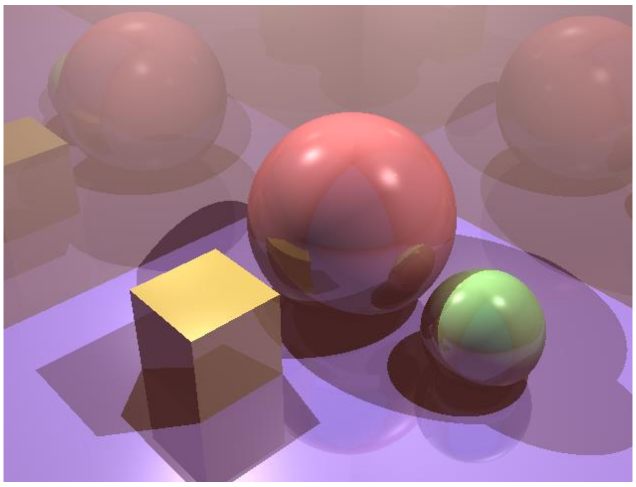
img_r1.png
reflectLevels = 1
hasShadow = false



img_r1s.png
reflectLevels = 1
hasShadow = true



img_r2.png
reflectLevels = 2
hasShadow = false



img_r2s.png
reflectLevels = 2
hasShadow = true

Each of these images should take **less than 10 seconds** to produce. On my laptop, **`img_r2s.png`** (the most time-consuming) took about 1 second. If you are using Microsoft Visual Studio, make sure you compile your program using the **Release** configuration.

Task 3

You are to complete the **`DefineScene2()`** function in **`Main.cpp`** to model a new scene for rendering. You must write your code only in places marked "**`WRITE YOUR CODE HERE`**".

You must use all the surface primitive types, namely, plane, sphere, and triangle, in your scene model. Your scene need not be complex. It will be assessed by the aesthetics of the rendered image, which should be produced with `reflectLevels=2` and `hasShadow=true`, and an image resolution of **640x480**. Name your image file **`img_scene2.png`**.

For this task, you have to **submit** your completed **`Main.cpp`**, and the generated image **`img_scene2.png`**.

GRADING

The maximum marks for this programming assignment is **100**, and it constitutes **8%** of your total marks for the module. The marks are allocated as follows:

- **Task 1 — 30 marks**
- **Task 2 — 50 marks**
- **Task 3 — 20 marks**

Note that marks will be deducted for bad coding style. If your program cannot be compiled and linked, you get 0 (zero) mark.

Good coding style. Comment your code adequately, use meaningful names for functions and variables, and indent your code properly. You must fill in your **Name**, **Student Number**, and **NUS email address** in the **header comment**.

SUBMISSION

For this assignment, you need to **submit only the following 11 files**:

- `Sphere.cpp` and `img_spheres.png`,
- `Raytrace.cpp` and `img_r0.png`, `img_r0s.png`, `img_r1.png`, `img_r1s.png`, `img_r2.png`, `img_r2s.png`,
- `Main.cpp` and `img_scene2.png`.

You must put it/them in a ZIP file and name your ZIP file ***your-student-number*****_lab4.zip**. For example, **A0123456X_lab4.zip**. All letters in your student number must be capitalized.

Submit your ZIP file to the **Lab 4 Submissions / Group T0*x*** folder in LumiNUS Files, where **T0*x*** is your officially allocated Tutorial group number. Before the submission deadline, you may upload your ZIP file as many times as you want to the correct folder. **We will take only your latest submission.** Once you have uploaded a new version to the folder, you **must delete the old versions**.

DEADLINE

Late submissions will NOT be accepted. The submission folder will automatically close at the deadline.

BUILDING THE PROJECT

For Windows:

You can just open the given Visual Studio 2017 solution file **Lab4.sln**. The executable **Lab4.exe** will be produced in the **Debug** or **Release** folder, depending on whether you have used the **Debug** or **Release** configuration when you built your project.

Installing Visual Studio

If you do not have Visual Studio already installed on your computer, you can download (from https://visualstudio.microsoft.com/downloads/) and install Visual Studio Community 2017 or 2019. For NUS students, you can also go to Microsoft's **Azure Dev Tools for Teaching** site (https://azureforeducation.microsoft.com/devtools) to download and install Visual Studio Enterprise 2017 or 2019. At the website, you need to sign in with your NUS email address.

When you are installing Visual Studio, make sure you include the **Desktop development with C++** module/workload. If you have already installed Visual Studio earlier, you can check and install this module by running Visual Studio Installer from the Windows Start Menu, and then clicking on "Modify" under your existing install of Visual Studio.

Note that if you are using the Community edition of Visual Studio, you have to sign in using a Microsoft account (you can create a free account if you do not already have one) before you can use it.

For macOS:

A **CMakeLists.txt** file is provided for you to configure a project to build your program. You will need to download and install CMake. In the following instructions, the `<...>` symbols in the commands are merely *placeholders* for your project directory, so you need to replace them with the actual directory.

1. Download CMake for macOS from the link below and drag the icon to the **Applications** folder.
   https://github.com/Kitware/CMake/releases/download/v3.18.1/cmake-3.18.1-Darwin-x86_64.dmg
2. Add **cmake** to the **PATH** variable, by executing the following command in Terminal:
   ```
   sudo "/Applications/CMake.app/Contents/bin/cmake-gui" --install
   ```
3. If you are using Xcode, execute the following command to generate build files for XCode:
   ```
   cmake "-H<path/to/Lab4_todo>" "-B<path/to/Lab4_todo/build>" -G "Xcode"
   ```
4. Navigate to the generated **build** subfolder and open the **Lab4.xcodeproj** file in Xcode.
5. If you are using another text editor, follow instruction Step **2** under 'For Linux' below to generate build files for **gcc**/**clang** and **make**.

For Linux:

A **CMakeLists.txt** file is provided for you to configure a project to build your program. You will need to download and install CMake. In the following instructions, the `<...>` symbols in the commands are merely *placeholders* for your project directory, so you need to replace them with the actual directory.

1. Download and install **cmake** from your distribution's package manager (e.g. **apt**, **yum**, **pacman**).

2. Execute the following command to generate Unix makefiles:
   ```
   cmake "-H<path/to/Lab4_todo>" "-B<path/to/Lab4_todo/build>" -G "Unix Makefiles"
   ```
3. Open the lab project directory in an editor/IDE of your choice.

——— End of Document ———