

# Module Interface Specification for ...

Author Name

November 21, 2019

# 1 Revision History

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

## 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [\[give url —SS\]](#)

[\[Also add any additional symbols, abbreviations or acronyms —SS\]](#)

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Revision History</b>                    | <b>i</b>  |
| <b>2</b> | <b>Symbols, Abbreviations and Acronyms</b> | <b>ii</b> |
| <b>3</b> | <b>Introduction</b>                        | <b>1</b>  |
| <b>4</b> | <b>Notation</b>                            | <b>1</b>  |
| <b>5</b> | <b>Module Decomposition</b>                | <b>2</b>  |
| <b>6</b> | <b>MIS of Input Parameters Module</b>      | <b>3</b>  |
| 6.1      | Module . . . . .                           | 3         |
| 6.2      | Uses . . . . .                             | 3         |
| 6.3      | Syntax . . . . .                           | 3         |
| 6.3.1    | Exported Constants . . . . .               | 3         |
| 6.3.2    | Exported Access Programs . . . . .         | 3         |
| 6.4      | Semantics . . . . .                        | 3         |
| 6.4.1    | State Variables . . . . .                  | 3         |
| 6.4.2    | Environment Variables . . . . .            | 3         |
| 6.4.3    | Assumptions . . . . .                      | 3         |
| 6.4.4    | Access Routine Semantics . . . . .         | 4         |
| 6.4.5    | Local Functions . . . . .                  | 4         |
| <b>7</b> | <b>MIS of Point3D</b>                      | <b>5</b>  |
| 7.1      | Module . . . . .                           | 5         |
| 7.2      | Uses . . . . .                             | 5         |
| 7.3      | Syntax . . . . .                           | 5         |
| 7.3.1    | Exported Constants . . . . .               | 5         |
| 7.3.2    | Exported Access Programs . . . . .         | 5         |
| 7.4      | Semantics . . . . .                        | 5         |
| 7.4.1    | State Variables . . . . .                  | 5         |
| 7.4.2    | Environment Variables . . . . .            | 5         |
| 7.4.3    | Assumptions . . . . .                      | 6         |
| 7.4.4    | Access Routine Semantics . . . . .         | 6         |
| 7.4.5    | Local Functions . . . . .                  | 6         |
| <b>8</b> | <b>MIS of Colour</b>                       | <b>7</b>  |
| 8.1      | Module . . . . .                           | 7         |
| 8.2      | Uses . . . . .                             | 7         |
| 8.3      | Syntax . . . . .                           | 7         |
| 8.3.1    | Exported Constants . . . . .               | 7         |
| 8.3.2    | Exported Access Programs . . . . .         | 7         |

|           |                                    |           |
|-----------|------------------------------------|-----------|
| 8.4       | Semantics . . . . .                | 7         |
| 8.4.1     | State Variables . . . . .          | 7         |
| 8.4.2     | Environment Variables . . . . .    | 7         |
| 8.4.3     | Assumptions . . . . .              | 8         |
| 8.4.4     | Access Routine Semantics . . . . . | 8         |
| 8.4.5     | Local Functions . . . . .          | 9         |
| <b>9</b>  | <b>MIS of Vector</b>               | <b>10</b> |
| 9.1       | Module . . . . .                   | 10        |
| 9.2       | Uses . . . . .                     | 10        |
| 9.3       | Syntax . . . . .                   | 10        |
| 9.3.1     | Exported Constants . . . . .       | 10        |
| 9.3.2     | Exported Access Programs . . . . . | 10        |
| 9.4       | Semantics . . . . .                | 10        |
| 9.4.1     | State Variables . . . . .          | 10        |
| 9.4.2     | Environment Variables . . . . .    | 10        |
| 9.4.3     | Assumptions . . . . .              | 11        |
| 9.4.4     | Access Routine Semantics . . . . . | 11        |
| 9.4.5     | Local Functions . . . . .          | 11        |
| <b>10</b> | <b>MIS of Light Type</b>           | <b>12</b> |
| 10.1      | Module . . . . .                   | 12        |
| 10.2      | Uses . . . . .                     | 12        |
| 10.3      | Syntax . . . . .                   | 12        |
| 10.3.1    | Exported Constants . . . . .       | 12        |
| 10.3.2    | Exported Access Programs . . . . . | 12        |
| 10.4      | Semantics . . . . .                | 12        |
| 10.4.1    | State Variables . . . . .          | 12        |
| 10.4.2    | Environment Variables . . . . .    | 12        |
| 10.4.3    | Assumptions . . . . .              | 12        |
| 10.4.4    | Access Routine Semantics . . . . . | 12        |
| 10.4.5    | Local Functions . . . . .          | 13        |
| <b>11</b> | <b>MIS of Polygon</b>              | <b>14</b> |
| 11.1      | Module . . . . .                   | 14        |
| 11.2      | Uses . . . . .                     | 14        |
| 11.3      | Syntax . . . . .                   | 14        |
| 11.3.1    | Exported Constants . . . . .       | 14        |
| 11.3.2    | Exported Access Programs . . . . . | 14        |
| 11.4      | Semantics . . . . .                | 14        |
| 11.4.1    | State Variables . . . . .          | 14        |
| 11.4.2    | Environment Variables . . . . .    | 15        |
| 11.4.3    | Assumptions . . . . .              | 15        |

