Module Interface Specification for ...

Author Name

November 6, 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

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 $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

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 (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
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	Scene Module Object Module Light Source Module Observer Module Shader Module Lighting Model Module
Software Decision Module	JSON Module Vector Math Module Polygon Mesh Module Normal Maps Module

Table 1: Module Hierarchy

6 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.

6.1 Module

Objects

6.2 Uses

Input,

6.3 Syntax

6.3.1 Exported Constants

$SCENE\_MAX\_X : \mathbb{R}$
$SCENE\_MIN\_X : \mathbb{R}$
$SCENE\_MAX\_Y : \mathbb{R}$
$SCENE\_MIN\_Y : \mathbb{R}$
$SCENE\_MAX\_Z : \mathbb{R}$
$SCENE\_MIN\_Z : \mathbb{R}$

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
InitObj	$type$: $Shape$, $mesh$: $Mesh$, $position$: $Point3D$, $size : \mathbb{Z}$, $base$: $Colour$, $spec$: $Colour$, $kd : \mathbb{Z}$, $ka : \mathbb{Z}$, $ks : \mathbb{Z}$, $alpha : \mathbb{N}$, $nmap$: $NMap$		
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	-	$nMap$	
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	$nMap$	-	-

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

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

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

GetObj_Type():

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

GetObj_Mesh():

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

GetObj_Position():

- output: $centre_point : Point3D$. The centre point of the object.
- 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

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

6.4.5 Local Functions

N/A

7 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.

7.1 Module

Objects

7.2 Uses

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

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

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

N/A

7.4.4 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

7.4.5 Local Functions

N/A

8 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.

8.1 Module

Objects

8.2 Uses

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
InitObsv	$position : Point3D, direction : Vec3$		
GetObsv_Direction	-	Vec3	-
GetObsv_Position	-	Point3D	-
SetObsv_Direction	-	Vec3	-
SetObsv_Position	-	Point3D	-

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

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

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

8.4.5 Local Functions

N/A

References

9 Appendix

[Extra information if required —SS]