Report of Mini-Project #4
Ray Tracing

Peihong Guo

Department of Computer Science and Engineering

December 15, 2011

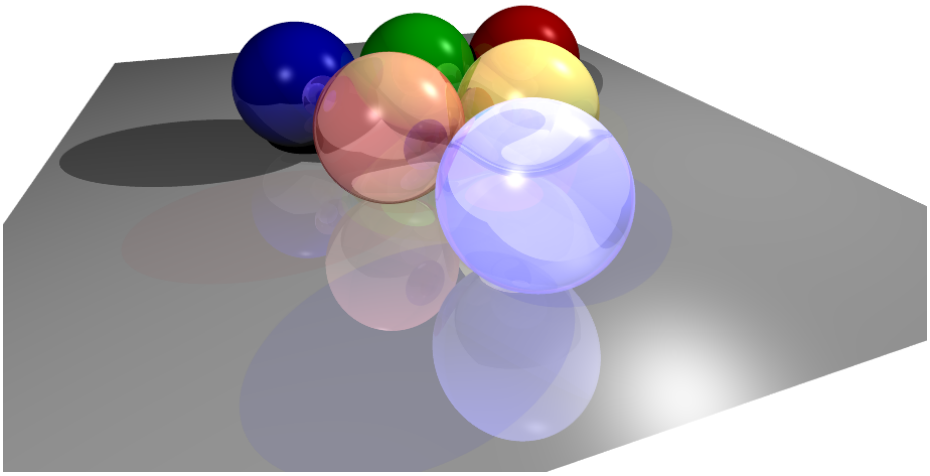


Figure 1: A sample image synthesized with the developed system.

1 Brief Introduction

A ray tracer is implemented in this project, which includes reflection, refraciton, shadow and local lighting features, following the lecture notes [3] and a ray tracing introduction website [2]. A sample image generated by the developed ray tracer is shown in Figure. 1.

2 Implemetation Details

A ray tracer consists of several major modules: scene setup, ray generation and ray-object intersection, and different kinds of lighting effects including local lighting, shadow, reflection and refraction.

2.1 Scene Setup

Generally, a scene contains a camera, a few light sources and a collection of objects. All these components are written in a scene file (.scn file) which are parsed by a scene parser included in the system. All necessary information must present in the scene file, such as camera position and orientation, light sources position and color, objects definitions and properties, etc. A sample scene file is show below.

```
1  .SCENE_INFO
2  SCENE_RANGE -5.0 5.0 -5.0 5.0 -5.0 5.0 # minX, maxX, minY, maxY, minZ, maxZ
3  HAS_BOUNDING_BOX false
4  HAS_GROUND true
5  .CAMERA_INFO
6  CAMERA_POS 0 0 15 #(x, y, z)
7  CAMERA_DIR 0 0 -15
8  CAMERA_UP 0 1 0
9  FOCAL_LENGTH 1.25
10 CANVAS_SIZE 2.0 1.5
11 .LIGHT_SOURCES
12 LIGHT_SOURCE_NUMBER 1
13 LIGHT_TYPE POINT
14 LIGHT_COLOR 1 1 1 1
15 LIGHT_POS -4 4 0 #(x, y, z)
16 .SHAPES
17 SHAPE_NUMBER 1
18 SHAPE sphere
19 COLOR 0.55 0.75 0.025 0.125
20 PARAMETERS 0.0 0 0 3.0 #(x, y, z) radius
21 REFRACTION_RATE 1.3
```

2.2 Ray Generation and Ray-Object Intersection

Eye rays are generated with perspective viewing: each eye ray are shoot from the camera position P_{camera} to a specified canvas position P_{canvas}, resulting in a ray whose origin is at P_{canvas} and oriented in the direction $\vec{d}_{ray} = P_{canvas} - P_{camera}$.

Camera information is given following the routines in OpenGL. Given orientation and focal length f of the camera, the canvas plane is first determined as a plane perpendicular to camera viewing direction d_{camera}. An *up* vector is also given in the scene file as part of the camera's orientaion information, which is used to determine two orthogonal directions, $\vec{u} and \vec{v}$, of the canvas plane. Canvas size then gives information of the visible angle of the camera.

The canvas center P_{cc} can be calculated with $P_{cc} = P_{camera} + f\vec{d}_{camera}$. For a canvas with a width of w and a height of h, any point P on the canvas can then be represented with $P = x\vec{u} + y\vec{v}$. With P and P_{camera}, every eye ray can be uniquely determined.

Ray-object intersection is implemented following the equations provided in the lecture notes [3] and thus omitted.

