# Module Guide for Family of Light Models

Sasha Soraine

November 29, 2019

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| Date 1 | 1.0 | Notes |
| Date 2 | 1.1 | Notes |

# 2 Reference Material

This section records information for easy reference.

## 2.1 Abbreviations and Acronyms

| symbol | description |
|---|---|
| AC | Anticipated Change |
| DAG | Directed Acyclic Graph |
| M | Module |
| MG | Module Guide |
| OS | Operating System |
| R | Requirement |
| SC | Scientific Computing |
| SRS | Software Requirements Specification |
| ProgName | Explanation of program name |
| UC | Unlikely Change |
| [etc. —SS] | [... —SS] |

# Contents


**1 Revision History** **i**

**2 Reference Material** **ii**

2.1 Abbreviations and Acronyms . . . ii

**3 Introduction** **1**

**4 Anticipated and Unlikely Changes** **2**

4.1 Anticipated Changes . . . 2

4.2 Unlikely Changes . . . 2

**5 Module Hierarchy** **3**

**6 Connection Between Requirements and Design** **4**

**7 Module Decomposition** **5**

7.1 Hardware Hiding Modules (M1) . . . 5

7.2 Behaviour-Hiding Module . . . 5

7.2.1 Input Parameters Module (M2) . . . 5

7.2.2 Output Parameters Module (M3) . . . 6

7.2.3 Polygon Module (M4) . . . 6

7.2.4 Light Type Module (M7) . . . 6

7.2.5 Colour Module (M5) . . . 6

7.2.6 3D Cartesian Coordinate Module (M??) . . . 6

7.2.7 Polygon Mesh Module (M8) . . . 6

7.2.8 Normal Maps Module (M9) . . . 7

7.2.9 Scene Module (M10) . . . 7

7.2.10 Object Module (M11) . . . 7

7.2.11 Light Source Module (M12) . . . 7

7.2.12 Observer Module (M13) . . . 7

7.2.13 Vector Math Module (M14) . . . 7

7.2.14 Shader Module (M15) . . . 8

7.2.15 Lighting Model Module (M16) . . . 8

7.3 Software Decision Module . . . 8

7.3.1 JSON Module (M17) . . . 8

7.3.2 Rendering Module (M18) . . . 8

7.3.3 Etc. . . . 9

**8 Traceability Matrix** **9**

**9 Use Hierarchy Between Modules** **10**

# List of Tables

1 Module Hierarchy . . . . . 4
2 Trace Between Requirements and Modules . . . . . 9
3 Trace Between Anticipated Changes and Modules . . . . . 10

# List of Figures

1 Use hierarchy among modules . . . . . 10

# 3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

# 4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

## 4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

**AC1:** The specific hardware on which the software is running.

**AC2:** The format of the initial input data.

**AC3:** The format of the output data.

**AC4:** The algorithm for calculating surface normals.

**AC5:** The algorithm for interpolating surface normals.

**AC6:** The algorithm for calculating the final colouring of objects.

**AC7:** The format for representing object shapes.

**AC8:** The format for representing light sources.

## 4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

**UC1:** Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

**UC2:** The format of 3D coordinates and vectors.

# 5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

**M1:** Hardware-Hiding Module

**M2:** Input Parameters Module

**M3:** Output Format Module

**M4:** Polygon Module

**M5:** Colour Module

**M6:** 3D Cartesian Coordinate Module

**M7:** Light Types Module

**M8:** Polygon Mesh Module

**M9:** Normal Map Module

**M10:** Scene Module

**M11:** Object Module

**M12:** Light Source Module

**M13:** Observer Module

**M14:** Vector Math Module

**M15:** Shader Module

**M16:** Lighting Model Module

**M17:** JSON Module

**M18:** Rendering Module

| Level 1 | Level 2 |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Input Parameters Module |
| | Output Format Module |
| | Polygon Module |
| | Colour Module |
| | 3D Cartesian Coordinate Module |
| | Light Type Module |
| | Polygon Mesh Module |
| | Normal Maps Module |
| | Scene Module |
| | Object Module |
| | Light Source Module |
| | Observer Module |
| | Vector Math Module |
| | Shader Module |
| | Lighting Model Module |
| Software Decision Module | JSON Module |
| | Rendering Module |

Table 1: Module Hierarchy

# 6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. In scientific examples, the choice of algorithm could potentially go here, if that is a decision that is exposed by the interface. —SS]

# 7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *ProgName* means the module will be implemented by the ProgName software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

## 7.1 Hardware Hiding Modules (M1)

The following section outlines the hardware hiding modules. Their descriptions follow the format below:

**Secrets:** The data structure and algorithm used to implement the virtual hardware.

**Services:** Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

**Implemented By:** OS

## 7.2 Behaviour-Hiding Module

The following section outlines the behaviour-hiding modules. Their descriptions follow the format below:

**Secrets:** The contents of the required behaviours.

**Services:** Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

**Implemented By:** –

### 7.2.1 Input Parameters Module (M2)

**Secrets:** The format and structure of inputs to the system.

**Services:** Converts information passed to the system into formats readable by the system.

**Implemented By:** [Your Program Name Here]

#### 7.2.2 Output Parameters Module (M3)

**Secrets:** The format and structure of outputs from the system.

**Services:** Converts information from system formats to storage formats for output.

**Implemented By:** [Your Program Name Here]

#### 7.2.3 Polygon Module (M4)

**Secrets:** The structure of a Polygon.

**Services:** Constrains polygons usable in meshes and defines their characteristics and behaviours.

**Implemented By:** [Your Program Name Here]

#### 7.2.4 Light Type Module (M7)

**Secrets:** The types of lights possible in a scene.

**Services:** Defines types of light and their respective behaviours.

**Implemented By:** [Your Program Name Here]

#### 7.2.5 Colour Module (M5)

**Secrets:** The structure of colours.

**Services:** Constrains colours usable in the system.

**Implemented By:** [Your Program Name Here]

#### 7.2.6 3D Cartesian Coordinate Module (M??)

**Secrets:** The structure of a 3D Cartesian Coordainte (point).

**Services:** Constrains valid points in the system.

**Implemented By:** [Your Program Name Here]

#### 7.2.7 Polygon Mesh Module (M8)

**Secrets:** The data structures and algorithms to represent and manipulate polygon meshes.

**Services:** Stores and retrieves data about polygon mesh.

**Implemented By:** —

#### 7.2.8 Normal Maps Module (M9)

**Secrets:** The data structures algorithms to represent and manipulate normal maps.

**Services:** Stores and retrieves data about normal maps associated to polygon meshes.

**Implemented By:** —

#### 7.2.9 Scene Module (M10)

**Secrets:** The structure of a scene.

**Services:** Coordinates the objects, light sources, and observers in the scene. Calculates final lighting based on lighting and shader models.

**Implemented By:** [Your Program Name Here]

#### 7.2.10 Object Module (M11)

**Secrets:** The structure of objects.

**Services:** Converts input data into the data structure used to represent objects.

**Implemented By:** [Your Program Name Here]

#### 7.2.11 Light Source Module (M12)

**Secrets:** The structure of a light source.

**Services:** Converts input data into the data structure used to represent light sources.

**Implemented By:** [Your Program Name Here]

#### 7.2.12 Observer Module (M13)

**Secrets:** The structure of an observer.

**Services:** Converts input data into the data structure used to represent observers.

**Implemented By:** [Your Program Name Here]

#### 7.2.13 Vector Math Module (M14)

**Secrets:** The structure and operations available to Vectors.

**Services:** Creates and manipulates vectors.

**Implemented By:** [Your Program Name Here]

#### 7.2.14 Shader Module (M15)

**Secrets:** The algorithm to calculate object normals.

**Services:** Calculates surface normals for objects, and interpolates them between points on the mesh.

**Implemented By:** [Your Program Name Here]

#### 7.2.15 Lighting Model Module (M16)

**Secrets:** The algorithm to calculate luminous intensity of light at objects.

**Services:** Calculates luminous intensity and final colours based on lighting model.

**Implemented By:** [Your Program Name Here]

### 7.3 Software Decision Module

The following section outlines the software decision modules. Their descriptions follow the format below:

**Secrets:** The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

**Services:** Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

**Implemented By:** –

#### 7.3.1 JSON Module (M17)

**Secrets:** The structure and parsing of JSON.

**Services:** Converts information from the system to formatted JSON for saving, and information from JSON to system readable data for loading.

**Implemented By:** —

#### 7.3.2 Rendering Module (M18)

**Secrets:** The algorithm to convert output files to rendered scene.

**Services:** Converts information from the JSON output files to rendered scene.

**Implemented By:** —

#### 7.3.3 Etc.

# 8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

| **Req.** | **Modules** |
|---|---|
| R1 | M1, M??, M??, M?? |
| R2 | M??, M?? |
| R3 | M?? |
| R4 | M??, M?? |
| R5 | M??, M??, M??, M??, M??, M?? |
| R6 | M??, M??, M??, M??, M??, M?? |
| R7 | M??, M??, M??, M??, M?? |
| R8 | M??, M??, M??, M??, M?? |
| R9 | M?? |
| R10 | M??, M??, M?? |
| R11 | M??, M??, M??, M?? |

Table 2: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC1 | M1 |
| AC2 | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC3 | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |
| AC?? | M?? |

Table 3: Trace Between Anticipated Changes and Modules

# 9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

Figure 1: Use hierarchy among modules

# References