|           |                                    |           |
|-----------|------------------------------------|-----------|
| 11.4.4    | Access Routine Semantics . . . . . | 15        |
| 11.4.5    | Local Functions . . . . .          | 15        |
| <b>12</b> | <b>MIS of VecMath</b>              | <b>16</b> |
| 12.1      | Module . . . . .                   | 16        |
| 12.2      | Uses . . . . .                     | 16        |
| 12.3      | Syntax . . . . .                   | 16        |
| 12.3.1    | Exported Constants . . . . .       | 16        |
| 12.3.2    | Exported Access Programs . . . . . | 16        |
| 12.4      | Semantics . . . . .                | 16        |
| 12.4.1    | State Variables . . . . .          | 16        |
| 12.4.2    | Environment Variables . . . . .    | 16        |
| 12.4.3    | Assumptions . . . . .              | 17        |
| 12.4.4    | Access Routine Semantics . . . . . | 17        |
| 12.4.5    | Local Functions . . . . .          | 17        |
| <b>13</b> | <b>MIS of Scene Module</b>         | <b>18</b> |
| 13.1      | Module . . . . .                   | 18        |
| 13.2      | Uses . . . . .                     | 18        |
| 13.3      | Syntax . . . . .                   | 18        |
| 13.3.1    | Exported Constants . . . . .       | 18        |
| 13.3.2    | Exported Access Programs . . . . . | 18        |
| 13.4      | Semantics . . . . .                | 18        |
| 13.4.1    | State Variables . . . . .          | 18        |
| 13.4.2    | Environment Variables . . . . .    | 19        |
| 13.4.3    | Assumptions . . . . .              | 19        |
| 13.4.4    | Access Routine Semantics . . . . . | 19        |
| 13.4.5    | Local Functions . . . . .          | 19        |
| <b>14</b> | <b>MIS of Objects Module</b>       | <b>20</b> |
| 14.1      | Module . . . . .                   | 20        |
| 14.2      | Uses . . . . .                     | 20        |
| 14.3      | Syntax . . . . .                   | 20        |
| 14.3.1    | Exported Constants . . . . .       | 20        |
| 14.3.2    | Exported Access Programs . . . . . | 21        |
| 14.4      | Semantics . . . . .                | 22        |
| 14.4.1    | State Variables . . . . .          | 22        |
| 14.4.2    | Environment Variables . . . . .    | 22        |
| 14.4.3    | Assumptions . . . . .              | 22        |
| 14.4.4    | Access Routine Semantics . . . . . | 22        |
| 14.4.5    | Local Functions . . . . .          | 25        |

|   |           |
|---|-----------|
| <b>15 MIS of Light Source Module</b>      | <b>26</b> |
| 15.1 Module . . . . .                     | 26        |
| 15.2 Uses . . . . .                       | 26        |
| 15.3 Syntax . . . . .                     | 26        |
| 15.3.1 Exported Constants . . . . .       | 26        |
| 15.3.2 Exported Access Programs . . . . . | 26        |
| 15.4 Semantics . . . . .                  | 26        |
| 15.4.1 State Variables . . . . .          | 26        |
| 15.4.2 Environment Variables . . . . .    | 26        |
| 15.4.3 Assumptions . . . . .              | 26        |
| 15.4.4 Access Routine Semantics . . . . . | 27        |
| 15.4.5 Local Functions . . . . .          | 27        |
| <b>16 MIS of Observer Module</b>          | <b>27</b> |
| 16.1 Module . . . . .                     | 27        |
| 16.2 Uses . . . . .                       | 27        |
| 16.3 Syntax . . . . .                     | 27        |
| 16.3.1 Exported Constants . . . . .       | 27        |
| 16.3.2 Exported Access Programs . . . . . | 27        |
| 16.4 Semantics . . . . .                  | 27        |
| 16.4.1 State Variables . . . . .          | 27        |
| 16.4.2 Environment Variables . . . . .    | 28        |
| 16.4.3 Assumptions . . . . .              | 28        |
| 16.4.4 Access Routine Semantics . . . . . | 28        |
| 16.4.5 Local Functions . . . . .          | 28        |
| <b>17 Appendix</b>                        | <b>30</b> |