2.3 Local Lighting, Shadow, Reflection and Refraction

Phong lighting model [1] is used for local lighting, which simulates lighting effects with ambient, diffuse and specular light. Ray reflection and refraction are implemented following the equations provided in the lecture notes [3]. However, for refraciton there are two different cases, i.e. ray enters media with higher refraction factor and that with lower refraction facotr. A few adaptation of the provided equation is needed to achieve correct refraction effect.

Shadow can be conviniently introduced by testing if a point is visible from light source or not when evaluating local lighting parameters. However, in cases translucent objects appear, it becomes a little bit complicated since translucent objects do not block the ray completely. A more general approach for shadow generation is to take translucent objects into consideration, alter light color and reduce light intensity if a ray passes through such objects. However this is still a very simple work-around strategy. More realistic results could be achieved by considering a light intensity decay model when travelling through translucent objects.

3 Results

The following presents a few images generated with the developed system.

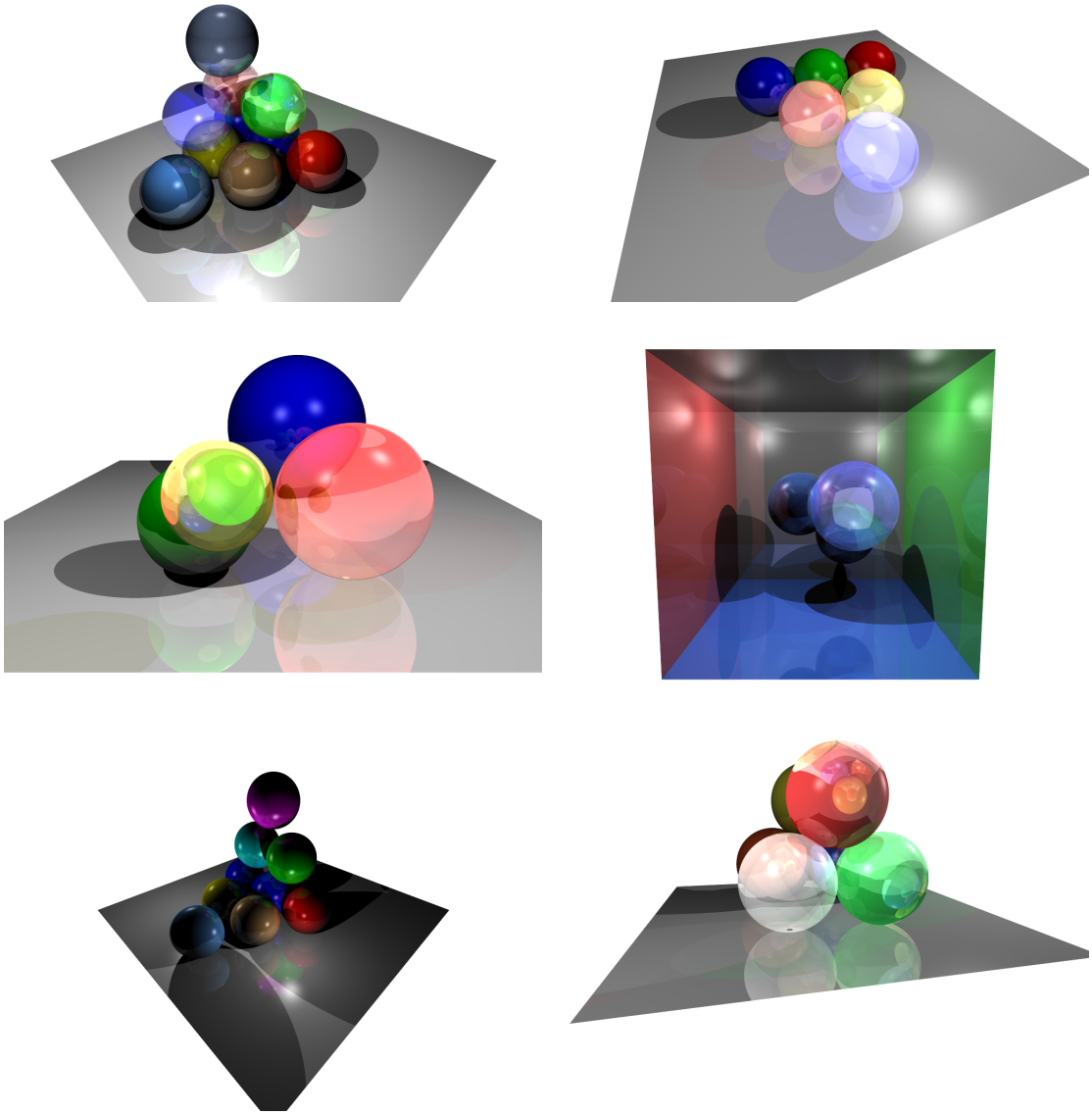
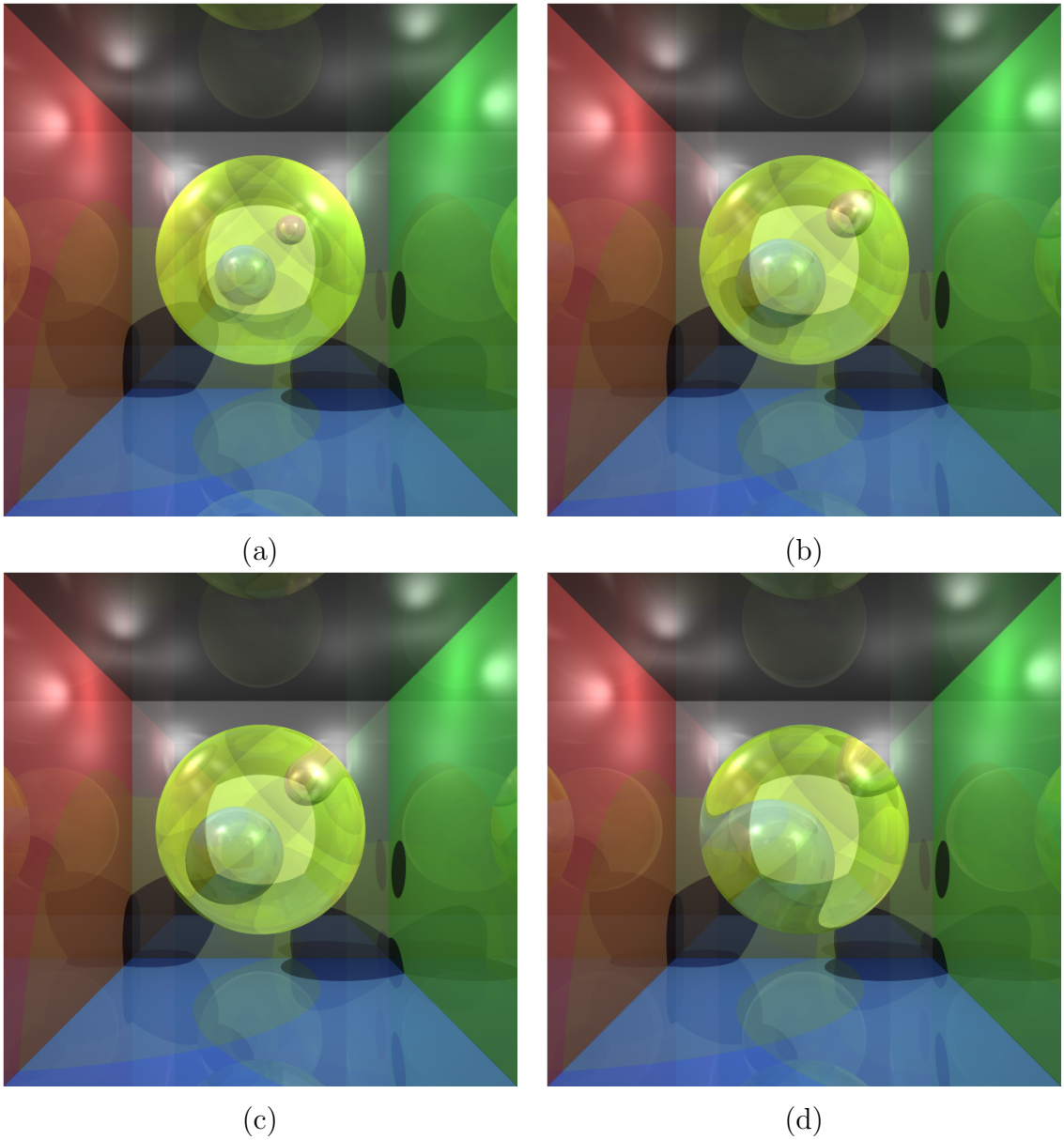


Figure 2: Synthesized images with different scenes.

The following images show a comparison for objects with different refraction factor.

