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

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	Input Parameters Module Output Format Module Shape Module Colour Module 3D Cartesian Coordinate 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 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 o : Object l : Light-Source v : Observer	INPUT_INVALID_FILE INPUT_FILE_EMPTY

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

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: exc := $(r < 0 || r > 255) \implies INVALID\_R$
  $|(g < 0 || g > 255) \implies INVALID\_G$
  $|(b < 0 || b > 255) \implies INVALID\_B$

.r():

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

.g():

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

.b():

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

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

- transition: $r := Ir$
- exception: exc := $(r < 0 || r > 255) \implies INVALID\_R$

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

- transition: $g := Ig$
- exception: exc := $(g < 0 || g > 255) \implies INVALID\_G$

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

- transition: $b := Ib$
- exception: exc := $(b < 0 || b > 255) \implies INVALID\_B$

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 $\times$ Point3D	–	–
Vector	$\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{R}$	–	–
.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

Vector($p : Point3D, q : Point3D$):

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

- exception: –

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

- transition: $ux, uy, uz, m := Ix, Iy, Iz, Im$

- exception: exc := $(ux < -1 || ux > 1) \implies INVALID\_UX$
  $|(ux < -1 || ux > 1) \implies INVALID\_UY$
  $|(ux < -1 || ux > 1) \implies INVALID\_UZ$
  $|(m < 0) \implies INVALID\_M$

.m():

- output: $self.m$

- exception: N/A

direction():

- output: $self.ux, self.uy, self.uz$

- 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} $\times(Point3D, Vector)^n$	–	–
.shape	–	{triangle, quad}	–
.bounds	–	Set of (Point3D, Vector)	–
.s_norm	–	Vector	–
getEdges	Point3D	Set of Vectors	–
getPoints		Set of Point3D	–

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 := Calculate norm as cross-product of two vectors from 1 vertex.
- 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: Set of Vectors := $\forall b : (Point3D, Vector) | (b \in self.bounds \wedge b[0] == p) \implies \cup b[1]$
- exception: N/A

getPoints(): This method retrieves the set of points in the polygon.

- output: Set of Point3D := $b : (Point3D, Vector) | \forall b \in self.bounds \cup b.[0]$
- exception: N/A

11.4.5 Local Functions

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

12 MIS of Mesh

??
The Mesh module is an abstract data type that captures the structure of polygon meshes as used by this program. It also provides methods to find out basic data about the polygon mesh.

12.1 Module

Mesh

12.2 Uses

Point3d **??**
Vector **??**
VecMath **??**
Polygon **??**

12.3 Syntax

12.3.1 Exported Constants

N/A

12.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
Mesh	Set of Polygons	–	–
.Surfaces	-	Set of Polygons	–
.Edges	-	Set of Vectors	–
.Vertices	-	Set of Point3D	–
isInMesh	Polygon	$\mathbb{B}$	–
numPoly	Point3D	$\mathbb{Z}^+$	–
intersects	Vector	Polygon	–

12.4 Semantics

12.4.1 State Variables

Vertices : Set of Point3D Edges : Set of Vectors Surfaces : Set of Polygons

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

12.4.4 Access Routine Semantics

Mesh($P : SetofPolygons$):

- transition: Surfaces := P
  Vertices := $(p : Polygon | \forall p \in P \cup p.getPoints)$
  (Vertices pulls its values from the bounds of the polygons in P)
  Edges := $(p : Polygon, v : Point3D | \forall p \in P \forall v \in p.getPoints \cup (p.getEdges(v)))$
  (Edges pulls its values from the bounds of the polygons in P)

- exception: exc :=

.Surfaces():

- output := self.Surfaces

- exception: exc :=

.Vertices():

- output := self.Vertices

- exception: exc :=

.Edges():

- output := self.Edges

- exception: exc :=

isInMesh($p : Polygon$):

- output := $(q : Polygon | \exists q \in self.Surfaceswhereq == p)$

- exception: exc :=

numPoly($p : Point3D$):

- output $\equiv$ counter := $p \in self.Vertices \implies (s : Polygon | \forall s \in self.Surfaces)$ if $p \in s.bounds$ then $counter++$

- exception: exc := $\{p \notin self.Vertices \implies ERR\_POINT\_NOT\_IN\_MESH\}$

intersects($r : Vector$):

- output := calculate whether the given vector intersects with any polygon on the mesh, and return the first polygon it intersects with.

- exception: exc :=

12.4.5 Local Functions

N/A

13 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.

13.1 Module

VecMath

13.2 Uses

Vector **??**

13.3 Syntax

13.3.1 Exported Constants

N/A

13.3.2 Exported Access Programs

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

13.4 Semantics

13.4.1 State Variables

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

13.4.4 Access Routine Semantics

add($v1 : Vector, v2 : Vector$):

- output: 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: exc :=

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

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

dot($v1 : Vector, v2 : Vector$):

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

cross($v1 : Vector, v2 : Vector$):

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

angleBetween($v1 : Vector, v2 : Vector$):

- output: $\cos^{-1}(\frac{dot(v1,v2)}{v1.m \times v2.m})$
- exception: exc :=

13.4.5 Local Functions

N/A

14 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.

14.1 Module

Scene

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
initScene	$max\_X : \mathbb{R}$		
	$max\_Y : \mathbb{R}$		
	$max\_Z : \mathbb{R}$		
	$o : Object$		

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

14.4.5 Local Functions

N/A

15 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.

15.1 Module

Objects

15.2 Uses

Input,

15.3 Syntax

15.3.1 Exported Constants

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

15.3.2 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	$nMap$	-	-

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

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

15.4.5 Local Functions

N/A

16 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.

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
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	$\mathbb{R}$	-	

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

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

16.4.5 Local Functions

N/A

17 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.

17.1 Module

Objects

17.2 Uses

17.3 Syntax

17.3.1 Exported Constants

17.3.2 Exported Access Programs

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

17.4 Semantics

17.4.1 State Variables

N/A

17.4.2 Environment Variables

N/A

17.4.3 Assumptions

N/A

17.4.4 Access Routine Semantics

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

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

17.4.5 Local Functions

N/A

References

18 Appendix

[Extra information if required —SS]