### 3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at . . . . [provide the url for your repo —SS]

### 4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from ?, with the addition that template modules have been adapted from ?. The mathematical notation comes from Chapter 3 of ?. For instance, the symbol  $:=$  is used for a multiple assignment statement and conditional rules follow the form  $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$ .

The following table summarizes the primitive data types used by Program Name.

| Data Type               | Notation     | Description  |
|-------------------------|--------------|--|
| character               | char         | a single symbol or digit   |
| integer                 | $\mathbb{Z}$ | a number without a fractional component in $(-\infty, \infty)$   |
| natural number          | $\mathbb{N}$ | a number without a fractional component in $[1, \infty)$   |
| real                    | $\mathbb{R}$ | any number in $(-\infty, \infty)$  |
| 3D Cartesian Coordinate | Point3D      | A 3-dimensional cartesian coordinate, represented as an (x,y,z)-tuple where all three are $\mathbb{R}$ values          |
| RGB Colour              | Colour       | A 3-tuple represented as (r,g,b)- where all three are $\mathbb{R}$ values  |
| Shape of Object         | Shape        | The abstract shape that an object mesh is classified as. It can be one of the following : sphere, cube, torus, teapot. |
| Polygon Mesh            | Mesh         | Mesh constructed of vertices, edges, and traingle surfaces to create one of the allowed shapes.                        |
| Normal Map of Object    | nMap         | A structure maintaining a list of the normal vectors for the measured points on the mesh.                              |

The specification of Program Name uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of



characters. Tuples contain a list of values, potentially of different types. In addition, Program Name uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

## 5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

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

Table 1: Module Hierarchy

## 6 MIS of Input Parameters Module

??

The Input Parameters Module converts the JSON data from the input file into the objects usable by the system. During this process, the input parameters

### 6.1 Module

Input Parameters

### 6.2 Uses

JSON, Object, Light Source, Observer, Scene

### 6.3 Syntax

#### 6.3.1 Exported Constants

#### 6.3.2 Exported Access Programs

| Name               | In        | Out                  | Exceptions         |
|--------------------|-----------|----------------------|--------------------|
| convertJSONtoScene | JSON File | s: Scene             | INPUT_INVALID_FILE |
|                    |           | o : Object           | INPUT_FILE_EMPTY   |
|                    |           | l : Light-<br>Source |                    |
|                    |           | v : Ob-<br>server    |                    |

### 6.4 Semantics

#### 6.4.1 State Variables

N/A

#### 6.4.2 Environment Variables

N/A

#### 6.4.3 Assumptions

N/A

#### 6.4.4 Access Routine Semantics

$\text{convertJSONtoScene}(in : JSON)$ :

- output:  $s : Scene, o : Object, l : LightSource, v : Observer | s.Valid(o, l, v)$
- exception: N/A

#### 6.4.5 Local Functions

N/A

## 7 MIS of Point3D

??

The Point3D module captures the structure of a 3D Caretsian Coordinate and functions that are useful for this structure.

### 7.1 Module

Point3D

### 7.2 Uses

-

### 7.3 Syntax

#### 7.3.1 Exported Constants

N/A

#### 7.3.2 Exported Access Programs

| Name         | In   | Out          | Exceptions |
|--------------|--|--------------|------------|
| Point        | $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$ | —            | —          |
| .x           | —  | $\mathbb{R}$ | —          |
| .y           | —  | $\mathbb{R}$ | —          |
| .z           | —  | $\mathbb{R}$ | —          |
| distance_abs | Point3D  | $\mathbb{R}$ | —          |

### 7.4 Semantics

#### 7.4.1 State Variables

$x : \mathbb{R}$

$y : \mathbb{R}$

$z : \mathbb{R}$

#### 7.4.2 Environment Variables

N/A

### 7.4.3 Assumptions

Point3D positions (x,y,z) are only set once (at initialization). This means there will be no individual setter methods.