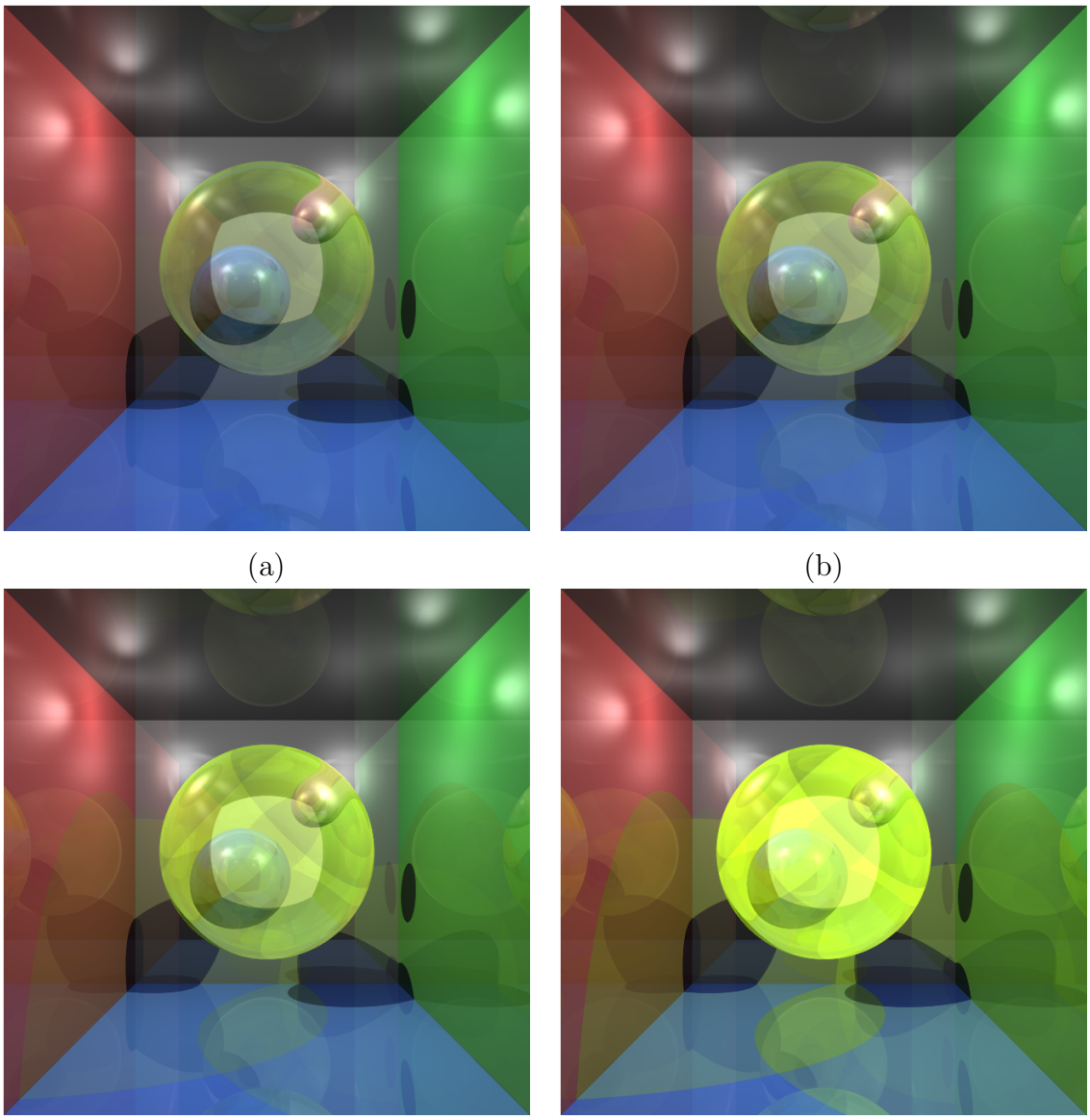


(a) (b)

(c) (d)

Figure 3: Comparison of different refraction factor: (a) 1.00001, (b)1.25, (c)1.3, (d)1.5

The following images show a comparison for objects with different transparency.



(a) (b)

(c) (d)

Figure 4: Comparison of different transparency: (a)0.025, (b)0.05, (c)0.125, (d)0.25

References

[1] Phong reflectance model. [web page] `http://en.wikipedia.org/wiki/Phong_reflection_model`.

[2] Paul Rademacher. Ray tracing: Graphics for the masses. [web page] `http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html`.

[3] Jin xiang Chai. Ray tracing. [web page] `http://faculty.cs.tamu.edu/jchai/csce641/lectures/ray-tracing.ppt`.

Manual

Qt library is required to build the system. To compile the source code, please make sure Qt 4 is correctly installed and configurated. The source code is tested ONLY in 64 bit Linux system.

Compile Procedures:

1. tar xvf *package*.tar.gz
2. cd *package*/
3. qmake (or qmake-qt4)
4. make

Usage: The program provides a GUI for convenience.

- Load scene: Use the "Load Scene" entry in "File" menu, or click on "Load" button in main tool bar.
- Render ray traced image: Click on "Render" button in main tool bar once a scene is loaded.