**CHAPTER 2**

**REQUIREMENT SPECIFICATION**

**2.1 Functional Requirements**

A functional specification (functional specifications document (FSD), functional requirements specification) in systems engineering and software development is a document that specifies the functions that a system or component must perform. A functional specification is the more technical response to a matching requirements document.

A functional specification does not define the inner workings of the proposed system; it does not include the specification of how the system function will be implemented. Instead, it focuses on what various outside agents might "observe" when interacting with the system.

Functional requirement for Lighthouse includes:

* Keyboard Keys: for movement

**2.2 Non-Functional Requirements**

In systems engineering and requirements engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.

The application should produce the informative error messages, if any errors are found in the input program. It should use memory as less as possible, dynamic memory allocation is preferable to accomplish this task.

**2.3 Hardware Requirements**

* Processor of 2.2G Hz or higher speed
* 20MB Hard Disk Space
* 1GB RAM
* 32 Bit color monitor

**2.4 Software Requirements**

* Dev-C++
* Windows-xp/vista/win7 Operating System
* Graphics package available in Dev-C++

**2.5 Software Tools Used**

Graphics programs can be written on any languages including C, C++, JAVA, Visual Studio, and Python using predefined set of library files supported in each language. Using different library function 2D or 3D applications can be developed by embedding these functionalities in any of the language.

**2.5.1 OpenGL**

OpenGL is a low level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn). Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimensional objects such as a sphere, a torus and even a teapot.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs. GLUT simplifies the implementation of programs using OpenGL rendering. The GLUT application programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT routines also take relatively few parameters.

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to simplify the programming tasks, including the following:

* The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. The library is provided as part of every OpenGL implementation.
* The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit. GLUT routines use the prefix glut.

**2.5.2 Dev-C++**

C is a minimalist programming language. Among its design goals were that it could be compiled in a straight forward manner using a relatively simple compiler, provided low-level access to memory, generated only a few machine language instructions for each of its core language elements and did not require extensive run-time support. As a result, C code is suitable for many system programming applications that had traditionally been implemented in assembly language.

Despite its low-level capabilities, the language was designed to encourage machine independent programming.

In order to get graphics in any programming languages, one needs to do one of four things:

* Write a C/C++/JAVA program and embed OpenGL primitives for graphics
* Make calls to the Windows API
* Make calls to the OpenGL subsystem
* Use a graphics library