### 7.4.4 Access Routine Semantics

Point( $Ix : \mathbb{R}, Iy : \mathbb{R}, Iz : \mathbb{R}$ ):

- transition:  $x, y, z := Ix, Iy, Iz$
- exception: N/A

.x():

- output:  $self.x$
- exception: N/A

.y():

- output:  $self.y$
- exception: N/A

.z():

- output:  $self.z$
- exception: N/A

distance\_abs(p:Point3D):

- output:  $\sqrt{(p.x - self.x)^2 + (p.y - self.y)^2 + (p.z - self.z)^2}$
- exception: N/A

### 7.4.5 Local Functions

N/A

## 8 MIS of Colour

??

The Colour module captures the structure of colours used in this program.

### 8.1 Module

Colour

### 8.2 Uses

-

### 8.3 Syntax

#### 8.3.1 Exported Constants

N/A

#### 8.3.2 Exported Access Programs

| Name   | In   | Out            | Exceptions |
|--------|--|----------------|------------|
| Colour | $\mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+$ | —              | —          |
| .r     | —  | $\mathbb{Z}^+$ | —          |
| .g     | —  | $\mathbb{Z}^+$ | —          |
| .b     | —  | $\mathbb{Z}^+$ | —          |
| .set_r | $\mathbb{Z}^+$   | —              | —          |
| .set_g | $\mathbb{Z}^+$   | —              | —          |
| .set_b | $\mathbb{Z}^+$   | —              | —          |

### 8.4 Semantics

#### 8.4.1 State Variables

$r : \mathbb{Z}^+$

$g : \mathbb{Z}^+$

$b : \mathbb{Z}^+$

#### 8.4.2 Environment Variables

N/A

### 8.4.3 Assumptions

- Colours can be changed at any point in time - therefore setters will be needed.
- Colours are represented by RGB values that (individually) range from 0 to 255.

### 8.4.4 Access Routine Semantics

Colour( $Ir : \mathbb{Z}^+, Ig : \mathbb{Z}^+, Ib : \mathbb{Z}^+$ ):

- transition:  $r, g, b := Ir, Ig, Ib$
- exception:  $\text{exc} := (r < 0 \parallel r > 255) \implies \text{INVALID\_R}$   
 $\quad \quad \quad (g < 0 \parallel g > 255) \implies \text{INVALID\_G}$   
 $\quad \quad \quad (b < 0 \parallel b > 255) \implies \text{INVALID\_B}$

.r():

- output:  $\text{self}.r$
- exception: N/A

.g():

- output:  $\text{self}.g$
- exception: N/A

.b():

- output:  $\text{self}.b$
- exception: N/A

.set\_r( $Ir : \mathbb{Z}^+$ ):

- transition:  $r := Ir$
- exception:  $\text{exc} := (r < 0 \parallel r > 255) \implies \text{INVALID\_R}$

.set\_g( $Ig : \mathbb{Z}^+$ ):

- transition:  $g := Ig$
- exception:  $\text{exc} := (g < 0 \parallel g > 255) \implies \text{INVALID\_G}$

.set\_b( $Ib : \mathbb{Z}^+$ ):

- transition:  $b := Ib$
- exception:  $\text{exc} := (b < 0 \parallel b > 255) \implies \text{INVALID\_B}$

#### 8.4.5 Local Functions

N/A



## 9 MIS of Vector

??

The Vector module captures the structure of Vector objects.

### 9.1 Module

Vector

### 9.2 Uses

Point3D ??

### 9.3 Syntax

#### 9.3.1 Exported Constants

N/A

#### 9.3.2 Exported Access Programs

| Name      | In  | Out  | Exceptions |
|-----------|---|--|------------|
| Vector_P  | Point3D ×<br>Point3D  | –  | –          |
| Vector    | $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times$<br>$\mathbb{R}$ | –  | –          |
| .m        |   | $\mathbb{R}$                                     | –          |
| direction |   | $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ | –          |

### 9.4 Semantics

#### 9.4.1 State Variables

$ux := \mathbb{Z}$

$uy := \mathbb{Z}$

$uz := \mathbb{Z}$

$m := \mathbb{R}$

#### 9.4.2 Environment Variables

N/A

### 9.4.3 Assumptions

- Vectors can be created infinitely; we will only set them once during initialization.

