Chapter 20

An Application: a Simple Raytracing Program

The world is really wild at heart and weird on top
Wild At Heart (Gifford, 1990)

The development of a simple raytracing program is presented in this final chapter. A raytracing program is an excellent realistic graphical application, of sufficient size, in which to apply many of the C++ and object-oriented programming features discussed in previous chapters.

The chapter begins by presenting an overview of the raytracing method and how to implement a raytracing program in C++ using object-oriented programming methodologies. Although the raytracing program introduces several new classes, the program relies heavily on classes developed throughout the book such as Vector, LinkedList, Vector3D, Point, Line, Complex, DIBitmap and World. Thus, the raytracing program demonstrates the importance of both code reusability and good class design for the development of larger programs.

A raytracing program is also a good graphical application in which to illustrate the object-oriented application of classes through graphical object abstraction, inheritance and polymorphism. Numerous new classes are introduced to assist in the implementation of a raytracing program which is designed with the view of being extended. For instance, classes RGBColour and NormalisedRGBColour model an RGB (Red-Green-Blue) colour with associated operations, whereas classes RectWindow, ViewPlane and Screen encapsulate the abstractions of a rectangular window, view plane and display screen so frequently used in computer graphics. A big part of a raytracing program is the ray–object intersections, and the chapter discusses the intersection of a ray for all world objects modelled, namely a plane, polygon, triangle, quadrilateral, tetrahedron, sphere and circle. A Surface class is presented which encapsulates the different components of lighting to produce realistic illumination, reflection and transparency effects. Consideration of simple illumination models, such as specular highlights, can produce visually effective images.

Class World pulls together all the features of the raytracing program by encapsulating data members which model a viewer, view plane and display screen, point light source, background and ambient light, a list of world objects and object and world data files. The chapter concludes by illustrating the raytracing program by means of a simple Windows program which allows a user to generate a raytraced image, generate and display an image, or simply display a previously generated image.
20.1 Why an Application and Why Raytracing?

This chapter discusses the development of a ray tracing program. Although the program is relatively simple it nevertheless models a realistic and popular graphical application. The program is of a sufficiently moderate size to help demonstrate the benefits of adopting an object-oriented approach for larger, more realistic programs – when object-oriented programming really comes into its own. It is intended that the ray tracing program will help illustrate the object-oriented features of C++ when applied to a particular problem and help pull together many of the classes developed in previous chapters throughout this book.

The present chapter makes good use of classes presented in previous chapters and illustrates just how much primitive classes can be used time and time again. You will observe in the present chapter that several of the classes presented in previous chapters have required minimal modifications. Good classes are developed over a period of time through real application. Only through real application will flaws in class design surface. For instance, one minor modification which has been introduced in the present chapter is implicit conversion functions between Point and Vector3D objects. The use of conversion functions should be minimised, but is advantageous in the present case so as to eliminate a great deal of explicit class user conversion between Point and Vector3D objects. It could be argued that this need for conversion functions has arisen because the distinction between a point and a vector has been introduced by developing separate Point and Vector3D classes. However, there are subtle distinctions between a point and a vector, and it is nice to be able to exercise these differences when writing program code.

Although a relatively recent technique, ray tracing is nevertheless a very popular one, and hence an excellent graphical application to illustrate the object-oriented features of C++. Ray tracing naturally lends itself to object-oriented design because of the similarity between world objects (such as a viewer, a light source and objects) and class objects and the underlying mechanism of ray tracing. Essentially, ray tracing consists of projecting a ray from a viewer through a view plane and detecting whether or not the ray intersects an object, irrespective of the particular type of object. It is exactly this property of the ray–object intersection algorithms being independent of the image rendering process that allows the ray tracing model to integrate neatly into a geometric class hierarchy.

It is important to note that in developing the ray tracing program that emphasis has not been placed on the behind the scenes algorithms; i.e. the fastest ray–object intersection algorithms available. That is not to say that such algorithms are not important (because they are very important), but rather that emphasis is placed on the object-oriented issues of a ray tracing program. Further, extravagant double precision and actual point intersection are used when float precision and parametric point intersection would generally be adopted. The philosophy behind such decisions is to design the relevant class member functions to be as general as possible so that they are easily transferable to applications other than ray tracing, such as geometric and solid modelling.

Hopefully, the simple ray tracing program presented in the present chapter will help illustrate just how sophisticated and clever ray tracing programs such as POV-Ray are; for instance refer to Wells and Young (1993).

20.2 Recursive Raytracing

The ray tracing method consists of projecting infinitesimally thin rays from a light source into a world domain to compute the light intensity of visible surfaces of objects. Figure 20.1