### 9.4.4 Access Routine Semantics

$\text{Vector}(p : \text{Point3D}, q : \text{Point3D})$ :

- transition:  $ux := (q.x - p.x)/m$   
 $uy := (q.y - p.y)/m$   
 $uz := (q.z - p.z)/m$   
 $m := p.\text{distance\_abs}(q)$

- exception: –

$\text{Vector}(Ix : \mathbb{Z}, Iy : \mathbb{Z}, Iz : \mathbb{Z}, Im : \mathbb{R})$ :

- transition:  $ux, uy, uz, m := Ix, Iy, Iz, Im$
- exception:  $\text{exc} := (ux < -1 \vee ux > 1) \implies \text{INVALID\_UX}$   
 $(ux < -1 \vee ux > 1) \implies \text{INVALID\_UY}$   
 $(ux < -1 \vee ux > 1) \implies \text{INVALID\_UZ}$   
 $(m < 0) \implies \text{INVALID\_M}$

$\text{.m}()$ :

- output:  $\text{self}.m$
- exception: N/A

$\text{direction}()$ :

- output:  $\text{self}.ux, \text{self}.uy, \text{self}.uz$
- exception: N/A

### 9.4.5 Local Functions

N/A

## 10 MIS of Light Type

??

The Light Type module is an abstract data type which captures information related to the different types of light sources.

### 10.1 Module

LightType

### 10.2 Uses

N/A

### 10.3 Syntax

#### 10.3.1 Exported Constants

N/A

#### 10.3.2 Exported Access Programs

| Name  | In | Out                                 | Exceptions |
|-------|----|-------------------------------------|------------|
| .name |    | ambient,point,spotlight,directional |            |
| .i    |    | ??                                  | —          |

### 10.4 Semantics

#### 10.4.1 State Variables

name := ambient, point, spotlight, directional

i := Function that describes how the light intensity changes as a function of distance. Every type of light has an associated function - so this should really be a set of functions.

#### 10.4.2 Environment Variables

N/A

#### 10.4.3 Assumptions

#### 10.4.4 Access Routine Semantics

.name():

- output: *self.name*

- exception: N/A

.i():

- output: *self.i*
- exception: N/A

#### **10.4.5 Local Functions**

N/A

# 11 MIS of Polygon

??

The Polygon module is an abstract data type captures the structure of polygons used in polygon meshes.

## 11.1 Module

Polygon

## 11.2 Uses

Point3d ??

Vector ??

## 11.3 Syntax

### 11.3.1 Exported Constants

N/A

### 11.3.2 Exported Access Programs

| Name     | In   | Out                      | Exceptions |
|----------|--|--------------------------|------------|
| Polygon  | {triangle, quad}<br>$\times (Point3D, Vector)^n$ | –                        | –          |
| .shape   | –  | {triangle, quad}         | –          |
| .bounds  | –  | Set of (Point3D, Vector) | –          |
| .s_norm  | –  | Vector                   | –          |
| getEdges | Point3D  | Set of Vectors           | –          |

## 11.4 Semantics

### 11.4.1 State Variables

shape := {triangle, quad} bounds := Set of (Point3D, Vector) tuples s\_norm := Vector

### 11.4.2 Environment Variables

N/A

### 11.4.3 Assumptions

### 11.4.4 Access Routine Semantics

Polygon( $t : \{triangle, quad\}, (p : Point3D, v : Vector)^n$ ):

- transition:  $shape := t$ ;  
 $bounds := \cup(p, v)$   
 $s\_norm := Calculatenormascross - productoftwovectorsfrom1vertex$ .
- exception:  $exc :=$

.shape():

- output:  $self.shape$
- exception: N/A

.bounds():

- output:  $self.bounds$
- exception: N/A

.s\_norm():

- output:  $self.s\_norm$
- exception: N/A

getEdges(p:Point3D): This method retrieves all the edges that are connected to the vertex represented by Point3D p. Individual polygons should have a maximum of two edges per vertex based on the polygon assumptions.

- output:  $SetofVectors := \forall b : (Point3D, Vector) | (b \in self.bounds \wedge b[0] == p) \implies \cup b[1]$
- exception: N/A

### 11.4.5 Local Functions

sizeOfBounds  $\equiv$  Number of elements in the set of (Point3D, Vector) tuples.

## 12 MIS of VecMath

??

The Vector Math module is a library of services that can be used with Vectors. All functions here take in 2 Vectors and output either a Vector or a scalar value.

### 12.1 Module

VecMath

### 12.2 Uses

Vector ??

### 12.3 Syntax

#### 12.3.1 Exported Constants

N/A

#### 12.3.2 Exported Access Programs

| Name         | In                    | Out          | Exceptions |
|--------------|-----------------------|--------------|------------|
| add          | Vector × Vector       | Vector       | –          |
| sclMult      | Vector × $\mathbb{R}$ | Vector       | –          |
| dot          | Vector × Vector       | $\mathbb{R}$ | –          |
| cross        | Vector × Vector       | Vector       | –          |
| angleBetween | Vector × Vector       | rad          | –          |

### 12.4 Semantics

#### 12.4.1 State Variables

#### 12.4.2 Environment Variables

N/A

### 12.4.3 Assumptions

### 12.4.4 Access Routine Semantics

$\text{add}(v1 : \text{Vector}, v2 : \text{Vector})$ :

- output:  $\text{Vector}((v1.x+v2.x), (v1.y+v2.y), (v1.z, v2.z), \sqrt{(v1.x + v2.x)^2 + (v1.y + v2.y)^2 + (v1.z, v2.z)^2})$
- exception:  $\text{exc} :=$

$\text{sclMult}(v1 : \text{Vector}, r : \mathbb{R})$ :

- output:  $\text{ux} := r \times v1.x$   
 $\text{uy} := r \times v1.y$   
 $\text{uz} := r \times v1.z$
- exception:  $\text{exc} :=$

$\text{dot}(v1 : \text{Vector}, v2 : \text{Vector})$ :

- output:  $\text{ux} := v1.x \times v2.x$   
 $\text{uy} := v1.y \times v2.y$   
 $\text{uz} := v1.z \times v2.z$
- exception:  $\text{exc} :=$

$\text{cross}(v1 : \text{Vector}, v2 : \text{Vector})$ :

- output:  $\text{ux} := (v1.y \times v2.z) - (v1.z \times v2.y)$   
 $\text{uy} := (v1.z \times v2.x) - (v1.x \times v2.z)$   
 $\text{uz} := (v1.x \times v2.y) - (v1.y \times v2.x)$
- exception:  $\text{exc} :=$

### 12.4.5 Local Functions

N/A



## 13 MIS of Scene Module

??

The Scene Module is an abstract object module that contains the structure for the overall scene. It maintains information about the entities in the scene (object, light source, observer) regarding their distances between each other. It constrains the positions, sizes, and directions of entities based on the specified size of the scene.

### 13.1 Module

Scene

### 13.2 Uses

Input,

### 13.3 Syntax

#### 13.3.1 Exported Constants

$SCENE\_MAX\_X : \mathbb{R}$

$SCENE\_MIN\_X : \mathbb{R}$

$SCENE\_MAX\_Y : \mathbb{R}$

$SCENE\_MIN\_Y : \mathbb{R}$

$SCENE\_MAX\_Z : \mathbb{R}$

$SCENE\_MIN\_Z : \mathbb{R}$

#### 13.3.2 Exported Access Programs

| Name      | In  | Out | Exceptions |
|-----------|---|-----|------------|
| initScene | $max\_X : \mathbb{R}$<br>$max\_Y : \mathbb{R}$<br>$max\_Z : \mathbb{R}$<br>$o : Object$ |     |            |

### 13.4 Semantics

#### 13.4.1 State Variables

N/A

### **13.4.2 Environment Variables**

N/A

### **13.4.3 Assumptions**

N/A

### **13.4.4 Access Routine Semantics**

### **13.4.5 Local Functions**

N/A

## 14 MIS of Objects Module

The Objects Module is an abstract object module that contains the structure for objects to be lit. This includes fields and methods associated with these objects. This module will not be accessed by the user; it will be used extensively by other modules in the system to find data about the objects in the scene, and to manipulate their data.

### 14.1 Module

Objects

### 14.2 Uses

Input,

### 14.3 Syntax

#### 14.3.1 Exported Constants

*SCENE\_MAX\_X* :  $\mathbb{R}$

*SCENE\_MIN\_X* :  $\mathbb{R}$

*SCENE\_MAX\_Y* :  $\mathbb{R}$

*SCENE\_MIN\_Y* :  $\mathbb{R}$

*SCENE\_MAX\_Z* :  $\mathbb{R}$

*SCENE\_MIN\_Z* :  $\mathbb{R}$

### 14.3.2 Exported Access Programs

| Name              | In  | Out          | Exceptions       |
|-------------------|---|--------------|------------------|
| InitObj           | <i>type</i> :<br><i>Shape</i> ,<br><i>mesh</i> :<br><i>Mesh</i> ,<br><i>position</i> :<br><i>Point3D</i> ,<br><i>size</i> : $\mathbb{Z}$ ,<br><i>base</i> :<br><i>Colour</i> ,<br><i>spec</i> :<br><i>Colour</i> ,<br><i>kd</i> : $\mathbb{Z}$ ,<br><i>ka</i> : $\mathbb{Z}$ ,<br><i>ks</i> : $\mathbb{Z}$ ,<br><i>alpha</i> : $\mathbb{N}$ ,<br><i>nmap</i> :<br><i>NMap</i> |              |                  |
| GetObj_Type       | -   | Shape        | -                |
| GetObj_Mesh       | -   | Mesh         | -                |
| GetObj_Position   | -   | Point3D      | -                |
| GetObj_Size       | -   | $\mathbb{Z}$ |                  |
| GetObj_BaseColour | -   | Colour       |                  |
| GetObj_SpecColour | -   | Colour       |                  |
| GetObj_kd         | -   | $\mathbb{Z}$ |                  |
| GetObj_ka         | -   | $\mathbb{Z}$ |                  |
| GetObj_ks         | -   | $\mathbb{Z}$ |                  |
| GetObj_alpha      | -   | $\mathbb{N}$ |                  |
| GetObj_NormalMap  | -   | <i>nMap</i>  |                  |
| SetObj_Position   | Point3D   | -            |                  |
| SetObj_Size       | $\mathbb{Z}$  | -            |                  |
| SetObj_BaseColour | Colour  | -            |                  |
| SetObj_SpecColour | Colour  | -            |                  |
| SetObj_kd         | $\mathbb{Z}$  | -            | IV_OUT_OF_BOUNDS |
| SetObj_ka         | $\mathbb{Z}$  | -            | IV_OUT_OF_BOUNDS |
| SetObj_ks         | $\mathbb{Z}$  | -            | IV_OUT_OF_BOUNDS |
| SetObj_alpha      | $\mathbb{Z}$  | -            | IV_OUT_OF_BOUNDS |
| SetObj_NormalMap  | <i>nMap</i>   | -            | -                |

## 14.4 Semantics

### 14.4.1 State Variables

N/A

### 14.4.2 Environment Variables

N/A

### 14.4.3 Assumptions

N/A

### 14.4.4 Access Routine Semantics

$\text{InitObj}(\text{type} : \text{Shape}, \text{mesh} : \text{Mesh}, \text{position} : \text{Point3D}, \text{size} : \mathbb{Z}, \text{base} : \text{Colour}, \text{spec} : \text{Colour}, \text{kd} : \mathbb{Z}, \text{ka} : \mathbb{Z}, \text{ks} : \mathbb{Z}, \text{alpha} : \mathbb{N}, \text{nmap} : \text{NMap})$ :

- transition: New object created with these properties.
- exception: N/A

$\text{GetObj\_Type}()$ :

- output:  $s : \text{Shape}$ . The shape of the object.
- exception: N/A

$\text{GetObj\_Mesh}()$ :

- output:  $m : \text{Mesh}$ . The mesh of the object.
- exception: N/A

$\text{GetObj\_Position}()$ :

- output:  $\text{centre\_point} : \text{Point3D}$ . The centre point of the object.
- exception: N/A

$\text{GetObj\_Size}()$ :

- output:  $\text{size} : \mathbb{Z}$ . The size of the object; this is the value that scales the polygon mesh up or down from the base model.
- exception: N/A

$\text{GetObj\_BaseColour}()$ :

- output:  $b : \textit{Colour}$ . The base colour of the object. This is the colour that would come through if the object is not specular or diffuse.
- exception: N/A

GetObj\_SpecColour():

- output:  $spec : \textit{Colour}$ . The specular colour of the object.
- exception: N/A

GetObj\_kd():

- output:  $kd : \mathbb{Z}$ . The diffuse coefficient.
- exception: N/A

GetObj\_ka():

- output:  $ka : \mathbb{Z}$ . The ambient coefficient.
- exception: N/A

GetObj\_ks():

- output:  $ks : \mathbb{Z}$ . The specular coefficient.
- exception: N/A

GetObj\_alpha():

- output:  $a : \mathbb{Z}$ . The shininess coefficient of the object.
- exception: N/A

GetObj\_NormalMap():

- output: A normal map of the object. This is a list of normals based on shader calculations, and a string literal that describes the type of normals (vertex, surface, pixel).
- exception: N/A

SetObj\_Type(Colour (r,g,b)):

- output: –

- exception:  $err :=$   
 $Colour.r > 255 \implies IV\_OUT\_OF\_BOUNDS$   
 $|$   
 $Colour.g > 255 \implies IV\_OUT\_OF\_BOUNDS$   
 $|$   
 $Colour.b > 255 \implies IV\_OUT\_OF\_BOUNDS$   
 $|$   
 $Colour.r < 1 \implies IV\_OUT\_OF\_BOUNDS$   
 $|$   
 $Colour.g < 1 \implies IV\_OUT\_OF\_BOUNDS$   
 $|$   
 $Colour.b < 1 \implies IV\_OUT\_OF\_BOUNDS$

SetObj\_Position(Point3D (x,y,z)):

- output: -
- exception: N/A

GetObj\_Size():

- output:  $size : \mathbb{Z}$ . The size of the object; this is the value that scales the polygon mesh up or down from the base model.
- exception: N/A

GetObj\_BaseColour():

- output:  $b : Colour$ . The base colour of the object. This is the colour that would come through if the object is not specular or diffuse.
- exception: N/A

GetObj\_SpecColour():

- output:  $spec : Colour$ . The specular colour of the object.
- exception: N/A

GetObj\_kd():

- output:  $kd : \mathbb{Z}$ . The diffuse coefficient.
- exception: N/A

GetObj\_ka():

- output:  $ka : \mathbb{Z}$ . The ambient coefficient.

- exception: N/A

GetObj\_ks():

- output:  $ks : \mathbb{Z}$ . The specular coefficient.
- exception: N/A

GetObj\_alpha():

- output:  $a : \mathbb{Z}$ . The shininess coefficient of the object.
- exception: N/A

GetObj\_NormalMap():

- output: A normal map of the object. This is a list of normals based on shader calculations, and a string literal that describes the type of normals (vertex, surface, pixel).
- exception: N/A

#### 14.4.5 Local Functions

N/A



## 15 MIS of Light Source Module

The Light Source Module is an abstract object module that contains the structure for light sources in a scene. This includes fields and methods associated with these light sources. This module will not be accessed by the user; it will be used extensively by other modules in the system to find data about the lights in the scene, and to manipulate their data.

### 15.1 Module

Objects

### 15.2 Uses

### 15.3 Syntax

#### 15.3.1 Exported Constants

#### 15.3.2 Exported Access Programs

| Name                   | In  | Out          | Exceptions |
|------------------------|---|--------------|------------|
| InitLight              | <i>type</i> :<br><i>Light, position</i> :<br><i>Point3D, base</i> :<br><i>Colour, intensity</i> :<br>$\mathbb{R}$ |              |            |
| GetLight_Type          | -   | Light        | -          |
| GetLight_Position      | -   | Point3D      | -          |
| GetLight_BaseColour    | -   | Colour       |            |
| GetLight_BaseIntensity | -   | $\mathbb{R}$ |            |
| SetLight_BaseColour    | Colour  | -            |            |
| SetLight_BaseIntensity | $\mathbb{R}$  | -            |            |

### 15.4 Semantics

#### 15.4.1 State Variables

N/A

#### 15.4.2 Environment Variables

N/A

#### 15.4.3 Assumptions

N/A

#### 15.4.4 Access Routine Semantics

InitLight(*type* : *Light*, *position* : *Point3D*, *base* : *Colour*, *intensity* :  $\mathbb{R}$ ):

- transition: Create a new light source in the scene with these properties.
- exception: N/A

#### 15.4.5 Local Functions

N/A

## 16 MIS of Observer Module

The Observer Module is an abstract object module that contains the structure for observers in a scene. This includes fields and methods associated with these observers. This module will not be accessed by the user; it will be used extensively by other modules in the system to find data about the observers in the scene, and to manipulate their data.

### 16.1 Module

Objects

### 16.2 Uses

### 16.3 Syntax

#### 16.3.1 Exported Constants

#### 16.3.2 Exported Access Programs

| Name              | In  | Out     | Exceptions |
|-------------------|---|---------|------------|
| InitObsv          | <i>position</i> :<br><i>Point3D</i> , <i>direction</i> :<br><i>Vec3</i> |         |            |
| GetObsv_Direction | -   | Vec3    | -          |
| GetObsv_Position  | -   | Point3D | -          |
| SetObsv_Direction | -   | Vec3    | -          |
| SetObsv_Position  | -   | Point3D | -          |

### 16.4 Semantics

#### 16.4.1 State Variables

N/A

### 16.4.2 Environment Variables

N/A

### 16.4.3 Assumptions

N/A

### 16.4.4 Access Routine Semantics

InitObsv(*position* : *Point3D*, *direction* : *Vec3*):

- transition: Create a new observer in the scene with these properties.
- exception: N/A

### 16.4.5 Local Functions

N/A

## References

## 17 Appendix

[Extra information if required —SS]